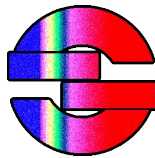




SPPAS

the automatic annotation
and analysis of speech

by Brigitte Bigi



Any help is welcome to improve this documentation!
Contact: brigitte.bigi@gmail.com
© 2011-2017 - Brigitte Bigi

Contents

1	Introduction	3
1.1	What is SPPAS?	3
1.1.1	Main features	3
1.1.2	Copyright and Licenses	4
1.1.3	User engagement: how to cite SPPAS	4
1.1.4	Need help	5
1.1.5	Supports	5
1.1.6	Contributors	5
1.2	Getting and installing	6
1.2.1	Websites	6
1.2.2	External programs	6
1.2.3	Download and install SPPAS	6
1.2.4	The SPPAS package	7
1.2.5	Update	8
1.3	Capabilities	8
1.3.1	What SPPAS can do?	8
1.3.2	How to use SPPAS?	9
1.3.3	Interoperability and compatibility	9
1.4	Main and important recommendations	11
1.4.1	About files	11
1.4.2	About automatic annotations	11
1.5	About this documentation	11
2	Interface	13
2.1	Introduction	13
2.2	GUI Main frame	13
2.2.1	Launch SPPAS	13

2.2.2	The tips	14
2.2.3	The menu	14
2.2.4	The file explorer	15
2.2.5	Settings	17
2.2.6	About	17
2.2.7	Help	19
2.2.8	Plugins	19
2.3	CLI	20
3	Automatic Annotations	23
3.1	Introduction	23
3.1.1	About this chapter	23
3.1.2	Annotations methodology	23
3.1.3	File formats and tier names	25
3.1.4	Recorded speech	25
3.1.5	Orthographic Transcription	25
3.1.6	Automatic Annotations with GUI	27
3.1.7	Automatic Annotations with CLI	27
3.1.8	The procedure outcome report	29
3.2	Inter-Pausal Units (IPUs) segmentation	30
3.2.1	Overview	30
3.2.2	Silence/Speech segmentation	32
3.2.3	Silence/Speech segmentation time-aligned with a transcription	33
3.2.4	Split into multiple files	35
3.2.5	Perform IPUs Segmentation with the GUI	35
3.2.6	Perform IPUs Segmentation with the CLI	35
3.3	Tokenization	37
3.3.1	Overview	37
3.3.2	Adapt Tokenization	37
3.3.3	Perform Tokenization with the GUI	37
3.3.4	Perform Tokenization with the CLI	38
3.4	Phonetization	39
3.4.1	Overview	39
3.4.2	Adapt Phonetization	40
3.4.3	Perform Phonetization with the GUI	40

3.4.4	Perform Phonetization with the CLI	40
3.5	Alignment	42
3.5.1	Overview	42
3.5.2	Adapt Alignment	42
3.5.3	Perform Alignment with the GUI	42
3.5.4	Perform Alignment with the CLI	43
3.6	Syllabification	44
3.6.1	Overview	44
3.6.2	Adapt Syllabification	45
3.6.3	Perform Syllabification with the GUI	46
3.6.4	Perform Syllabification with the CLI	46
3.7	Repetitions	46
3.7.1	Overview	46
3.7.2	Perform Repetitions with the GUI	47
3.7.3	Perform Repetitions with the CLI	47
3.8	Momel and INTSINT	47
3.8.1	Momel (modelling melody)	48
3.8.2	Encoding of F0 target points using the “INTSINT” system	48
3.8.3	Perform Momel and INTSINT with the GUI	50
3.8.4	Perform Momel and INTSINT with the CLI	50
4	Resources for Automatic Annotations	51
4.1	What are SPPAS resources and where they come from?	51
4.1.1	Overview	51
4.1.2	About the phonemes	52
4.1.3	How to add a new language	52
4.2	French Resources	53
4.3	Italian resources	55
4.4	Spanish resources	56
4.5	Catalan resources	58
4.6	English	60
4.7	Mandarin Chinese	61
4.8	Southern Min (or Min Nan) resources	63
4.9	Cantonese resources	65
4.10	Polish Resources	67
4.11	Portuguese Resources	68

5	Analyses	71
5.1	Introduction	71
5.2	DataRoamer	73
5.3	AudioRoamer	73
5.3.1	Properties of the audio file	74
5.3.2	Playing audio files	75
5.3.3	Want more?	75
5.4	IPUscriber	77
5.5	Vizualizer	78
5.6	DataFilter	78
5.6.1	Filtering annotations of a tier: SingleFilter	79
5.6.2	Filtering on time-relations between two tiers	81
5.7	Statistics	83
5.7.1	Descriptive statistics	83
5.7.2	TGA - Time Group Analyzer	84
5.7.3	User agreement	85
6	Scripting with Python and SPPAS	87
6.1	Introduction	87
6.2	A gentle introduction to programming	87
6.2.1	Definitions	87
6.2.2	Comments and blocks	88
6.2.3	Variables: Assignment and Typing	88
6.2.4	Basic Operators	89
6.2.5	Conditions	89
6.2.6	Loops	90
6.3	Scripting with Python	90
6.3.1	Hello World!	90
6.3.2	Functions	92
6.3.3	Reading/Writing files	94
6.3.4	Dictionaries	97
6.3.5	Exercises to practice	97
6.4	The API of SPPAS to manage data	97
6.4.1	Overview	97
6.4.2	Why developing a new API?	98

6.4.3	The API class diagram	98
6.5	Creating scripts with the SPPAS API	99
6.5.1	Preparing the data	99
6.5.2	Read/Write annotated files	102
6.5.3	Manipulating a Transcription object	103
6.5.4	Manipulating a Tier object	104
6.5.5	Main information on Annotation/Location/Label objects	105
6.5.6	Exercises	105
6.5.7	Search in annotations: Filters	105
6.5.8	Exercises	108
7	References	109
7.1	Publications about SPPAS	109
7.1.1	Main reference	109
7.1.2	Specific references	110
7.1.3	Related references	112
7.2	SPPAS in research projects	113
7.2.1	MULTIPHONIA	113
7.2.2	Amennpro	113
7.2.3	Evalita 2011: Italian phonetization and alignment	113
7.2.4	Cofee: Conversational Feedback	114
7.2.5	Variamu: Variations in Action: a MUltilingual approach	114
8	SPPAS Release notes	115
8.1	the Prehistory	115
8.2	the Middle Ages	116
8.3	the Renaissance	127
8.4	the Early modern period	130
8.5	the Modern period	136

Introduction

1.1 What is SPPAS?

1.1.1 Main features

SPPAS - the automatic annotation and analyses 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.

SPPAS is daily developed with the aim to provide a robust and reliable software. Available for free, with open source code, there is simply no other package for linguists to simple use in both the automatic annotations of speech and the analyses of any kind of annotated data. You can imagine the annotations or analyses you need, SPPAS does the rest.

As the primary functionality, SPPAS proposes a set of automatic or semi-automatic annotations of recordings. Corpus annotation “can be defined as the practice of adding interpretative, linguistic information to an electronic corpus of spoken and/or written language data. ‘Annotation’ can also refer to the end-product of this process” (Leech, 1997). The annotation of recordings is concerned by many Linguistics sub-fields as Phonetics, Prosody, Gestures or Discourse... Corpora are annotated with detailed information at various linguistic levels thanks to annotation software. As large multimodal corpora become prevalent, new annotation and analysis requirements are emerging. The annotations must be time-synchronized: annotations need to be time-aligned in order to be useful for purposes such as qualitative or quantitative analyses. Temporal information makes it possible to describe behaviors from different subjects that happen at the same time, and time-analysis of multi-level annotations can reveal Linguistic structures. In the past, studies was mostly based on limited data. In current trends, it is expected that models have to be built on the acoustic analysis of large quantity of speech data with valid statistical analyses. Annotating is very labor-intensive and cost-ineffective since it has to be performed manually by experienced researchers with many hours of work. SPPAS can automatize annotations and allows users to save time. In order to use efficiently this automatic annotation software, a rigorous methodology to collect data and to prepare them is expected. This implies a rigorous framework to ensure compatibilities between annotations and time-saving. The expected result is time-aligned data, for all annotated levels.

Indeed, “when multiple annotations are integrated into a single data set, inter-relationships between the annotations can be explored both qualitatively (by using database queries that combine levels) and quantitatively (by running statistical analyses or machine learning algorithms)” (Chiarcos 2008). Some special features are

also offered in SPPAS for managing corpora of annotated files; particularly, it includes a tool to filter multi-levels annotations (Bigi and Saubesty, 2015). Some other tools are dedicated to the analysis of time-aligned data; as for example to estimate descriptive statistics, and a version of the Time Group Analyzer (Gibbon 2013), etc.

Linguistics annotation, especially when dealing with multiple domains, makes use of different tools within a given project. In recent years, many annotation software/tools have become available for annotation of audio-video data. For a researcher looking for an annotation software, it is difficult to select the most appropriate. Due to the diversity of linguistic phenomena, annotation tools lead to a variety of models, theories and formalisms. This diversity results in heterogeneous description formats, each tool developing its own framework. The choice of all annotation software is part of the annotation framework and must be done carefully and of course before the creation of the corpus. SPPAS annotation files are in a specific XML format (xra). Annotations can be imported from and exported to a variety of other formats, including Praat (TextGrid, PitchTier, IntensityTier), Elan (eaf), Transcriber (trs), Annotation Pro (antx), Phonedit (mrk), Sclite (ctm, stm), HTK (lab, mlf), subtitles formats (srt, sub) and CSV files.

Automatic annotations can be used either with a Command-line User Interface or a Graphical User Interface. So, there's no specific difficulty by using this software. Advanced users can also access directly the Application Programming Interface. The program was implemented using the programming language Python 2.7. The only potential brake on the usage of automatic annotations of SPPAS is the need to integrate it in a rigorous methodology for the corpus construction, annotations and analyses.

Data analysis of SPPAS are mainly proposed in the Graphical User Interface. However, advanced users can also access directly the Application Programming Interface, for example to estimate statistics or to manipulate annotated data.

1.1.2 Copyright and Licenses

(c) 2011-2016 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: how to cite SPPAS

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

See the “References” chapter of this documentation to get the list or the home page <http://www.lpl-aix.fr/~bigi/> to get printable versions of the references.

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, contact the author by e-mail.

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/>

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>
- Campus France, Procore PHC.

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

Since January 2016, the main website of SPPAS is located at the following URL:

<http://www.sppas.org>

The source code with recent stable releases is hosted on github. From this website, anyone can download the development version, contribute, send comments and/or declare an issue:

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

1.2.2 External programs

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 5 up to 10 minutes depending on the operating system.

The following software are required:

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

It is very (very very) important to take care about the version of Python. An installation guide is available on the website, depending on the operating system. **Please, closely follow the instructions.** Notice that administrator rights are required to perform the installations.

