

**SPPAS**

# Documentation

(c) 2011-2015 - Brigitte Bigi



# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	What is SPPAS? . . . . .	3
1.1.1	Main features . . . . .	3
1.1.2	Copyright and Licenses . . . . .	3
1.1.3	User engagement . . . . .	4
1.1.4	Need help . . . . .	4
1.1.5	Supports . . . . .	4
1.1.6	Contributors . . . . .	5
1.2	Getting and installing . . . . .	5
1.2.1	Websites . . . . .	5
1.2.2	Dependencies . . . . .	5
1.2.3	Download and install SPPAS . . . . .	5
1.2.4	The SPPAS package . . . . .	6
1.2.5	Update . . . . .	7
1.3	Capabilities . . . . .	7
1.3.1	What SPPAS can do? . . . . .	7
1.3.2	How to use SPPAS? . . . . .	8
1.3.3	Interoperability and compatibility . . . . .	8
1.4	Main and important recommendations . . . . .	10
1.4.1	About files . . . . .	10
1.4.2	About automatic annotations . . . . .	10
<b>2</b>	<b>Automatic Speech Segmentation and Annotation</b>	<b>11</b>
2.1	Introduction . . . . .	11
2.1.1	About this chapter . . . . .	11
2.1.2	File formats and tier names . . . . .	11
2.1.3	Recorded speech . . . . .	12

2.1.4	Orthographic Transcription . . . . .	12
2.2	Inter-Pausal Units (IPUs) segmentation . . . . .	14
2.2.1	Silence/Speech segmentation . . . . .	14
2.2.2	Silence/Speech segmentation time-aligned with a transcription . . . . .	15
2.2.3	Split into multiple files . . . . .	17
2.3	Tokenization . . . . .	17
2.4	Phonetization . . . . .	19
2.5	Alignment . . . . .	20
2.6	Syllabification . . . . .	21
2.7	Repetitions . . . . .	23
2.8	Momel and INTSINT . . . . .	24
2.8.1	Momel (modelling melody) . . . . .	24
2.8.2	Encoding of F0 target points using the “INTSINT” system . . . . .	25
<b>3</b>	<b>Resources for Automatic Annotations</b>	<b>27</b>
3.1	What are SPPAS resources and where they come from? . . . . .	27
3.1.1	Overview . . . . .	27
3.1.2	About the phone sets . . . . .	28
3.1.3	How to add a new language . . . . .	28
3.2	French Resources . . . . .	29
3.2.1	Pronunciation Dictionary . . . . .	29
3.2.2	Acoustic Model . . . . .	29
3.2.3	Syllabification configuration . . . . .	31
3.3	Italian resources . . . . .	31
3.3.1	Pronunciation Dictionary . . . . .	31
3.3.2	Acoustic Model . . . . .	31
3.3.3	Syllabification configuration . . . . .	32
3.4	Spanish resources . . . . .	32
3.4.1	Pronunciation Dictionary . . . . .	32
3.4.2	Acoustic Model . . . . .	32
3.5	Catalan resources . . . . .	34
3.5.1	Pronunciation Dictionary . . . . .	34
3.5.2	Acoustic Model . . . . .	34
3.6	English . . . . .	36
3.6.1	Dictionary . . . . .	36

3.6.2	Acoustic Model . . . . .	36
3.7	Mandarin Chinese . . . . .	37
3.7.1	Pronunciation dictionary . . . . .	38
3.7.2	Acoustic model . . . . .	38
3.8	Southern Min (or Min Nan) resources . . . . .	38
3.9	Cantonese resources . . . . .	38
3.9.1	Dictionary . . . . .	38
3.9.2	Acoustic Model . . . . .	41
3.10	Polish Resources . . . . .	41
3.10.1	Pronunciation Dictionary . . . . .	41
3.10.2	Acoustic Model . . . . .	42
<b>4</b>	<b>Graphical User Interface - GUI</b>	<b>43</b>
4.1	Getting started . . . . .	43
4.1.1	The Tips Frame . . . . .	43
4.1.2	The main frame . . . . .	44
4.2	File List Panel . . . . .	46
4.2.1	The file explorer . . . . .	46
4.2.2	The toolbar . . . . .	48
4.3	Automatic Annotations Panel . . . . .	49
4.3.1	Description . . . . .	49
4.3.2	AAP Usage . . . . .	50
4.3.3	The procedure outcome report . . . . .	50
4.4	Components Panel . . . . .	52
4.4.1	The main toolbar . . . . .	52
4.4.2	The list of files . . . . .	53
4.4.3	DataRoamer . . . . .	53
4.4.4	SndRoamer . . . . .	54
4.4.5	IPUscribe . . . . .	54
4.4.6	SppasEdit . . . . .	54
4.4.7	DataFilter . . . . .	56
4.4.8	Statistics . . . . .	60
4.5	Plugins Panel . . . . .	63
4.5.1	Installing a Plugin . . . . .	64
4.5.2	Marsatag-Plugin . . . . .	64
4.5.3	TierMapping . . . . .	64

<b>5</b>	<b>Command-Line user Interface - CLI</b>	<b>67</b>
5.1	Overview . . . . .	67
5.2	Programs to perform an automatic annotation . . . . .	67
5.2.1	annotation.py . . . . .	68
5.2.2	wavsplit.py . . . . .	68
5.2.3	tokenize.py . . . . .	70
5.2.4	phonetize.py . . . . .	71
5.2.5	alignment.py . . . . .	71
5.2.6	syllabify.py . . . . .	72
5.2.7	repetition.py . . . . .	72
5.2.8	Momel and INTSINT . . . . .	72
5.3	Programs to execute a GUI . . . . .	73
5.3.1	sppasgui.py . . . . .	73
5.3.2	dataroamer.py . . . . .	73
5.3.3	wavplayer.py . . . . .	73
5.3.4	iputranscriber.py . . . . .	73
5.3.5	dataeditor.py . . . . .	73
5.3.6	datafilter.py . . . . .	73
5.3.7	stats.py . . . . .	73
5.4	Other Programs . . . . .	73
5.4.1	Merge annotation files . . . . .	73
5.4.2	Convert annotation files . . . . .	74
5.4.3	Get information about a tier of an annotated file. . . . .	74
5.4.4	extract.py . . . . .	74
5.4.5	reformat.py . . . . .	74
5.4.6	equalize.py . . . . .	75
5.4.7	channelsmixer.py . . . . .	75
5.4.8	channelsmixersimulator.py . . . . .	75
5.4.9	fragmentextracter.py . . . . .	75
5.4.10	audiotoaster . . . . .	76
<b>6</b>	<b>Scripting with Python and SPPAS</b>	<b>77</b>
6.1	Introduction . . . . .	77
6.2	A gentle introduction to programming . . . . .	77
6.2.1	Definitions . . . . .	77

6.2.2	Comments and blocks . . . . .	78
6.2.3	Variables: Assignment and Typing . . . . .	78
6.2.4	Basic Operators . . . . .	79
6.2.5	Conditions . . . . .	79
6.2.6	Loops . . . . .	80
6.3	Scripting with Python . . . . .	80
6.3.1	Hello World! . . . . .	80
6.3.2	Functions . . . . .	82
6.3.3	Reading/Writing files . . . . .	84
6.3.4	Dictionaries . . . . .	87
6.3.5	Exercises to practice . . . . .	87
6.4	The API of SPPAS to manage data . . . . .	87
6.4.1	Overview . . . . .	87
6.4.2	Why developing a new API? . . . . .	88
6.4.3	The API class diagram . . . . .	88
6.5	Creating scripts with the SPPAS API . . . . .	89
6.5.1	Preparing the data . . . . .	89
6.5.2	Read/Write annotated files . . . . .	92
6.5.3	Manipulating a Transcription object . . . . .	93
6.5.4	Manipulating a Tier object . . . . .	94
6.5.5	Main information on Annotation/Location/Label objects . . . . .	95
6.5.6	Exercises . . . . .	95
6.5.7	Search in annotations: Filters . . . . .	95
6.5.8	Exercises . . . . .	98
<b>7</b>	<b>References</b>	<b>99</b>
7.1	Publications about SPPAS . . . . .	99
7.2	SPPAS in research projects . . . . .	101
7.2.1	MULTIPHONIA . . . . .	101
7.2.2	Amennpro . . . . .	102
7.2.3	Evalita 2011: Italian phonetization and alignment . . . . .	102
7.2.4	Orthographic Transcription: Impact on Phonetization . . . . .	102
7.2.5	Cofee: Conversational Feedback . . . . .	103
7.2.6	Variamu: Variations in Action: a MULTilingual approach . . . . .	103





# Introduction

## 1.1 What is SPPAS?

### 1.1.1 Main features

SPPAS - Automatic Annotation of Speech is a scientific computer software package written and maintained by Brigitte Bigi of the Laboratoire Parole et Langage, in Aix-en-Provence, France.

Available for free, with open source code, there is simply no other package for linguists to simple use in automatic segmentation of speech. SPPAS is daily developed with the aim to provide a robust and reliable software for the automatic annotation and for the exploitation of annotated-data.

This documentation will assume that you are using a relatively recent version of SPPAS (1.7.1).

There's no reason not to download the latest version whenever released: it's easy and fast!

### 1.1.2 Copyright and Licenses

*(c) 2011-2015 Brigitte Bigi, Laboratoire Parole et Langage, Aix-en-Provence, France*

SPPAS software is distributed under the terms of the **GNU GENERAL PUBLIC LICENSE**.

SPPAS resources are distributed:

- under the terms of the GNU GENERAL PUBLIC LICENSE, or
- on the terms of the “**Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International Public License**”

A copy of both licenses is available in the package. See the “Resources” chapter for details about the license of each proposed resource.

### 1.1.3 User engagement

By using SPPAS, **you agree to cite references in your publications.**

See the “References” section of this documentation and/or see PDF files included in the package.

### 1.1.4 Need help

1. When looking for more detail about some subject, one can search this documentation. This documentation is available in-line (see the SPPAS website), it is also included in the package (in PDF format) and it can also be explored with the Graphical User Interface by clicking on the ‘Help’ button.
2. Many problems can be solved by updating the version of SPPAS.
3. There is a SPPAS Users discussion group where queries and allied topics are discussed, with responses from colleagues or from the author. Topics can range from elementary “how do I” queries to advanced issues in scriptwriting. There’s (or there will be) something there for everybody. It is recommended to sign up to become a member on the website: <https://groups.google.com/forum/#!forum/sppas-users> (neither spam or e-mails will be sent directly to members).
4. If none of the above helps, you may send e-mail to the author. It is very important to indicate clearly:  
1/ your operating system and its version, 2/ the version of SPPAS (supposed to be the last one), and 3/ for automatic annotations, send the log file, and a sample of the data on which a problem occurs.

And/Or, if you have any question, if you want to contribute to SPPAS either to improve the quality of resources or to help in development, or anything else, do not hesitate to contact the author by e-mail at: [brigitte.bigi@gmail.com](mailto:brigitte.bigi@gmail.com).

### 1.1.5 Supports

#### 2011-2012:

Partly supported by ANR OTIM project (Ref. Nr. ANR-08-BLAN-0239), Tools for Multimodal Information Processing.

Read more at: <http://www.lpl-aix.fr/~otim/>

#### 2013-2015:

Partly supported by ORTOLANG (Ref. Nr. ANR-11-EQPX-0032) funded by the « Investissements d’Avenir » French Government program managed by the French National Research Agency (ANR).

Read more at: <http://www.ortolang.fr/index.php?page=accueil&lang=en>

#### 2014-2015:

SPPAS is also partly carried out thanks to the support of the following projects or groups:

- CoFee - Conversational Feedback <http://cofee.hypotheses.org>
- Variamu - Variations in Action: a MUltilingual approach <http://variamu.hypotheses.org>
- Team C3i of LPL <http://www.lpl-aix.fr/~c3i>

### 1.1.6 Contributors

Here is the list of contributors:

- Since January 2011: Brigitte Bigi is the main author;
- April 2012-June 2012: Alexandre Ranson;
- April 2012-July 2012: Cazembé Henry;
- April 2012-June 2013: Bastien Herbaut;
- March 2013-March 2014: Tatsuya Watanabe;
- April 2015-June 2015: Nicolas Chazeau;
- April 2015-June 2015: Jibril Saffi.

## 1.2 Getting and installing

### 1.2.1 Websites

In the past, SPPAS - Automatic Annotation of Speech, was hosted by “Laboratoire Parole et Langage” (see <http://www.lpl-aix.fr>). SPPAS is hosted by Speech and Language Data Repository (SLDR), since January 2015, and is located at the following URL:

<http://sldr.org/sldr000800/preview>

The source code with recent stable releases is now migrated on github.

<https://github.com/brigittebigi/>

From this website, anyone can download the development version, contribute, send comments and/or declare an issue.

### 1.2.2 Dependencies

On the main website, you will find information about the software requirements. In fact, other programs are required for SPPAS to operate. Of course, they must be installed before using SPPAS, and *only once*. This operation takes from 5 to 15 minutes depending on the operating system. The following software are required:

1. Python, version 2.7.x
2. wxPython >= 3.0
3. julius >= 4.1

An installation guide is available on the website, depending on your operating system. **Please, closely follow the instructions.** Administrator rights are required to perform these installations.

### 1.2.3 Download and install SPPAS

The website lets to go to the Download Page to download a new version or Subscribe to the User’s group.

SPPAS is ready to run, so it does not need elaborate installation, except for its dependencies (other software required for SPPAS to work properly). All you need to do is to copy the SPPAS package from the website

to somewhere on your computer. Preferably, choose *a location without spaces nor accentuated characters in the name of the path*.

The SPPAS package is compressed and zipped, so you will need to *decompress and unpack* it once you've got it.

There is a unique version of SPPAS which does **not depend** on your operating system. The only obvious difference depending on the system is how it looks on the computer screen.



Figure 1.1: Operating systems

#### 1.2.4 The SPPAS package

Unlike many other software, SPPAS is not what is called a “black box”. Instead, everything is done so that users can check / change operation. It is particularly suitable for automatic annotations. It allows any user to adapt automatic annotations to its own needs.

The package of SPPAS is then a folder with content as files and sub-folders.

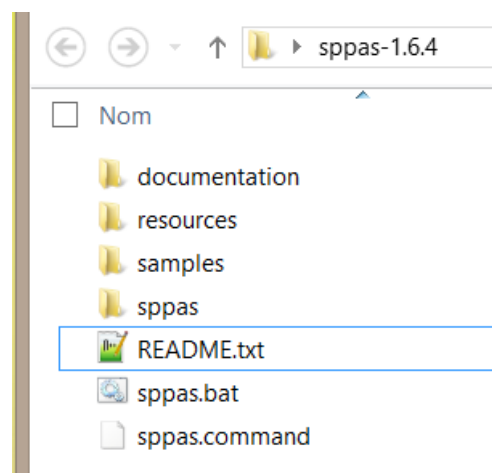


Figure 1.2: SPPAS Package content

The SPPAS package contains:

- the `README.txt` file, which aims to be read by users!
- the files `sppas.bat` and `sppas.command` to execute the Graphical User Interface of SPPAS

- the `resources` used by automatic annotations (lexicons, dictionaries, ...)
- the `samples` are sets of annotations freely distributed to test SPPAS
- the `sppas` directory contains the program itself
- the `documentation`, which contains:
  - the file `CHANGES.txt` is a Release History It shows an overview of the differences between the succeeding versions of SPPAS
  - the copyright and a copy of the licenses
  - the `documentation` in PDF
  - the slides of the document `SPPAS for Dummies`
  - the `references` sub-folder includes PDF files of some publications about SPPAS
  - the `solutions` of the exercises proposed in the chapter “Scripting with Python and SPPAS”
  - the `etc` directory is for internal use: never modify or remove it!

### 1.2.5 Update

SPPAS is constantly being improved and new packages are published frequently (about 10 versions a year). It is important to update regularly in order to get the latest functions and corrections.

Updating SPPAS is very (very very!) easy and fast:

1. Optionally, put the old package into the Trash,
2. Download and unpack the new version.

## 1.3 Capabilities

### 1.3.1 What SPPAS can do?

Here is the list of functionalities available to annotate automatically speech data and to analyse annotated files:

#### 1. Automatic Annotations

- **Momel/INTSINT**: modelling melody
- **IPUs segmentation**: utterance level segmentation
- **Tokenization**: text normalization
- **Phonetization**: grapheme to phoneme conversion
- **Alignment**: phonetic segmentation
- **Syllabification**: group phonemes into syllables
- **Repetitions**: detect self-repetitions, and other-repetitions (not in the GUI).

#### 2. Components

- *IPUScribe*: Manual orthographic transcription
- *SndPlayer*: Play sounds (mono wav) and display main information
- *Statistics*: Estimates/Save statistics on annotated files

- *DataRoamer*: Manipulate annotated files
- *DataFilter*: Extract data from annotated files
- *SppasEdit*: Display sound and annotated files (development version, unstable)

### 3. Plugins

- *TierMapping-plugin*: Create tier by mapping annotation labels
- *MarsaTag-plugin*: Use the POS-Tagger MarsaTag from SPPAS (French only)

## 1.3.2 How to use SPPAS?

There are three main ways to use SPPAS:

1. The Graphical User Interface (GUI) is as user-friendly as possible:
  - double-click on the `sppas.bat` file, under Windows;
  - double-click on the `sppas.command` file, under MacOS or Linux.
2. The Command-line User Interface (CLI), with a set of programs, each one essentially independent of the others, that can be run on its own at the level of the shell.
3. Scripting with Python and SPPAS provides the more powerful way.

## 1.3.3 Interoperability and compatibility

SPPAS is able to open/save files from the following software, with the expected extension:

- Praat: TextGrid, PitchTier, IntensityTier
- Elan: eaf
- Sclite: ctm, stm
- HTK: lab, mlf
- Subtitles: srt, sub
- Phonedit: mrk
- Signaux: hz
- Excel/OpenOffice/R: csv

It can also import and export data from:

- Annotation Pro: antx

And it can also import data from:

- ANVIL: anvil
- Transcriber: trs

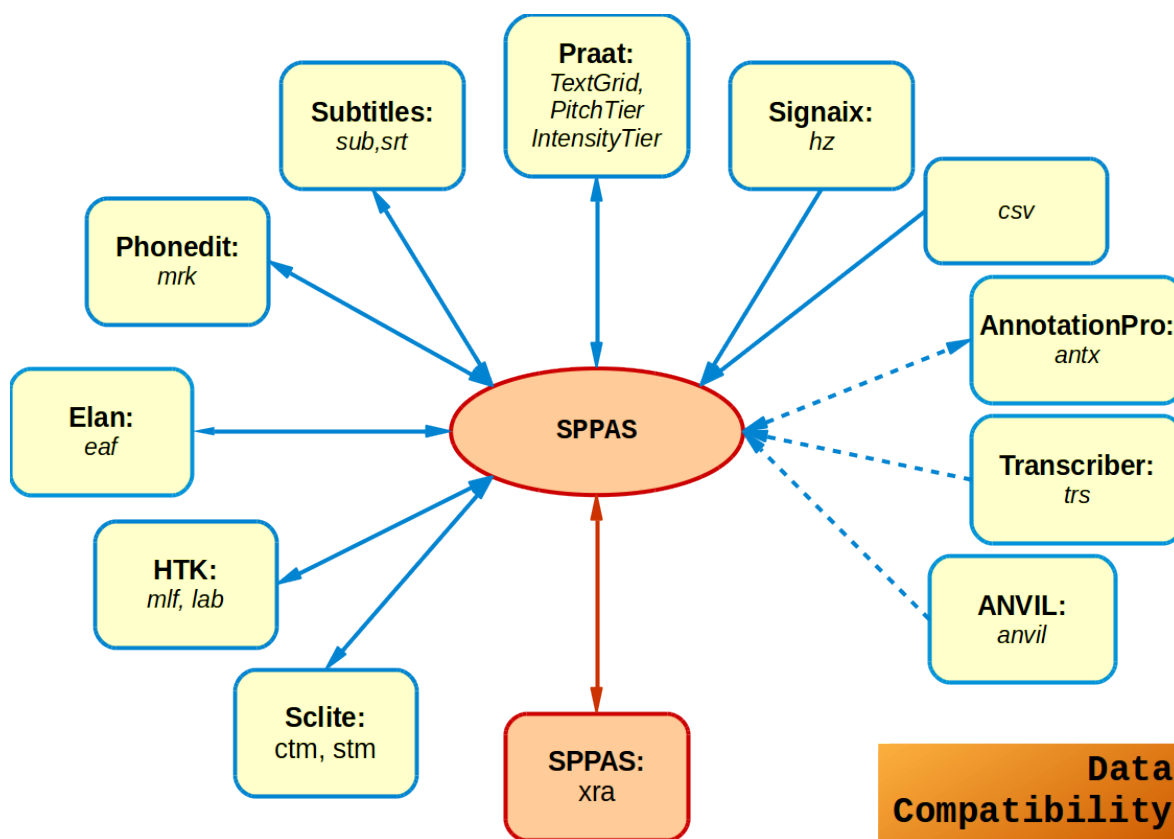


Figure 1.3: File formats SPPAS can open/save and import

## 1.4 Main and important recommendations

### 1.4.1 About files

There is a list of important things to keep in mind while using SPPAS. They are summarized as follows and detailed in chapters of this documentation:

1. Speech files:

- only `wav`, `aiff` and `au` audio files
- only mono (= one channel)
- frame rates preferably: 16000hz, 32000hz, 48000hz
- bit rate preferably: 16 bits
- good recording quality is expected. It is, of course, important to never convert from a compressed file (as mp3 for example).

2. Annotated files:

- UTF-8 encoding only
- it is recommended to NOT use accentuated characters in file names nor in the paths

### 1.4.2 About automatic annotations

The quality of the results for some annotations is highly influenced by the quality of the data the annotation takes in input (this is a politically correct way to say: **Garbage in, garbage out!**)

Annotations are based on the use of **resources**, that are freely available in the SPPAS package. The quality of the automatic annotations is also largely influenced by such resources. However... all the resources can be modified by any user!



# Automatic Speech Segmentation and Annotation

## 2.1 Introduction

### 2.1.1 About this chapter

This chapter is not a description on how each automatic annotation is implemented and how it's working: the references are available for that specific purpose!

Instead, this chapter describes how each automatic annotation can be used in SPPAS, i.e. what is the goal of the annotation, what are the requirements, what kind of resources are used, and what is the expected result. Each automatic annotation is then illustrated as a workflow schema, where:

- blue boxes represent the name of the automatic annotation,
- red boxes represent tiers (with their name mentioned in white),
- green boxes indicate the resource,
- yellow boxes indicate the annotated file (given as input or produced as output).

At the end of each automatic annotation process, SPPAS produces a Procedure Outcome Report that aims to be read!

Among others, SPPAS is able to produce automatically annotations from a recorded speech sound and its orthographic transcription. Let us first introduce what is the exactly meaning of “recorded speech” and “orthographic transcription”.

### 2.1.2 File formats and tier names

When using the Graphical User Interface, the file format for input and output can be fixed in the Settings and is applied to all annotations, and file names of each annotation is already fixed and can't be changed. When using the Command-Line interface, or when using scripts, each annotation can be configured independently (file format and file names). In all cases, **the name of the tiers are fixed and can't be changed!**

### 2.1.3 Recorded speech

First of all:

Only `wav`, `aiff` and `au` audio files and only as mono are supported by SPPAS.

SPPAS verifies if the wav file is 16 bits and 16000 Hz sample rate. Otherwise it automatically converts to this configuration. For very long files, this may take time. So, the following are possible:

1. be patient
2. prepare by your own the required wav/mono/16000Hz/16bits files to be used in SPPAS

Secondly, a relatively good recording quality is expected. Providing a guideline or recommendation for that is impossible, because it depends: “IPU segmentation” requires a better quality compared to what is expected by “Alignment”, and for that latter, it depends on the language.

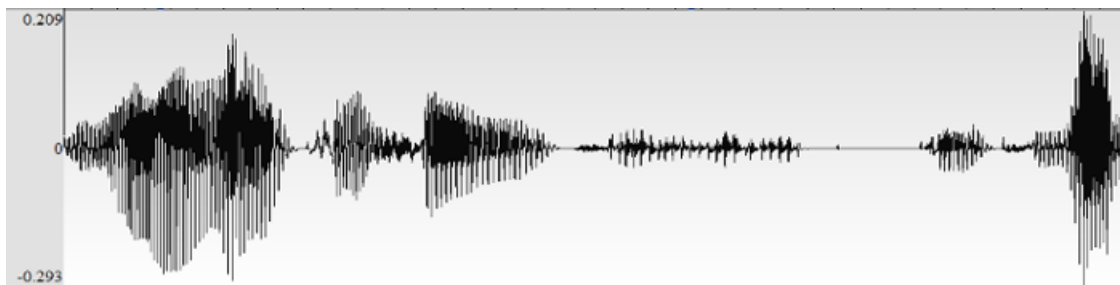


Figure 2.1: Example of recorded speech

### 2.1.4 Orthographic Transcription

#### Overview

Only UTF-8 encoding is supported by SPPAS.

Clearly, there are different ways to pronounce the same utterance. Different speakers have different accents and tend to speak at different rates. There are commonly two types of Speech Corpora. First is related to “Read Speech” which includes book excerpts, broadcast news, lists of words, sequences of numbers. Second is often named as “Spontaneous Speech” which includes dialogs - between two or more people (includes meetings), narratives - a person telling a story, map- tasks - one person explains a route on a map to another, appointment-tasks - two people try to find a common meeting time based on individual schedules. One of the characteristics of Spontaneous Speech is an important gap between a word’s phonological form and its phonetic realizations. Specific realization due to elision or reduction processes are frequent in spontaneous data. It also presents other types of phenomena such as non-standard elisions, substitutions or addition of phonemes which intervene in the automatic phonetization and alignment tasks.

Consequently, when a speech corpus is transcribed into a written text, the transcriber is immediately confronted with the following question: how to reflect the orality of the corpus? *Transcription conventions* are then designed to provide rules for writing speech corpora. These conventions establish phenomena to transcribe and also how to annotate them.

In that sense, the orthographic transcription must be a representation of what is “perceived” in the signal. Consequently, it **must** include:

- filled pauses;
- short pauses;
- repeats;
- noises and laugh items (not available for: English, Japanese and Cantonese).

In speech (particularly in spontaneous speech), many phonetic variations occur. Some of these phonologically known variants are predictable and can be included in the pronunciation dictionary but many others are still unpredictable (especially invented words, regional words or words borrowed from another language).

SPPAS is the only automatic annotation software that deals with **Enriched Orthographic Transcriptions**.

### Convention

The transcription must use the following convention:

- truncated words, noted as a '-' at the end of the token string (an ex- example);
- noises, noted by a '\*' (not available for: English, Japanese and Cantonese);
- laughs, noted by a '@' (not available for: English, Japanese and Cantonese);
- short pauses, noted by a '+';
- elisions, mentioned in parenthesis;
- specific pronunciations, noted with brackets [example,eczap];
- comments are noted inside braces or brackets without using comma {this} or [this and this];
- liaisons, noted between '=' (an =n= example);
- morphological variants with <like,lie ok>,
- proper name annotation, like \$John S. Doe\$.

SPPAS also allows to include in the transcription:

- regular punctuations,
- numbers: they will be automatically converted to their written form.

The result is what we call an enriched orthographic construction, from which two derived transcriptions are generated automatically: the **standard transcription** (the list of orthographic tokens) and a specific transcription from which the phonetic tokens are obtained to be used by the grapheme-phoneme converter that is named **faked transcription**.