1.2.3 Download and install SPPAS

The main website contains the `Download Page` to download a new version.

SPPAS is ready to run, so it does not need elaborate installation, except for the 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. Choose *a location with only US-ASCII 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 the 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 tool, 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 directory with content as files and folders.

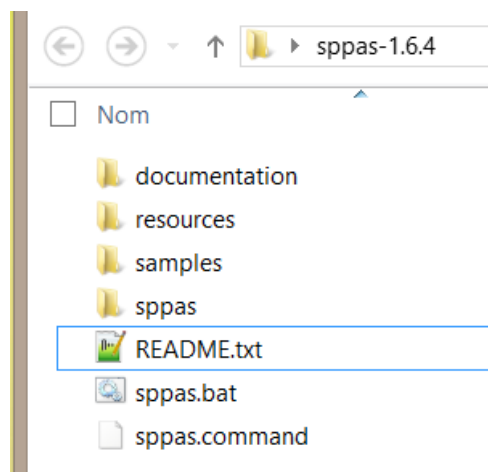


Figure 1.2: SPPAS Package content

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` directory contains data that are used by automatic annotations (lexicons, dictionaries, ...)
- the `samples` directory contains folders of data of various languages, they are sets of annotations freely distributed to test features of SPPAS
- the `sppas` directory contains the program itself
- the `documentation` directory contains:

- the copyright
- the printable documentation
- the printable version of the main reference published in “the Phonetician” journal
- the folder `scripting_solutions` is a set of python scripts corresponding to the exercises proposed in the chapter “Scripting with Python and SPPAS”

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 easy and fast:

1. Download the last updated package from the SPPAS main web site;
2. Unpack the downloaded package;
3. Optionally, put the old one into the Trash.

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 and semi-automatic annotations
 - *Momel*: modelling melody
 - *INTSINT*: Intonation
 - *IPUs segmentation*: speech/silence segmentation
 - *Tokenization*: text normalization
 - *Phonetization*: grapheme to phoneme conversion
 - *Alignment*: phonetic segmentation
 - *Syllabification*: group phonemes into syllables
 - *Repetitions*: detect self-repetitions, and other-repetitions
2. Analysis
 - *IPUscriber*: Manual orthographic transcription, after IPUs segmentation
 - *AudioRoamer*: Play, show information and manage speech audio files
 - *Statistics*: Estimates/Save statistics on annotated files
 - *DataRoamer*: Manipulate annotated files
 - *DataFilter*: Extract data from annotated files (i.e. querying data)
 - *Vizualizer*: Display sound and annotated files
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

In the scope of the compatibility between SPPAS data and annotated data from other software tools or programs, SPPAS is able to open/save and convert files. The conversion of a file to another file is the process of changing the form of the presentation of the data, and not the data itself. Every time, when data file is to be used, they must be converted to a readable format for the next application. A data conversion is normally an automated process to some extent. SPPAS provide the possibility to automatically import and export the work done on some various file formats from a wide range of other software tools. For the users, the visible change will be only a different file extension but for software it is the difference between understanding of the contents of the file and the inability to read it.

SPPAS supports the following software with their file extensions:

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

And the followings can be imported:

- ANVIL: anvil
- Transcriber: trs
- Xtrans: tdf

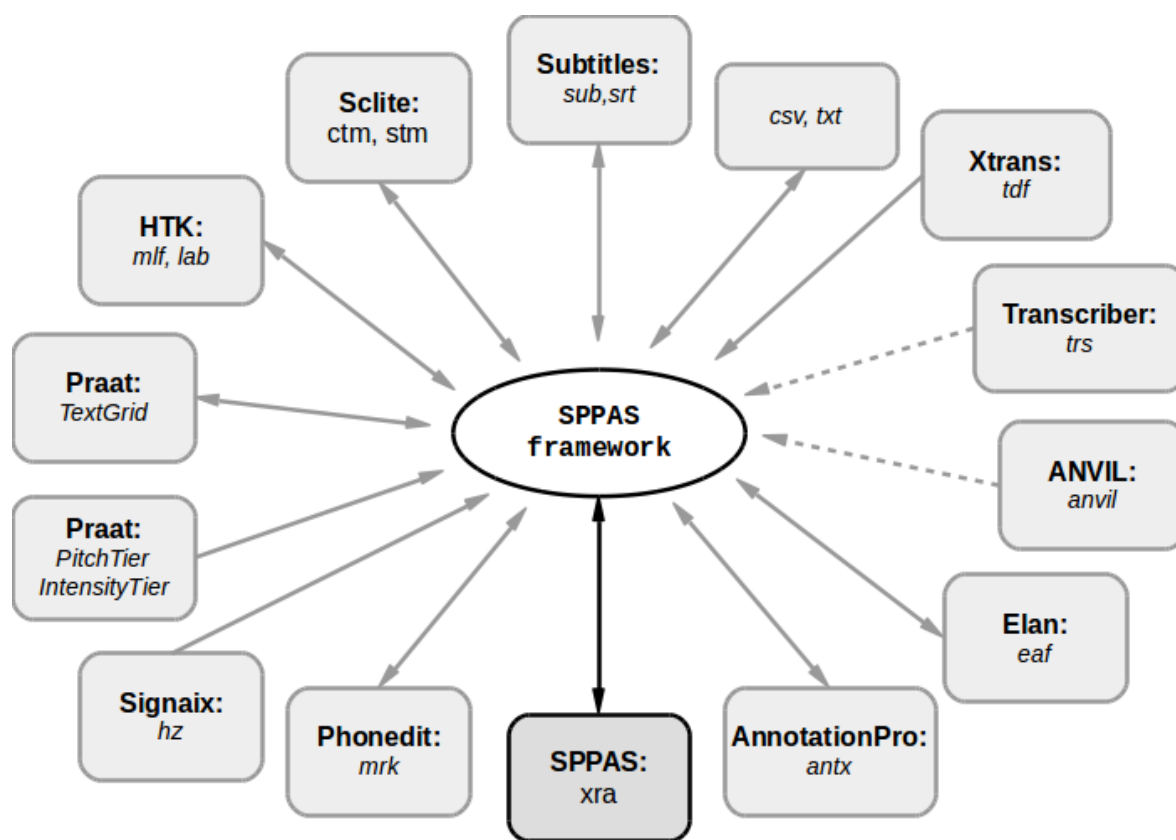


Figure 1.3: File formats SPPAS can open/save and import/export

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 the chapters of this documentation:

1. Speech audio 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 required to never convert from a compressed file (as mp3 for example).

2. Annotated data files:

- UTF-8 encoding only
- it is recommended to use only US-ASCII characters in file names (including their path)

1.4.2 About automatic annotations

The quality of the results for most of the automatic 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 gently shared and freely available in the SPPAS package. The quality of the automatic annotations is largely influenced by such resources, and the users can contribute to improve them. In that sense, users need automatic tools and such tools need users. Users are of crucial importance for resource development. They can contribute to improve them, and the author releases the improvements to the public, so that the whole community benefits. Resources are distributed under the terms of public licenses, so that SPPAS users are “free”: free to study the source code and the resources of the software they use, free to share the software and resources with other people, free to modify the software and resources, and free to publish their modified versions of the software and resources.

1.5 About this documentation

This documentation will assume that you are using a relatively recent version of SPPAS. There’s no reason not to download the latest version whenever released: it’s easy and fast!

Any and all constructive comments are welcome.

Interface

2.1 Introduction

The current chapter describes how to use SPPAS with the Graphical User Interface (GUI) and with the Command-line User Interface (CLI). Each feature implemented in SPPAS corresponds to a program that can be invoked by the GUI or the CLI.

2.2 GUI Main frame

2.2.1 Launch SPPAS

Under Windows, once the SPPAS package is opened in the File Explorer, double-click on the `sppas.bat` file. In recent versions of Windows (e.g. 10), the first time you try to run SPPAS by clicking on `sppas.bat` you may get a message: “” The solution is to click on the text “”. It will display a button to launch the program, then click on it.

Under MacOS, once the SPPAS package is opened in the Finder/File Explorer, double-click on the `sppas.command` file. In recent versions of MacOS X (e.g. 10.11 El Capitan), the first time you try to run SPPAS by clicking on `sppas.command` you may get a message: “sppas.command can’t be opened because it is from an unidentified developer.”. The solution is to run SPPAS with a right click (alt-click) on `sppas.command`. This time you will get a message: “sppas.command is from an unidentified developer. Are you sure you want to open it?” Click on Open and SPPAS will now run. It will also now work each time you try to run it.

Under 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. It will then check if the julius program can be accessed.

The main windows will open automatically. It is made of a menu (left), a title (top), the tips (middle), the list of files (left-middle), and the action buttons (right).

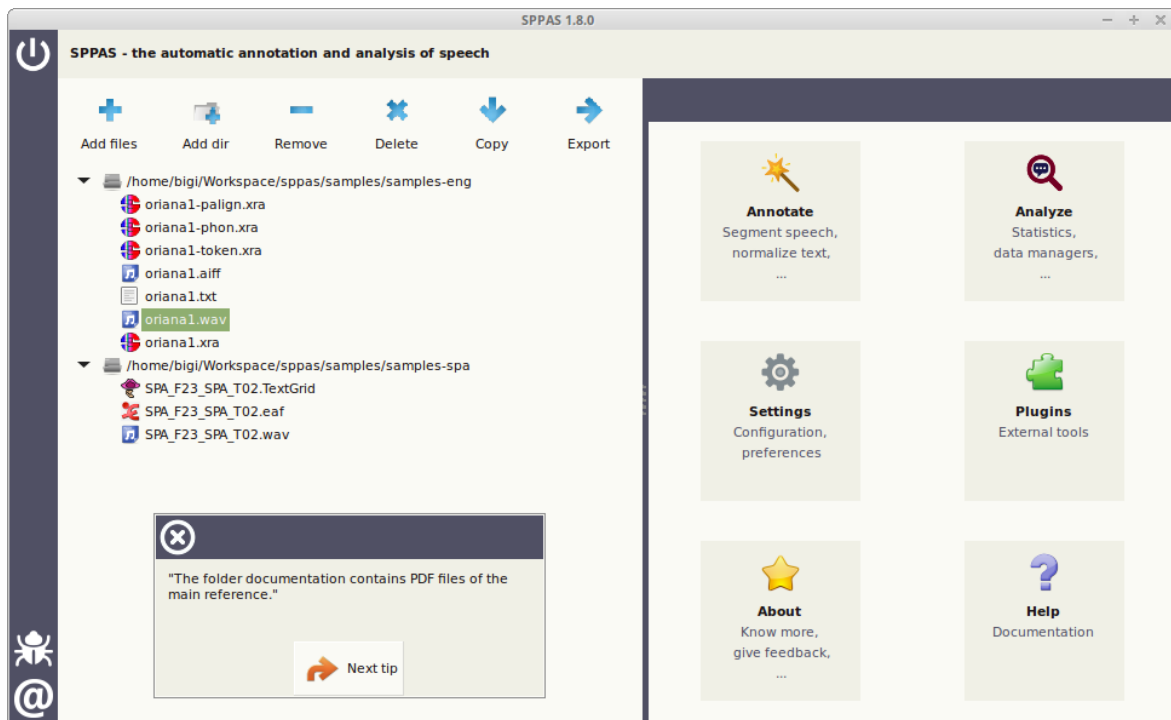


Figure 2.1: SPPAS Main Frame of version 1.8.0

2.2.2 The tips

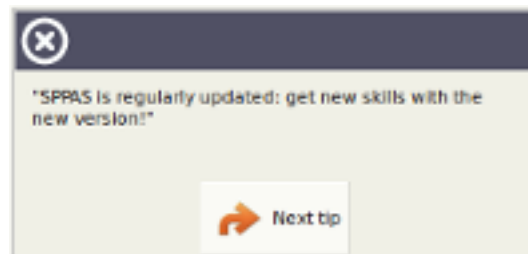


Figure 2.2: The tips included in the main frame

The frame includes message tips that are picked up randomly and printed to help users. Click on the button **Next Tip** to read another message, or click on the top-left button to close the tips frame. The **Settings** allows to show/hide tips at start-up. If you want to suggest new tips to help the other users, they have to be sent to the author by e-mail. They will be included in the next version.