### Example

*This is + hum... an enrich(ed) transcription {loud} number 1!*

The derived transcriptions are:

- standard: this is hum an enriched transcription number one
- faked: this is + hum an enrich transcription number one

## 2.2 Inter-Pausal Units (IPUs) segmentation

The “IPUs segmentation” automatic annotation can perform 3 actions:

1. find silence/speech segmentation of a recorded file,
2. find silence/speech segmentation of a recorded file including the time-alignment of utterances of a transcription file,
3. split/save a recorded file into multiple files, depending on segments indicated in a time-aligned file.

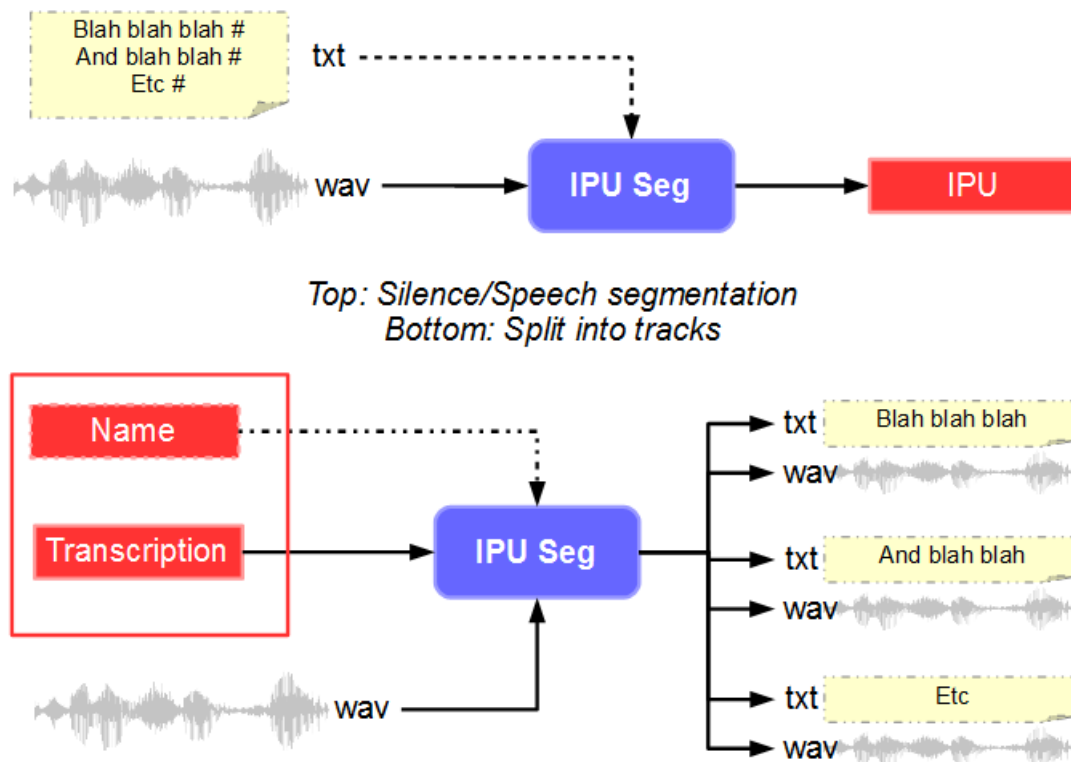


Figure 2.2: IPU Segmentation workflow

### 2.2.1 Silence/Speech segmentation

The IPUs Segmentation annotation performs a silence detection from a recorded file. This segmentation provides an annotated file with one tier named “IPU”. The silence intervals are labelled with the “#” symbol, as speech intervals are labelled with “ipu\_” followed by the IPU number.

The following parameters must be fixed:

- Minimum volume value (in seconds):  
If this value is set to zero, the minimum volume is automatically adjusted for each sound file. Try with it first, then if the automatic value is not correct, fix it manually. The Procedure Outcome Report indicates the value the system choose. The SndRoamer component can also be of great help: it indicates min, max and mean volume values of the sound.

- Minimum silence duration (in seconds): By default, this is fixed to 0.2 sec., an appropriate value for French. This value should be at least 0.25 sec. for English.
- Minimum speech duration (in seconds): By default, this value is fixed to 0.3 sec. The most relevant value depends on the speech style: for isolated sentences, probably 0.5 sec should be better, but it should be about 0.1 sec for spontaneous speech.
- Speech boundary shift (in seconds): a duration which is systematically added to speech boundaries, to enlarge the speech interval.

The procedure outcome report indicates the values (volume, minimum durations) that was used by the system for each sound file given as input. It also mentions the name of the output file (the resulting file). The file format can be fixed in the Settings of SPPAS (xra, TextGrid, eaf, ...).

```

-----
                        IPU Segmentation
-----
... Segmentation of C:\Users\Brigitte\Desktop\sppas-1.6.2\samples\samples-jpn\JAJA_M16_T33.wav
... .. Threshold volume value:      514
... .. Threshold silence duration: 0.2
... .. Threshold speech duration:  0.3
... .. C:\Users\Brigitte\Desktop\sppas-1.6.2\samples\samples-jpn\JAJA_M16_T33.TextGrid [ OK ]
-----

```

Figure 2.3: Procedure outcome report of IPU Segmentation

The annotated file can be checked manually (preferably in Praat than Elan nor Anvil). If such values was not correct, then, delete the annotated file that was previously created, change the default values and re-annotate.

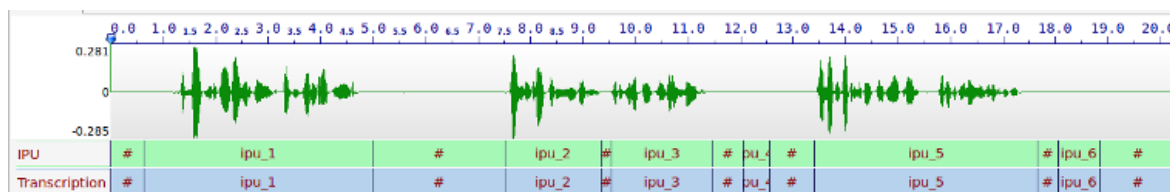


Figure 2.4: Result of IPU Segmentation: silence detection

Notice that the speech segments can be transcribed using the “IPUScribe” component.

## 2.2.2 Silence/Speech segmentation time-aligned with a transcription

Inter-Pausal Units segmentation can also consist in aligning macro-units of a document with the corresponding sound.

SPPAS identifies silent pauses in the signal and attempts to align them with the inter-pausal units proposed in the transcription file, under the assumption that each such unit is separated by a silent pause. This algorithm is language-independent: it can work on any language.

In the transcription file, **silent pauses must be indicated** using both solutions, which can be combined:

- with the symbol ‘#’,
- with newlines.

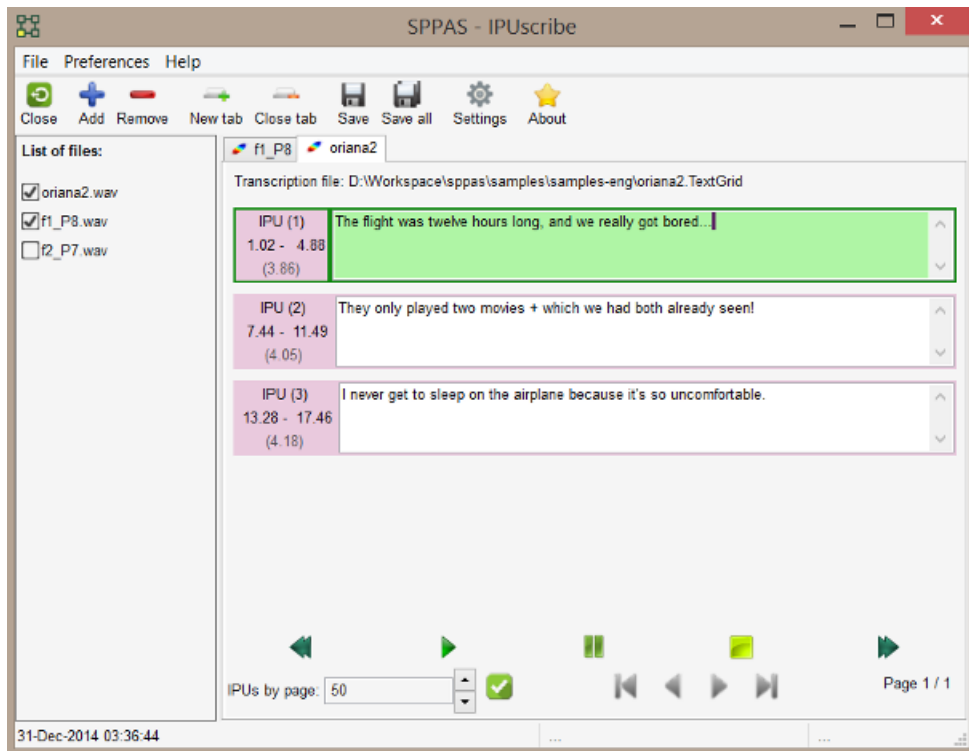


Figure 2.5: Orthographic transcription based on IPUs Segmentation

A recorded speech file must strictly correspond to a transcription, except for the extension expected as .txt for this latter. The segmentation provides an annotated file with one tier named “IPU”. The silence intervals are labelled with the “#” symbol, as speech intervals are labelled with “ipu\_” followed by the IPU number then the corresponding transcription.

The same parameters than those indicated in the previous section must be fixed.

Important:

This segmentation was tested on documents no longer than one paragraph (about 1 minute speech).

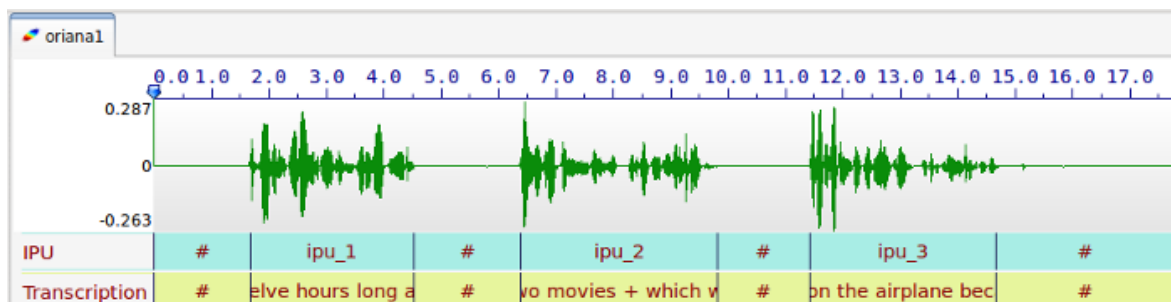


Figure 2.6: Silence/Speech segmentation with orthographic transcription

### 2.2.3 Split into multiple files

IPU segmentation can split the sound into multiple files (one per IPU), and it creates a text file for each of the tracks. The output file names are “track\_0001”, “track\_0002”, etc.

Optionally, if the input annotated file contains a tier named exactly “Name”, then the content of this tier will be used to fix output file names.



Figure 2.7: Data to split

In the example above, the automatic process will create 6 files: FLIGHT.wav, FLIGHT.txt, MOVIES.wav, MOVIES.txt, SLEEP.wav and SLEEP.txt. It is up to the user to perform another IPU segmentation of these files to get another file format than txt (xra, TextGrid, ...) thanks to the previous section “Silence/Speech segmentation time-aligned with a transcription”.

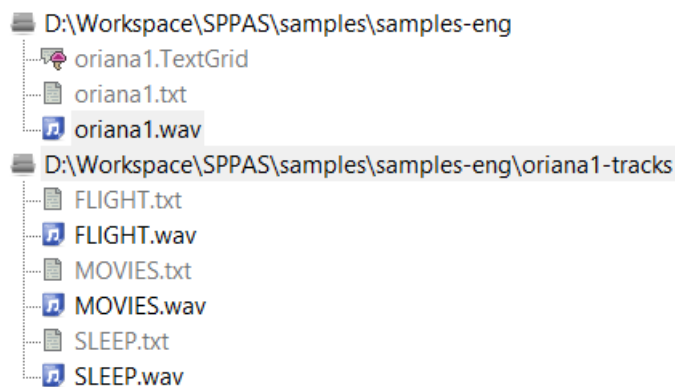


Figure 2.8: Data split

## 2.3 Tokenization

Tokenization is also known as “Text Normalization” the process of segmenting a text into tokens. In principle, any system that deals with unrestricted text need the text to be normalized. Texts contain a variety of “non-standard” token types such as digit sequences, words, acronyms and letter sequences in all capitals, mixed case words, abbreviations, roman numerals, URL’s and e-mail addresses... Normalizing or rewriting such texts using ordinary words is then an important issue. The main steps of the text normalization proposed in SPPAS are:

- Remove punctuation;
- Lower the text;
- Convert numbers to their written form;

- Replace symbols by their written form, thanks to a “replacement” dictionary, located into the sub-directory “repl” in the “resources” directory. Do not hesitate to add new replacements in this dictionary.
- Word segmentation based on the content of a lexicon. If the result is not corresponding to your expectations, fill free to modify the lexicon, located in the “vocab” sub-directory of the “resources” directory. The lexicon contains one word per line.

For more details, see the following reference:

**Brigitte Bigi (2011).** *A Multilingual Text Normalization Approach*. 2nd Less-Resourced Languages workshop, 5th Language Technology Conference, Poznań (Poland).

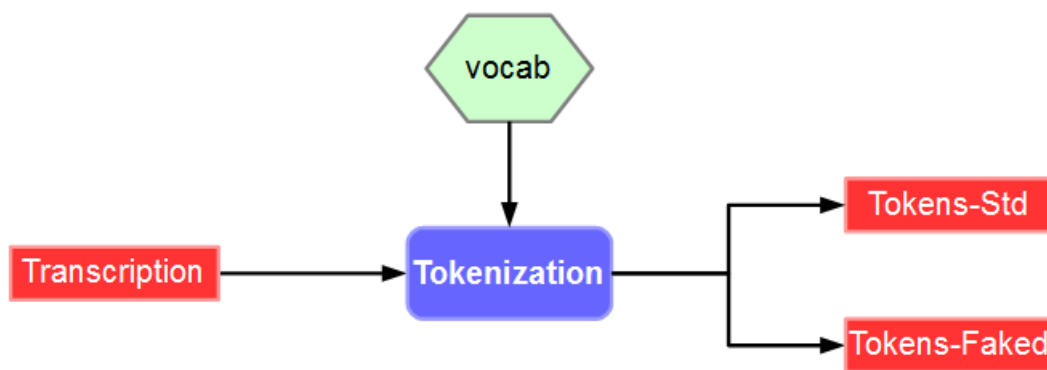


Figure 2.9: Text normalization workflow

The SPPAS Tokenization system takes as input a file including a tier with the orthographic transcription. The name of this tier must contains one of the following strings:

- trs
- trans
- ipu
- ortho
- toe

The first tier that matches is used (case insensitive search).

By default, it produces a file including only one tier with the tokens. To get both transcription tiers faked and standard, check such option!

- Tokens-std: the text normalization of the standard transcription,
- Tokens-faked: the text normalization of the faked transcription.

Read the “Introduction” of this chapter to understand the difference between “standard” and “faked” transcriptions.



## 2.4 Phonetization

Phonetization, also called grapheme-phoneme conversion, is the process of representing sounds with phonetic signs.

SPPAS implements a dictionary based-solution which consists in storing a maximum of phonological knowledge in a lexicon. In this sense, this approach is language-independent. SPPAS phonetization process is the equivalent of a sequence of dictionary look-ups.

The SPPAS phonetization takes as input an orthographic transcription previously normalized (by the Tokenization automatic system or manually). The name of this tier must contains one of the following strings:

- “tok” and “std”
- “tok” and “faked”

The first tier that matches is used (case insensitive search).

The system produces a phonetic transcription.

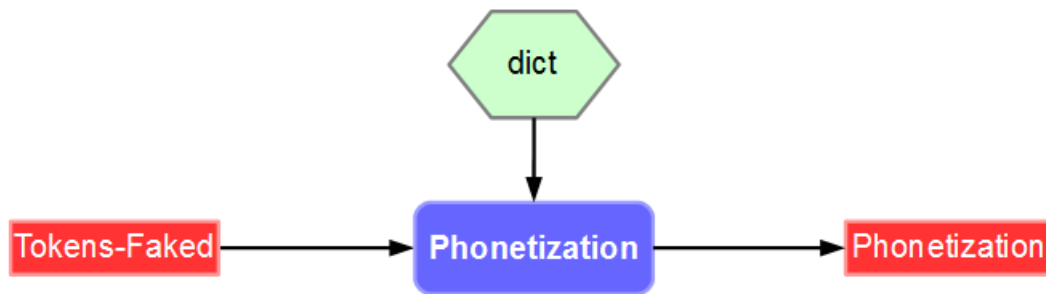


Figure 2.10: Phonetization workflow

Actually, some words can correspond to several entries in the dictionary with various pronunciations, all these variants are stored in the phonetization result. By convention, spaces separate words, dots separate phones and pipes separate phonetic variants of a word. For example, the transcription utterance:

- Tokenization: the flight was twelve hours long
- Phonetization: dh.ax|dh.ah|dh.iy f.l.ay.t w.aa.z|w.ah.z|w.ax.z|w.ao.z  
t.w.eh.l.v aw.er.z|aw.r.z l.ao.ng

Many of the other systems assume that all words of the speech transcription are mentioned in the pronunciation dictionary. On the contrary, SPPAS includes a language-independent algorithm which is able to phonetize unknown words of any language as long as a dictionary is available! If such case occurs during the phonetization process, a WARNING mentions it in the Procedure Outcome Report.

For details, see the following reference:

**Brigitte Bigi (2013).** *A phonetization approach for the forced-alignment task*, 3rd Less-Resourced Languages workshop, 6th Language & Technology Conference, Poznan (Poland).

Since the phonetization is only based on the use of a pronunciation dictionary, the quality of such a phonetization only depends on this resource. If a pronunciation is not as expected, it is up to the user to change it in the dictionary. All dictionaries are located in the sub-directory “dict” of the “resources” directory.

SPPAS uses the same dictionary-format as proposed in VoxForge, i.e. the HTK ASCII format. Here is a piece of the eng.dict file:

THE	[THE]	D @
THE (2)	[THE]	D V
THE (3)	[THE]	D i:
THEA	[THEA]	T i: @
THEALL	[THEALL]	T i: l
THEANO	[THEANO]	T i: n @U
THEATER	[THEATER]	T i: @ 4 3:r
THEATER'S	[THEATER'S]	T i: @ 4 3:r z

The first column indicates the word, followed by the variant number (except for the first one). The second column indicated the word between brackets. The last columns are the succession of phones, separated by a whitespace.

## 2.5 Alignment

Alignment, also called phonetic segmentation, is the process of aligning speech with its corresponding transcription at the phone level. The alignment problem consists in a time-matching between a given speech unit along with a phonetic representation of the unit.

SPPAS is based on the Julius Speech Recognition Engine (SRE). Speech Alignment also requires an Acoustic Model in order to align speech. An acoustic model is a file that contains statistical representations of each of the distinct sounds of one language. Each phoneme is represented by one of these statistical representations. SPPAS is working with HTK-ASCII acoustic models, trained from 16 bits, 16000 Hz wav files.

Speech segmentation was evaluated for French: in average, automatic speech segmentation is 95% of times within 40ms compared to the manual segmentation (tested on read speech and on conversational speech). Details about these results are available in the slides of the following reference:

*Brigitte Bigi (2014). Automatic Speech Segmentation of French: Corpus Adaptation. 2nd Asian Pacific Corpus Linguistics Conference, p. 32, Hong Kong.*

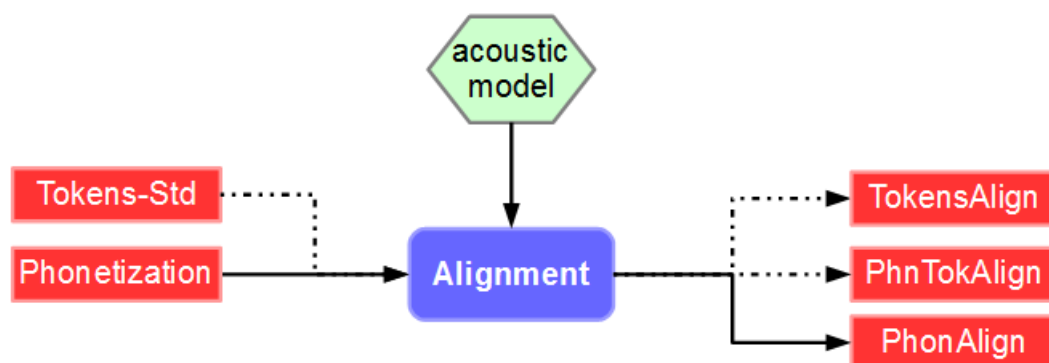


Figure 2.11: Alignment workflow

The SPPAS aligner takes as input the phonetization and optionally the tokenization. The name of the phonetization tier must contain the string “phon”. The first tier that matches is used (case insensitive search).

The annotation provides one annotated file with 3 tiers:

- “PhonAlign”, is the segmentation at the phone level;
- “PhnTokAlign” is the segmentation at the word level, with phonemes as labels;
- “TokensAlign” is the segmentation at the word level.

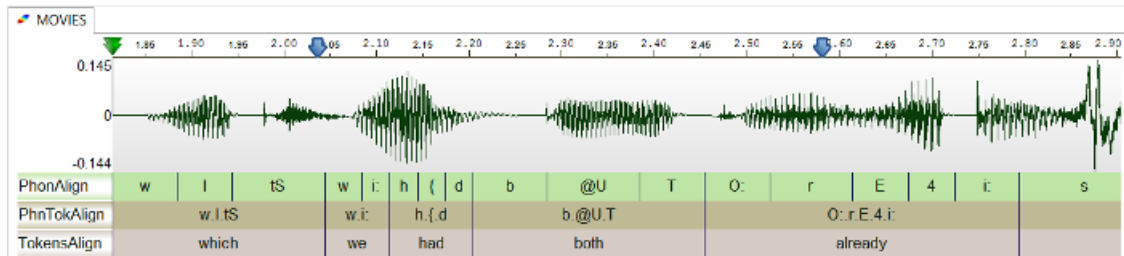


Figure 2.12: SPPAS alignment output example

The following options are available to configure alignment:

- Expend option: If expend is checked, SPPAS will expend the last phoneme and the last token of each unit to the unit duration.
- Extend option: If extend is checked, SPPAS will extend the last phoneme and the last token to the way duration, otherwise SPPAS adds a silence.
- Remove temporary files: keep only alignment result and remove intermediary files (they consists in one way file per unit and a set of text files per unit).
- Speech segmentation system can be either: julius, hvite or basic
- Guess short pauses after each token

## 2.6 Syllabification

The syllabification of phonemes is performed with a rule-based system from time-aligned phonemes. This phoneme-to-syllable segmentation system is based on 2 main principles:

- a syllable contains a vowel, and only one;
- a pause is a syllable boundary.

These two principles focus the problem of the task of finding a syllabic boundary between two vowels. As in state-of-the-art systems, phonemes were grouped into classes and rules established to deal with these classes. We defined general rules followed by a small number of exceptions. Consequently, the identification of relevant classes is important for such a system.

We propose the following classes, for both French and Italian set of rules:

- V - Vowels,
- G - Glides,
- L - Liquids,
- O - Occlusives,

- F - Fricatives,
- N - Nasals.

The rules we propose follow usual phonological statements for most of the corpus. A configuration file indicates phonemes, classes and rules. This file can be edited and modified to adapt the syllabification.

For more details, see the following reference:

**B. Bigi, C. Meunier, I. Nesterenko, R. Bertrand (2010).** *Automatic detection of syllable boundaries in spontaneous speech*. Language Resource and Evaluation Conference, pp 3285-3292, La Valetta, Malte.

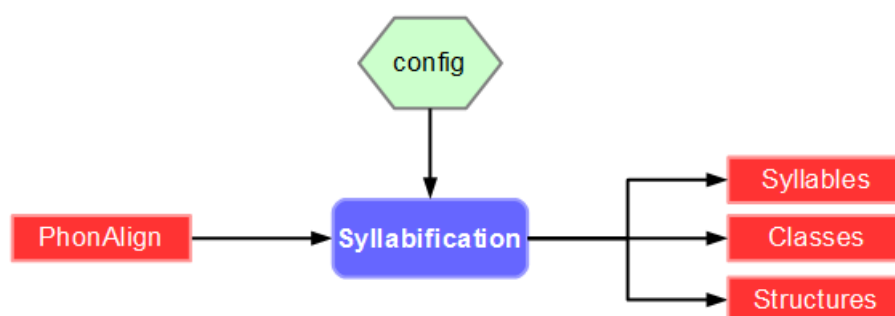


Figure 2.13: Syllabification workflow

The Syllabification annotation takes as input one file with (at least) one tier containing the time-aligned phonemes. The annotation provides one annotated file with 3 tiers (Syllables, Classes and Structures).

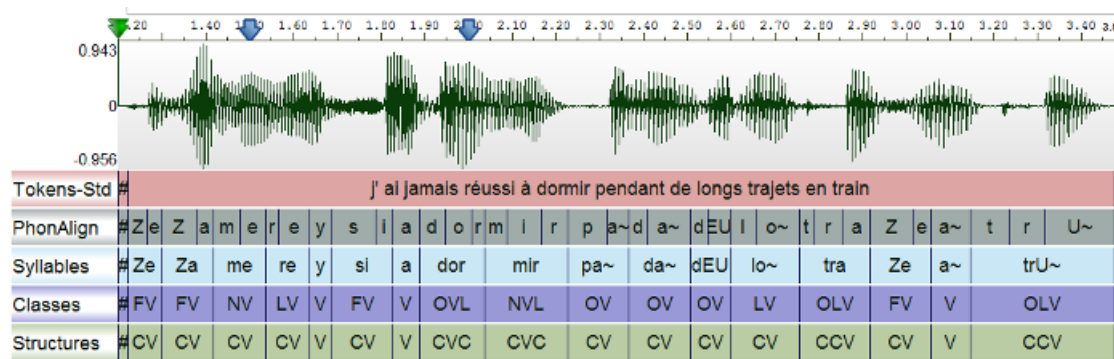


Figure 2.14: Syllabification example

If the syllabification is not as expected, you can change the set of rules. The configuration file is located in the sub-directory “syll” of the “resources” directory.

The syllable configuration file is a simple ASCII text file that any user can change as needed. At first, the list of phonemes and the class symbol associated with each of the phonemes are described as, for example:

- PHONCLASS e V
- PHONCLASS p O

The couples phoneme/class are made of 3 columns: the first column is the key-word PHONCLASS, the second column is the phoneme symbol, the third column is the class symbol. The constraints on this definition are:

- a pause is mentioned with the class-symbol #,
- a class-symbol is only one upper-case character, except:
  - the character X is forbidden;
  - the characters V and W are used for vowels.

The second part of the configuration file contains the rules. The first column is a keyword, the second column describes the classes between two vowels and the third column is the boundary location. The first column can be:

- GENRULE,
- EXCRULE, or
- OTHRULE.

In the third column, a 0 means the boundary is just after the first vowel, 1 means the boundary is one phoneme after the first vowel, etc. Here are some examples, corresponding to the rules described in this paper for spontaneous French:

- GENRULE V XV 0
- GENRULE V XXV 1
- EXCRULE V FLV 0
- EXCRULE V O LGV 0

Finally, to adapt the rules to specific situations that the rules failed to model, we introduced some phoneme sequences and the boundary definition. Specific rules contain only phonemes or the symbol “ANY” which means any phoneme. It consists of 7 columns: the first one is the key-word OTHRULE, the 5 following columns are a phoneme sequence where the boundary should be applied to the third one by the rules, the last column is the shift to apply to this boundary. In the following example:

```
OTHRULE ANY ANY p s k -2
```

## 2.7 Repetitions

This automatic detection focus on word repetitions, which can be an exact repetition (named strict echo) or a repetition with variation (named non-strict echo).

SPPAS implements *self-repetitions* and *other-repetitions* detection. The system is based only on lexical criteria. The proposed algorithm is focusing on the detection of the source.