2.2.3 The menu

The menu is located at the left-side of the window. At its top, a button allows to exit the program, and at bottom you can declare an issue or contact the author.

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 declare an issue, click on the bug button of the menu, then your default web browser will be opened at the appropriate URL. Take a quick look at the list of already declared issues and if any, click on the green

button “New Issue”.

To contact the author, 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.

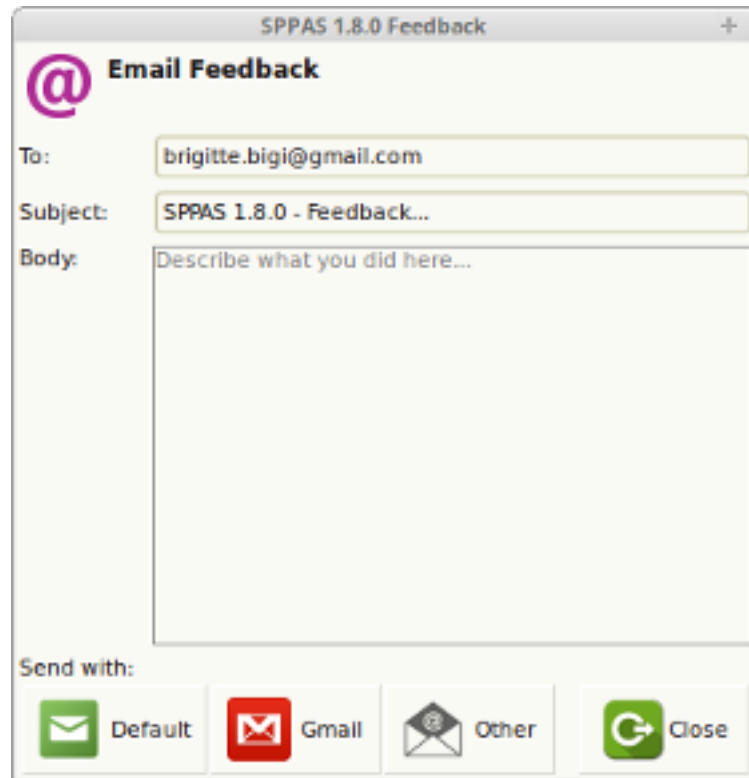


Figure 2.3: The Feedback Frame

2.2.4 The file explorer

The list contains directories and files the user added, but only files that SPPAS can deal with, e.g. depending on the file extension.

To select:

- 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.

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.

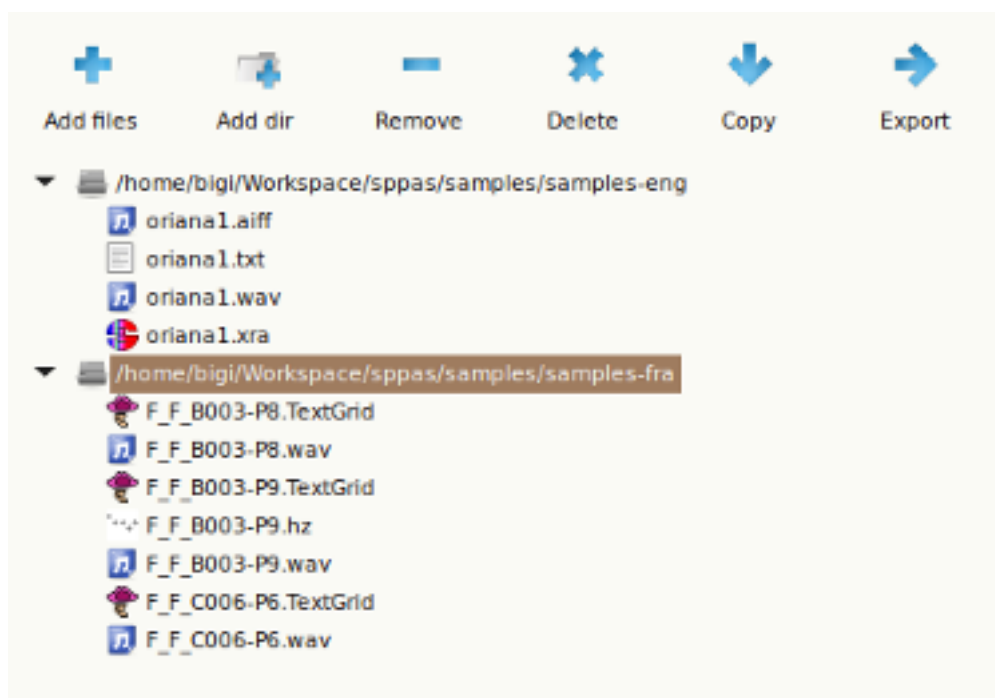


Figure 2.4: File List Panel (FLP)

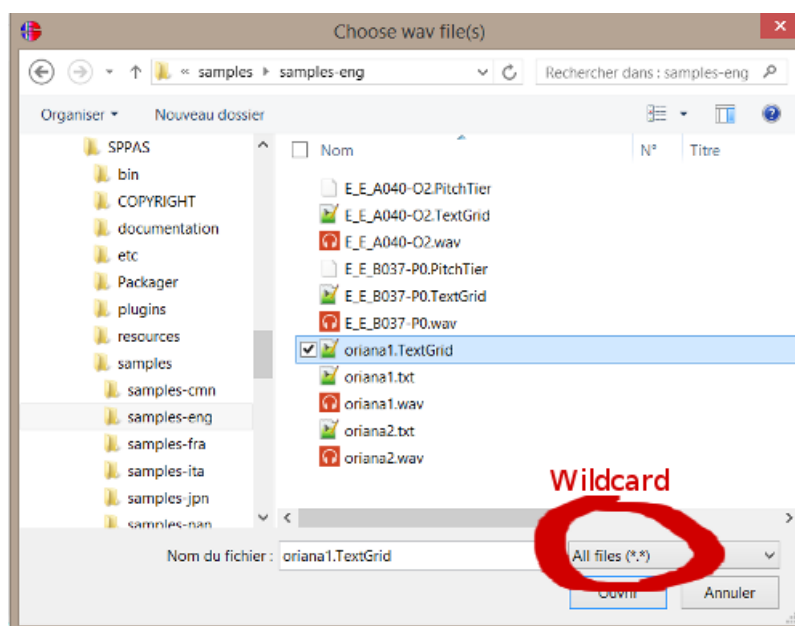


Figure 2.5: Adding specific files

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 way 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, ...):

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

2.2.5 Settings

To change preferences, click on the `Settings` icon, then choose your preferred options:

- General: fix background color, font color 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.

2.2.6 About

The `About` action button allows to display the main information about SPPAS: authors, license, web site URL, etc.

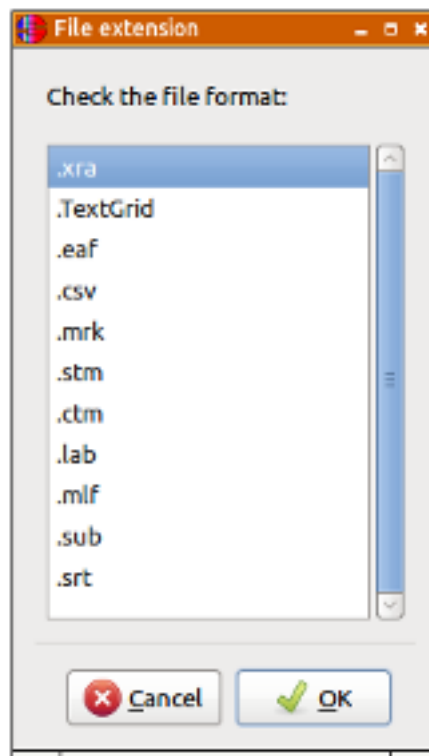


Figure 2.6: Export: Check the expected format in the list

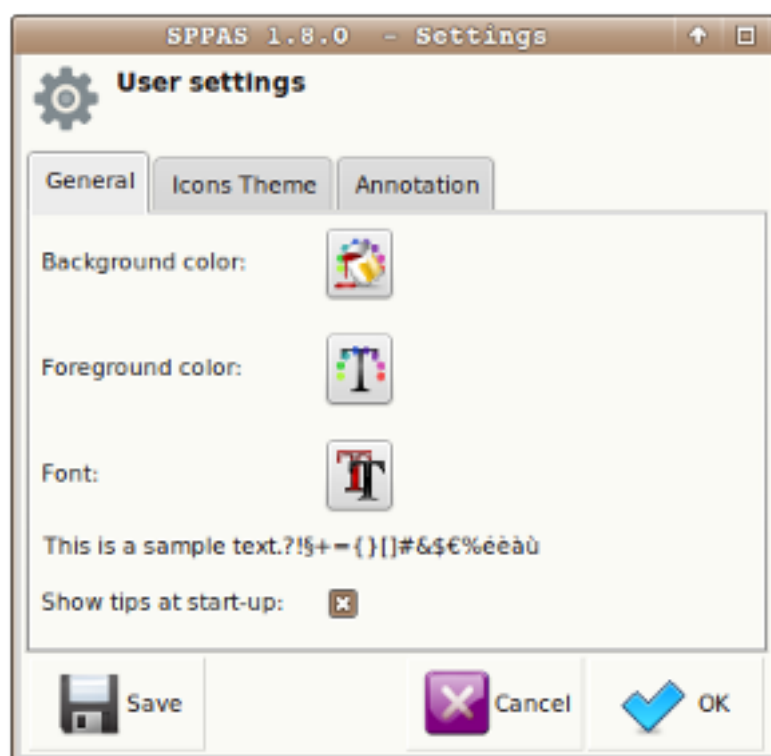


Figure 2.7: Settings is used to manage user preferences

2.2.7 Help

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

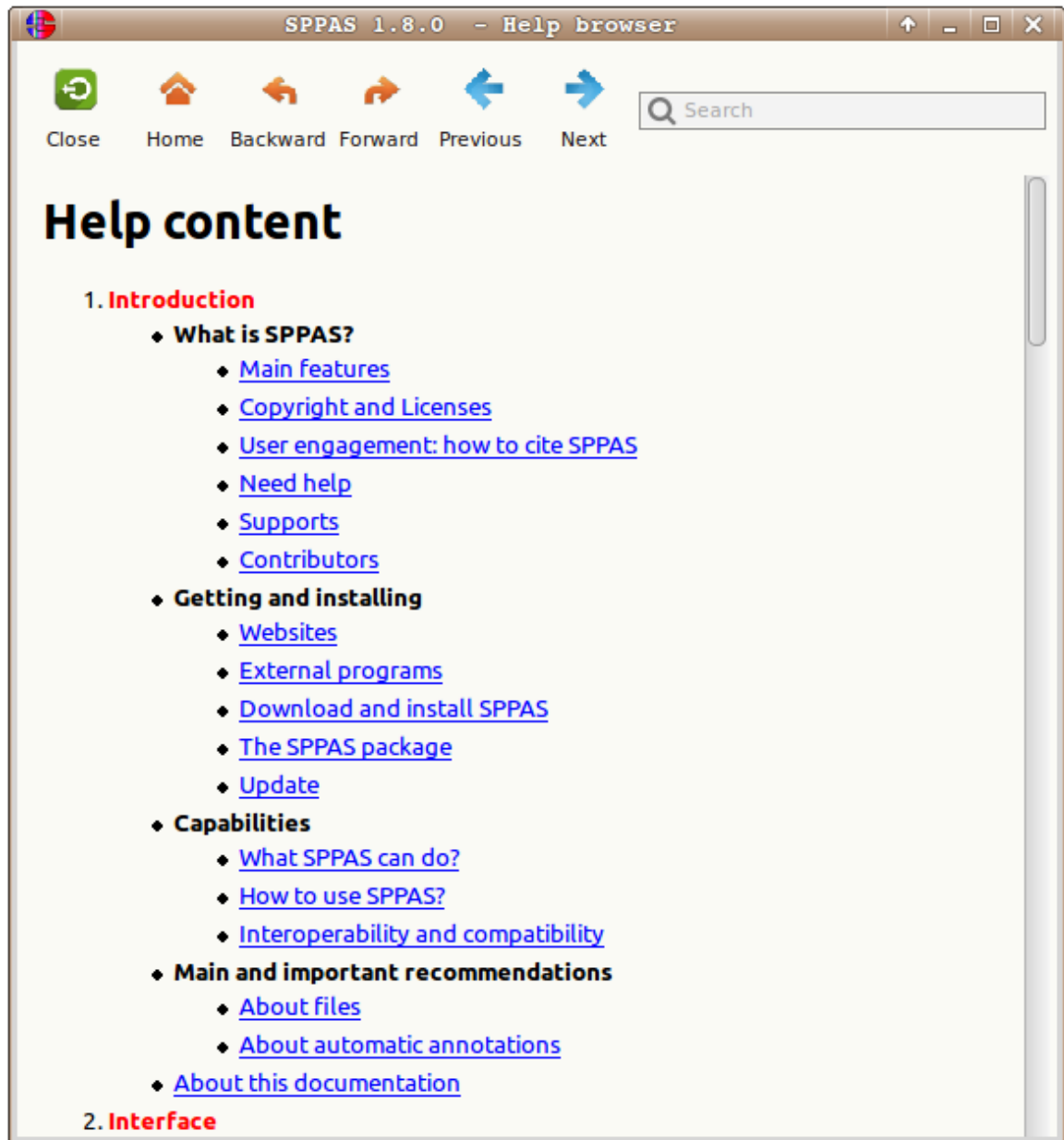


Figure 2.8: The Help Browser Frame

2.2.8 Plugins

Installing plugins is a very useful solution to extend the features of SPPAS. Several plugins are available for download in the main site of SPPAS. The plugins of SPPAS are installed in a folder with name “plugins” in the main directory of SPPAS. Then, do not remove/rename this folder!

The plugin system of SPPAS were fully changed at version 1.8.2. Old plugins and new ones are not compatibles.

Installing a plugin

To install a plugin, follow this workflow:

1. Download the plugin package (a zip file).
2. Execute SPPAS.
3. Click on the 'Plugin' icon then click on the 'Install' button of the toolbar.
4. Indicate the package of the plugin.
5. See the new plugin icon in the plugins list.

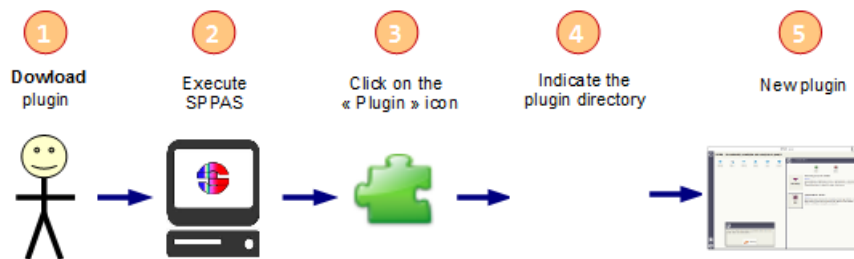


Figure 2.9: Installing a new Plugin

Deleting a plugin

To delete a plugin, click on the “Delete” button of the toolbar. Choose the plugin in the given list then click on the OK button. Notice that the plugin is definitively deleted of the disk.

Using a plugin

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.

2.3 CLI

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).

Such programs are located in the `bin` folder of the `sppas` directory included in the SPPAS package. All these programs are written with the programming language Python 2.7.

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` directory then `bin` folder:

1. programs to execute an automatic annotation;
2. programs to execute an analysis tool.

Moreover, the program “`plugin.py`” allows to install, execute and remove a plugin of SPPAS.

Automatic Annotations

3.1 Introduction

3.1.1 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, what is the expected result and how to perform it with SPPAS. Each automatic annotation is 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).

3.1.2 Annotations methodology

The kind of process to implement in the perspective of obtaining rich and broad-coverage multimodal/multi-levels annotations of a corpus is illustrated in next Figure. It describes each step of the annotation workflow. This Figure must be read from top to bottom and from left to right, starting by the recordings and ending to the analysis of annotated files. Yellow boxes represent manual annotations, blue boxes represent automatic ones.

After the recording, the first annotation to perform is IPUs segmentation. Indeed, at a first stage, the audio signal must be automatically segmented into Inter-Pausal Units (IPUs) which are blocks of speech bounded by silent pauses of more than X ms, and time-aligned on the speech signal. An orthographic transcription has to be performed manually inside the set of IPUs. Then tokenization will normalize the transcription. The phonetization process will convert the normalized text in a set of pronunciations using X-SAMPA standard. Alignment will perform segmentation at phonemes and tokens levels, etc.

At the end of each automatic annotation process, SPPAS produces a Procedure Outcome Report that contains important information about the annotations.

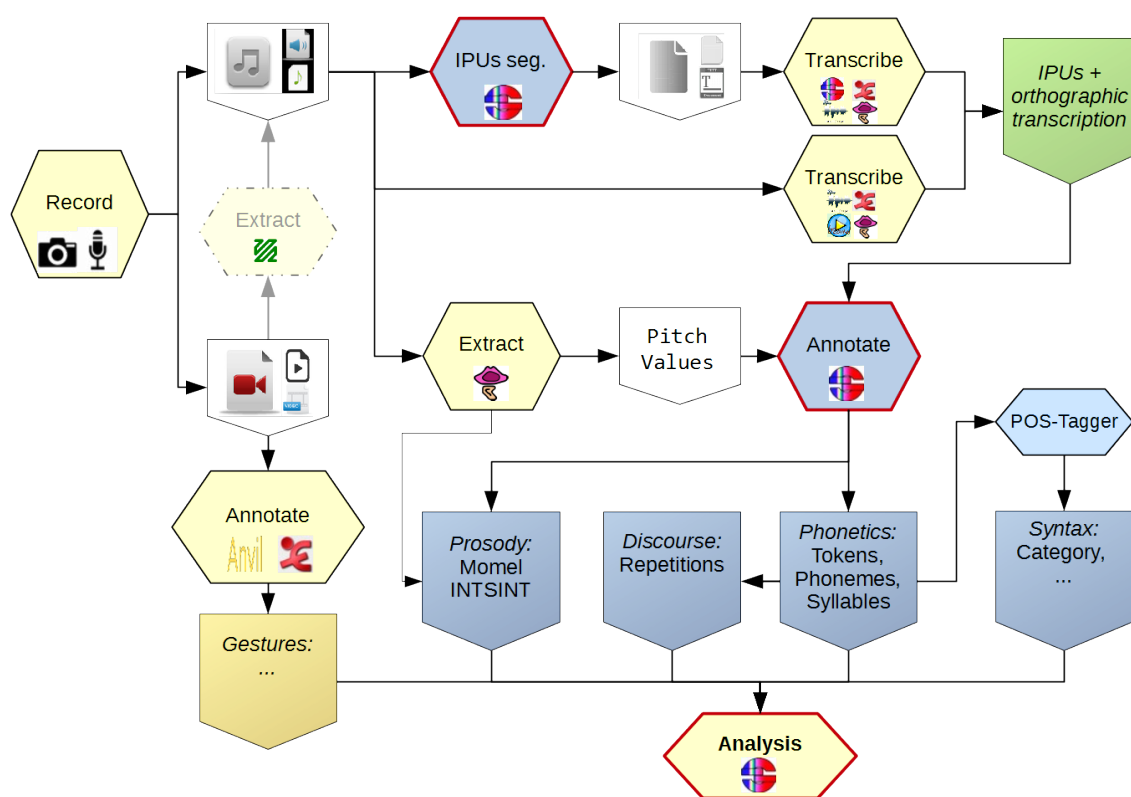


Figure 3.1: Annotation methodology

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

3.1.3 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, however 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!**

3.1.4 Recorded speech

First of all:

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

SPPAS verifies if the audio 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.