The Graphical User Interface only allows to detect self-repetitions. Use the Command-Line User Interface if you want to get other-repetitions.

For more details, see the following paper:

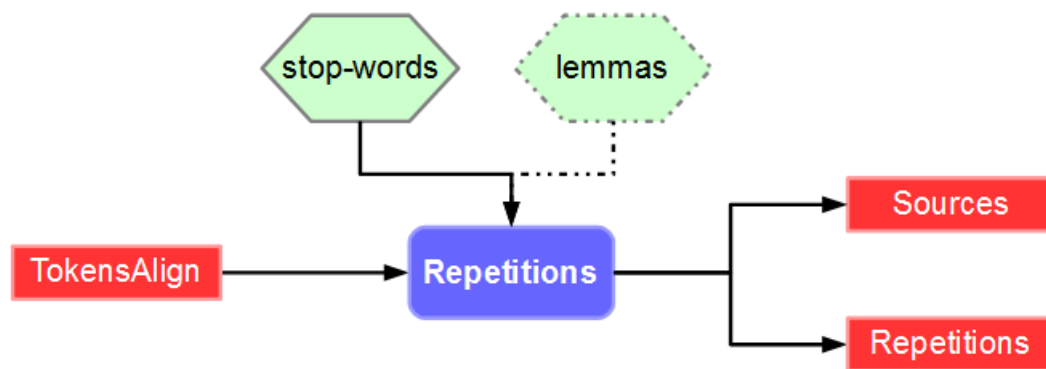


Figure 2.15: Repetition detection workflow

**Brigitte Bigi, Roxane Bertrand, Mathilde Guardiola** (2014). *Automatic detection of other-repetition occurrences: application to French conversational speech*, 9th International conference on Language Resources and Evaluation (LREC), Reykjavik (Iceland).

The automatic annotation takes as input a file with (at least) one tier containing the time-aligned tokens of the speaker (and another file/tier for other-repetitions). The annotation provides one annotated file with 2 tiers (Sources and Repetitions).

This process requires a list of stop-words, and a dictionary with lemmas (the system can process without it, but the result is better with it). Both lexicons are located in the “vocab” sub-directory of the “resources” directory.

## 2.8 Momel and INTSINT

### 2.8.1 Momel (modelling melody)

Momel is an algorithm for the automatic modelling of fundamental frequency (F0) curves using a technique called assymetric modal quaratic regression.

This technique makes it possible by an appropriate choice of parameters to factor an F0 curve into two components:

- a macroprosodic component represented by a quadratic spline function defined by a sequence of target points < ms, hz >.
- a microprosodic component represented by the ratio of each point on the F0 curve to the corresponding point on the quadratic spline function.

The algorithm which we call Asymmetrical Modal Regression comprises the following four stages:

For details, see the following reference:

**Daniel Hirst and Robert Espesser** (1993). *Automatic modelling of fundamental frequency using a quadratic spline function*. Travaux de l’Institut de Phonétique d’Aix. vol. 15, pages 71-85.

The SPPAS implementation of Momel requires a file with the F0 values, sampled at 10 ms. Two extensions are supported:

- .PitchTier, from Praat.
- .hz, from any tool, is a file with one F0 value per line.

These options can be fixed:

- Window length used in the “cible” method
- F0 threshold: Maximum F0 value
- F0 ceiling: Minimum F0 value
- Maximum error: Acceptable ratio between two F0 values
- Window length used in the “reduc” method
- Minimal distance
- Minimal frequency ratio
- Eliminate glitch option: Filter f0 values before ‘cible’

### 2.8.2 Encoding of F0 target points using the “INTSINT” system

INTSINT assumes that pitch patterns can be adequately described using a limited set of tonal symbols, T,M,B,H,S,L,U,D (standing for : Top, Mid, Bottom, Higher, Same, Lower, Up-stepped, Down-stepped respectively) each one of which characterises a point on the fundamental frequency curve.

The rationale behind the INTSINT system is that the F0 values of pitch targets are programmed in one of two ways : either as absolute tones T, M, B which are assumed to refer to the speaker’s overall pitch range (within the current Intonation Unit), or as relative tones H, S, L, U, D assumed to refer only to the value of the preceding target point.

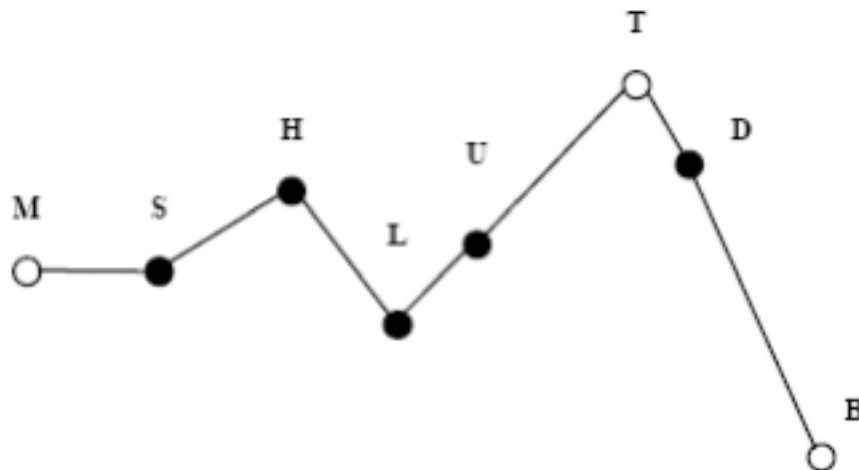
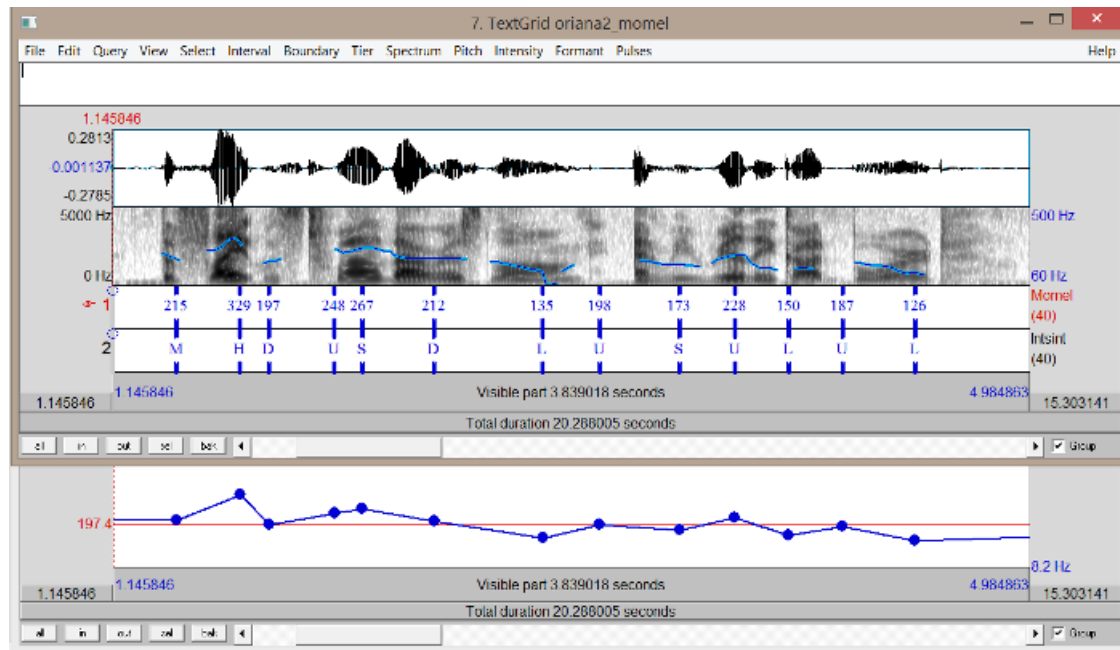


Figure 2.16: INTSINT example

The rationale behind the INTSINT system is that the F0 values of pitch targets are programmed in one of two ways : either as absolute tones T, M, B which are assumed to refer to the speaker’s overall pitch range (within the current Intonation Unit), or as relative tones H, S, L, U, D assumed to refer only to the value of the preceding target point.

A distinction is made between non-iterative H, S, L and iterative U, D relative tones since in a number of descriptions it appears that iterative raising or lowering uses a smaller F0 interval than non-iterative raising or lowering. It is further assumed that the tone S has no iterative equivalent since there would be no means of deciding where intermediate tones are located.



**D.-J. Hirst** (2011). *The analysis by synthesis of speech melody: from data to models*, Journal of Speech Sciences, vol. 1(1), pages 55-83.



## Resources for Automatic Annotations

### 3.1 What are SPPAS resources and where they come from?

#### 3.1.1 Overview

All automatic annotations included in SPPAS are implemented with language-independent algorithms... this means that adding a new language in SPPAS only consists in adding resources related to the annotation (like lexicons, dictionaries, models, set of rules, etc).

All available resources to perform automatic annotations are located in the sub-directory 'resources'. There are 5 sub-directories:

- Lexicon (list of words used during Tokenization) are located in the *vocab* sub-directory.
- A list of replacements to perform during tokenization in the *repl* sub-directory.
- Pronunciation dictionaries (used during Phonetization) are located in the *dict* sub-directory.
- The acoustic models (used during Alignment) are located in the *models* directory.
- The Syllabification configuration files are located in the *syll* directory.

All resources can be edited, modified, changed or deleted by any user.

Caution: all the files are in UTF-8, and this encoding must not be changed.

The language names are based on the ISO639-3 international standard. See <http://www-01.sil.org/iso639-3/> for the list of all languages and codes. Here is the list of available languages in SPPAS resources:

- French: fra
- English: eng
- Spanish: spa
- Italian: ita
- Catalan: cat
- Japanese: jpn

- Mandarin Chinese: cmn
- Southern Min (or Min Nan): nan
- Cantonese: yue

SPPAS can deal with a new language by simply adding the language resources to the appropriate sub-directories. Of course, file formats must correspond to which expected by SPPAS! Lexicon and dictionaries can be edited/modified/saved with a simple-editor as **Notepad++** for example (*under Windows: above all, don't use the windows' notepad*). Idem for the syllabification rules.

The only step in the procedure which is probably beyond the means of a linguist without external aid is the creation of a new acoustic model when it does not yet exist for the language being analysed. This only needs to be carried out once for each language, though, and we provide detailed specifications of the information needed to train an acoustic model on an appropriate set of recordings and dictionaries or transcriptions. Acoustic models obtained by such a collaborative process will be made freely available to the scientific community.

The current acoustic models can be improved too (except for eng and jpn): send your data (wav and transcription files) to the author. Notice that such data will not be published in any form without your authorization. They will be included in the training procedure to create a new (and better) acoustic model, that will be distributed in the next version of SPPAS.

### 3.1.2 About the phone sets

Most of the resources are using SAMPA to represent phonemes.

In addition, all models (except jpn and yue) include the following fillers:

- dummy: untranscribed speech
- gb: garbage
- @@: laughing

### 3.1.3 How to add a new language

#### 1. Dictionary and lexicon:

1.1 Copy the phonetic dictionary `LANG.dict` in the `dict` directory

1.2 Copy the vocabulary list `LANG.vocab` in the `vocab` directory

#### 2. Create a directory `models/models-LANG`; then copy the acoustic model in this directory

#### 3. Optionally, copy the file `syllConfig-LANG.txt` in the `syll` directory.

Required Input Data formats:

- The dictionary is HTK ASCII, like: word [word] phon1 phon2 phon3 Columns are separated by spaces.
- Acoustic models are in HTK ASCII format (16bits,16000hz).

Notice that the Graphical User Interface dynamically creates the list of available languages by exploring the sub-directory “resources” included in the SPPAS package. This means that all changes in the “resources” directory will be automatically take into account.

## 3.2 French Resources

(C) Laboratoire Parole et Langage, Aix-en-Provence, France.

### 3.2.1 Pronunciation Dictionary

The French dictionary is under the terms of the “GNU Public License”.

The French pronunciation dictionary was created by Brigitte Bigi by merging a some free dictionaries loaded from the web. Some word pronunciations were added using the LIA\_Phon tool. Many words were manually corrected and a large set of missing words and pronunciation variants were added.

### 3.2.2 Acoustic Model

See COPYRIGHT.txt (in the “model-fra” directory) for the details of the license: “Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International Public License”.

The French acoustic model was created by Brigitte Bigi from various corpora recorded at Laboratoire Parole et Langage (see the “References” section of this documentation). It includes:

- CID - Corpus of Interactional Data (7h30) <http://www.sldr.org/sldr000027/>,
- Grenelle (7min) <http://www.sldr.org/sldr000729/>,
- Broadcast news (40min) and Read speech (not public).

References to these corpora:

**P. Blache, R. Bertrand, B. Bigi, E. Bruno, E. Cela, R. Espesser, G. Ferré, M. Guardiola, D. Hirst, E.-P. Magro, J.-C. Martin, C. Meunier, M.-A. Morel, E. Murisasco, I Nesterenko, P. Nocera, B. Pallaud, L. Prévot, B. Priego-Valverde, J. Seinturier, N. Tan, M. Tellier, S. Rauzy** (2010). *Multimodal Annotation of Conversational Data*, The Fourth Linguistic Annotation Workshop, ACL 2010, pages 186-191, Uppsala, Sweden.

**B. Bigi, C. Portes, A. Steuckardt, M. Tellier** (2011). *Multimodal Annotations and Categorization for Political Debates*, ICMI Workshop on Multimodal Corpora for Machine learning (ICMI-MMC), Alicante, Spain.

Here is the phoneset used in the acoustic model:

SPPAS	- IPA -	Examples
b	b	beau
d	d	doux
f	f	fête pharmacie
g	g	gain guerre second
k	k	cabas psychologie quatre kelvin
l	l	loup

SPPAS	- IPA -	Examples
m	m	mou femme
n	n	nous bonne
p	p	passé
R	ʁ	roue rhume
s	s	sa hausse ce garçon option scie
S	ʃ	choux schème shampooing
t	t	tout thé grand-oncle
v	v	vous wagon neuf heures
z	z	hasard zéro transit
Z	ʒ	joue geai
N	ŋ	camping bingo
j	j	fief payer fille travail
w	w	oui loi moyen web whisky wagon
h	ɥ	huit Puy
a	a	patte là
a	ɑ	pâte glas
e	e	clé les chez aller pied journée
E	ɛ	crème est faite peine
E	ɛ:	fête maître mètre reître reine caisse Lemaistre Lévesque
eu	ə	le reposer monsieur faisons
eu	ø	ceux jeûner queue deux
9	œ	sœur jeune neuf
i	i	si île régie y
o	o	sot hôtel haut bureau
o	ɔ	sort minimum
u	u	coup clown roue
y	y	tu sûr rue
a~	ã	sans champ vent temps Jean taon
U~	ẽ	vin impair pain daim plein Reims synthèse sympa
U~	œ	un parfum
o~	õ	son nom
fp		euh

### 3.2.3 Syllabification configuration

The syllabification configuration file corresponds to the rules defined in the following paper:

*B. Bigi, C. Meunier, I. Nesterenko, R. Bertrand (2010). Automatic detection of syllable boundaries in spontaneous speech, Language Resource and Evaluation Conference, pp 3285-3292, La Valetta, Malte.*

## 3.3 Italian resources

*(C) Laboratoire Parole et Langage, Aix-en-Provence, France.*

### 3.3.1 Pronunciation Dictionary

The Italian dictionary is under the terms of the “GNU Public License”.

The Italian dictionary was downloaded from the Festival synthesizer tool. A part of the phonetization were manually corrected and a large set of missing words and pronunciation variants were added.

### 3.3.2 Acoustic Model

See COPYRIGHT.txt (in the “model-ita” directory) for the details of the license: “Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International Public License”.

The Italian acoustic model were created during the Evalita 2011 evaluation campaign, from the CLIPS Map-Task corpus (3h30).

Here is the phoneset used in the acoustic model:

SPPAS	- IPA -	Examples
b	b	banca cibo
d	d	dove idra
dz	dz	zaino zelare mezzo
dZ	dʒ	giungla magia fingere pagina
f	f	fatto fosforo
g	g	gatto agro glifo ghetto
k	k	cavolo acuto anche quei
l	l	lato lievemente
L	ʎ	gli glielo maglia
m	m	mano amare campo
n	n	nano punto pensare anfibio
N	ɲ	fango unghia panchina dunque
J	ɲ	gnocco ogni

SPPAS	- IPA -	Examples
p	p	primo ampio copertura
r	r	Roma quattro morte
s	s	sano scatola presentire pasto
S	ʃ	scena sciame pesci
t	t	tranne mito
ts	ts	sozzo canzone marzo
tS	tʃ	Cennini cinque ciao
v	v	vado povero
z	z	sbavare presentare asma
j	j	ieri scoiattolo più Jesi
w	w	uovo fuoco qui
a	a	alto sarà
e	e	vero perché
E	ɛ	elica cioè
i	i	imposta colibrì zie
o	o	ombra come
o	ɔ	otto posso sarò
u	u	ultimo caucciù tuo

### 3.3.3 Syllabification configuration

The syllabification configuration file corresponds to the rules defined in the following paper:

*B. Bigi, C. Petrone (2014). A generic tool for the automatic syllabification of Italian, Proceedings of the first Italian Computational Linguistics Conference, Pisa, Italy.*

## 3.4 Spanish resources

### 3.4.1 Pronunciation Dictionary

The pronunciation dictionary was downloaded from the CMU web page. Brigitte Bigi converted this CMU phoneset to SAMPA, and changed the file format.

### 3.4.2 Acoustic Model

*(C) Laboratoire Parole et Langage, Aix-en-Provence, France.*

See COPYRIGHT.txt (in the “model-spa” directory) for the details of the license: “Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International Public License”.

The acoustic model was trained from Glissando. We address special thanks to Juan-Maria Garrido for giving us access to this corpus: <http://veus.barcelonamedia.org/glissando/node/10>

**GARRIDO, J. M. - ESCUDERO, D. - AGUILAR, L. -CARDEÑOSO, V. - RODERO, E. - DE-LA-MOTA, C. - GONZÁLEZ, C. - RUSTULLET, S. - LARREA, O. - LAPLAZA, Y. - VIZCAÍNO, F. - CABRERA, M. - BONAFONTE, A.** (2013). *Glissando: a corpus for multidisciplinary prosodic studies in Spanish and Catalan*, Language Resources and Evaluation, DOI 10.1007/s10579-012-9213-0.

Here is the phoneset used in the acoustic model:

SPPAS	- IPA -	Examples
b	b	bestia embuste vaca
b	β	bebé obtuso vivir curva
d	d	dedo cuando aldaba
d	ð	dádiva arder anddmirar
f	f	fase café
g	g	gato lengua gatouerra
g	ɣ	trigo amargo sigue signo
h	h	jamón eje reloj general México
k	k	caña laca quise kilo
l	l	lino alhaja principal
L	ʎ	llave pollo roughly
m	m	madre completelymer campo convertir
n	n	nido anillo anhelo sin álbum
J	ɲ	ñandú cañón enyesar
N	ŋ	cinco venga conquista
p	p	pozo topo
rr	r	rumbo carro honra subrayo amor
r	ʀ	caro bravo amor eterno
s	s	saco zapato cientos espita xenón
T	θ	xenón cereal encima zorro enzima paz
t	t	tamiz átomo
tS	tʃ	chubasco acechar
x	x	jamón eje reloj general México
z	z	isla mismo deshuesar

SPPAS	- IPA -	Examples
S	ʃ	English abacaxi Shakira
ts	ts	Ertzaintza abertzale Pátzcuaro
j	j	aliada cielo amplio ciudad
w	w	cuadro fuego
a	a	azahar
e	e	vehemente
i	i	dimitir mío
o	o	boscoso
u	u	cucurucho dúo

## 3.5 Catalan resources

### 3.5.1 Pronunciation Dictionary

The Catalan dictionary is under the terms of the “GNU Public License”.

The catalan pronunciation dictionary was downloaded from the Ralf catalog of dictionaries for the Simon ASR system at <http://spirit.blau.in/simon/import-pls-dictionary/>. It was then converted (format and phoneset) by Brigitte Bigi. Some new words were also added and phonetized manually.

### 3.5.2 Acoustic Model

(C) *Laboratoire Parole et Langage, Aix-en-Provence, France.*

See COPYRIGHT.txt (in the “model-cat” directory) for the details of the license: “Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International Public License”.

The acoustic model was *not* trained from data. Monophones of other models were cut and pasted to create this one.

Here is the phoneset used in the acoustic model:

SPPAS	- IPA -	Examples
@	ə	sec
E	ε	sec
D	ð	
N	ŋ	
J	ɲ	
L	ʎ	
S	ʃ	



SPPAS	- IPA -	Examples
O	ɔ	
Z	ʒ	
U	ʊ	
a	a	sac
b	b	
b	β	
b	v	
d	d	
d	ð	
e	e	séc
f	f	
g	g	
g	ʝ	
k	k	
l	l	
m	m	
n	n	
i	ɪ	
i	i	sic
o	o	sóc
o	ɔ	soc
p	p	
rr	r	
r	R	
4	r	
s	s	si
t	t	
tS	tʃ	
x	x	
x	ɣ	
z	z	
j	j	
w	w	
u	u	suc

---

SPPAS   - IPA -   Examples

---

## 3.6 English

### 3.6.1 Dictionary

The CMU Pronouncing Dictionary (also known as CMUdict) is a public domain pronouncing dictionary created by Carnegie Mellon University (CMU). It defines a mapping from English words to their North American pronunciations; it contains over 125,000 words and their transcriptions. See <http://www.speech.cs.cmu.edu/cgi-bin/cmudict> for details.

The Carnegie Mellon Pronouncing Dictionary, in its current and previous versions is Copyright (C) 1993-2008 by Carnegie Mellon University.

Use of this dictionary for any research or commercial purpose is completely unrestricted. If you make use of or redistribute this material, the CMU requests that you acknowledge its origin in your descriptions.

### 3.6.2 Acoustic Model

The acoustic model distributed in SPPAS resources were downloaded (in 2013) from the VoxForge project at <http://www.voxforge.org/>. It was then converted to SAMPA by Brigitte Bigi.

The English acoustic model is under the terms of the “GNU Public License”.

Here is the phoneset used in the acoustic model:

SPPAS	- IPA -	Examples
b	b	buy cab
d	d	dye cad do
D	ð	thy breathe father
dZ	dʒ	giant badge jam
f	f	phi caff fan
g	g	guy bag
h	h	high ahead
j	j	yes yacht
k	k	sky crack
l	l	lie sly gal
m	m	my smile cam
n	n	nigh snide can
N	ŋ	sang sink singer
T	θ	thigh math
p	p	pie spy cap

SPPAS	- IPA -	Examples
r	r	rye try very
s	s	sigh mass
S	ʃ	shy cash emotion
t	t	tie sty cat atom
tS	tʃ	China catch
v	v	vie have
w	w	wye swine
hw	hw	why
z	z	zoo has
Z	ʒ	equation pleasure vision beige
x	x	ugh loch Chanukah
A	ɑ:	PALM father bra
A	ɒ	LOT pod John
{	æ	TRAP pad shall ban
aI	aɪ	PRICE ride file fine pie
aU	aʊ	MOUTH loud foul down how
@	ɛ	DRESS bed fell men
eI	eɪ	FACE made fail vein pay
I	ɪ	KIT lid fill bin
O:	ɔ:	THOUGHT Maud dawn fall straw
OI	ɔɪ	CHOICE void foil coin boy
@U	oʊ	GOAT code foal bone go
U	ʊ	FOOT good full woman
u:	u:	GOOSE food soon chew do
V	ʌ	STRUT mud dull gun
i	i	HAPPY serious
i:	i:	FLEECE seed feel mean sea
3:r	ɜ:r	LINER foundered current
4	ɹ	Adam atom coda

### 3.7 Mandarin Chinese

(C) Laboratoire Parole et Langage, Aix-en-Provence, France.

### 3.7.1 Pronunciation dictionary

The pronunciation dictionary was manually created for the syllables by Zhi Na.

### 3.7.2 Acoustic model

The acoustic model was created by Brigitte Bigi from data recorded at Shanghai by Zhi Na, and another one by Hongwei Ding. We address special thanks to hers for giving us access to the corpus. These recordings are a Chinese version of the Eurom1 corpus. See the following publication for details:

**Daniel Hirst, Brigitte Bigi, Hyongsil Cho, Hongwei Ding, Sophie Herment, Ting Wang** (2013). *Building OMProDat: an open multilingual prosodic database*, Proceedings of Tools and Resources for the Analysis of Speech Prosody, Aix-en-Provence, France, Eds B. Bigi and D. Hirst, ISBN: 978-2-7466-6443-2, pp. 11-14.

Notice that the current model was trained from a very small amount of data: this will impact on the results: Do not expect to get good performances for the automatic alignment.

#### More Mandarin Chinese data are welcome!

Here is the phoneset used in the acoustic model:

| SPPAS | - IPA - | Examples | :—:|:—:|:—:|:—:| @ | | | N | ŋ | | | S | ʃ | |  
 | a | a | | | e | e | | | f | f | ʔ | | i | i | ʔ ʔ | | i\_d | | ʔ ʔ | | i | |  
 ʔ ʔ ʔ | | k | k | ʔ | | k\_h | | | | ʔ | | m | m | | n | n | | o | o | ʔ | | p | p | ʔ | | p\_h | | | s | s | ʔ | | S | | ʔ  
 | | s | | ʔ ʔ | | ss | | ʔ | | t | t | ʔ ʔ | | t\_h | | ʔ ʔ ʔ ʔ | | ts | ts | ʔ |  
 | | ts\_h | | ʔ ʔ | | ts\_h | | ʔ ʔ | | ts\_hs | | ʔ ʔ | | ts | | ʔ ʔ | | tss | | ʔ ʔ | | u | u  
 | ʔ | | x | x | ʔ ʔ | | y | y | ʔ ʔ | | z | | |

## 3.8 Southern Min (or Min Nan) resources

(C) *Laboratoire Parole et Langage, Aix-en-Provence, France.*

We address special thanks to S-F Wang for giving us access to the corpus.

**S-F Wang, J. Fon** (2013). *A Taiwan Southern Min spontaneous speech corpus for discourse prosody*, Proceedings of Tools and Resources for the Analysis of Speech Prosody, Aix-en-Provence, France, Eds B. Bigi and D. Hirst, ISBN: 978-2-7466-6443-2, pp. 20-23.

## 3.9 Cantonese resources

### 3.9.1 Dictionary

Here is the phoneset used in the dictionary:

SPPAS	- IPA -	Examples
	b	

---

SPPAS   - IPA -   Examples

---

c

d

f

g

gw

h

j

k

kw

l

m

n

ng

p

s

t

w

z

aa

aai

aak

aam

aan

aang

aap

aat

aau

ai

ak

am

an

ang

ap

at

---

SPPAS	- IPA -	Examples
-------	---------	----------

---

au

e

ei

ek

eng

eoi

eon

eot

i

ik

im

in

ing

ip

it

iu

m

ng

o

oe

oek

oeng

oi

ok

on

ong

ot

ou

u

ui

uk

un

ung

ut

SPPAS	- IPA -	Examples
		yu
		yun
		yut

### 3.9.2 Acoustic Model

(C) DSP and Speech Technology Laboratory, Department of Electronic Engineering, the Chinese University of Hong Kong.

This is a monophone Cantonese acoustic model, based on Jyutping of the Linguistic Society of Hong Kong (LSHK). Each state is trained with 32 Gaussian mixtures. The model is trained with HTK 3.4.1. The corpus for training is CUSENT, also developed in our laboratory.

Generally speaking, you may use the model for non-commercial, academic or personal use.

See COPYRIGHT for the details of the license: “Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International Public License”.