3.1.5 Orthographic Transcription

An orthographic transcription is often the minimum requirement for a speech corpus so it is at the top of the annotation procedure, and it is the entry point for most of the automatic annotations, including automatic speech segmentation. But clearly, there are different ways to pronounce the same utterance. 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 is a subjective representation of what is “perceived” in the signal. It **must** includes:

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

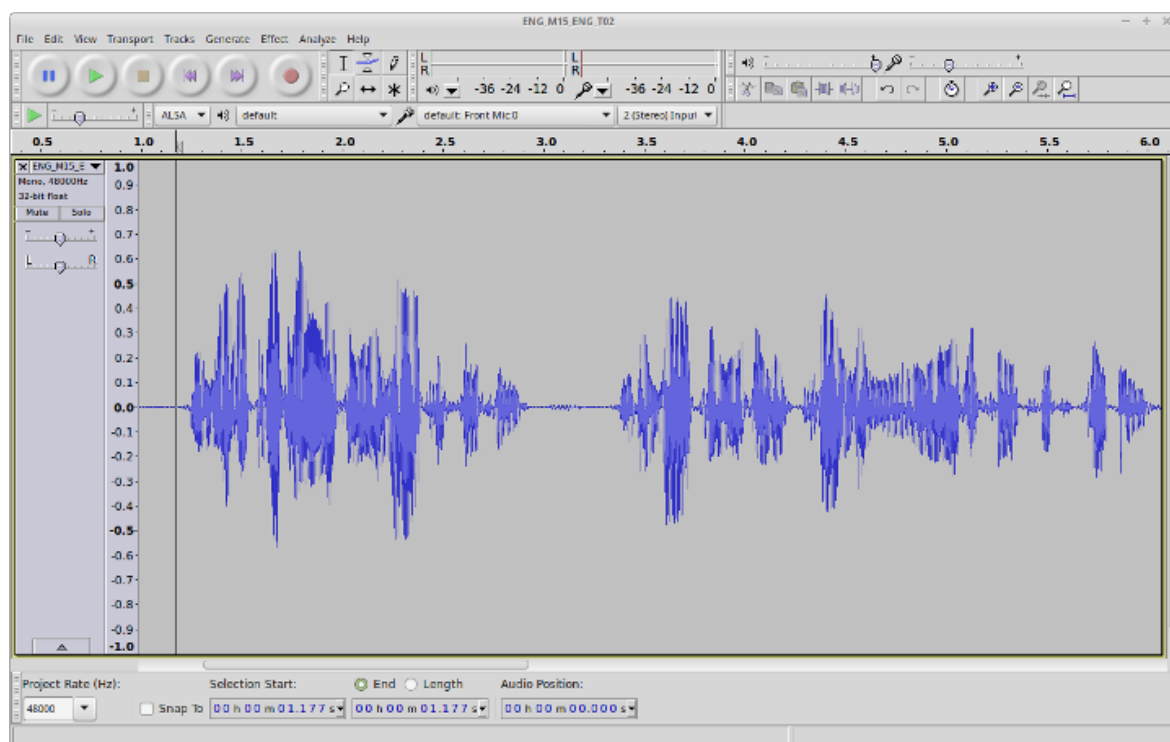


Figure 3.2: Example of recorded speech

In 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 like invented words, regional words or words borrowed from another language. These specifics have a direct consequence on the automatic phonetization procedure as shown in (Bigi 2012). As a consequence, from the beginning of its development it was considered to be essential for SPPAS to deal with **Enriched Orthographic Transcriptions**.

The transcription must use the following convention to represent speech phenomena. All the symbols must be surrounded by whitespace.

- 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.

This convention is not software-dependent which means that it can be performed with IPUScriber tool of SPPAS, Praat, Annotation Pro, ...

The result is what we call an Enriched Orthographic construction, from which two derived transcriptions can be generated automatically:

1. the **standard transcription** is the list of orthographic tokens
2. a specific transcription from which the phonetic tokens are obtained to be used by the grapheme-phoneme converter that is named **faked transcription**.

As for example with the following sentence:

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

3.1.6 Automatic Annotations with GUI

To perform automatic annotations with SPPAS Graphical User Interface, select first the list of audio files and/or a directory, then click on the “Annotate” button.

1. Enable each annotation to perform by clicking on the button in red. It will be turned green.
2. Each annotation can be configured by clicking on the “Configure...” text in blue.
3. Select the language for all annotations in one, or for each one independently by clicking on the “chains” button.
4. Click on the *Perform annotations* button... and wait! Please, be patient! 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!
5. It is important to read the Procedure Outcome report to check that everything happened normally during the automatic annotations.

3.1.7 Automatic Annotations with CLI

To perform automatic annotations with SPPAS Command-line User Interface, there is a main program `annotation.py` and each annotation has also its own program with more options than the main one.

This main program performs automatic annotations on a given file or on all files of a directory. It strictly corresponds to the button `Perform annotations` of the GUI. All annotations are pre-configured: no specific option can be specified. This configuration is fixed in the source code packages, via a file with extension `.cfg` for each annotation.

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

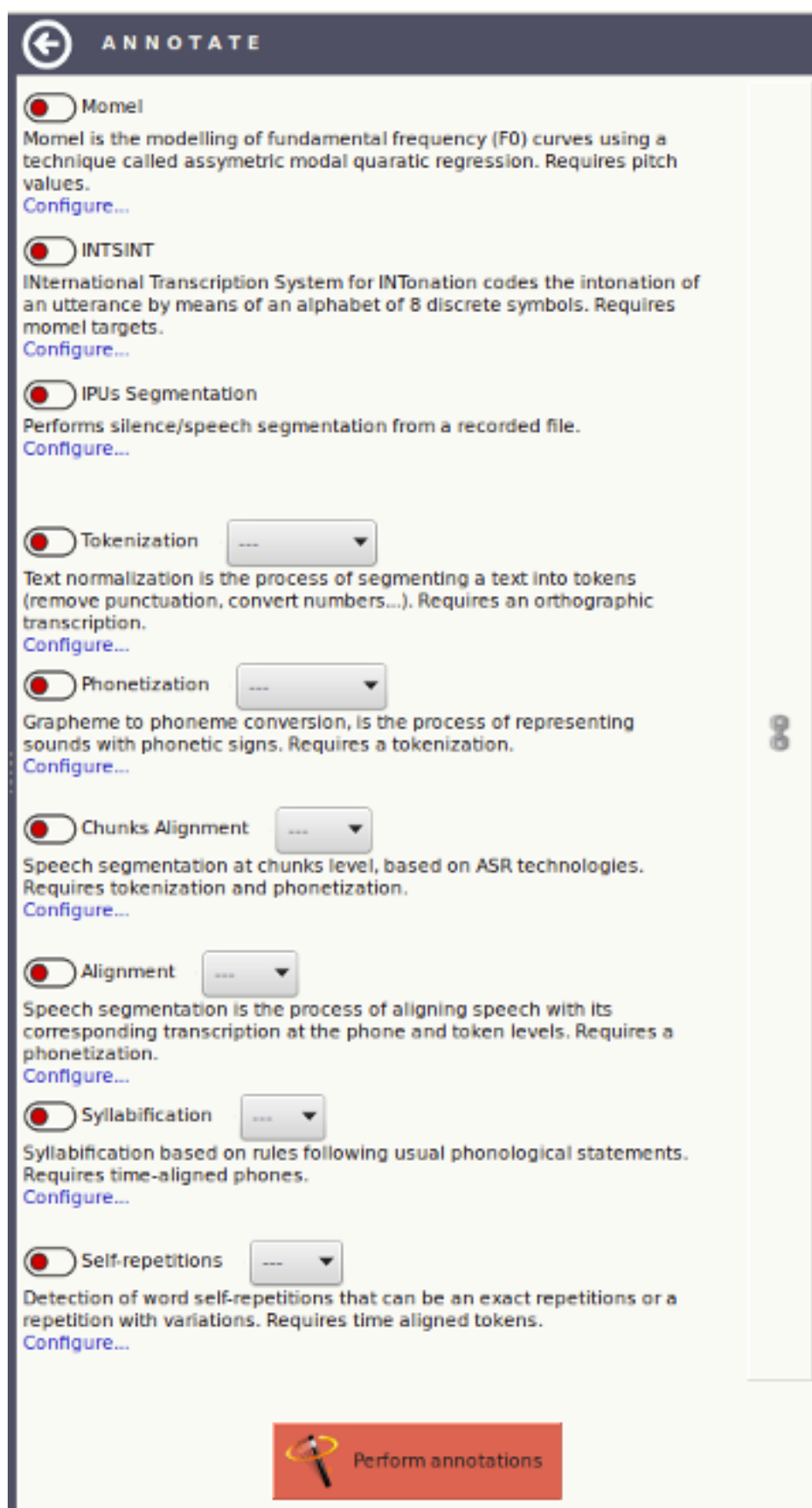


Figure 3.3: the annotate panel

```

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 IPUs Segmentation
  --tok                Activate Tokenization
  --phon               Activate Phonetization
  --align              Activate Alignment
  --syll               Activate Syllabification
  --rep                Activate Repetitions
  --all                Activate ALL automatic annotations
  --merge              Create a merged TextGrid file, if more than two automatic
                        annotations (this is the default).
  --nomerge            Do not create a merged TextGrid file.

```

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, ...).



```

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% [=====] IPU 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 3.4: CLI: annotation.py output example

3.1.8 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. Four levels of information must draw your attention:

1. “[OK]” means that everything happened normally. The annotation was performed successfully.
2. “[IGNORED]” means that SPPAS ignored the file and didn’t do anything.
3. “[WARNING]” means that something happened abnormally, but SPPAS found a solution, and the annotation was still performed successfully.
4. “[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.

3.2 Inter-Pausal Units (IPUs) segmentation

3.2.1 Overview

After recording, the first annotation to perform is IPUs segmentation. Indeed, at a first stage, the audio signal must be automatically segmented in Inter-Pausal Units (IPUs) which are blocks of speech bounded by silent pauses of more than X ms. This X duration depends on the language and it is commonly set to 200ms for French and 250ms for English. IPUs are time-aligned on the speech signal. This segmentation should be verified manually, depending on the quality of the recording: the better quality, thus the better 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.

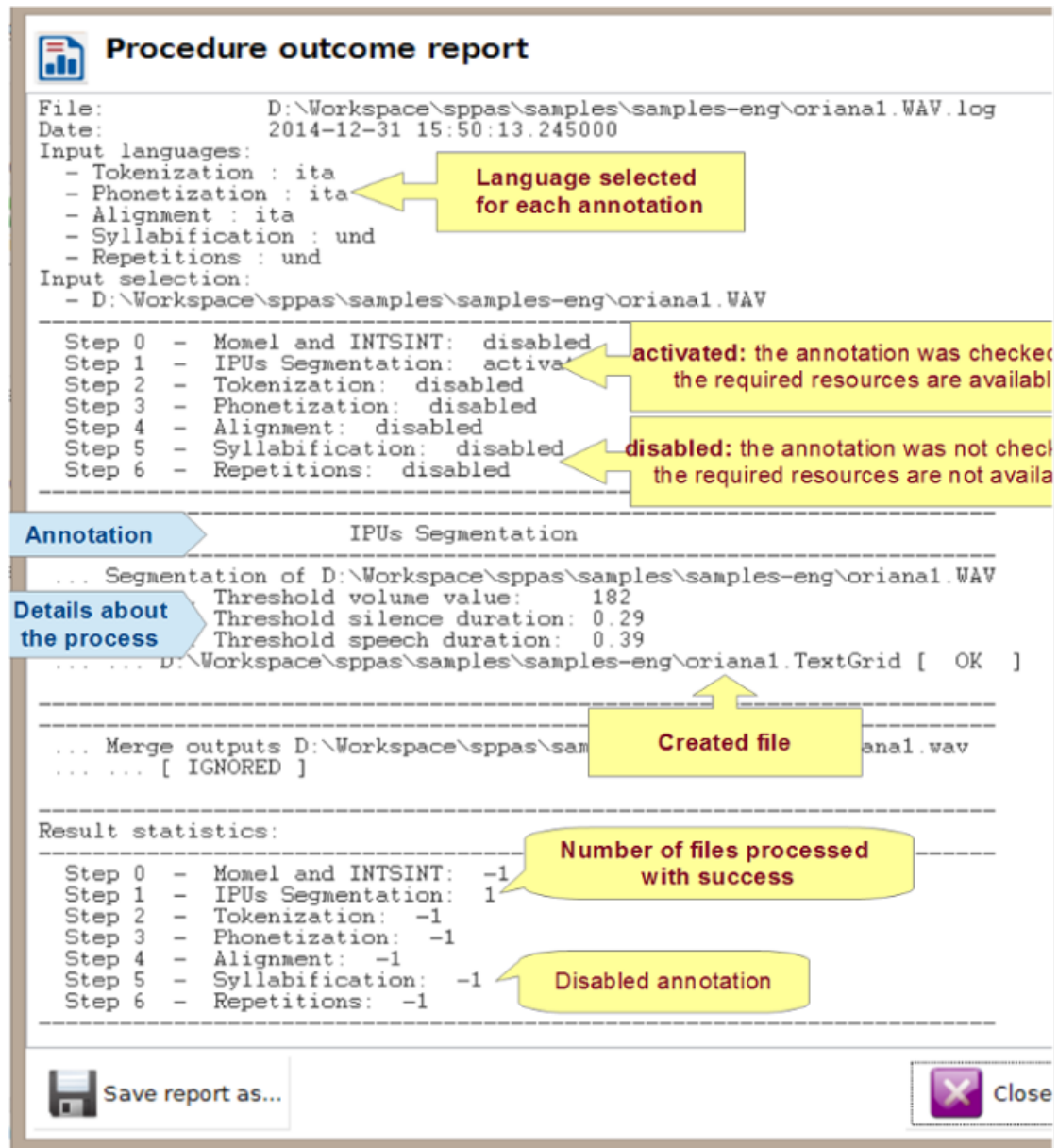


Figure 3.5: Procedure outcome report

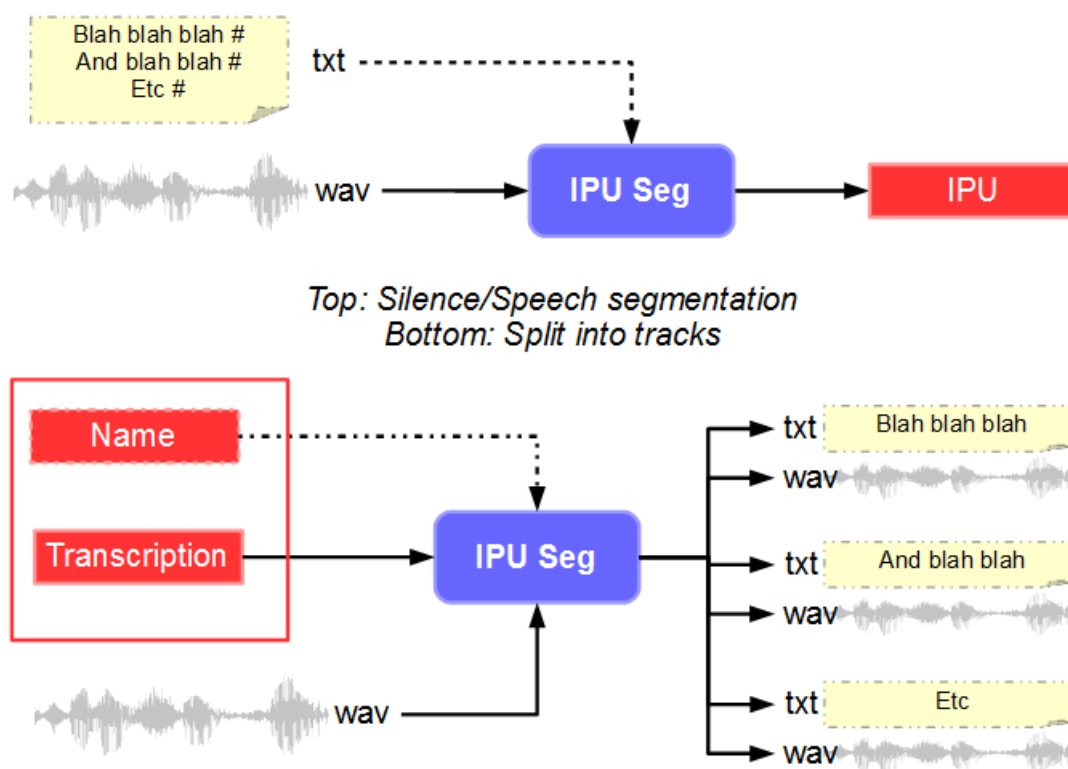


Figure 3.6: IPU Segmentation workflow

3.2.2 Silence/Speech segmentation

The IPU 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
-----
... IPU Segmentation of file /home/big1/workspace/sppas/samples/samples-eng/oriana1.wav
... [ INFO ] A transcription was found, perform Silence/Speech segmentation time-aligned with a transcription
... Options:
...   - save_as_trs: False
...   - dirtracks: False
...   - addipuidx: True
... Diagnosis:
...   - /home/big1/workspace/sppas/samples/samples-eng/oriana1.wav: Valid.
...   - Threshold volume value: 122
...   - Threshold silence duration: 0.25
...   - Threshold speech duration: 0.35
... [ OK ] /home/big1/workspace/sppas/samples/samples-eng/oriana1.xra
-----

```

Figure 3.7: 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.

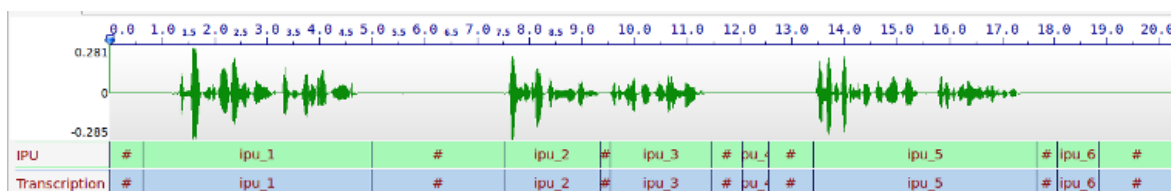


Figure 3.8: Result of IPU Segmentation: silence detection

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

3.2.3 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.

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).

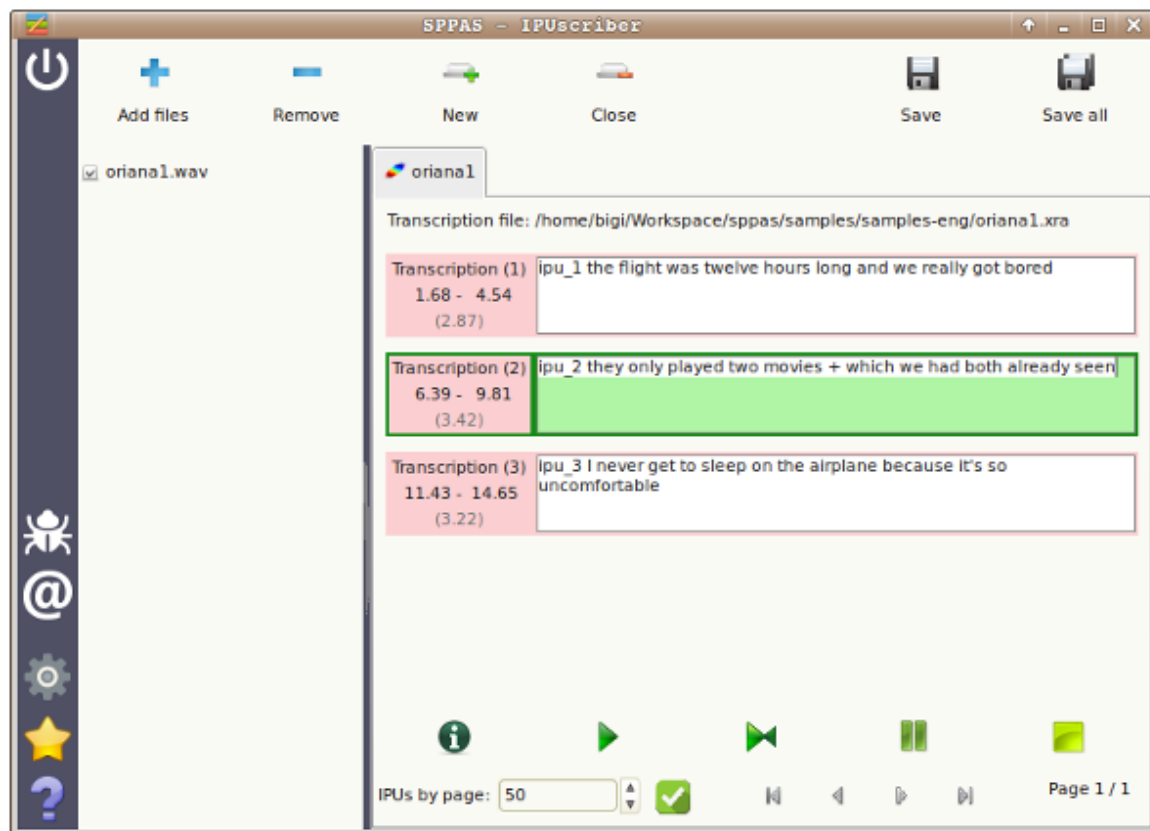


Figure 3.9: Orthographic transcription based on IPUs Segmentation

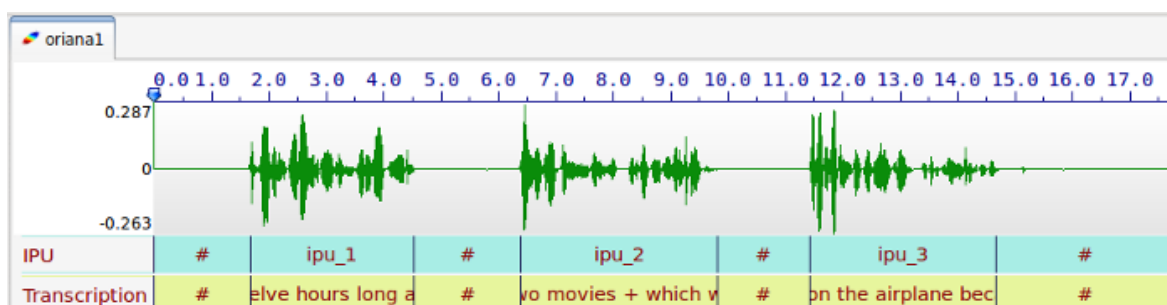


Figure 3.10: Silence/Speech segmentation with orthographic transcription

3.2.4 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 3.11: 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”.