We also have other well-trained Cantonese acoustic models. If you would like to use the models and/or the CUSENT corpus for commercial applications or development, please contact Professor Tan LEE for appropriate license terms.

The character pronunciation comes from Jyutping phrase box from the Linguistic Society of Hong Kong.

“The copyright of the Jyutping phrase box belongs to the Linguistic Society of Hong Kong. We would like to thank the Jyutping Group of the Linguistic Society of Hong Kong for permission to use the electronic file in our research and/or product development.”

If you use the Cantonese acoustic model for academic research, please cite:

**Tan Lee, W.K. Lo, P.C. Ching, Helen Meng** (2002). *Spoken language resources for Cantonese speech processing*, Speech Communication, Volume 36, Issues 3–4, Pages 327-342

- Website: <http://dsp.ee.cuhk.edu.hk>
- Email: [tanlee@ee.cuhk.edu.hk](mailto:tanlee@ee.cuhk.edu.hk)

## 3.10 Polish Resources

\*(C) Brigitte Bigi

### 3.10.1 Pronunciation Dictionary

The Polish dictionary is under the terms of the “GNU Public License”, v3.

The Polish pronunciation dictionary was downloaded from the Ralf catalog of dictionaries for the Simon ASR system at <http://spirit.blau.in/simon/import-pls-dictionary/>. It was then converted (format and phoneset) and corrected by Brigitte Bigi, thanks to the help of Katarzyna Klessa.

### 3.10.2 Acoustic Model

	SPPAS
	a
	b
	c
	x
	d
	dz
	dZ
	E
	E~
	f
	g
	x
	i
	j
	n
	k
	l
	m
	n
	N
	O
	O
	o~
	p
	Q
	r
	r
	s
s   ɸ       S   ʃ       t   t       v   v       w   w       x   x       y   y       z   z	u
	z'
	Z



# Graphical User Interface - GUI

## 4.1 Getting started

The current chapter describes how to use the Graphical User Interface.

Under Windows, once the SPPAS package is opened in the File Explorer, *double-click on the `sppas.bat` file*.

Under MacOS or Linux, once the SPPAS package is opened in the Finder/File Explorer, *double-click on the `sppas.command` file*. The program will first check the version of wxpython and eventually ask to update.

Then, two windows will open automatically.

1. Above, a window with a tips aims at displaying messages to help users to discover SPPAS capabilities.
2. Below, the main frame: to perform automatic annotations, get statistics, filter data, view wav/annotated data, etc.

### 4.1.1 The Tips Frame

This frame picks up randomly and prints a message to help users. Click on the button `Next tip` to read another message, or click on the `Close` button to close the frame.



Figure 4.1: The Tips Frame

The `Settings` frame also allows to show/hide the Tips frame at start-up.

If you want to suggest new tips (to help the other users), send them by e-mail to the author at [brigitte.bigi@gmail.com](mailto:brigitte.bigi@gmail.com). They will be included in the next version.

### 4.1.2 The main frame

The main frame is made of a menu, a toolbar and the main content. All of them are described in the next sub-sections.

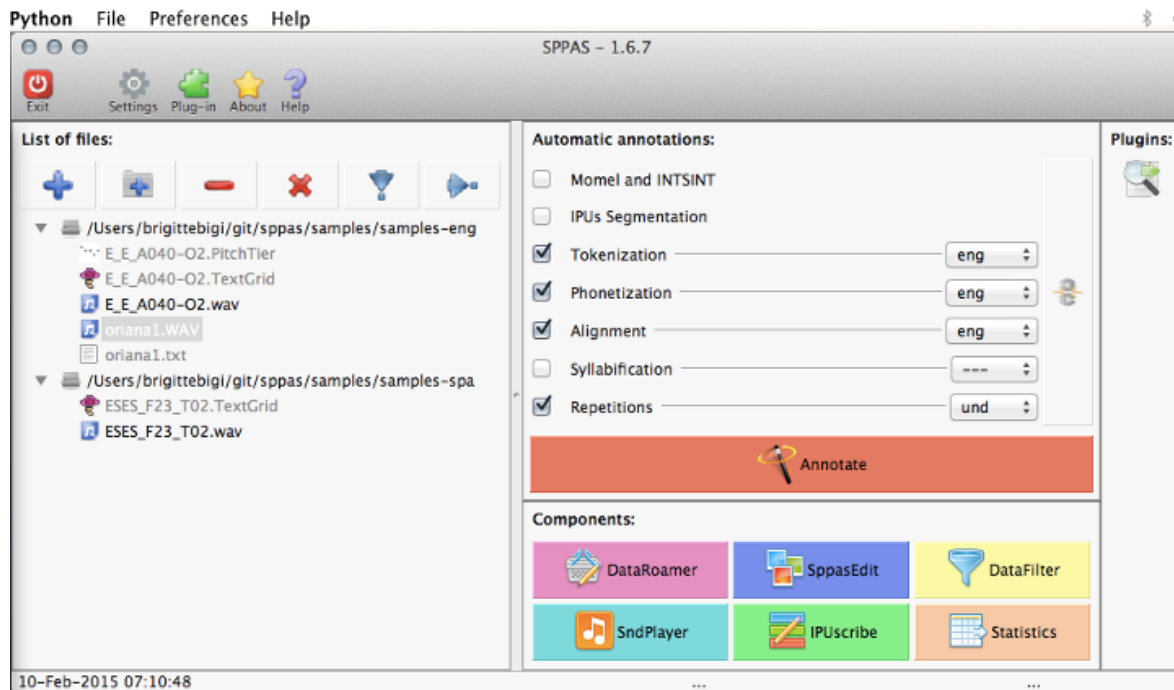


Figure 4.2: The Main Frame of version 1.6.7

#### The menu

It allows to access to all functions of the GUI, and to manage it.

The “File” menu can be used to manage files (add, remove, ...) and to exit the program.

In the “Preferences” menu, check boxes allow to show/hide the toolbar and the status bar, and to fix settings (see below for details).

The “Help” menu includes an item to open the SPPAS website in a web browser, to access inline documentation, to declare a bug and to send feedback to the author. In the latter case, replace the text “Describe what you did here...” by your own comment, question or suggestion and choose how to send the e-mail: with your own default mailer, with gmail (opened in your web browser) or by another e-mail client. Then, close the frame by clicking on the “Close” button.

#### The toolbar

The toolbar includes 5 buttons:

1. Exit;

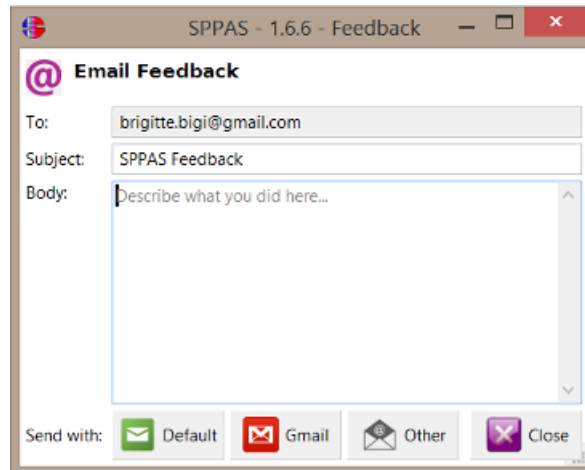


Figure 4.3: The Feedback Frame

2. Settings;
3. Plug-in;
4. About;
5. Help.

The look of the toolbar can change depending of the Theme of the icons.

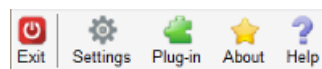


Figure 4.4: Main toolbar, with Default icon theme

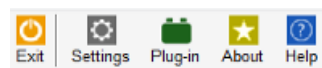


Figure 4.5: Main toolbar, with Metro icon theme

The `Exit` button closes all SPPAS frames *properly*. Please, do not kill SPPAS by clicking on the arrow of the windows manager! **Use this Exit button** to close SPPAS.

To fix new settings, click on the `Settings` icon, then choose your preferred options:

- General: fix background colour, font colour and font of the GUI, and enable/disable tips at start-up.
- Theme: fix the icons of the GUI.
- Annotations: fix automatic annotations parameters.

The Settings can be saved in a file to be used each time SPPAS is executed. To close the frame, click on:

- “Cancel” button to ignore changes,
- “Close” to apply changes then close the frame.

The `Plug-in` button allows to install a new plugin.

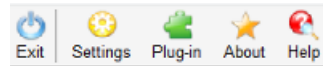


Figure 4.6: Main toolbar, with CrystalClear icon theme

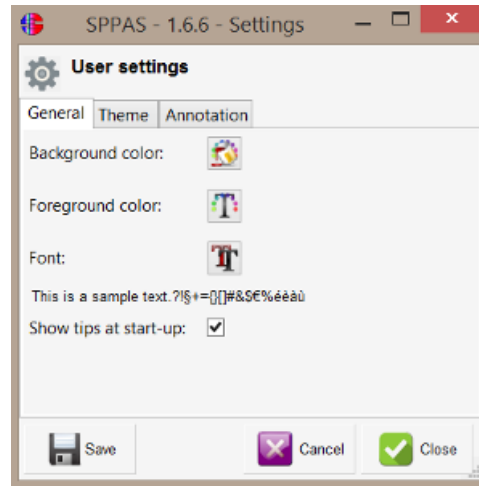


Figure 4.7: Settings is used to manage user preferences

The `About` button opens a new frame which gives the main information about SPPAS: authors, license, web site URL, etc.

The `Help` button opens the documentation and allows to browse in the chapters and sections.

### The content of the frame.

The content of the main frame is made of 4 main panels:

1. the file list panel (FLP), at left;
2. the automatic annotation panel (AAP); at middle-top;
3. the components panel (CCP), at middle-bottom;
4. the plug-ins panel (P&P), at right.

## 4.2 File List Panel

The File List Panel (FLP) consists of:

- a file explorer: a tree-style set of files and directories.
- at top, a toolbar: a set of buttons to perform actions.

### 4.2.1 The file explorer

The list contains Directories and Files the user added, but only files that SPPAS can deal with (depending on the file extension).

To select:

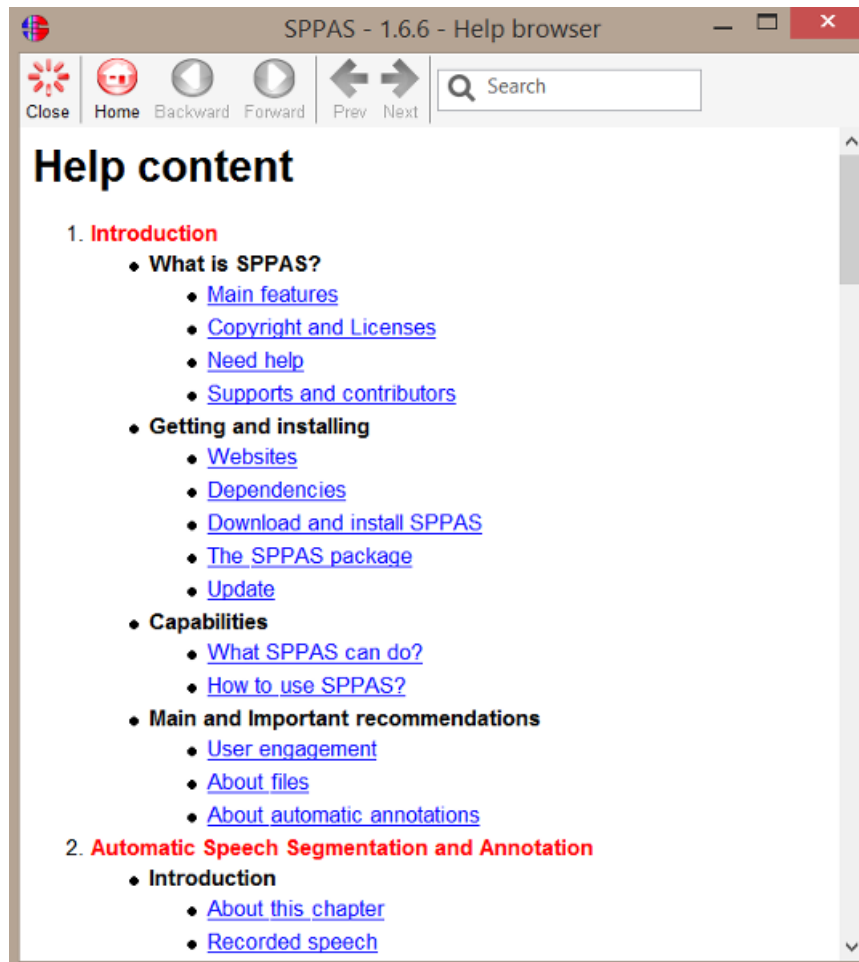


Figure 4.8: The Help Browser Frame

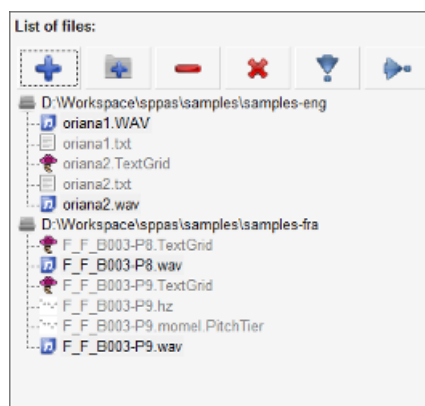


Figure 4.9: File List Panel (FLP)

- a single click on a file name highlights and selects the chosen file (displayed in the list).
- a single click on a directory name highlights and selects the chosen directory (displayed in the list).

Like in any other file explorer, while clicking and pressing the “CTRL” key (“COMMAND” on MacOS) on the keyboard, you can select multiple files and/or directories. Idem with the “SHIFT” key.

### 4.2.2 The toolbar

#### Add file(s).

A single-click on the `Add File` button opens a frame that allows to select the files to get. By default, only “wav” files are proposed. If you select a *wav file(s)*, all files with the same name will be automatically added into the file explorer. It is possible to change the wildcard of this frame and to select each file to add. In both cases, only files with an appropriate extension will be added in the file explorer.

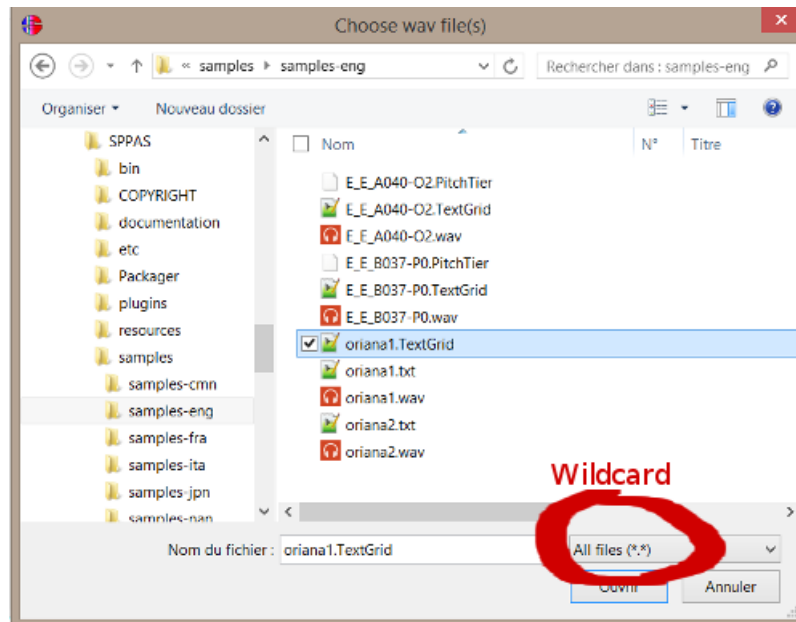


Figure 4.10: Adding specific files

Remark: The files are added in the list, but they are not opened.

#### Add a directory.

A single-click on the `Add Dir` button opens a frame that allows to select the directory to get. Each wav file, and all related files (i.e. with the same name and with an appropriate extension) will be added into the file explorer.

#### Remove file(s).

A single-click on the `Remove` button removes the selected files/directories.

Notice that files are not deleted from the disk, they are just removed of the FLP.

**Delete file(s).**

A single-click on the `Delete` button deletes definitively the selected files/directories of your computer, and remove them of the FLP. Notice that there is no way to get them back!

A dialogue frame will open, and you'll have to confirm or cancel the definitive file(s) deletion.

**Copy file(s).**

A single-click on the `Copy` button allows to copy the selected file(s), change their name and/or change the location (eventually, change the file extension).

**Export file(s).**

Export annotated files in an other format (csv, txt, ...):

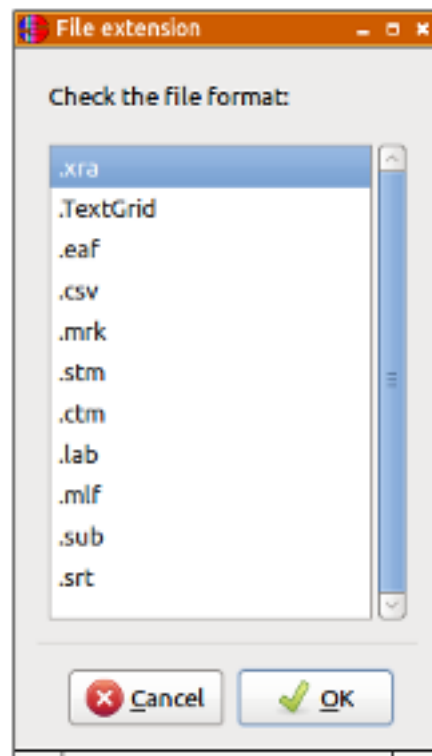


Figure 4.11: Export: Check the expected format in the list

After the export, a new frame will open to report if file(s) were exported successfully or not.

## 4.3 Automatic Annotations Panel

### 4.3.1 Description

The Automatic Annotations Panel (AAP) consists of a list of buttons to check (the left column), the annotation name (middle) and buttons to fix the language of each annotation (at right).

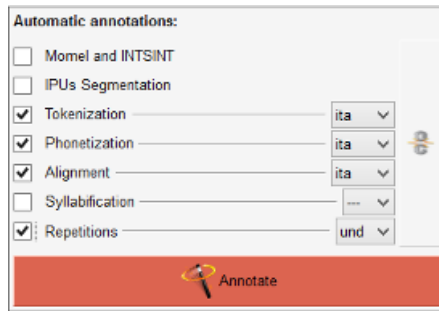


Figure 4.12: Automatic Annotations Panel (AAP)

### 4.3.2 AAP Usage

Remark: to understand what each annotation is for, please refer to the chapter “Automatic Speech Segmentation and Annotation” of this document.

1. Select each annotation to perform. Each annotation can be configured by clicking on the annotation name.
2. Select the language for all annotations, or for each one independently by clicking on the “chains” button.
3. Click on the *Annotate* button... and wait! Please, wait! Particularly for Tokenization or Phonetization: loading resources (lexicons or dictionaries) is very long. Sometimes, the progress bar does not really “progress”...it depends on the system. So, just wait!
4. It is important **to read the Procedure Outcome report** to check that everything happened normally during the automatic process.

### 4.3.3 The procedure outcome report

It is very important to read conscientiously this report: it mentions exactly what happened during the automatic annotation process. This text can be saved: it is recommended to be kept it with the related data because it contains information that are interesting to know for anyone using the annotations!

The text first indicates the version of SPPAS that was used. This information is very important, mainly because annotations in SPPAS and their related resources are regularly improved and then, the result of the automatic process can change from one version to the other one.

Secondly, the text mentions information related to the given input:

1. the selected language of each annotation, only if the annotation is language-dependent). For some language-dependent annotations, SPPAS can still perform the annotation even if the resources for a given language are not available: in that case, select “und”, which is the iso639-3 code for “undetermined”.
2. the list of files to be annotated;
3. the list of annotations and if each annotation was activated or disabled. In that case, activated means that the checkbox of the AAP was checked by the user and that the resources are available for the given language. On the contrary, disabled means that either the checkbox of the AAP was not checked or the required resources are not available.



In the following, each automatic annotation is described in details, for each annotated file. Three levels of information must draw your attention:

1. “[ OK ]” means that everything happened normally. The annotation was performed successfully.
2. “[ WARNING ]” means that something happened abnormally, but SPPAS found a solution, and the annotation was still performed successfully.
3. “[ ERROR ]” means that something happened abnormally, and SPPAS failed to found a solution. The annotation was either not performed, or performed with a bad result.

Finally, the “Result statistics” section mentions the number of files that was annotated for each step, or -1 if the annotation was disabled.

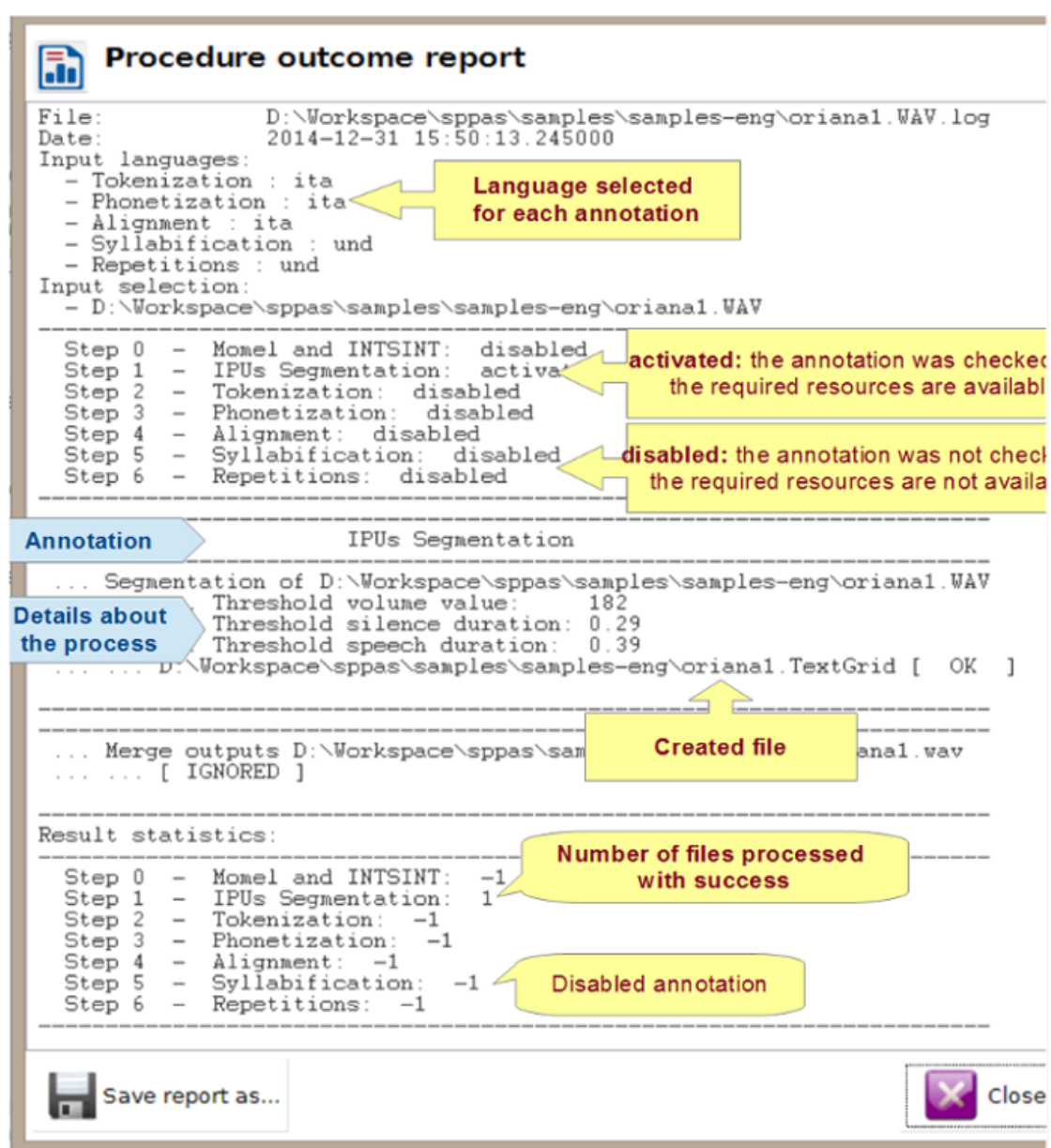


Figure 4.13: Procedure outcome report

## 4.4 Components Panel

The components are useful for the analysis of annotated files: display the automatic alignments with the audio signal, estimates statistics on the annotations, filter the annotated data to get only the annotations you are interested in, etc.

To execute a specific component, select file(s) in the file explorer, then click on the button of a component. It will open the component frame, and add the selected file(s) in the file manager of the component. Refer to the documentation of each component to know how to use it.

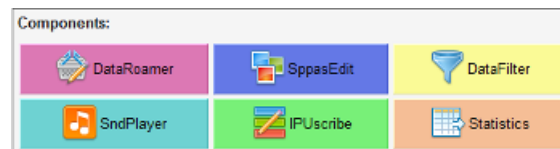


Figure 4.14: Component Panel (CCP)

Six components are available:

1. `DataRoamer` allows to explore the annotated files: cut/copy/paste/rename/duplicate tiers, move a tier from one file to another one, etc.
2. `SndPlayer` allows to play your speech files.
3. `IPUscribe` is useful to perform manual orthographic transcription.
4. `SppasEdit` displays speech files and annotated files, and is very useful to take a screenshot! Easy way to zoom/scroll, change colours, choose the tiers to display, etc;
5. `DataFilter` allows to select annotations: fix a set of filters to create new tiers with only the annotations you are interested in!
6. `Statistics` estimates the number of occurrences, the duration, etc. of the annotations, and allows to save in CSV (for Excel, OpenOffice, R, MatLab,...).

All of the components share the same style:

- a menu, a toolbar
- at left: the list of files
- at right: a notebook to open files in tabs.

### 4.4.1 The main toolbar



Figure 4.15: The toolbar of the components

Seven buttons are available:

- Exit: go out of the component;
- Add: add file(s) in the list;
- Remove: remove all un-checked (i.e. un-used) files of the list;

- New tab: open an empty new tab in the notebook;
- Close tab: close the selected tab;
- Settings: the same that the main frame of SPPAS;
- About: get information about the component.

#### 4.4.2 The list of files

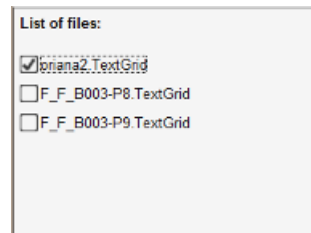


Figure 4.16: Components: list of files

The most important information to know is that when files are added in the list, they are not opened: only checked files are loaded.

#### 4.4.3 DataRoamer

DataRoamer displays detailed information about annotated files and allows to manage the tiers: cut/copy/paste/rename/duplicate tiers, move a tier from one file to another one, etc.