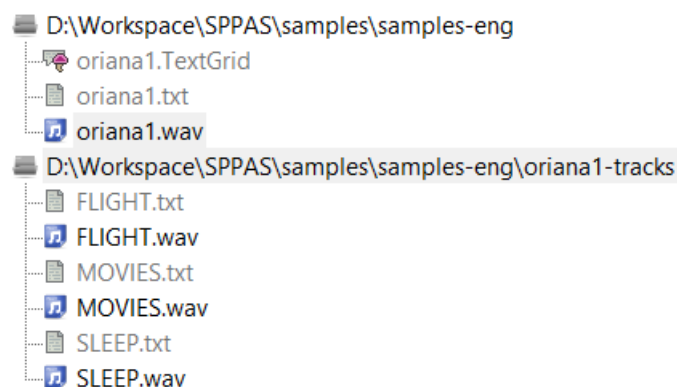


Figure 3.12: Data split

3.2.5 Perform IPUs Segmentation with the GUI

Click on the IPUs Segmentation activation button and on the “Configure...” blue text to fix options.

Notice that all time values are indicated in seconds.

3.2.6 Perform IPUs Segmentation with the CLI

wavsplit.py is the program to perform 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:

```
-w file          audio input file name
-d delta shift  Add this time value to each start
                 bound of the IPUs
-D delta shift  Add this time value to each end
                 bound of the IPUs
-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 (default: None)
-n value        Input transcription tier number (default: None)
-N             Adjust the volume cap until it splits
                 into nb tracks (default: 0=automatic)
```

Output options:

```
-o dir          Output directory name          (default: None)
-e ext          Output tracks extension        (default: txt)
-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 0.01
-D 0.01
-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
```

3.3 Tokenization

3.3.1 Overview

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 implemented in SPPAS (Bigi 2011) 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.

3.3.2 Adapt Tokenization

Word segmentation of SPPAS is mainly based on the use of a lexicon. If a segmentation is not as expected, it is up to the user to modify the lexicon. Lexicons of all supported languages are all located in the folder “vocab” of the “resources” directory. They are in the form of “one word at a line”.

3.3.3 Perform Tokenization with the GUI

The SPPAS Tokenization system takes as input a file (or a list of files) for which the name strictly match the name of the audio file except the extension. For example, if a file with name “oriana1.wav” is given, SPPAS will search for a file with name “oriana1.xra” at a first stage if “.xra” is set as the default extension, then it will search for other supported extensions until a file is found.

This file must include a tier with the orthographic transcription. At a first stage, SPPAS tries to find a tier with `transcription` as name. If such a tier does not exist, SPPAS tries to find a tier that contains one of the following strings:

1. `trans`
2. `trs`
3. `ipu`
4. `ortho`
5. `toe`

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

Tokenization produces a file with “-tokens” appended to its name, i.e. “oriana1-tokens.xra” for the previous example. This file is including only one tier with the resulting tokenization and with name “Tokenization”.

In case of an Enriched Orthographic Transcription, to get both faked and standard tokenized tiers, check the corresponding option. Then, two tiers will be created:

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

Read the “Introduction” of this chapter for a better understanding of the difference between “standard” and “faked” transcriptions.

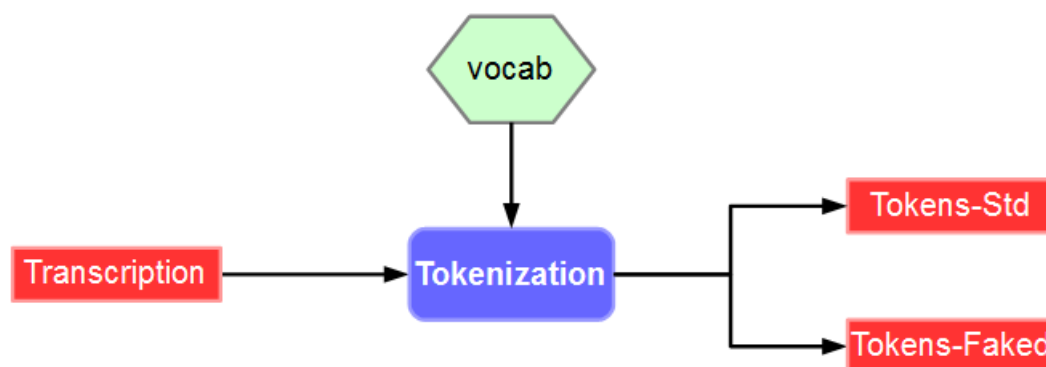


Figure 3.13: Text normalization workflow

To perform the text normalization process, click on the Tokenization activation button, select the language and click on the “Configure...” blue text to fix options.

3.3.4 Perform Tokenization with the CLI

`tokenize.py` is the program to perform Tokenization, i.e. the text normalization of a given file or a raw text.

```
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 parenthesis 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
```

3.4 Phonetization

3.4.1 Overview

Phonetization, also called grapheme-phoneme conversion, is the process of representing sounds with phonetic signs. However, converting from written text into actual sounds, for any language, cause several problems that have their origins in the relative lack of correspondence between the spelling of the lexical items and their sound contents.

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.

Actually, some words can correspond to several entries in the dictionary with various pronunciations. These pronunciation variants are stored in the phonetization result. By convention, whitespace separate words, minus characters separate phones and pipe character separate phonetic variants of a word. For example, the transcription utterance:

- **Transcription:** The flight was 12 hours long.
- **Tokenization:** the flight was twelve hours long
- **Phonetization:** D-@|D-V|D-i: f-l-aI-t w-A-z|w-V-z|w-@-z|w-O:-z t-w-E-l-v aU-3:r-z|aU-r-z l-O:-N

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 (minimum) dictionary is available (Bigi 2013)! If such case occurs during the phonetization process, a **WARNING** indicates it in the Procedure Outcome Report.

Since the phonetization is only based on the use of a pronunciation dictionary, the quality of such a phonetization only depends on this resource.

3.4.2 Adapt Phonetization

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 folder “dict” of the “resources” directory.

SPPAS uses the HTK ASCII format for dictionaries. As example, below 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 indicates the word between brackets; however brackets can also be empty. The last columns are the succession of phones, separated by a whitespace.

SPPAS dictionaries are mainly based on X-SAMPA international standard for phoneme encoding. See the chapter “Resources” of this documentation to know the list of accepted phones for a given language. This list can't be extended nor modified while using SPPAS. However, it can be improved from one version to the other one thanks to user comments (to be sent to the author).

3.4.3 Perform Phonetization with the GUI

The Phonetization process takes as input a file that strictly match the audio file name except for the extension and that “-tokens” is appended. For example, if the audio file name is “oriana1.wav”, the expected input file name is “oriana1-tokens.xra” if .xra is the default extension for annotations. This file must include a **normalized** orthographic transcription. The name of such tier must contains one of the following strings:

1. “tok”
2. “trans”

The first tier that matches one of these requirements is used (this match is case insensitive).

Phonetization produces a file with “-phon” appended to its name, i.e. “oriana1-phon.xra” for the previous example. This file is including only one tier with the resulting phonetization and with name “Phonetization”.

To perform the grapheme-to-phoneme conversion, click on the Phonetization activation button, select the language and click on the “Configure...” blue text to fix options.

3.4.4 Perform Phonetization with the CLI

`phonetize.py` is the program performs Phonetization, i.e. grapheme to phoneme conversion on a given file.

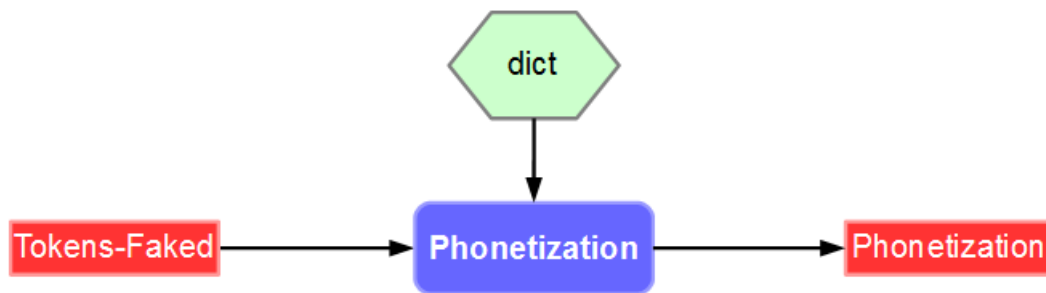


Figure 3.14: Phonetization workflow

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

optional arguments:

```

-r dict      Pronunciation dictionary file name (HTK-ASCII format)
-m map       Pronunciation mapping table
-i file      Input file name
-o file      Output file name
--nunk       Disable unknown word phonetization
-h, --help   Show the help message and exit

```

Examples of use:

```

$ echo "the te totu" |\
./sppas/bin/phonetize.py
-r ./resources/dict/eng.dict
--nunk
$ D-@|D-V|D-i: t-i: UNK
$
$ echo "the te totu" |\
./sppas/bin/phonetize.py -r ./resources/dict/eng.dict
$ D-@|D-V|D-i: t-i: t-u-t-u|t-i-t-u|t-A-t-u|t-@-t-u

```

If we suppose that the previous text was read by a French native speaker, the previous example can be phonetized as:

```

$ echo "the te totu" |\
./sppas/bin/phonetize.py
-r ./resources/dict/eng.dict
-m ./resources/dict/eng-fra.map
$ D-@|z-@|v-@|z-9|D-V|v-9|v-V|D-9|z-V|D-i:|z-i|v-i|D-i|v-i:|z-i:
t-i:|t-i
t-u-t-u|t-i-t-u|t-A-t-u|t-@-t-u