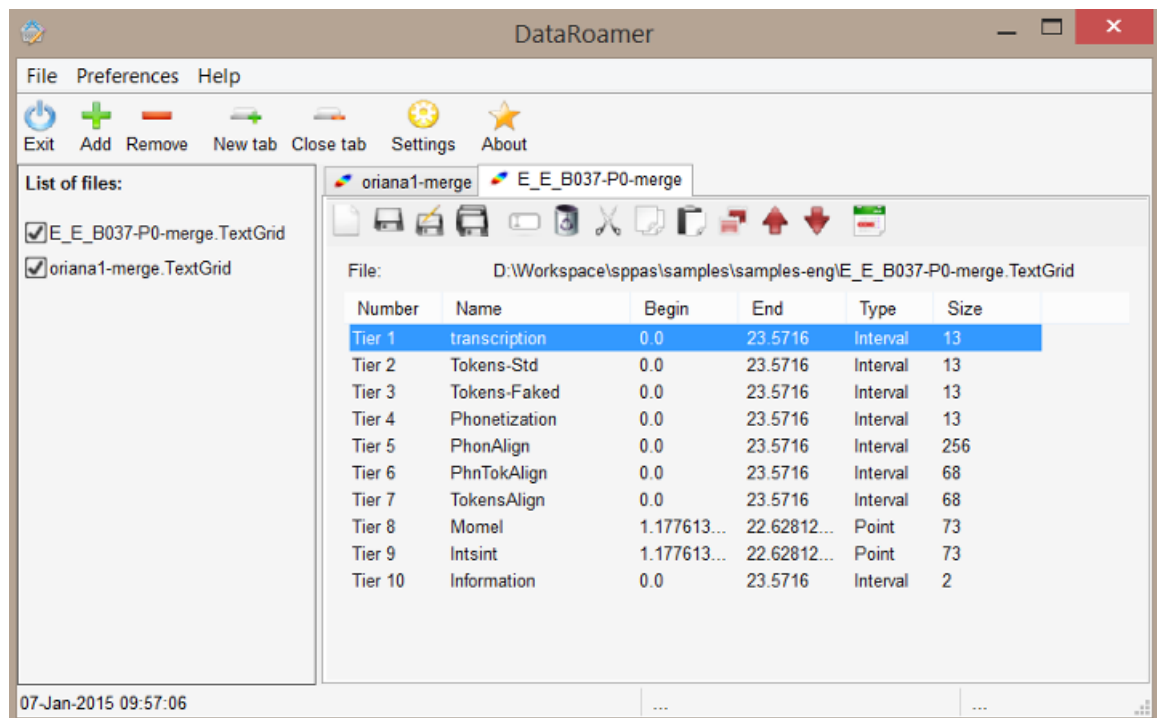


Figure 4.17: Component: DataRoamer

#### 4.4.4 SndRoamer

SndRoamer allows to play your speech files and to get information.

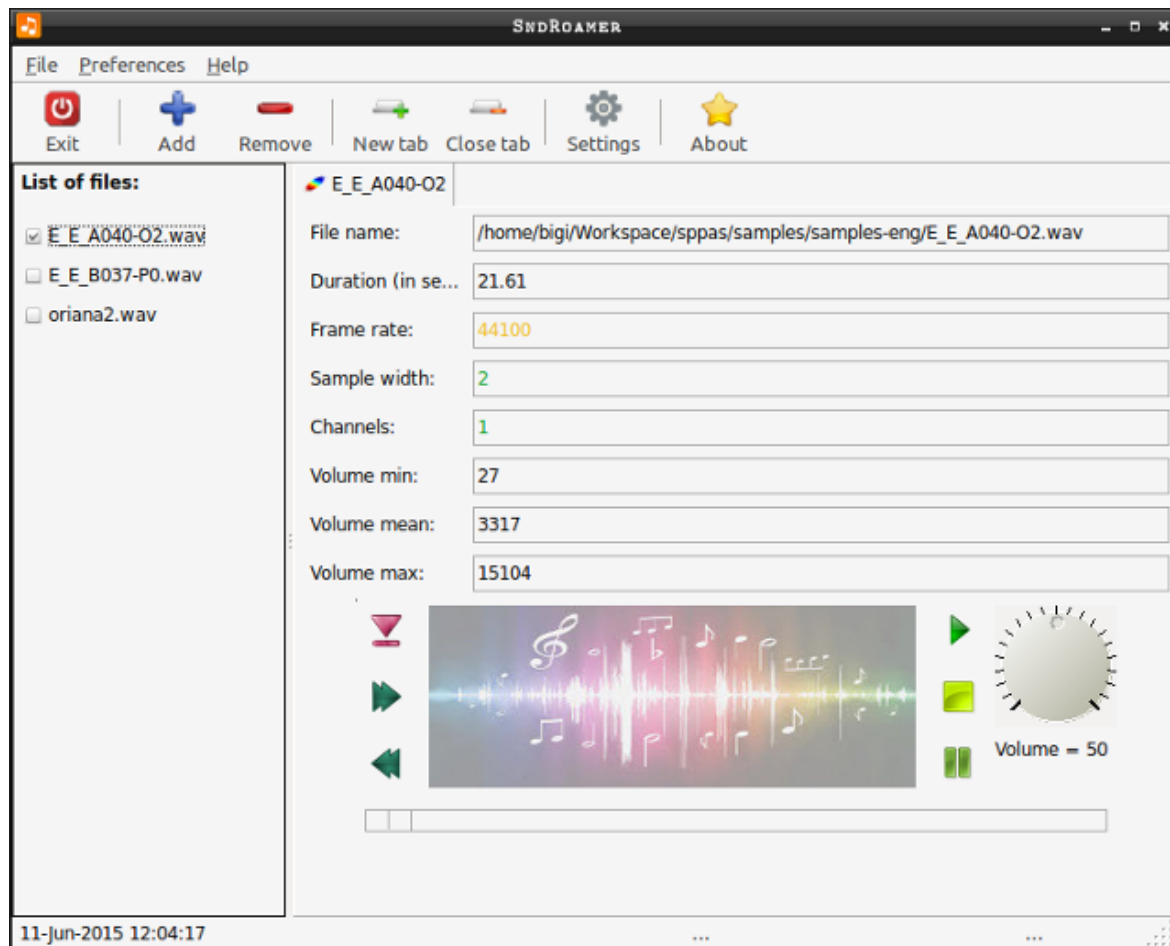


Figure 4.18: Component: SndRoamer

#### 4.4.5 IPUscribe

IPUscribe is useful to perform manual orthographic transcription.

#### 4.4.6 SppasEdit

This component is still under-development, some “troubles/crashes” can occur while using it... but the data will never been corrupted!!

DataViewer displays speech files and annotated files, and is very useful to take a nice screenshot! Most of the screenshots of annotated data of this document were taken with it...

Try the Demo of this component: easy way to zoom/scroll, change colours, choose the tiers to display, etc!

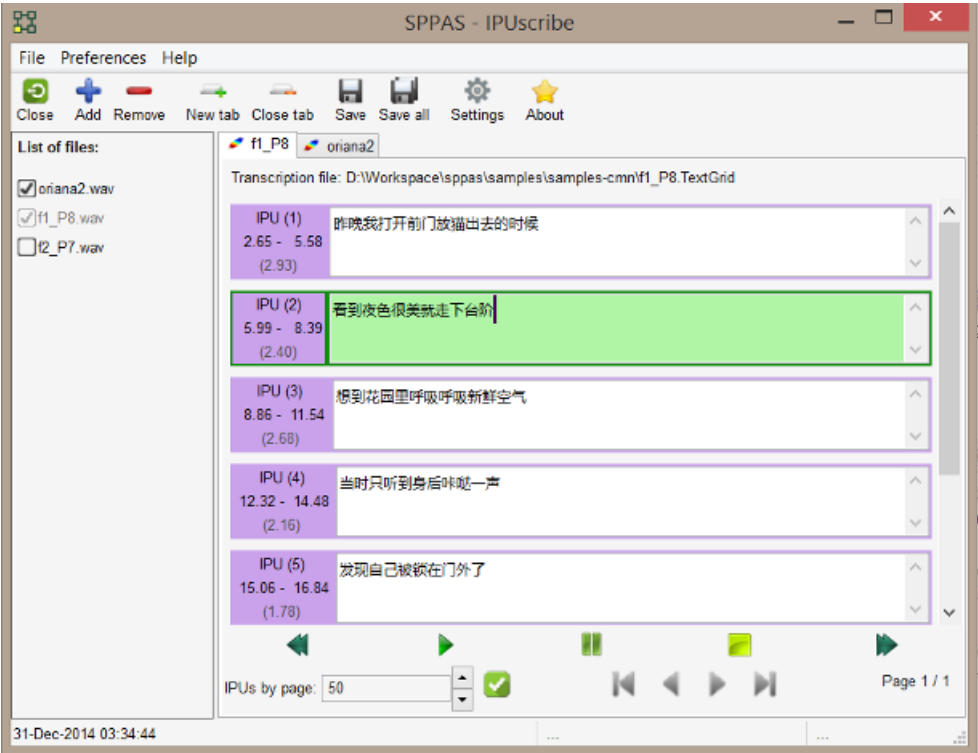


Figure 4.19: Component: IPUTranscriber

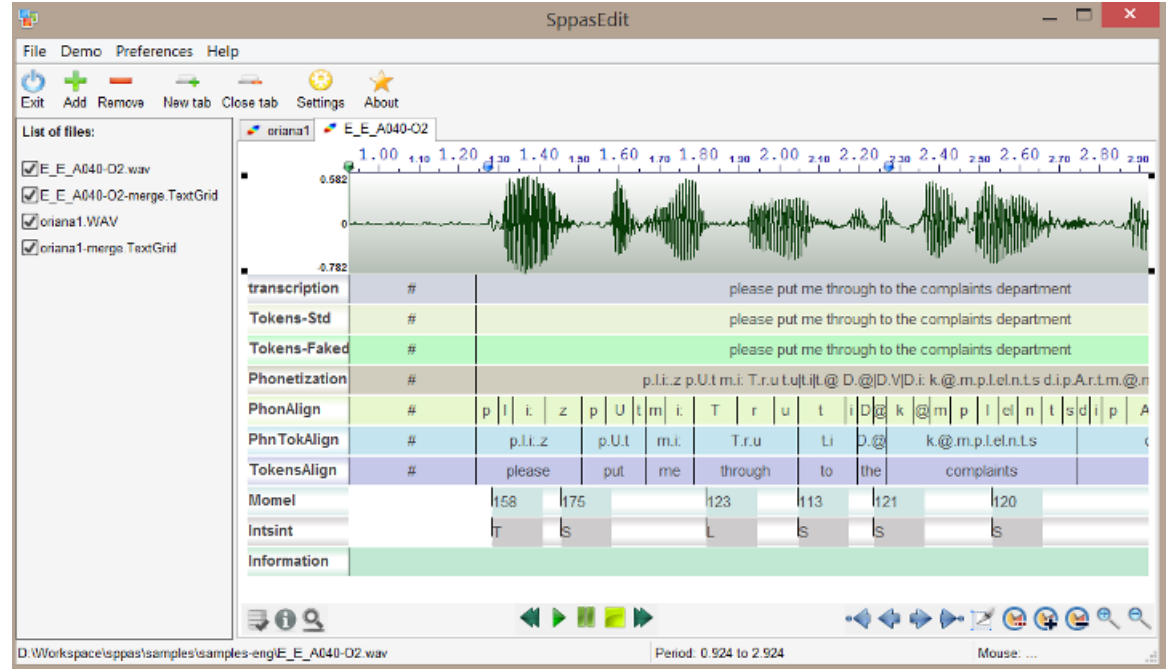


Figure 4.20: Component: SppasEdit

### 4.4.7 DataFilter

DataFilter allows to select annotations: fix a set of filters to create new tiers with only the annotations you are interested in! This system is based on the creation of 2 different types of filters:

1. single filters, i.e. search in a/several tier(s) depending on the data content, the time values or the duration of each annotation;
2. relation filters, i.e. search on annotations of a/several tier(s) in time-relation with annotations of another one.

These later are applied on tiers of many kind of input files (TextGrid, eaf, trs, csv...). The filtering process results in a new tier, that can re-filtered and so on.

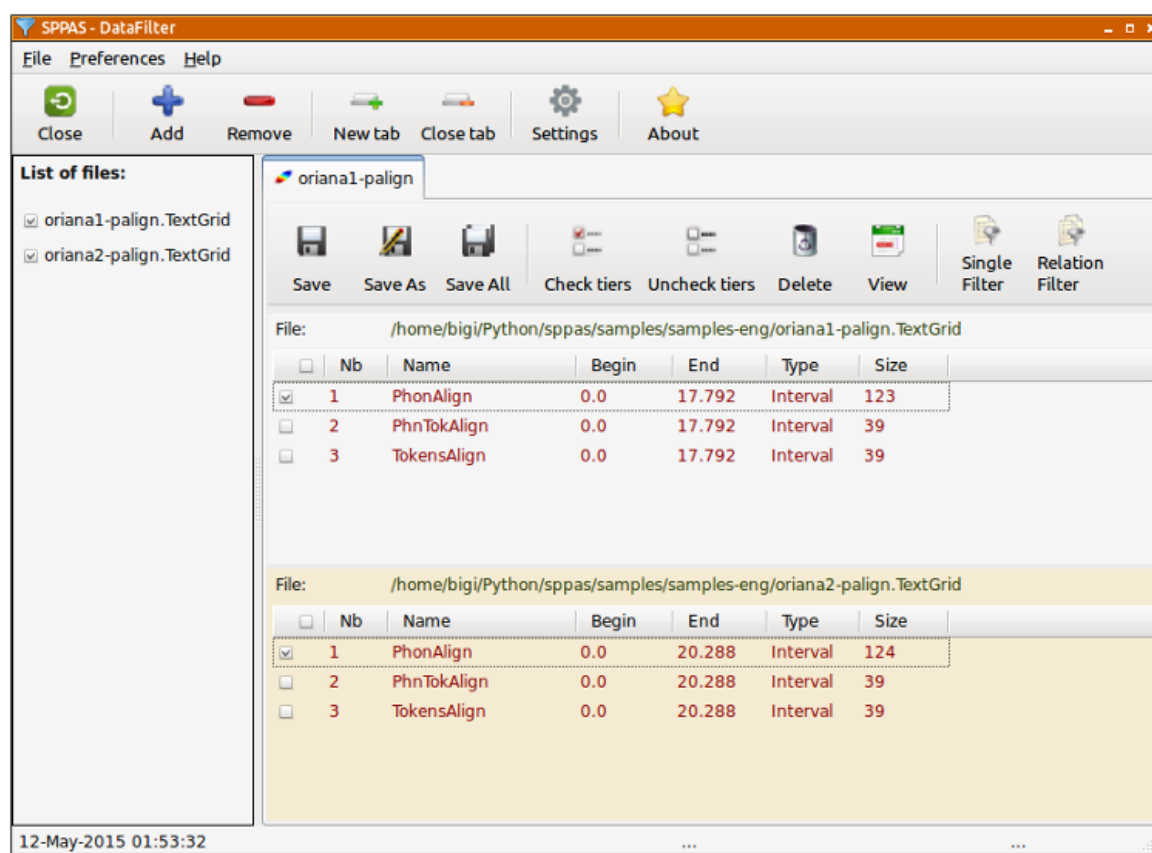


Figure 4.21: Component: DataFilter

#### Filtering annotations of a tier: SingleFilter

Pattern selection is an important part to extract data of a corpus and is obviously and important part of any filtering system. Thus, if the label of an annotation is a string, the following filters are proposed in DataFilter:

- exact match: an annotation is selected if its label strictly corresponds to the given pattern;
- contains: an annotation is selected if its label contains the given pattern;
- starts with: an annotation is selected if its label starts with the given pattern;

- ends with: an annotation is selected if its label ends with the given pattern.

All these matches can be reversed to represent respectively: does not exactly match, does not contain, does not start with or does not end with. Moreover, this pattern matching can be case sensitive or not.

For complex search, a selection based on regular expressions is available for advanced users.

A multiple pattern selection can be expressed in both ways:

- enter multiple patterns at the same time (separated by commas) to mention the system to retrieve either one pattern or the other, etc.
- enter one pattern at a time and choose the appropriate button: “Apply All” or “Apply any”.

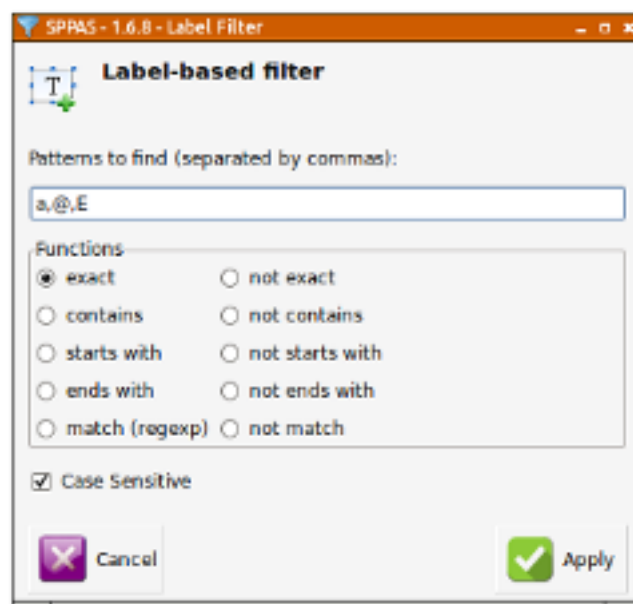


Figure 4.22: Frame to create a filter on annotation labels. In that case, filtering annotations that exactly match either a, @ or E

Another important feature for a filtering system is the possibility to retrieve annotated data of a certain duration, and in a certain range of time in the timeline.

Search can also starts and/or ends at specific time values in a tier.

All the given filters are then summarized in the “SingleFilter” frame. To complete the filtering process, it must be clicked on one of the apply buttons and the new resulting tiers are added in the annotation file(s).

In the given example:

- click on “Apply All” to get either a, @ or E vowels during more than 80ms, after the 5th minute.
- click on “Apply Any” to get a, @ or E vowels, and all annotations during more than 80 ms, and all annotations after the 5th minute.

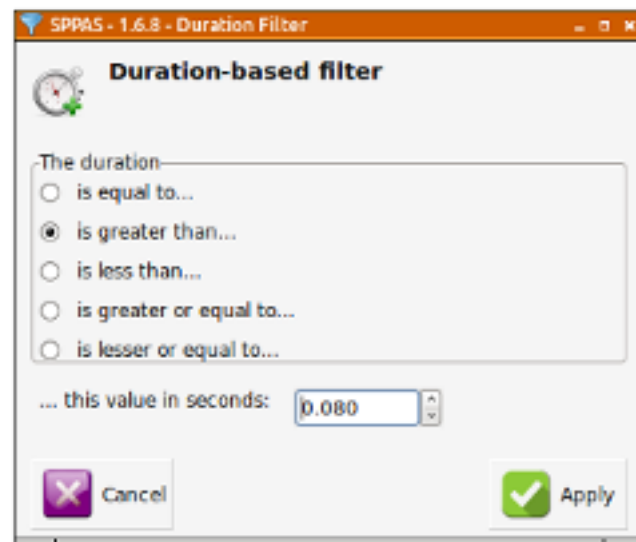


Figure 4.23: Frame to create a filter on annotation durations. In that case, filtering annotations that are during more than 80 ms

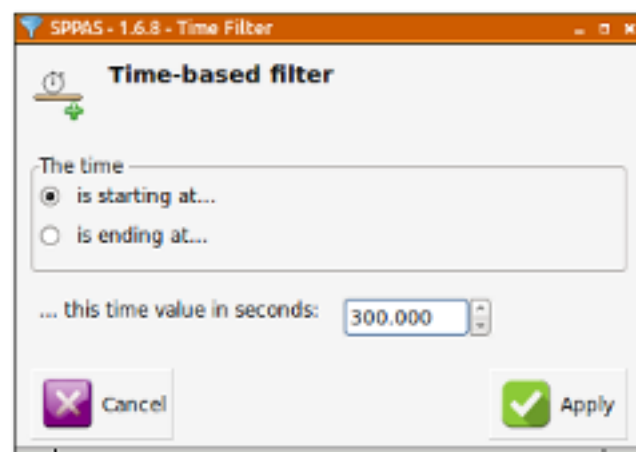


Figure 4.24: Frame to create a filter on annotation time values. In that case, filtering annotations that are starting after the 5th minute.



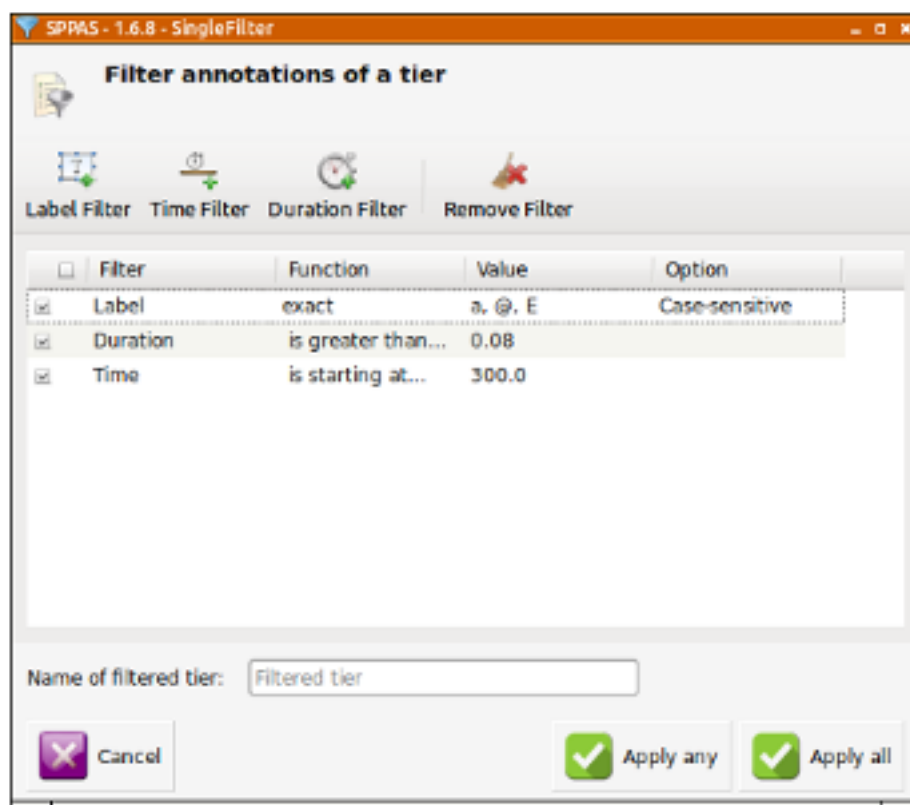


Figure 4.25: DataFilter: SingleFilter frame

### Filtering on time-relations between two tiers

Regarding the searching problem, linguists are typically interested in locating patterns on specific tiers, with the possibility to relate different annotations a tier from another. The proposed system offers a powerful way to request/extract data, with the help of Allen's interval algebra.

In 1983 James F. Allen published a paper in which he proposed 13 basic relations between time intervals that are distinct, exhaustive, and qualitative:

- distinct because no pair of definite intervals can be related by more than one of the relationships;
- exhaustive because any pair of definite intervals are described by one of the relations;
- qualitative (rather than quantitative) because no numeric time spans are considered.

These relations and the operations on them form Allen's interval algebra. These relations were extended to Interval-Tiers as Point-Tiers to be used to find/select/filter annotations of any kind of time-aligned tiers.

For the sake of simplicity, only the 13 relations of the Allen's algebra are available in the GUI. But actually, we implemented the 25 relations proposed Pujari and al. (1999) in the INDU model. This model is fixing constraints on INtervals (with Allen's relations) and on DURATION (duration are equals, one is less/greater than the other). Such relations are available while requesting with Python.

At a first stage, the user must select the tiers to be filtered and click on "RelationFilter". The second stage is to select the tier that will be used for time-relations.

The next step consists in checking the Allen's relations that will be applied. The last stage is to fix the name of the resulting tier. The above screenshots illustrates how to select the first phoneme of each token, except

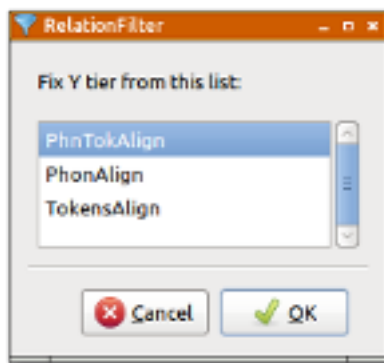


Figure 4.26: Fix time-relation tier name

for tokens that are containing only one phoneme (in this later case, the “equal” relation should be checked).

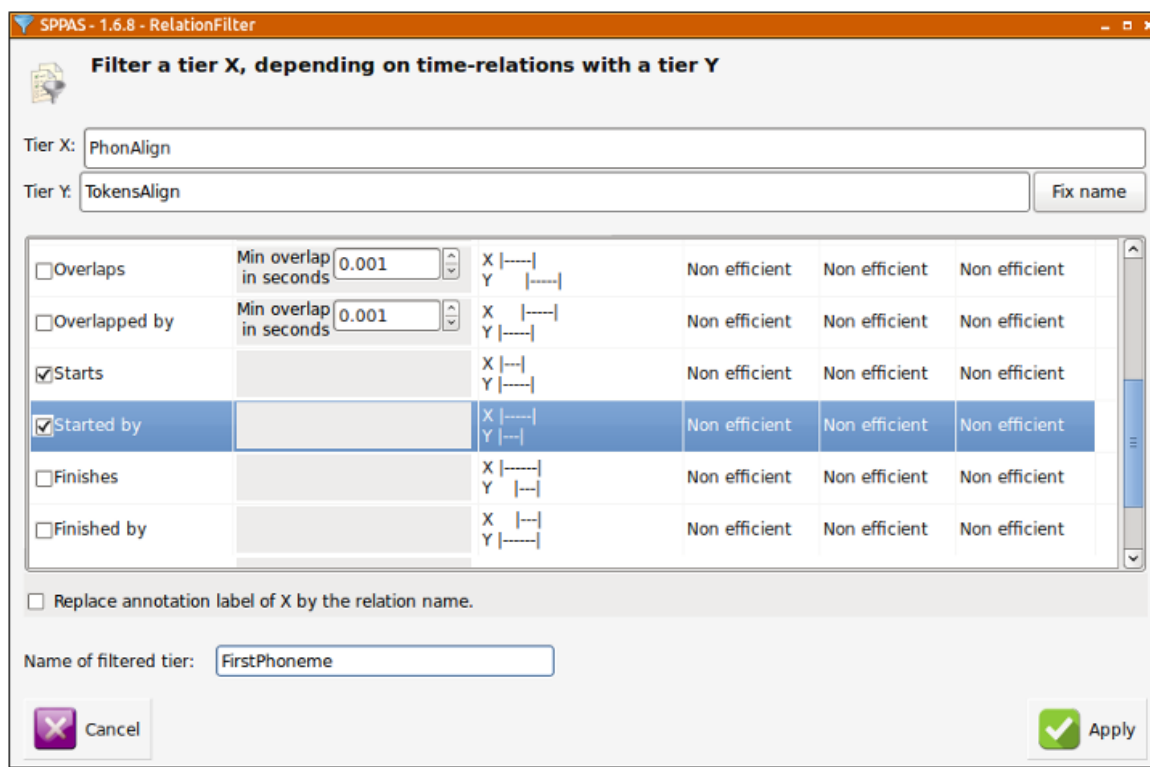


Figure 4.27: DataFilter: RelationFilter frame

To complete the filtering process, it must be clicked on the “Apply” button and the new resulting tiers are added in the annotation file(s).

#### 4.4.8 Statistics

Statistics allows to get descriptives statistics about a set of selected tiers. It also allows to estimate a user agreement rate (Kappa as a first stage) and includes TGA (Time Group Analyzer), originally available at <http://wwwhomes.uni-bielefeld.de/gibbon/TGA/>, a tool developed by Dafydd Gibbon, emeritus professor of English and General Linguistics at Bielefeld University.

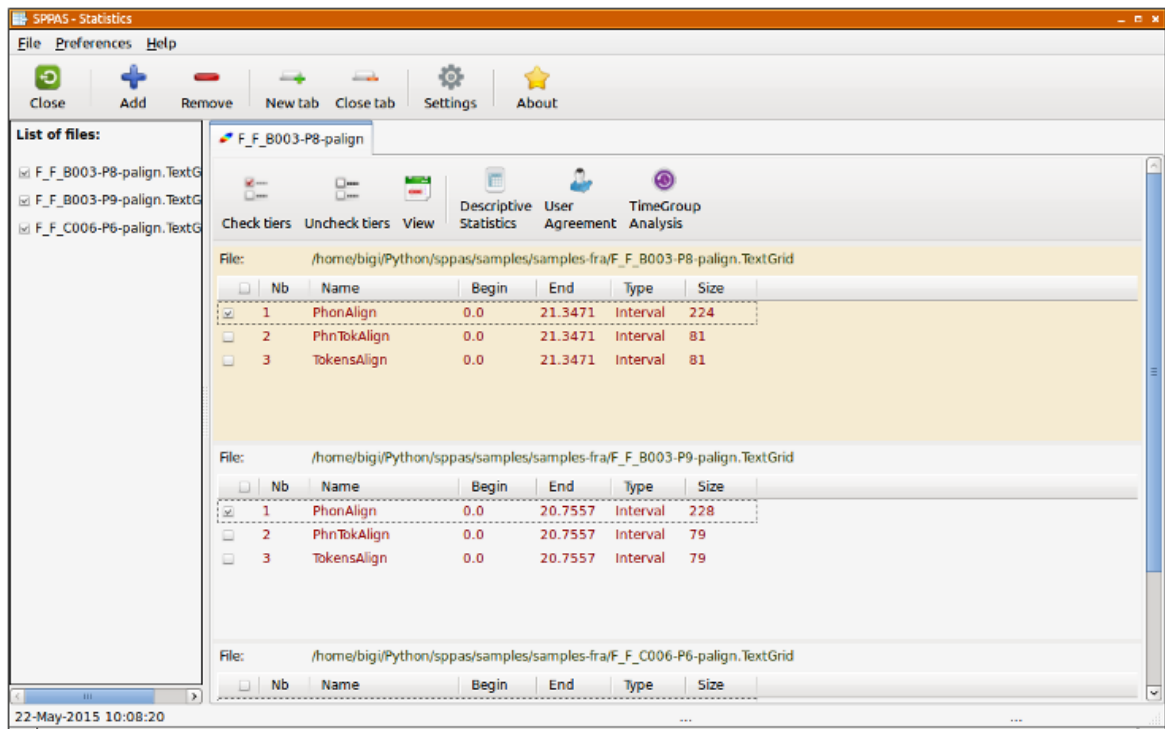


Figure 4.28: Component: Statistics

### Descriptive statistics

It allows to estimate the number of occurrences, the duration, etc. of the annotations of a set of selected tiers, and allows to save in CSV (for Excel, OpenOffice, R, MatLab,...). It offers a serie of sheets organized in a notebook. The first tab is displaying a summary of descriptive statistics of the set of given tiers. The other tabs are indicating one of the statistics over the given tiers. The followings are estimated:

- occurrences: the number of observations
- total durations: the sum of the durations
- mean durations: the arithmetic mean of the duration
- median durations: the median value of the distribution of durations
- std dev. durations: the standard deviation value of the distribution of durations

All of them can be estimated on a single annotation label or on a serie of them. The length of this context can be optionally changed while fixing the “N-gram” value (available from 1 to 5), just above the sheets.

Each displayed sheet can be saved as a CSV file, which is a useful file format to be read by R, Excel, OpenOffice, LibreOffice, and so... To do so, display the sheet you want to save and click on the button “Save sheet”, just below the sheets. If you plan to open this CSV file with Excel under Windows, it is recommended to change the encoding to UTF-16. For the other cases, UTF-8 is probably the most relevant.

The annotation durations are commonly estimated on the Midpoint value, without taking the radius into account; see (Bigi et al, 2012) for explanations about the Midpoint/Radius. Optionnally, the duration can either be estimated by taking the vagueness into account, then check “Add the radius value” button, or by ignoring the vagueness and estimating only on the central part of the annotation, then check “Deduct the radius value”.

For those who are estimating statistics on XRA files, you can either estimate stats only on the best label (the label with the higher score) or on all labels, i.e. the best label and all its alternatives (if any).

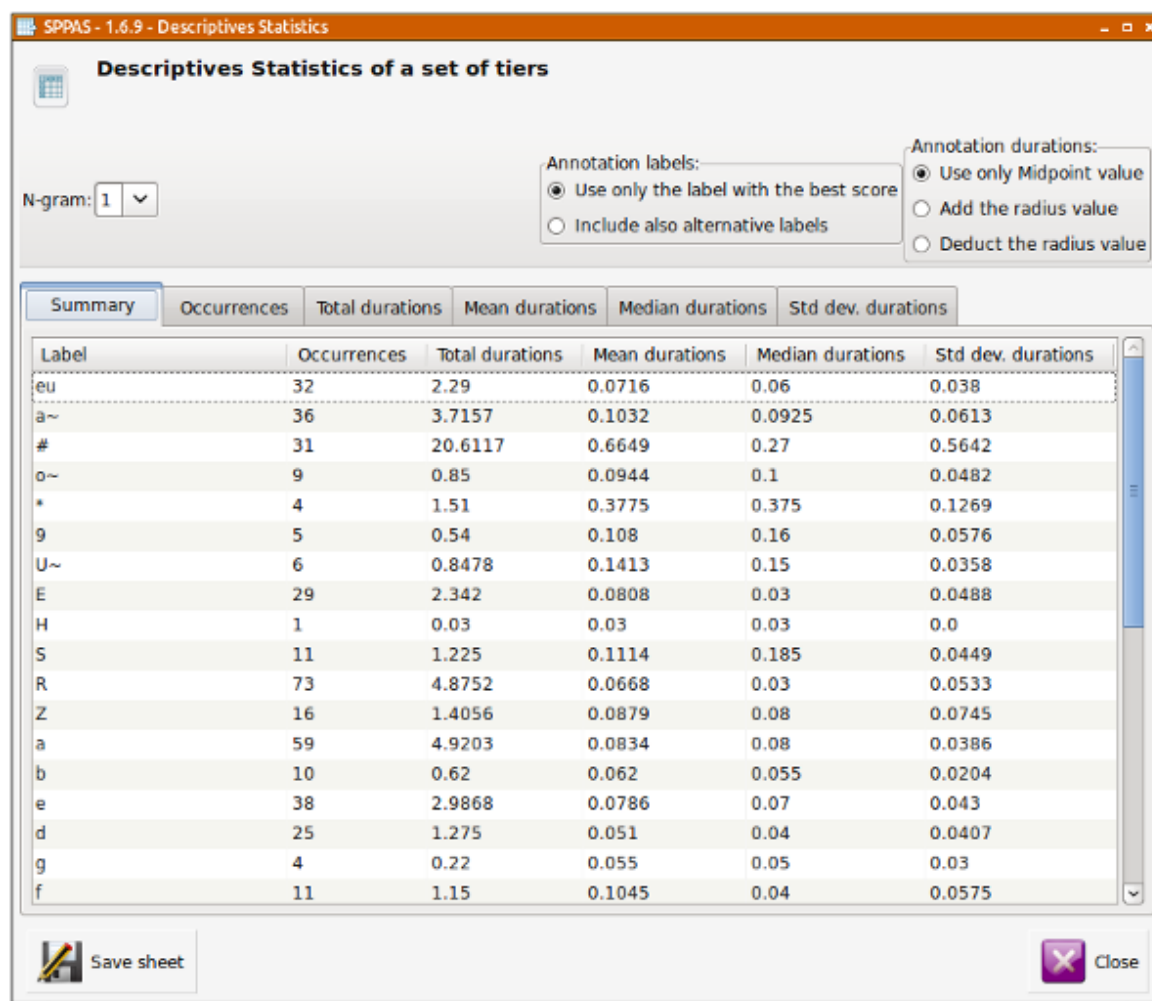


Figure 4.29: Component: Statistics

## User agreement

SPPAS integrates the estimation of the Cohen's Kappa. It is currently limited to the evaluation of this user agreement between labels of 2 tiers with the same number of intervals.

## TGA - Time Group Analyzer

*Dafydd Gibbon* (2013). **TGA: a web tool for Time Group Analysis**, Tools and Resources for the Analysis of Speech Prosody, Aix-en-Provence, France, pp. 66-69.

The TGA is an online batch processing tool which provides a parametrised mapping from time-stamps in speech annotation files in various formats to a detailed analysis report with statistics and visualisations. TGA software calculates, inter alia, mean, median, rPVI, nPVI, slope and intercept functions within interpausal groups, provides visualisations of timing patterns, as well as correlations between these, and parses interpausal

groups into hierarchies based on duration relations. Linear regression is selected mainly for the slope function, as a first approximation to examining acceleration and deceleration over large data sets.

The TGA online tool was designed to support phoneticians in basic statistical analysis of annotated speech data. In practice, the tool provides not only rapid analyses but also the ability to handle larger data sets than can be handled manually.

*Katarzyna Klessa, Dafydd Gibbon (2014). **Annotation Pro + TGA: automation of speech timing analysis**, 9th International conference on Language Resources and Evaluation (LREC), Reykjavik (Iceland). pp. 1499-1505, ISBN: 978-2-9517408-8-4.*

The integrated Annotation Pro + TGA tool incorporates some TGA features and is intended to support the development of more robust and versatile timing models for a greater variety of data.

The integration of TGA statistical and visualisation functions into Annotation Pro+TGA results in a powerful computational enhancement of the existing AnnotationPro phonetic workbench, for supporting experimental analysis and modelling of speech timing.

So... What's the novelty...

TGA is partly implemented in SPPAS. The Statistics component of SPPAS allows to estimates TGA within the SPPAS framework. It results in the following advantages:

- it can read either TextGrid, csv, Elan, HTK, or Scilite, or any file format supported by SPPAS,
- it can save TGA results either as a table in a CSV file or as an annotation file (of any of the format supported by SPPAS),
- it estimates 2 linear regressions (the y-axis is the duration in both cases):
  1. with x-axis based on positions, like in the inline TGA
  2. with x-axis based on time-stamps, like in the AnnotationPro+TGA

## 4.5 Plugins Panel

This panel includes the icons of plugins that were previously installed. To execute a plug-in, select file(s) in the File explorer, click on the icon of the plug-in and follow instructions of the plugged program.



Figure 4.30: Plugins Panel with two plug-ins: TierMapping and MarsaTag

### 4.5.1 Installing a Plugin

To install a plugin, follow this workflow:

1. Download and unpack the Plugin package in a new folder (this folder will be removed after the plugin installation).
2. Execute SPPAS.
3. Click on the 'Plugin' icon of the toolbar: it will open a new frame.
4. Indicate the folder of the new Plugin in the text entry.
5. Follow the Plugin instructions (if any).
6. See the new Plugin icon in the Plugins panel of the main frame.

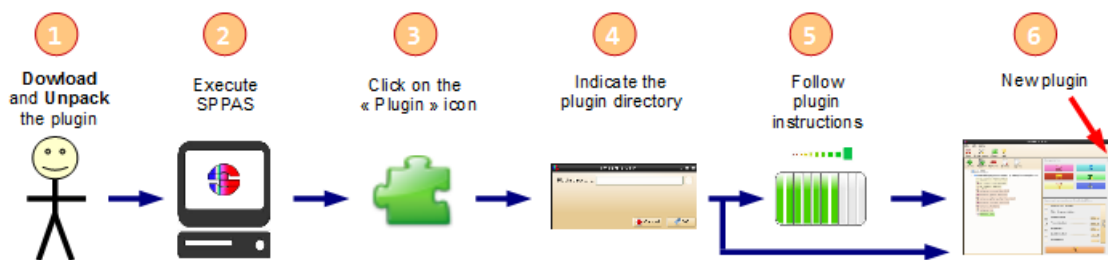


Figure 4.31: Installing a new Plugin

### 4.5.2 Marsatag-Plugin

MarsaTag-Plugin is a plugin to use the French POS-tagger *MarsaTag* directly from SPPAS. Using this plugin offers two main advantages:

1. the POS-Tags are time-aligned; and
2. MarsaTag is ready-to-use: all steps to execute MarsaTag are pre-configured.

Notice that you must install MarsaTag first: <http://sldr.org/sldr000841>

### 4.5.3 TierMapping

This plugin allows to create new tiers by mapping the labels of an existing tier, with new labels. Some mapping-tables are included in the plugin:

- map the phonemes of the tier “PhonAlign” from SAMPA to IPA,
- map the phonemes of the tier “PhonAlign” to their articulatory properties.

Furthermore, any user can easily create a mapping table. The only requirements are that all tables must be UTF-8 encoded in a CSV file, with columns separated by semi-columns. These tables can be imported/modified/saved with OpenOffice, LibreOffice, Excel, ... The most important is to not change its properties (UTF8, semi-columns) and location (inside the `resources` directory of the plug-in directory).

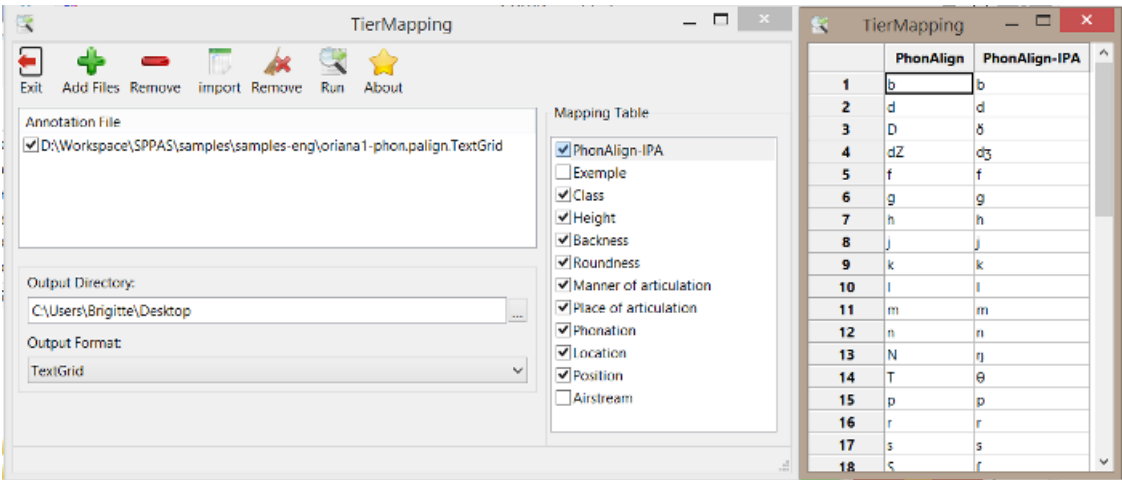


Figure 4.32: Plug-in: TierMapping





# Command-Line user Interface - CLI

## 5.1 Overview

A command-line interface (CLI), also known as command-line user interface, is a means of interacting with a computer program where the user issues commands to the program in the form of successive lines of text (command lines). Command-line interfaces provide a more concise and powerful means to control the program than the GUI. Operating system command line interfaces are called a command-line interpreter, command processor or shell. It displays a prompt, accept a “command line” typed by the user terminated by the Enter key, then execute the specified command and provide textual display of results or error messages. When a shell is active a program is typically invoked by typing its name followed by command-line arguments (if any).

Each capability implemented in SPPAS corresponds to a program that can be invoked by its name in a shell. Such programs are located in the `bin` sub-directory of the `sppas` directory included in the SPPAS package. It is usual for a program to be able to display a brief summary of its parameters. Each program included in SPPAS provides its usage by using the option `--help`, as for example:

```
bin> annotation.py --help
```

There are 2 types of programs in the `sppas` then `bin` sub-directory:

1. programs to execute an automatic annotation;
2. programs to execute a GUI (i.e. the SPPAS main frame or a component)

All programs are written with the programming language Python. **The version of Python must be 2.7.something.** Nothing will work with Python `>= 3.0`!

## 5.2 Programs to perform an automatic annotation

This chapter describes how to use the automatic annotations of SPPAS with a Command-Line Interface. It describes all available command and how to use it. However, for scripts that execute an automatic annotation, refer to the chapter “Automatic Annotation” to understand exactly each option!

Generally, resources are given to the CLI with the argument “-r”, an input wav file is given with “-w” option, an input annotated file is given with “-i” and the output is specified with “-o”.

### 5.2.1 annotation.py

This program performs automatic annotations on a given file or on all files of a directory. It strictly corresponds to the button Annotate of the GUI. All annotations are pre-configured: no specific option can be used.

```
usage: annotation.py -w file|folder [options]
```

optional arguments:

-h, --help	show this help message and exit
-w file folder	Input wav file name, or folder
-l lang	Input language, using iso639-3 code
-e extension	Output extension. One of: xra, textgrid, eaf, csv, ...
--momel	Activate Momel and INTSINT
--ipu	Activate IPU Segmentation
--tok	Activate Tokenization
--phon	Activate Phonetization
--align	Activate Alignment
--syll	Activate Syllabification
--rep	Activate Repetitions
--all	Activate ALL automatic annotations

Examples of use:

```
./sppas/bin/annotation.py ./samples/samples-eng  
-l eng  
--ipu --tok --phon --align
```

A progress bar is displayed for each annotation. At the end of the process, a message indicates the name of the procedure outcome report file, which is `./samples/samples-eng.log` in our example. This file can be opened with any text editor (as Notepad++, vim, TextEdit, ...).

### 5.2.2 wavsplit.py

This program performs the IPU-segmentation, i.e. silence/speech segmentation. When this program is used from an audio speech sound and eventually a transcription, it consists in aligning macro-units of a document with the corresponding sound.

Notice that all time values are indicated in seconds.

```
usage: wavsplit.py -w file [options]
```

Generic options:

```

Brigitte@asus-bigi:/cygdrive/d/workspace/SPPAS$ ./bin/annotation.py -i ./samples/samples-eng -l eng --ipu --tok --phon --align

SPPAS - Version 1.6.2
Copyright (C) 2011-2014 LPL Laboratory
http://www.lpl-aix.fr/~bigi/sppas/

-----
100% [=====] IPUs Segmentation
                        Finished.
100% [=====] Tokenization
                        Finished.
100% [=====] Phonetization
                        Finished.
100% [=====] Alignment
                        Finished.

SPPAS finished.
See ./samples/samples-eng.log for details.
Thank you for using SPPAS.
Brigitte@asus-bigi:/cygdrive/d/workspace/SPPAS$ !

```

Figure 5.1: CLI: annotation.py output example

```

-w file          audio input file name
-d delta shift   Add this time value to each start/end
                  bounds of the IPUs (for -o option only)
-h, --help       show the help message and exit

```

Options that can be fixed for the Speech/Silence segmentation:

```

-r float         Window size to estimate rms, in seconds
                  (default value is: 0.010)
-m value         Drop speech shorter than m seconds long
                  (default value is: 0.300)
-s value         Drop silences shorter than s seconds long
                  (default value is: 0.200)
-v value         Assume that a rms lower than v is a silence
                  (default value is: 0 which means to auto-adjust)

```

Options that can be fixed for the Speech/Silence segmentation with a given orthographic transcription. It must be choose one of -t or -n options:

```

-t file          Input transcription file with txt or
                  textgrid extension (default: None)
-n value         Input transcription tier number
                  (in case of input as textgrid) (default: 0)
-N              Adjust the volume cap until it splits
                  into nb tracks (default: 0)

```

Output options:

-o dir	Output directory name	(default: None)
-e ext	Output tracks extension	(default: txt)
-l file	File with units' boundaries	(default: None)
-p file	File with the segmentation	(default: None)

Examples of use to get each IPU in a wav file and its corresponding textgrid:

```
./sppas/bin/wavsplit.py -w ./samples/samples-eng/oriana1.WAV
-d 0.02
-t ./samples/samples-eng/oriana1.txt
-p ./samples/samples-eng/oriana1.xra

./sppas/bin/wavsplit.py -w ./samples/samples-eng/oriana1.WAV
-t ./samples/samples-eng/oriana1.xra
-o ./samples/samples-eng/oriana1
-e textgrid
```

### 5.2.3 tokenize.py

This program performs Tokenization, i.e. text normalization on a given file.

```
usage: tokenize.py -r vocab [options]

optional arguments:
  -r vocab          Vocabulary file name
  -i file           Input file name
  -o file           Output file name
  --delimiter char Use a delimiter character instead of a space for word delimiter.
  -h, --help       Show the help message and exit
```

The following situations are possible:

1. no input is given: the input is stdin and the output is stdout (if an output file name is given, it is ignored).  
In case of enriched orthographic transcription, only the faked tokenization is printed.
2. an input is given, but no output: the result of the tokenization is added to the input file.
3. an input and an output are given: the output file is created (or erased if the file already exists) and the result of the tokenization is added to this file..

Example of use, using stdin/stdout:

```
$ echo "The te(xt) to normalize 123." | \
./sppas/bin/tokenize.py
-r ./resources/vocab/eng.vocab
$ the te to normalize one_hundred_twenty-three
```

In that case, the elision mentionned with the parentheses is removed and the number is converted to its written form. The character “\_” is used for compound words (it replaces the whitespace).

Example of use on a transcribed file:

```
$ ./sppas/bin/tokenize.py -r ./resources/vocab/eng.vocab
-i ./samples/samples-eng/oriana1.xra
-o ./samples/samples-eng/oriana1-token.xra
```

### 5.2.4 phonetize.py

This program performs Phonetization, i.e. grapheme to phoneme conversion on a given file.

```
usage: phonetize.py -r dict [options]

optional arguments:
  -r dict      Pronunciation dictionary file name
  -i file      Input file name
  -o file      Output file name
  --nouns      Disable unknown word phonetization
  -h, --help  Show the help message and exit
```

Examples of use: ~~~~ \$ echo “the te to normalize one\_hundred\_twenty-three” |  
./sppas/bin/phonetize.py -d ./resources/dict/eng.dict -unk \$ D.@lD.VlD.i: t.i: t.ult.ilt.@ n.O:r.m.@.l.aI.z  
UNK ~~~~

Example of use on a tokenized file: ~~~~ ./sppas/bin/phonetize.py -d ./resources/dict/eng.dict -i  
../samples/samples-eng/oriana1-token.xra -o ../samples/samples-eng/oriana1-phon.xra ~~~~

### 5.2.5 alignment.py

This program performs automatic speech segmentation on a given file.

```
usage: alignment.py -w file -i file -r file -o file [options]

optional arguments:
  -w file      Input wav file name
  -i file      Input file name with the phonetization
  -I file      Input file name with the tokenization
  -r file      Model directory
  -o file      Output file name with alignments
  -a name      Aligner name. One of: julius, hvite, basic (default: julius)
  --extend     Extend last phoneme/token to the wav duration
  -h, --help  Show the help message and exit
```

Typical usage:

```
./sppas/bin/alignment.py
-r ./resources/models/models-eng
-w ./samples/samples-eng/oriana1.WAV
-i ./samples/samples-eng/oriana1-phon.xra
-I ./samples/samples-eng/oriana1-token.xra
-o ./samples/samples-eng/oriana1-palign.xra
```

### 5.2.6 syllabify.py

This program performs automatic syllabification on a given file.

```
usage: syllabify.py -r config [options]

optional arguments:
  -r config    Rules configuration file name
  -i file      Input file name (time-aligned phonemes)
  -o file      Output file name
  -e file      Reference file name to syllabify between intervals
  -t string    Reference tier name to syllabify between intervals
  --nophn     Disable the output of the result that does not use the reference tier
  -h, --help  Show the help message and exit
```

### 5.2.7 repetition.py

This program performs automatic detection of self-repetitions or other-repetitions if a second speaker is given.

It can be language-dependent (better results) or language-independent.

If an input is given, but no output: the result is appended to the input file.

```
usage: repetition.py -i file [options]

optional arguments:
  -h, --help  show this help message and exit
  -i file     Input file name with time-aligned tokens of the self-speaker
  -r folder   Directory with resources
  -l lang     Language code in iso639-3
  -I file     Input file name with time-aligned tokens of the echoing-speaker
              (if ORs)
  -o file     Output file name
```

### 5.2.8 Momel and INTSINT

```
usage: momel-intsint.py [options] -i file

optional arguments:
  -i file     Input file name (extension: .hz or .PitchTier)
  -o file     Output file name (default: stdout)
  --win1 WIN1 Target window length (default: 30)
```

```
--lo LO          f0 threshold (default: 50)
--hi HI          f0 ceiling (default: 600)
--maxerr MAXERR  Maximum error (default: 1.04)
--win2 WIN2      Reduct window length (default: 20)
--mind MIND      Minimal distance (default: 5)
--minr MINR      Minimal frequency ratio (default: 0.05)
--non-elim-glitch
-h, --help       Show the help message and exit
```

## 5.3 Programs to execute a GUI

Each program opens a frame. The option `-i` allows to add a file in the frame; it can be added as many times as wanted.

THIS DOCUMENTATION IS TO DO... Any help is welcome!!!

### 5.3.1 sppasgui.py

### 5.3.2 dataroamer.py

### 5.3.3 wavplayer.py

### 5.3.4 iputranscriber.py

### 5.3.5 dataeditor.py

### 5.3.6 datafilter.py

### 5.3.7 stats.py

## 5.4 Other Programs

Some scripts are also available to be used with the Command-Line Interface. They are located in the `scripts` sub-directory.

### 5.4.1 Merge annotation files

```
usage: trsmerge.py -i file -o file [options]
```

optional arguments:

```
-h, --help  show this help message and exit
-i file     Input annotated file name (as many as wanted)
-o file     Output annotated file name
--quiet     Disable the verbosity
```

### 5.4.2 Convert annotation files

```
usage: trsconvert.py -i file -o file [options]
```

optional arguments:

```
-h, --help    show this help message and exit
-i file       Input annotated file name
-o file       Output annotated file name
-t value      A tier number (use as many -t options as wanted). Positive or
              negative value: 1=first tier, -1=last tier.
--quiet       Disable the verbosity
```

### 5.4.3 Get information about a tier of an annotated file.

```
usage: tierinfo.py -i file [options]
```

optional arguments:

```
-h, --help    show this help message and exit
-i file       Input annotated file name
-t value      Tier number (default: 1)
```

### 5.4.4 extract.py

This script extract the channel chosen from the input file and write it in an output file.

```
usage: extract.py -w inputfile -o outputfile -c channel
```

optional arguments:

```
-h, --help    show this help message and exit
```

### 5.4.5 reformat.py

This script extract the channel chosen from input file and write the result in an output file. It can be:

- change framerate (-r)
- change sample width (-s)
- multiply by a factor each frame (-m)
- bias each frame by a value (-b)

or any combination of such options.

```
usage: reformat.py -w inputfile -o outputfile [options]
```



```
optional arguments:
  -h, --help            show this help message and exit
  -c                    Channel (default 1)
  -r                    Framerate
  -s                    Sample width
  -m                    Factor to multiply each frame
  -b                    Value to bias each frame
```

#### 5.4.6 equalize.py

This script equalize the number of frames of each mono channel input files and write the results in output files.

```
usage: equalize.py -w inputfile [inputfile]*
```

```
optional arguments:
  -h, --help            show this help message and exit
```

#### 5.4.7 channelsmixer.py

This script mix all channels from mono input files and write the channel result in an output file.

```
usage: channelsmixer.py -w input_file [inputfile]* -o outputfile
```

```
optional arguments:
  -h, --help            show this help message and exit
```

#### 5.4.8 channelsmixersimulator.py

This script simulate a mix of all channels from input files and print the maximum value of the result.

```
usage: channelsmixer.py -w input_file [inputfile]*
```

```
optional arguments:
  -h, --help            show this help message and exit
```

#### 5.4.9 fragmentextracter.py

This script extract a fragment from an audio file.

```
usage: ./scripts/fragmentextracter.py -w inputfile -o outputfile [options]
```

optional arguments:

-h, --help	show this help message and exit
-bs	The position (in seconds) when begins the mix
-bf	The position (in number of frames) when begins the mix
-es	The position (in seconds) when ends the mix
-ef	The position (in number of frames) when ends the mix

Example of use:

```
usage: ./scripts/fragmentextracter.py -w audiofile.wav -o fragment.wav -bs 2 -es 4
```

#### 5.4.10 audiotaster

This script performs a mix according to the settings given in parameter. A factor permits to attenuate the separation between the two output channels (the value is from 0 to 1).

Settings file must have the required format:

- a header: **file speaker volume panoramic**
- a line by channel for example: **TRACK0\_0.wav HB 100 R100**

The program will look for each file by the filename found on each line. Audio files have to be in the same folder as the settings file.

```
usage: ./scripts/audiotaster.py -s settingfile -o outputfile [options]
```

optional arguments:

-h, --help	show this help message and exit
-l	Factor to attenuate the difference between the two output channels. (default value : 0) x = 0 will perform two channels as asked in the settings 0 < x < 1 will attenuate the difference x = 1 will perform two identical channels
-b	The position (in seconds) when begins the mix
-e	The position (in seconds) when ends the mix
-v	Verbosity level: 0, 1 or 2 (default=1)

Example of use:

```
usage: ./scripts/audiotaster.py -s ./samples/settings.txt -o mix.wav -l 0
```

# Scripting with Python and SPPAS

## 6.1 Introduction

SPPAS implements an Application Programming Interface (API), named *annotationdata*, to deal with annotated files.

*annotationdata* API is a free and open source Python library to access and search data from annotated data. It can convert file formats like Elan's EAF, Praat's TextGrid and others into a `Transcription` object and convert this object into anyone of these formats. This object allows unified access to linguistic data from a wide range sources.

In this chapter, we are going to create Python scripts with the SPPAS Application Programming Interface *annotationdata* and then run them. This chapter includes examples and some exercises to practice. Solutions of the exercises are included in the package sub-directory *documentation*, then *solutions*.

*annotationdata* is based on the Programming Language Python, version 2.7. This chapter firstly introduces basic programming concepts, then it will gradually introduce how to write scripts with Python. Those who are familiar with programming in Python can directly go to the last section related to the description of the *annotationdata* API and how to use it in Python scripts.

In this chapter, it is assumed that Python 2.7 is already installed and configured. It is also assumed that the Python IDLE is ready-to-use. For more details about Python, see:

The Python Website: <http://www.python.org>

## 6.2 A gentle introduction to programming

This section is partially made of selected parts of *the Python website* and *wikipedia*.

### 6.2.1 Definitions

*Programming* is the process of writing instructions for computers to produce software for users. More than anything, it is a creative and problem-solving activity. Any problem can be solved in a large number of ways.

An *algorithm* is the step-by-step solution to a certain problem: algorithms are lists of *instructions* which are followed step by step, from top to bottom and from left to right. To practice writing programs in a programming language, it is required first to think of the problem and consider the logical solution before writing it out.

Writing a program is done so using a programming language. Thankfully even though some languages are vastly different than others, most share a lot of common ground, so that once anyone knows his/her first language, other languages are much easier to pick up.

Any program is made of statements. A *statement* is more casually (and commonly) known as a line of code is the smallest standalone element which expresses some action to be carried out while executing the program. It can be one of: comment, assignment, conditions, iterations, etc. Most languages have a fixed set of statements defined by the language.s

Lines of code are grouped in *blocks*. Blocks are delimited by brackets, braces or by the indentation (depending on the programming language).

The prompt indicates the system is ready to receive input. Most of times, it is represented by '>'. In the following, the prompt is not mentioned.

### 6.2.2 Comments and blocks

Comments are not required by the program to work. But comments are necessary! It is commonly admitted that about 25% to 30% of lines of a program must be comments.

```
# this is a comment in python, bash and others.  
# I can write what I want after the # symbol :-)~
```

### 6.2.3 Variables: Assignment and Typing

A *variable* in any programming language is a named piece of computer memory, containing some information inside. When declaring a variable, it is usually also stated what kind of data it should carry. Assignment is setting a variable to a value. Dynamically typed languages, such as Python, allow automatic conversions between types according to defined rules.

Python variables do not have to be explicitly declared: the declaration happens automatically when a value is assigned to a variable. In most languages, the equal sign (=) is used to assign values to variables.

```
a = 1  
b = 1.0  
c = 'c'  
hello = 'Hello world!'  
vrai = True
```

In the previous example, `a`, `b`, `c`, `hello` and `vrai` are variables, `a = 1` is a declaration with a typed-statement.

Here is a list of some fundamental data types, and their characteristics:

- *char* Character and/or small integer, 1 byte (signed: -128 to 127 or unsigned: 0 to 255)
- *int* Integer, 4 bytes (signed: -2147483648 to 2147483647 or unsigned: 0 to 4294967295)

- *bool* Boolean value, can take two values `True` or `False`
- *float* Floating point number, 4 bytes

Notice that the number 0 represents the boolean value `False`.

Assignments are performed with *operators*. Python Assignment Operators are:

```
a = 10    # simple assignment operator
a += 10   # add and assignment operator
a -= 10
a *= 10
a /= 10
```

Notice that a Character is a single letter, number, punctuation or other value; and a String is a collection of these character values, often manipulated as if it were a single value.

Complex data types are often used, for example: arrays, lists, tree, hash-tables, etc. For example, with Python:

```
lst = ['a', 'bb', 'ccc', 'dddd', 'eeee' ]
sublst = lst[1:2]
```

## 6.2.4 Basic Operators

Python Arithmetic Operators:

```
a = 10
b = 20
a + b  # Addition
a - b  # Subtraction
a * b  # Multiplication
a / b  # Division
float(a) / float(b) # try it! and compare with the previous one
```

## 6.2.5 Conditions

Decision making structures require that the programmer specify one or more conditions to be evaluated or tested by the program. The condition statement is a simple control that tests whether a statement is true or false. The condition can include a variable, or be a variable. If the condition is true, then an action occurs. If the condition is false, nothing is done.

Conditions are performed with the help of comparison operators, as equal, less than, greater than, etc.

In the following, we give example of conditions/comparisons in Python.

```
var = 100
if var == 100: print "Value of expression is 100"
```

Python programming language assumes any non-zero and non-null values as `True`, and if it is either zero or null, then it is assumed as `False` value.

```

if a == b:
    print 'equals'
elif a > b:
    print 'a greater than b'
else:
    print 'b greater than a'

```

Python Comparison Operators:

```

a == b  # check if equals
a != b  # check if different
a > b   # check if a is greater than b
a >= b  # check if a is greater or equal to b
a < b
a <= b

```

Other Python Operators:

- `and` Called Logical AND operator. If both the operands are true then the condition becomes true.
- `or` Called Logical OR Operator. If any of the two operands are non zero then the condition becomes true.
- `not` Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.
- `in` Evaluates to true if it finds a variable in the specified sequence and false otherwise.

### 6.2.6 Loops

A “loop” is a process in which a loop is initiated until a condition has been met.

A `while` loop statement repeatedly executes a target statement as long as a given condition is true.

The `for` loop has the ability to iterate over the items of any sequence, such as a list or a string. The following Python program prints items of a list on the screen:

```

l = ['fruits', 'viande', 'poisson', 'oeufs']
for item in l:
    print item

```

## 6.3 Scripting with Python

### 6.3.1 Hello World!

We are going to create a very simple Python script and then run it. First of all, create a new folder (on your Desktop for example); you can name it “pythonscripts” for example.

Execute the python IDLE (available in the Application-Menu of your operating system).

Then, create a new empty file:



Figure 6.1: The Python IDLE logo

- by clicking on “File” menu, then “New File”,
- or with the shortcut “CTRL”+N.

Copy the following code in this newly created file.

```
1      print 'Hello world!'
```

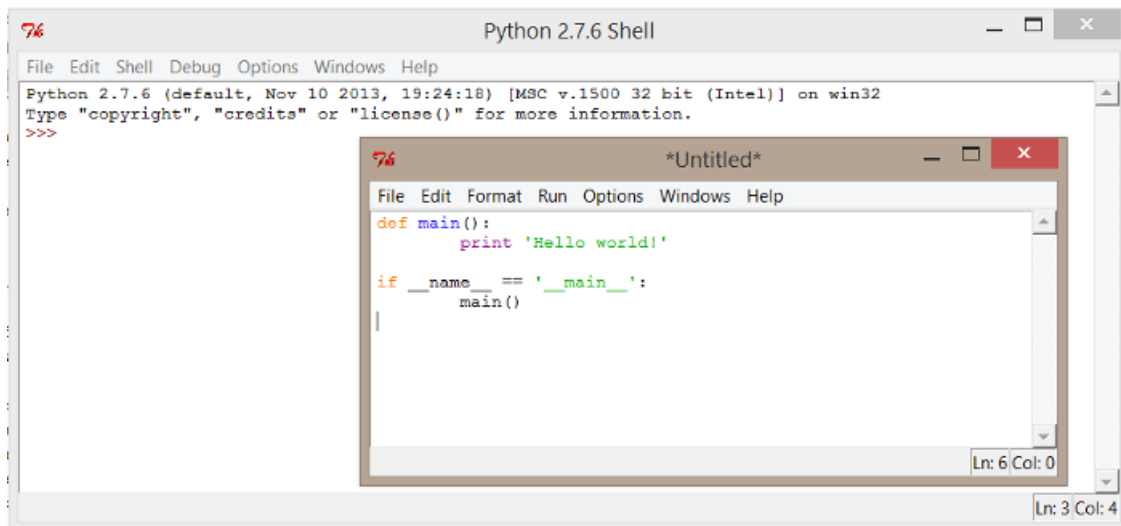


Figure 6.2: Hello world! in a Python script

Then, save the file in the “pythonscripts” folder. By convention, Python source files end with a `.py` extension, so I suggest the name `01_helloworld.py`.

Notice that `main` (in the code above) is a function.

A function does something. This particular function prints, or outputs to the screen, the text, or string, between apostrophes or quotation marks. We’ve decided to call this function `main`: the name `main` is just a convention. We could have called it anything.

To execute the program, you can do one of:

- with the mouse: Click on the Menu “Run”, then “Run module”
- with the keyboard: Press F5

The expected output is as follow:

A better practice while writing scripts is to describe by who, what and why this script was done. I suggest to create a skeleton for your future scripts, it is useful each time a new script will have to be written.

Here is an example:



Figure 6.3: Output of the first script

```

1      # -----
2      # Author: Me
3      # Date: Today
4      # Brief: Simple script to do nothing.
5      # -----
6
7      import os
8      import sys
9
10     # -----
11
12     def main():
13         """ This is the main function to do something. """
14         pass
15
16     # -----
17     # This is the python entry point:
18     # Here, we just ask to execute the main function.
19     if __name__ == '__main__':
20         main()
21     # -----

```

This ready-to-use script is available in the SPPAS package, its name is `skeleton.py`.

In the example above, the main function is documented: documentation is the text between `"""`. In this chapter, all these “docstrings” follow a convention, named “The Epytext Markup Language”. Epytext is a simple lightweight markup language that lets you add formatting and structure to docstrings. The software Epydoc uses that formatting and structure to produce nicely formatted API documentation. For details, see:

Epydoc web site: <http://epydoc.sourceforge.net/manual-epytext.html>

## 6.3.2 Functions

### Simple function

Now, we’ll play with functions! So, create a copy of the file `skeleton.py`, and add the following function `print_vowels()`. This function declare a list named `vowels`. Each item of the list is a string representing



a vowel in French encoded in SAMPA (at the phonetic level). Of course, such list can be overridden with any other set of phonemes. Then, the function print each item of the list, by means of a loop.

```

21     def print_vowels():
22         """ Print the list of French vowels on the screen. """
23         vowels = [ 'a', 'e', 'E', 'i', 'o', 'u', 'y', '@', '2', '9', 'a~', 'o~', 'U~' ]
24         for v in vowels:
25             print v

```

The `main()` function must be changed: instead of printing 'Hello World!', it will call the newly created function `print_vowels()`.

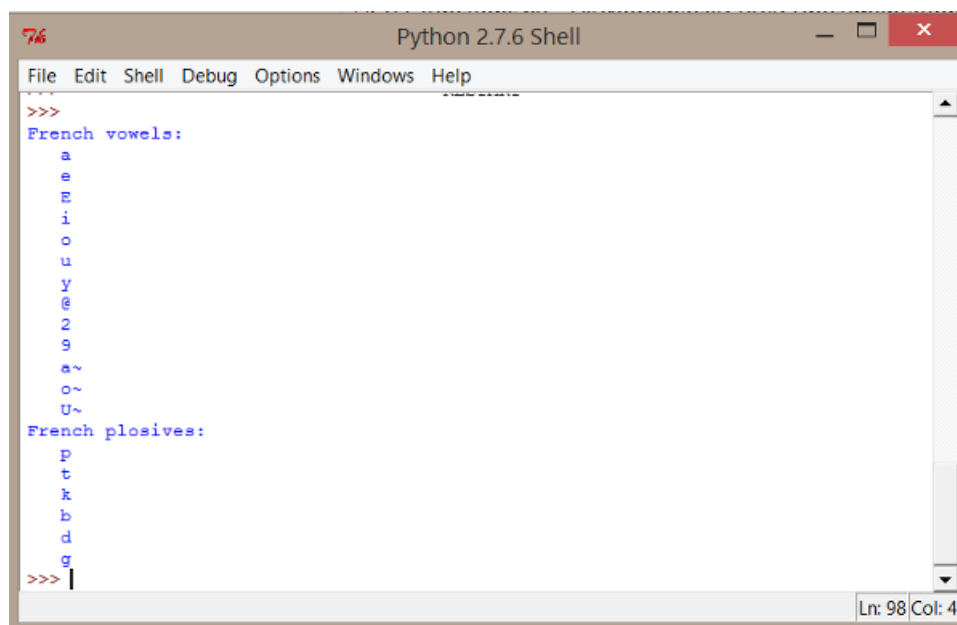
```

31     def main():
32         """ This is the main function. """
33         print_vowels()

```

Then, save the file in the “pythonscripts” folder and execute the program.

*Practice:* Add a function to print plosives and call it in the main function (solution: 02\_functions.py).



```

Python 2.7.6 Shell
File Edit Shell Debug Options Windows Help
>>>
French vowels:
a
e
E
i
o
u
y
@
2
9
a~
o~
U~
French plosives:
p
t
k
b
d
g
>>>
Ln: 98 Col: 4

```

Figure 6.4: Output of the second script

One can also create a function to print glides, another one to print affricates, and so on. Hum... this sounds a little bit fastidious! Lets update, or refactor, our printing function to *make it more generic*.

### Function with parameters

There are times when we need to do something different with only slight variation each time. Rather than writing the same code with only minor differences over and over, we group the code together and use a

mechanism to allow slight variations each time we use it. A function is a smaller program with a specific job. In most languages they can be “passed” data, called parameters, which allow us to change the values they deal with.

Notice that the number of parameters of a function is not limited!

In the example, we can replace the `print_vowels()` function and the `print_plosives()` function by a single function `print_list(mylist)` where `mylist` can be any list containing strings or characters. If the list contains other typed-variables (as `int` or `float`), they must be converted to string to be printed out.

```

21     def print_list(mylist, message=""):
22         """
23         Print a list on the screen.
24
25         @param mylist (list) is the list to print
26         @param message (string) is an optional message to print before each element
27
28         """
29
30         for element in mylist:
31             print message, str(element)
32
33     # -----

```

## Function return values

Languages usually have a way to return information from a function, and this is called the return data. This is done with the `return` key-word. The function stops at this stage, even if some code is following in the block.

In the following example, the function would return a boolean value (True if the given string has no character).

```

21     def is_empty(mystr):
22         """ Return True if mystr is empty. """
23         return not len(mystr.strip())

```

Practice: Add this function in a new script and try to print various lists (solution: 03\_functions.py)

### 6.3.3 Reading/Writing files

#### Reading data from a file

Now, we'll try to get data from a file. Create a new empty file with the following lines (and add as many lines as you want), then, save it with the name “phonemes.csv” (by using UTF-8 encoding):

```

occlusives ; b ; b
occlusives ; d ; d
fricatives ; f ; f

```

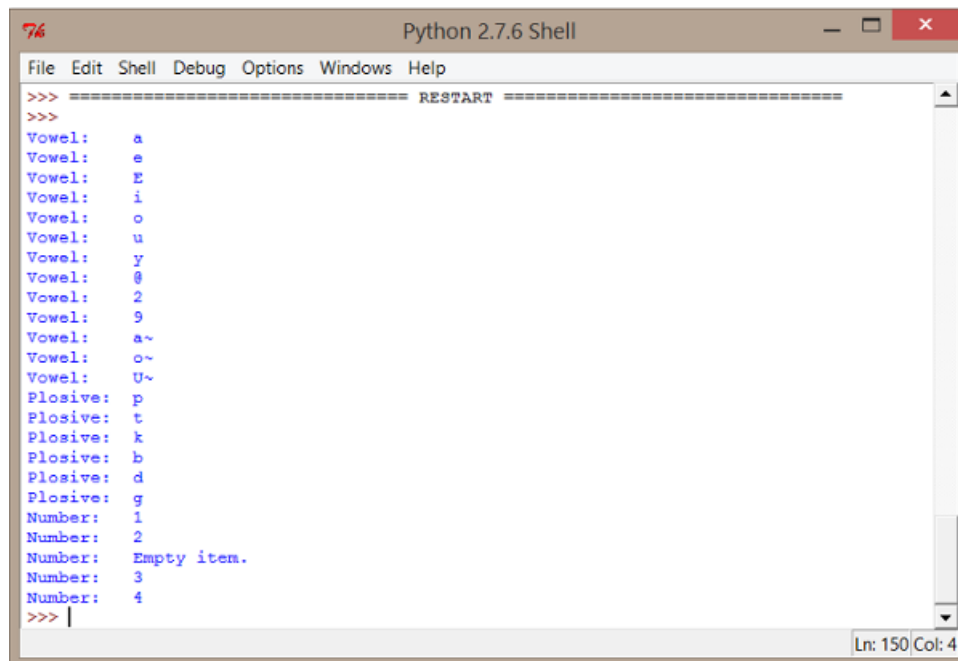


Figure 6.5: Expected output of the 3rd script

```

liquids ; l ; l
nasals ; m ; m
nasals ; n ; n
occlusives ; p ; p
glides ; w ; w
vowels ; a ; a
vowels ; e ; e

```

The following statements are typical statements used to read the content of a file. The first parameter of the function `open` is the file name, including the path (relative or absolute); and the second argument is the opening mode ('r' is the default value, used for reading).

```

21     def main():
22         f = open("C:\Users\Me\Desktop\pythonscripts\phonemes.csv", 'r')
23         for l in f:
24             # do something with the line stored in variable l
25             print l
26         f.close()

```

See the file `04_reading_simple.py` for the whole program and try-it (do not forget to change the file name to your own file!).

Like any program... it exists more than one way to solve the problem. The following is a more generic solution, with the ability to deal with various file encodings, thanks to the `codecs` library:

```

21     def read_file(filename):
22         """

```

```

23         Read the whole file, return lines into a list.
24
25         @param filename (string) Name of the file to read, including path.
26         @return List of lines
27
28         """
29         with codecs.open(filename, 'r', encoding="utf8") as f:
30             return f.readlines()

```

In the previous code, the `codecs.open` functions got 3 arguments: the file name, the mode (in that case, 'r' means 'read'), and the encoding. The `readlines()` function get each line of the file `f` and store it into a list.

The main function can be as follow:

```

35     def main():
36         """ This is the main function. """
37
38         lines = read_file("C:\Users\Me\Desktop\pythonscripts\phonemes.csv")
39
40         # before doing something, check the data!
41         if not len(lines):
42             print 'Hum... the file is empty!'
43             sys.exit(1)
44
45         # do something with the lines
46         for l in lines:
47             print l.strip()

```

See the file `05_reading_file.py` for the whole program, and try-it (do not forget to change the file name to your own file!).

Notice that Python `os` module provides methods that can help you to perform file-processing operations, such as renaming and deleting files. See Python documentation for details: <https://docs.python.org/2/>

## Writing files

Writing a file requires to open it in a writing mode:

- 'w' is the mode used to write; it will erase any existing file;
- 'a' is the mode used to append data in an existing file.

A file can be opened in an encoding and saved in another one. This could be useful to write a script to convert the encoding of a set of files in a given folder to UTF-8 for example. The following could help to create such a script:

```

10     # getting all files of a given folder:
11     path = 'C:\Users\Me\data'
12     dirs = os.listdir( path )

```

```

15     # Converting encoding of a file:
16     file_stream = codecs.open(file_location, 'r', file_encoding)
17     file_output = codecs.open(file_location+'utf8', 'w', 'utf-8')
18
19     for l in file_stream:
20         file_output.write(l)

```

### 6.3.4 Dictionaries

A dictionary is another container type that can store any number of Python objects, including other container types. Dictionaries consist of pairs (called items) of keys and their corresponding values. Each key is separated from its value by a colon (:), the items are separated by commas, and the whole thing is enclosed in curly braces. An empty dictionary without any items is written with just two curly braces, like this: {}. To access dictionary elements, you can use the familiar square brackets along with the key to obtain its value.

The next example is a portion of a program that can be used to convert a list of phonemes from SAMPA to IPA.

To get values from a dictionary, one way is to use directly `dict[key]`, but it is required to test if key is really in dict, otherwise, Python will stop the program and send an error. Alternatively, the `get` function can be used, as `dict.get(key, default=None)` where `default` is the value to return if the key is missing. In the previous example, it is possible to replace `sampadict[phone]` by `sampadict.get(phone, phone)`. Two other functions are useful while using dictionaries:

- `dict.keys()` return a list with the keys
- `dict.values()` return a list with values

### 6.3.5 Exercises to practice

Exercise 1: How many vowels are in a list of phonemes? (solution: 06\_list.py)

Exercise 2: Write a SAMPA to IPA converter. (solution: 07\_dict.py)

Exercise 3: Compare 2 sets of data using NLP techniques (Zipf law, Tf.Idf) (solution: 08\_counter.py)

## 6.4 The API of SPPAS to manage data

### 6.4.1 Overview

We are now going to write Python scripts with the help of the *annotationdata* API included in SPPAS. This API is useful to read/write and manipulate files annotated from various annotation tools as Praat or Elan for example.

First of all, it is important to understand the data structure included in the API to be able to use it efficiently. Details can be found in the following publication:

*Brigitte Bigi, Tatsuya Watanabe, Laurent Prévot* (2014). **Representing Multimodal Linguistics Annotated data**, Proceedings of the 9th edition of the Language Resources and Evaluation Conference, 26-31 May 2014, Reykjavik, Iceland.

### 6.4.2 Why developing a new API?

In the Linguistics field, multimodal annotations contain information ranging from general linguistic to domain specific information. Some are annotated with automatic tools, and some are manually annotated. Linguistics annotation, especially when dealing with multiple domains, makes use of different tools within a given project. Many tools and frameworks are available for handling rich media data. The heterogeneity of such annotations has been recognized as a key problem limiting the interoperability and re-usability of NLP tools and linguistic data collections.

In annotation tools, annotated data are mainly represented in the form of “tiers” or “tracks” of annotations. The genericity and flexibility of “tiers” is appropriate to represent any multimodal annotated data because it simply maps the annotations on the timeline. In most tools, tiers are series of intervals defined by:

- a time point to represent the beginning of the interval;
- a time point to represent the end of the interval;
- a label to represent the annotation itself.

In Praat, tiers can be represented by a time point and a label (such tiers are named `PointTiers` and `IntervalTiers`). Of course, depending on the annotation tool, the internal data representation and the file formats are different. For example, in Elan, unlabelled intervals are not represented nor saved. On the contrary, in Praat, tiers are made of a succession of consecutive intervals (labelled or un-labelled).

The *annotationdata* API used for data representation is based on the common set of information all tool are currently sharing. This allows to manipulate all data in the same way, regardless of the file type.

The API supports to merge data and annotation from a wide range of heterogeneous data sources for further analysis.

### 6.4.3 The API class diagram

After opening/loading a file, its content is stored in a `Transcription` object. A `Transcription` has a name, and a list of `Tier` objects. This list can be empty. Notice that in a `Transcription`, two tiers can't have the same name.

A `Tier` has a name, and a list of `Annotation` objects. It also contains a set of meta-data and it can be associated to a controlled vocabulary.

A common solution to represent annotation schemes is to put them in a tree. One advantage in representing annotation schemes through those trees, is that the linguists instantly understand how such a tree works and can give a representation of “their” annotation schema. However, each annotation tool is using its own formalism and it is unrealistic to be able to create a generic scheme allowing to map all of them. SPPAS implements another solution that is compatible with trees and/or flat structures (as in Praat). Actually, subdivision relations can be established between tiers. For example, a tier with phonemes is a subdivision reference for syllables, or for tokens, and tokens are a subdivision reference for the orthographic transcription in IPUs. Such subdivisions can be of two categories: alignment or constituency. This data representation allows to keep the `Tier` representation, which is shared by most of the annotation tools and it allows too to map data on a tree if required: the user can freely create tiers with any names and arrange them in such custom and very easy hierarchy system.

An annotation is made of 2 objects:

- a `Location` object,

- a `Label` object.

A `Label` object is representing the “content” of the annotation. It is a list of `Text` associated to a score.

A `Location` is representing where this annotation occurs in the media. Then, a `Location` is made of a list of `Localization` which includes the `BasePlacement` associated with a score. A `BasePlacement` object is one of: \* a `TimePoint` object; or \* a `TimeInterval` object, which is made of 2 `TimePoint` objects; or \* a `TimeDisjoint` object which is a list of `TimeInterval`; or \* a `FramePoint` object; or \* a `FrameInterval` object; or \* a `FrameDisjoint` object.

**The whole API documentation is available at the following URL:** <http://sldr.org/000800/preview/manual/module-tree.html>

### Label representation

Each annotation holds at least one label, mainly represented in the form of a string, freely written by the annotator or selected from a list of categories, depending on the annotation tool. The “*annotationdata*” API, aiming at representing any kind of linguistic annotations, allows to assign a score to each label, and allows multiple labels for one annotation. The API also allows to define a controlled vocabulary.

### Location representation

In the *annotationdata* API, a `TimePoint` is considered *as an imprecise value*. It is possible to characterize a point in a space immediately allowing its vagueness by using:

- a midpoint value (center) of the point;
- a radius value.

### Example

The screenshot below shows an example of multimodal annotated data, imported from 3 different annotation tools. Each `TimePoint` is represented by a vertical dark-blue line with a gradient color to refer to the radius value (0ms for prosody, 5ms for phonetics, discourse and syntax and 40ms for gestures in this screenshot).

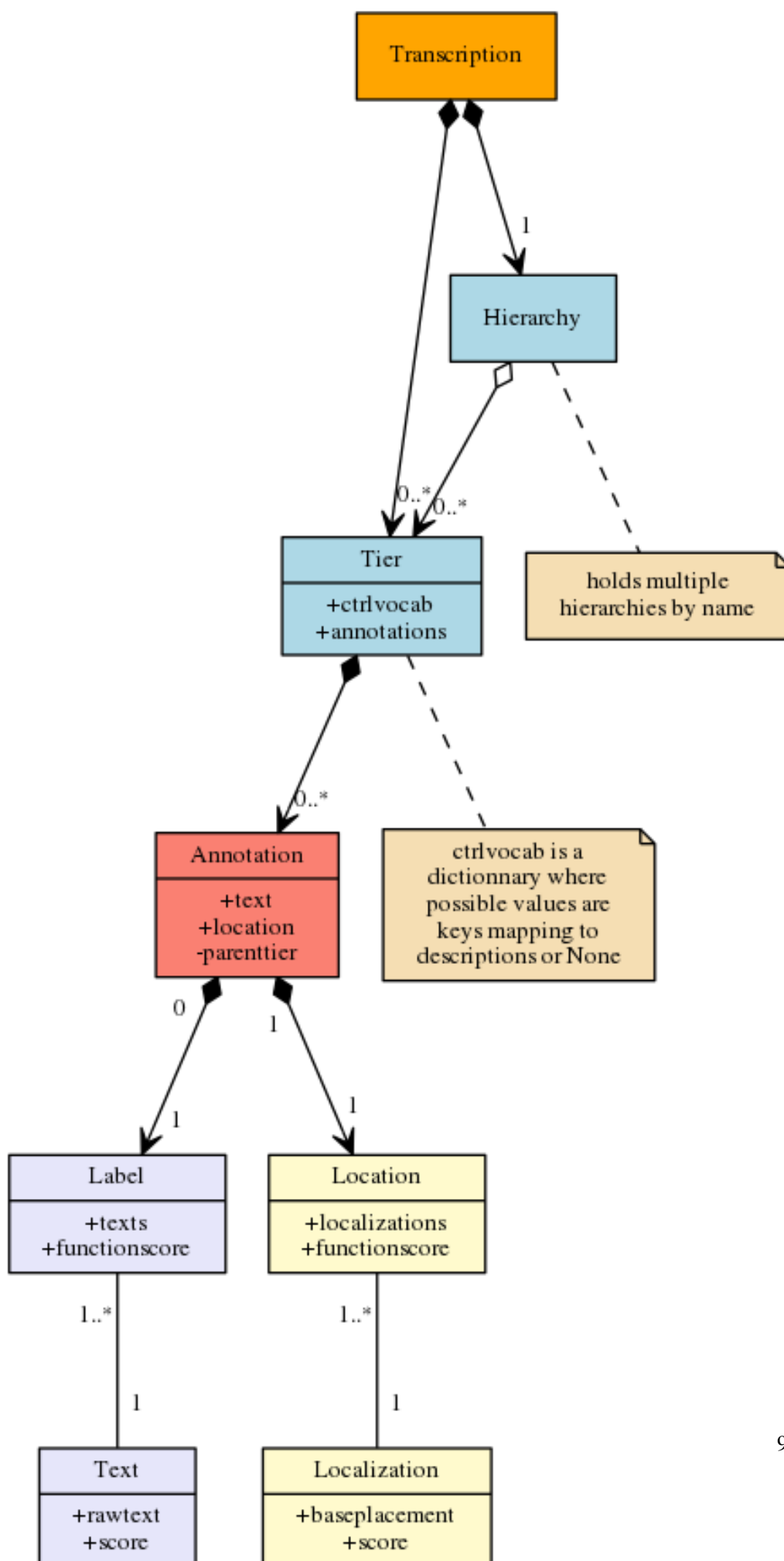
## 6.5 Creating scripts with the SPPAS API

### 6.5.1 Preparing the data

If it is not already done, create a new folder (on your Desktop for example); you can name it “*pythonscripts*” for example.

Open a File Explorer window and go to the SPPAS folder location. Then, open the `sppas` directory then `src` sub-directory. Copy the `annotationdata` folder then paste-it into the newly created `pythonscripts` folder.

Open the python IDLE and create a new empty file. Copy the following code in this newly created file, then save the file in the `pythonscripts` folder. By convention, Python source files end with a `.py` extension; I suggest `skeleton-sppas.py`. It will allow to use the SPPAS API in your script.





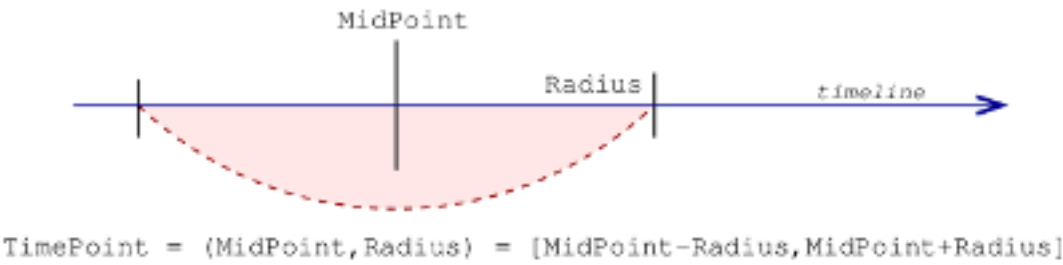


Figure 6.7: Representation of a TimePoint

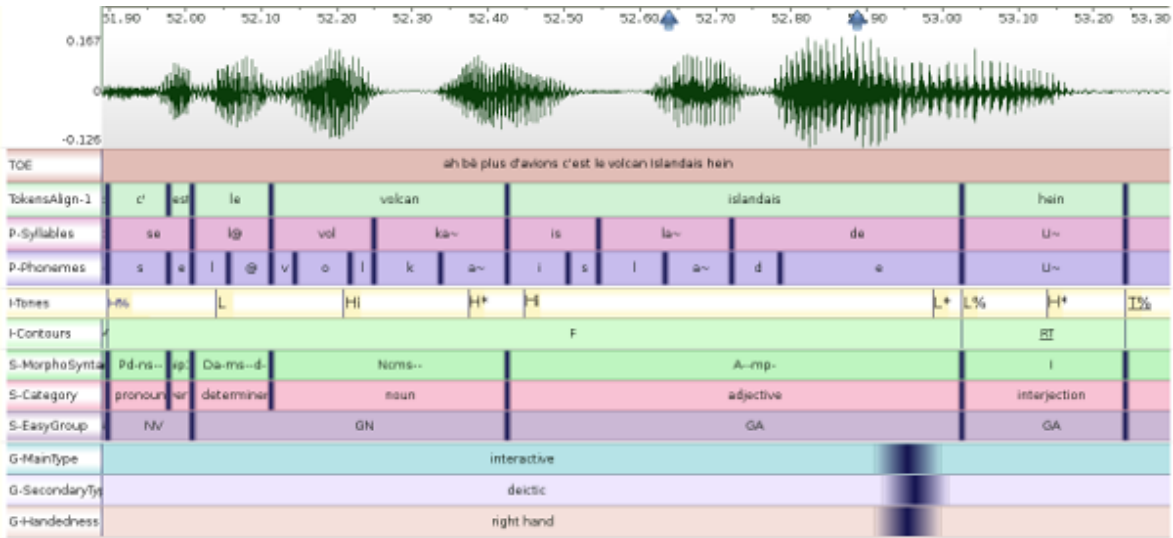


Figure 6.8: Example of multimodal data

```

1      # -----
2      # Author: Me
3      # Date: Today
4      # Brief: Script using the SPPAS API
5      # -----
6
7      # Get SPPAS API
8      import annotationdata.io
9      from annotationdata import Transcription
10     from annotationdata import Tier
11     from annotationdata import Annotation
12     from annotationdata import Label
13     from annotationdata import TimePoint
14     from annotationdata import TimeInterval
15     from annotationdata import Bool, Rel
16
17     import os
18     import sys
19
20     # -----
21
22     def main():
23         """ This is the main function. """
24         pass
25
26     # -----
27     # This is the python entry point:
28     # Here, we just ask to execute the main function.
29     if __name__ == '__main__':
30         main()
31     # -----

```

Navigate to the folder containing your script, and open-it with the python IDLE. To execute the file:

- Menu: “Run”, then “Run module”
- Keyboard: F5

It will do... nothing! But now, we are ready to do something with the API!

### 6.5.2 Read/Write annotated files

Open/Read a file of any format (TextGrid, Elan, Transcriber, CSV) and store it into a `Transcription` object instance, named `trs` in the following code, is mainly done as:

```

1      trs = annotationdata.io.read(filename_in)

```

Save/Write a `Transcription` object instance in a file of any format is mainly done as:

```
2      annotationdata.io.write(filename_out, trs)
```

These two lines of code loads any annotation file (Elan, Praat, Transcriber...) and writes the data into another file. The format of the file is given by its extension, as for example “.xra” is the SPPAS native format, “.TextGrid” is one of the Praat native format, and so on.

So... only both lines are used to convert a file from one format to another one!

In any script, to get the list of accepted extensions as input, just call “annotationdata.io.extensions\_in”, and the list of accepted extensions as output is “annotationdata.io.extensions\_out”.

Currently, accepted input file extensions are:

- xra
- csv, txt
- TextGrid, PitchTier, IntensityTier
- eaf
- trs
- mrk
- sub, srt
- hz
- antx

Possible output file extensions are:

- xra
- csv, txt, lab, ctm, stm
- TextGrid, PitchTier
- eaf
- mrk
- sub, srt
- antx

Practice: Write a script to convert a TextGrid file into CSV (solution: 10\_read\_write.py)

### 6.5.3 Manipulating a Transcription object

The most useful functions used to manage a Transcription object are:

- `Append(tier), Pop()`
- `Add(tier, index), Remove(index)`
- `Find(name, case_sensitive=True)`

`Append()` is used to add a tier at the end of the list of tiers of the Transcription object; and `Pop()` is used to remove the last tier of such list.

`Add()` and `Remove()` do the same, except that it does not put/delete the tier at the end of the list but at the given index.

`Find()` is useful to get a tier of the list from its name.

There are useful “shortcuts” that can be used. For example, `trs[0]` returns the first tier, `len(trs)` returns the number of tiers and loops can be written as:

```

15     for tier in trs:
16         # do something with the tier
17         print tier.GetName()
```

Practice: Write a script to select a set of tiers of a file and save them into a new file (solution: `11_transcription.py`).

### 6.5.4 Manipulating a Tier object

As it was already said, a tier is made of a name and a list of annotations. To get the name of a tier, or to fix a new name, the easier way is to use `tier.GetName()`.

Test the following code into a script:

```

15     trs = annotationdata.io.read(filename)
16     tier = trs[0]
17     print tier.GetName()
18     tier.SetName( "toto" )
19     print tier.GetName()
20     print trs[0].GetName()
```

The most useful functions used to manage a Tier object are:

- `Append(annotation), Pop()`
- `Add(annotation), Remove(begin, end, overlaps=False)`
- `IsDisjoint(), IsInterval(), IsPoint()`
- `Find(begin, end, overlaps=True)`
- `Near(time, direction)`
- `SetRadius(radius)`

Practice: Write a script to open an annotated file and print information about tiers (solution: `12_tiers_info.py`)

#### Goodies:

the file `12_tiers_info_wx.py` proposes a GUI to print information of one file or all files of a directory, and to ask the file/directory name with a dialogue frame, instead of fixing it in the script. This script can be executed simply by double-clicking on it in the File Explorer of your system. Many functions of this script can be cut/pasted in any other script.

### 6.5.5 Main information on Annotation/Location/Label objects

The most useful function used to manage an Annotation object are:

- `IsSilence()`, `IsLabel()`
- `IsPoint()`, `IsInterval()`, `IsDisjoint()`
- `GetBegin()`, `SetBegin(time)`, only if time is a `TimeInterval`
- `GetEnd()`, `SetEnd(time)`, only if time is a `TimeInterval`
- `GetPoint()`, `SetPoint(time)`, only if time is a `TimePoint`

The following example shows how to get/set a new label, and to set a new time to an annotation:

```

1      if ann.GetLabel().IsEmpty():
2          ann.GetLabel().SetValue( "dummy" )
3      if ann.GetLocation().IsPoint():
4          p = ann.GetLocation().GetPoint()
5          p.SetValue( 0.234 )
6          p.SetRadius( 0.02 )
7      if ann.GetLocation().IsInterval():
8          ann.GetLocation().GetBegin().SetValue( 0.123 )
9          ann.GetLocation().GetEnd().SetValue( 0.234 )

```

If something forbidden is attempted, the object will raise an `Exception`. This means that the program will stop (except if the program “raises” the exception).

### 6.5.6 Exercises

Exercise 1: Write a script to print information about annotations of a tier (solution: `13_tiers_info.py`)

Exercise 2: Write a script to estimates the frequency of a specific annotation label in a file/corpus (solution: `14_freq.py`)

### 6.5.7 Search in annotations: Filters

#### Overview

This section focuses on the problem of *searching and retrieving* data from annotated corpora.

The filter implementation can only be used together with the `Tier()` class. The idea is that each `Tier()` can contain a set of filters, that each reduce the full list of annotations to a subset.

SPPAS filtering system proposes 2 main axis to filter such data:

- with a boolean function, based on the content, or on the time,
- with a relation function between intervals of 2 tiers.

A set of filters can be created and combined to get the expected result, with the help of the boolean function and the relation function.

To be able to apply filters to a tier, some data must be loaded first. First, you have to create a new `Transcription()` when loading a file. In the next step, you have to select the tier to apply filters on. Then, if the input file was not XRA, it is widely recommended to fix a radius value depending on the annotation type. Now everything is ready to create filters for these data.

### Creating a boolean function

In the following, let `Bool` and `Rel` two predicates, a tier `T`, and a filter `f`.

Pattern selection is an important part to extract data of a corpus. In this case, each filter consists of search terms for each of the tiers that were loaded from an input file. Thus, the following matching predicates are proposed to select annotations (intervals or points) depending on their label. Notice that `P` represents the text pattern to find:

- exact match: `pr = Bool(exact=P)`, means that a label is valid if it strictly corresponds to the expected pattern;
- contains: `pr = Bool(contains=P)`, means that a label is valid if it contains the expected pattern;
- starts with, `pr = Bool(startswith=P)`, means that a label is valid if it starts with the expected pattern;
- ends with, `pr = Bool(endswith=P)`, means that a label is valid if it ends with the expected pattern.

These predicates are then used while creating a filter on labels. All these matches can be reversed, to represent does not exactly match, does not contain, does not start with or does not end with, as for example:

```
tier = trs.Find("PhonAlign")
ft = Filter(tier)
f1 = LabelFilter( Bool(exact='a'), ft)
f2 = LabelFilter( ~Bool(iexact='a'), ft)
```

In this example, `f1` is a filter used to get all phonemes with the exact label 'a'. On the other side, `f2` is a filter that ignores all phonemes matching with 'a' (mentioned by the symbol '~') with a case insensitive comparison (`iexact` means insensitive-exact).

For complex search, a selection based on regular expressions is available for advanced users, as `pr = Bool(regex=R)`.

A multiple pattern selection can be expressed with the operators `|` to represent the logical "or" and the operator `&` to represent the logical "and".

With this notation in hands, it is possible to formulate queries as, for example: *Extract words starting by "ch" or "sh"*, as:

```
pr = Bool(startswith="ch") | Bool(startswith="sh")
```

Filters on duration can also be created on annotations if Time instance is of type `TimeInterval`. In the following, `v` represents the value to be compared with:

- lower: `pr = Bool(duration_lt=v)`, means that an annotation of `T` is valid if its duration is lower than `v`;
- lower or equal: `pr = Bool(duration_le=v)`;
- greater: `pr = Bool(duration_gt=v)`;
- greater or equal: `pr = Bool(duration_ge=v)`;
- equal: `pr = Bool(duration_e=v)`;

Search can also starts and ends at specific time values in a tier by creating filters with `begin_ge` and `end_le`.

The, the user must apply the filter to get filtered data from the filter.

```

1      # creating a complex boolean function
2      predicate = (Bool(icontains="a") | Bool(icontains="e")) & Bool(duration_ge=0.08)
3
4      # to create a filter:
5      ft = Filter(tier)
6      flab = LabelFilter(predicate, ft)
7
8      # to get filtered data from the filter:
9      tier = flab.Filter()
10     tier.SetName( 'Filtered with a-e-0.8' )

```

### Creating a relation function

Relations between annotations is crucial if we want to extract multimodal data. The aim here is to select intervals of a tier depending on what is represented in another tier.

We implemented the 13 Allen interval relations: before, after, meets, met by, overlaps, overlapped by, starts, started by, finishes, finished by, contains, during and equals. Actually, we implemented the 25 relations proposed in the INDU model. This model is fixing constraints on INtervals (with Allen's relations) and on DURATION (duration are equals, one is less/greater than the other).

[List of Allen interval relations]<./etc/screenshots/allen.png>

Below is an example of request: *Which syllables stretch across 2 words?*

```

1      # Get tiers from a Transcription object
2      tiersyll = trs.Find("Syllables")
3      tiertoks = trs.Find("TokensAlign")
4
5      # Create filters
6      fsyll = Filter(tiersyll)
7      ftoks = Filter(tiertoks)
8
9      # Create the filter with the relation function (link both filters)
10     predicate = Rel("overlaps") | Rel("overlappedby")
11     f = RelationFilter(relation, fsyll, ftoks)

```

### 6.5.8 Exercises

Exercise 1: Create a script to filter annotated data on their label (solution: 15\_annotation\_label\_filter.py).

Exercise 2: Idem with a filter on duration or time. (solution: 16\_annotation\_time\_filter.py).

Exercise 3: Create a script to get tokens followed by a silence. (solution: 17\_annotations\_relation\_filter1.py).

Exercise 4: Create a script to get tokens preceded by OR followed by a silence. (solution: 17\_annotations\_relation\_filter2.py).

Exercise 5: Create a script to get tokens preceded by AND followed by a silence. (solution: 17\_annotations\_relation\_filter3.py).



## References

### 7.1 Publications about SPPAS

PDF versions of the publications are available in the SPPAS package (folder `documentation`, sub-folder `references`).

**By using SPPAS, you agree to cite one of these references.**

*Brigitte Bigi, Christine Meunier, Irina Nesterenko, Roxane Bertrand* (2010). **Automatic detection of syllable boundaries in spontaneous speech**, Language Resource and Evaluation Conference, pages 3285-3292, La Valetta, Malte.

Summary: This paper presents the outline and performance of an automatic syllable boundary detection system. The syllabification of phonemes is performed with a rule-based system. The proposed phonemes, classes and rules are listed in an external configuration file of the tool.

*Brigitte Bigi* (2012). **The SPPAS participation to Evalita 2011**, Working Notes of EVALITA 2011, Rome (Italy), ISSN: 2240-5186.

Summary: EVALITA is an initiative devoted to the evaluation of Natural Language Processing and Speech tools for Italian<sup>2</sup>. In Evalita 2011 the “Forced Alignment on Spontaneous Speech” task was added. Training data is about 15 map task dialogues recorded by couples of speakers exhibiting a wide variety of Italian variants.

*Brigitte Bigi* (2012). **SPPAS: a tool for the phonetic segmentations of Speech**, The eight international conference on Language Resources and Evaluation, Istanbul (Turkey), pages 1748-1755, ISBN 978-2-9517408-7-7.

Summary: This paper is a detailed presentation of SPPAS. It presents all the features of the software: an overview, annotation steps, resources and the architecture of the tool.

*Brigitte Bigi, Daniel Hirst* (2012) **SPEECH PHONETIZATION ALIGNMENT AND SYLLABIFICATION (SPPAS): A TOOL FOR THE AUTOMATIC ANALYSIS OF SPEECH PROSODY**, Speech Prosody, Tongji University Press, ISBN 978-7-5608-4869-3, pages 19-22, Shanghai (China).

Summary: This paper is an overview of SPPAS annotation steps and resources.

*Brigitte Bigi, Daniel Hirst* (2013). **What's new in SPPAS 1.5?**, Tools and Resources for the Analysis of Speech Prosody, Aix-en-Provence, France, pp. 62-65.

Summary: During Speech Prosody 2012, we presented SPPAS, SPEECH PHONETIZATION ALIGNMENT AND SYLLABIFICATION, a tool to automatically produce annotations which include utterance, word, syllabic and phonemic segmentations from a recorded speech sound and its transcription. SPPAS is open source software issued under the GNU Public License. SPPAS is multi-platform (Linux, MacOS and Windows) and it is specifically designed to be used directly by linguists in conjunction with other tools for the automatic analysis of speech prosody. This paper presents various improvements implemented since the previously described version.

*Brigitte Bigi* (2013). **A phonetization approach for the forced-alignment task**, 3rd Less-Resourced Languages workshop, 6th Language & Technology Conference, Poznan (Poland).

*Dafydd Gibbon* (2013). **TGA: a web tool for Time Group Analysis**, Tools and Resources for the Analysis of Speech Prosody, Aix-en-Provence, France, pp. 66-69.

Summary: Speech timing analysis in linguistic phonetics often relies on annotated data in de facto standard formats, such as Praat TextGrids, and much of the analysis is still done largely by hand, with spreadsheets, or with specialised scripting (e.g. Praat scripting), or relies on cooperation with programmers. The TGA (Time Group Analyser) tool provides efficient ubiquitous web-based computational support for those without such computational facilities. The input module extracts a specified tier (e.g. phone, syllable, foot) from inputs in common formats; user-defined settings permit selection of sub-sequences such as inter-pausal groups, and duration difference thresholds. Tabular outputs provide descriptive statistics (including modified deviation models like PIM, PFD, nPVI, rPVI), linear regression, and novel structural information about duration patterns, including difference n-grams and Time Trees (temporal parse trees).

*Brigitte Bigi* (2014). **Automatic Speech Segmentation of French: Corpus Adaptation**. 2nd Asian Pacific Corpus Linguistics Conference, p. 32, Hong Kong.

*Brigitte Bigi, Roxane Bertrand, Mathilde Guardiola* (2014). **Automatic detection of other-repetition occurrences: application to French conversational speech**, 9th International conference on Language Resources and Evaluation (LREC), Reykjavik (Iceland), pages 2648-2652. ISBN: 978-2-9517408-8-4.

Summary: This paper investigates the discursive phenomenon called other-repetitions (OR), particularly in the context of spontaneous French dialogues. It focuses on their automatic detection and characterization. A method is proposed to retrieve automatically OR: this detection is based on rules that are applied on the lexical material only. This automatic detection process has been used to label other-repetitions on 8 dialogues of CID - Corpus of Interactional Data. Evaluations performed on one speaker are good with a F1-measure of 0.85. Retrieved OR occurrences are then statistically described: number of words, distance, etc.

*Brigitte Bigi, Tatsuya Watanabe, Laurent Prévot* (2014). **Representing Multimodal Linguistics Annotated Data**, 9th International conference on Language Resources and Evaluation (LREC), Reykjavik (Iceland), pages 3386-3392. ISBN: 978-2-9517408-8-4.

Summary: The question of interoperability for linguistic annotated resources requires to cover different aspects. First, it requires a representation framework making it possible to compare, and potentially merge, different annotation schema. In this paper, a general description level representing the multimodal linguistic annotations is proposed. It focuses on time and data content representation: This paper reconsiders and enhances the current and generalized representation of annotations. An XML schema of such annotations is proposed. A Python API is also proposed. This framework is implemented in a multi-platform software and distributed under the terms of the GNU Public License.

*Brigitte Bigi* (2014). **A Multilingual Text Normalization Approach**, Human Language Technologies Challenges for Computer Science and Linguistics LNAI 8387, Springer, Heidelberg. ISBN: 978-3-319-14120-6. Pages 515-526.

Summary: The creation of text corpora requires a sequence of processing steps in order to constitute, normalize, and then to directly exploit it by a given application. This paper presents a generic approach for text normalization and concentrates on the aspects of methodology and linguistic engineering, which serve to develop a multi-purpose multilingual text corpus. This approach was applied on written texts of French, English, Spanish, Vietnamese, Khmer and Chinese and on speech transcriptions of French, English, Italian, Chinese and Taiwanese. It consists in splitting the text normalization problem in a set of minor sub-problems as language-independent as possible. A set of text corpus normalization tools with linked resources and a document structuring method are proposed and distributed under the terms of the GPL license.

*Brigitte Bigi, Caterina Petrone, Leonardo Lancia* (2014). **Automatic Syllabification of Italian: adaptation from French**. Laboratory Approaches to Romance Phonology VII, Aix-en-Provence (France).

## 7.2 SPPAS in research projects

### 7.2.1 MULTIPHONIA

SPPAS was used to annotate the MULTIPHONIA corpus (MULTImodal database of PHONetics teaching methods in classroom InterActions) created by Charlotte ALAZARD, Corine ASTESANO, Michel BILLIÈRES.

This database consists of audio-video classroom recording comparing two methods of phonetic correction (the “traditional” articulatory method, and the Verbo-Tonal Method). This database is composed of 96 hours of pronunciation classes with beginners and advanced students of French as a Foreign Language. Every class lasted approximatively 90 minutes. This multimodal database constitutes an important resource for Second Language Acquisition’s researchers. Hence, MULTIPHONIA will be enriched at many different levels, to allow for segmental, prosodic, morphosyntactic, syntactic, lexical and gestural analyses of L2 speech.

*Charlotte Alazard, Corine Astésano, Michel Billières* **MULTIPHONIA: a MULTImodal database of PHONetics teaching methods in classroom InterActions**, Language Resources and Evaluation Conference, Istanbul (Turkey), May 2012.

MULTIPHONIA: <http://www.sldr.org/sldr000780/en>

### 7.2.2 Amennpro

SPPAS was used for the annotation of the French part of the AixOx corpus.

- four way recordings of French and English texts;
- read by English and French speakers;
- non-native speakers were divided into advanced and beginners.

Download the AixOx corpus: <http://www.sldr.fr/sldr000784/>

Remark:

Some examples are located in the samples-fra directory (files F\_F\_\*.\*) and in the samples-eng (files E\_E\_\*.\*) .

### 7.2.3 Evalita 2011: Italian phonetization and alignment

Evalita 2011 was the third evaluation campaign of Natural Language Processing and Speech tools for Italian, supported by the NLP working group of AI\*IA (Associazione Italiana per l'Intelligenza Artificiale/Italian Association for Artificial Intelligence) and AISV (Associazione Italiana di Scienze della Voce/Italian Association of Speech Science).

SPPAS participated to the Forced Alignment on Spontaneous Speech for both tasks:

- Phone segmentation
- Word segmentation

The corpus was a set of Dialogues, map-tasks:

- 3h30 speech;
- 15% phones are: “sil”, filled-pauses, garbage.

### 7.2.4 Orthographic Transcription: Impact on Phonetization

SPPAS Phonetization was evaluated on a French Corpus (MARC-Fr). SPPAS Alignment was also evaluated on this corpus.

Results are reported in the following publications:

*B. Bigi, P. Péri R. Bertrand* (2012). **Orthographic Transcription: Which Enrichment is required for Phonetization?**, Language Resources and Evaluation Conference, Istanbul (Turkey), May 2012

*Brigitte Bigi* (2014). Automatic Speech Segmentation of French: Corpus Adaptation. 2nd Asian Pacific Corpus Linguistics Conference, p. 32, Hong Kong.

After registration, MARC-Fr can be freely downloaded at: <http://www.sldr.fr/sldr000786/fr>

### 7.2.5 Cofee: Conversational Feedback

In a conversation, feedback is mostly performed through short utterances produced by another participant than the main current speaker. These utterances are among the most frequent in conversational data. They are also considered as crucial communicative tools for achieving coordination in dialogue. They have been the topic of various descriptive studies and often given a central role in applications such as dialogue systems. Cofee project addresses this issue from a linguistic viewpoint and combines fine-grained corpus analyses of semi- controlled data with formal and statistical modelling.

Cofee is managed by Laurent Prévot: <http://cofee.hypotheses.org/>

Cofee corpora and the use of SPPAS on such corpora is presented in:

**Jan Gorish, Corine Astésano, Ellen Gurman Bard, Brigitte Bigi, Laurent Prévot (2014).**  
*Aix Map Task corpus: The French multimodal corpus of task-oriented dialogue*, 9th International conference on Language Resources and Evaluation (LREC), Reykjavik (Iceland).

### 7.2.6 Variamu: Variations in Action: a Multilingual approach

Variamu is an international collaborative joint project co-funded by Amidex. The scientific object of the collaboration is the issue of language variation addressed from a comparative perspective. Speech and language knowledge supports a growing number of strategic domains such as Human Language Technologies (HLT), Language Learning (LL), and Clinical Linguistics (CL). A crucial issue to these domains is language variation, which can result from dysfunction, proficiency, dialectal specificities, communicative contexts or even inter-individual differences. Variation is often an important part of the question itself.

This network will be structured around 3 research axes, all centered on the variation concept:

1. Language technologies,
2. Linguistics and Phonetics,
3. Speech and Language Pathologies.

SPPAS is mainly concerned by the first axe.

The first result of this project is the participation of SPPAS at the Evalita 2014 campaign, for the FACS task: Forced-Alignment on Children Speech.

The second result of this project is the support of Cantonese into SPPAS, thanks to a collaboration with Prof. Tan Lee of the University of Hong Kong.