```

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
```

3.5 Alignment

3.5.1 Overview

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.

Speech segmentation was evaluated for French: in average, automatic speech segmentation is 95% of times within 40ms compared to the manual segmentation and was tested on read speech and on conversational speech (Bigi 2014).

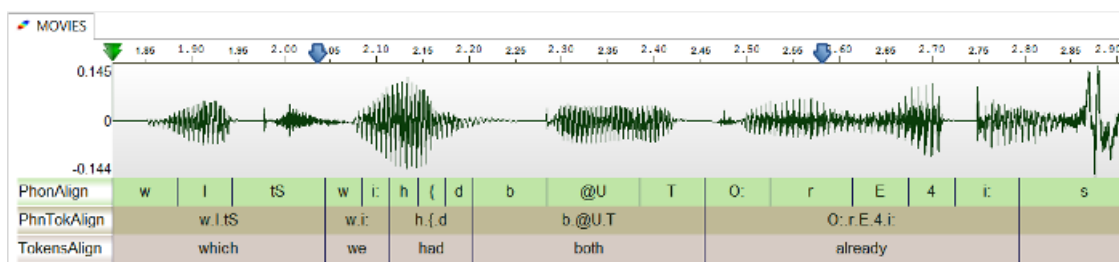


Figure 3.15: SPPAS alignment output example

3.5.2 Adapt Alignment

For Speech segmentation, the better Acoustic Model, the better results. In order to get a better result, any user can append or replace the models included in SPPAS in the “models” folder of the “resources” directory. SPPAS only supports HTK-ASCII acoustic models, trained from 16 bits 16000 Hz WAVE files. The models can be improved if they are re-trained with more data. So, any new data is welcome (send an e-mail to the author).

3.5.3 Perform Alignment with the GUI

The Alignment process takes as input one or two files that strictly match the audio file name except for the extension and that “-phon” is appended for the first one and “-tokens” for the optionnal second one. For example, if the audio file name is “oriana1.wav”, the expected input file name is “oriana1-phon.xra” with phonetization and optionnally “oriana1-tokens.xra” with tokenization, if .xra is the default extension for annotations.

The speech segmentation process provides one file with name “-palign” appended to its name, i.e. “oriana1-palign.xra” for the previous example. This file includes one or two tiers:

- “PhonAlign” is the segmentation at the phone level;
- “TokensAlign” is the segmentation at the word level (if a file with tokenization was found).

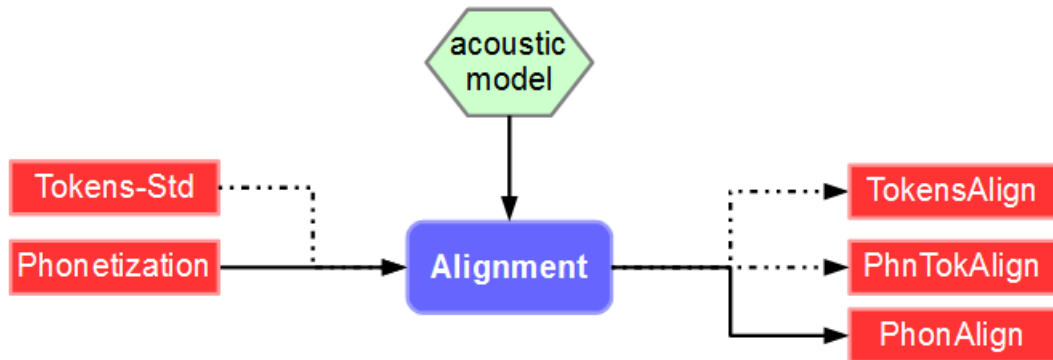


Figure 3.16: Alignment workflow

The following options are available to configure Alignment:

- choose the speech segmentation system. It can be either: julius, hvite or basic
- perform basic alignment if the aligner failed, instead such intervals are empty.
- remove working directory will keep only alignment result: it will remove working files. Working directory includes one wav file per unit and a set of text files per unit.
- create the Activity tier will append another tier with activities as intervals, i.e. speech, silences, laughter, noises...
- create the PhnTokAlign will append another tier with intervals of the phonetization of each word.

To perform speech segmentation, click on the Alignment activation button, select the language and click on the “Configure...” blue text to fix options.

3.5.4 Perform Alignment with the CLI

`alignment.py` is the program to perform automatic speech segmentation of 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      Directory of the acoustic model of the language of the text
-R file      Directory of the acoustic model of the mother language
              of the speaker
-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
--basic      Perform a basic alignment if error with the aligner
--inversp    Add 'sp' at the end of each token and let the aligner
              to decide the relevance
--noclean    Do not remove temporary data
-h, --help   Show the help message and exit

```

Example of use:

```
./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
```

3.6 Syllabification

3.6.1 Overview

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.

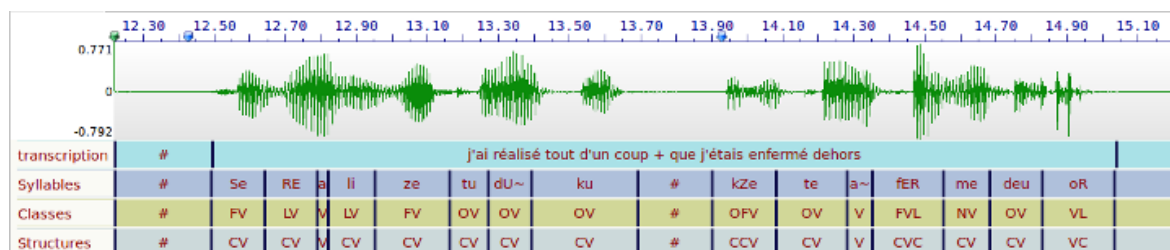


Figure 3.17: Syllabification example

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 (Bigi et al. 2010).

3.6.2 Adapt Syllabification

If the syllabification is not as expected, any user can change the set of rules. The configuration file is located in the folder “syll” of the “resources” directory. The syllable configuration file is a simple ASCII text file that can be edited with Notepad++ (Windows) or TextEdit (MacOS) or any other text editor.

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 VXV 0
- GENRULE VXXV 1
- EXCRULE VFLV 0
- EXCRULE VOLGV 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
```

3.6.3 Perform Syllabification with the GUI

The Syllabification process takes as input a file that strictly match the audio file name except for the extension and that “-palign” is appended. For example, if the audio file name is “oriana1.wav”, the expected input file name is “oriana1-palign.xra” if .xra is the default extension for annotations. This file must include a tier containing the time-aligned phonemes with name “PhonAlign”.

The annotation provides an annotated file with “-salign” appended to its name, i.e. “oriana1-salign.xra” for the previous example. This file is including 3 tiers: Syllables, Classes and Structures.

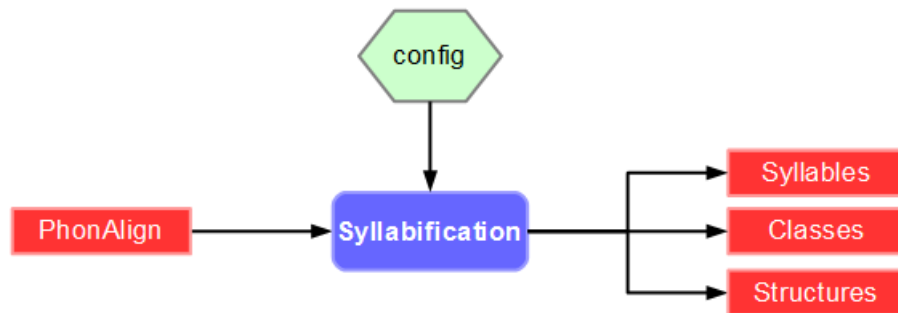


Figure 3.18: Syllabification workflow

Click on the Syllabification activation button, select the language and click on the “Configure...” blue text to fix options.

3.6.4 Perform Syllabification with the CLI

`syllabify.py` is the program performs automatic syllabification of a given file with time-aligned phones.

```
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
```

3.7 Repetitions

3.7.1 Overview

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 (Bigi et al. 2014). 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.

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” folder of the “resources” directory.

3.7.2 Perform Repetitions with the GUI

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.

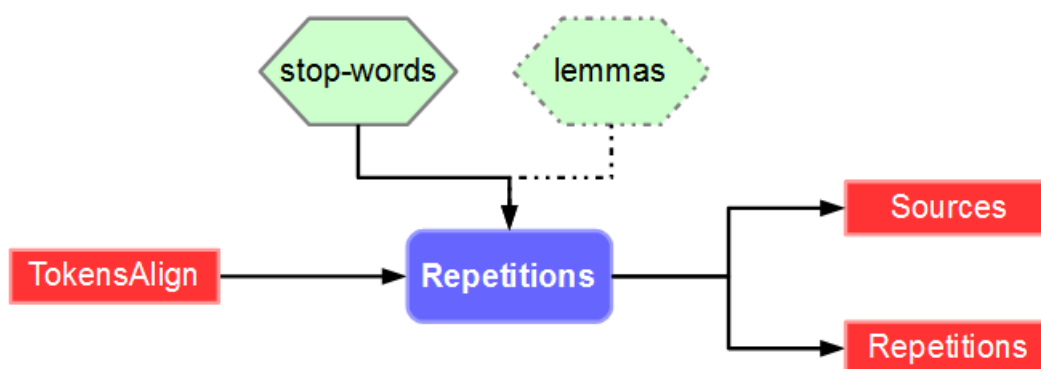


Figure 3.19: Repetition detection workflow

Click on the Self-Repetitions activation button, select the language and click on the “Configure...” blue text to fix options.

3.7.3 Perform Repetitions with the CLI

`repetition.py` is the program to perform automatic detection of self-repetitions or other-repetitions if a second speaker is given.

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

```
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
  
```

3.8 Momel and INTSINT

3.8.1 Momel (modelling melody)

Momel is an algorithm for the automatic modelling of fundamental frequency (F0) curves using a technique called asymmetric modal quadratic 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. It must be **sampled at 10 ms**. Two extensions are supported:

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

The following 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’

3.8.2 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.

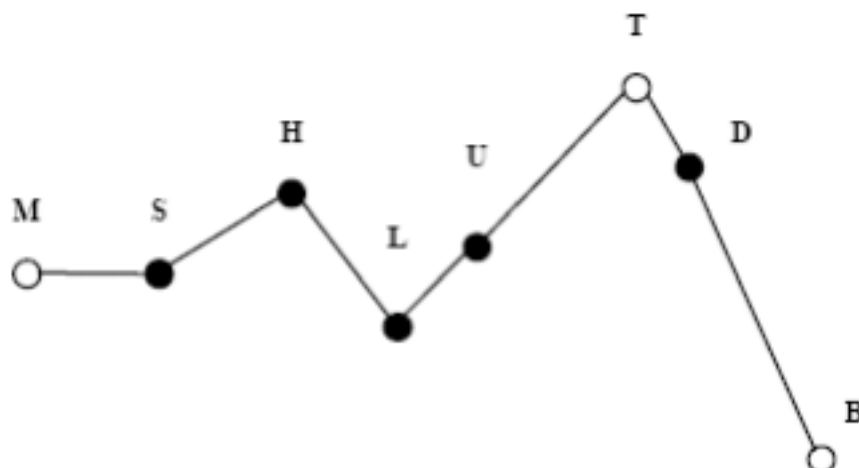
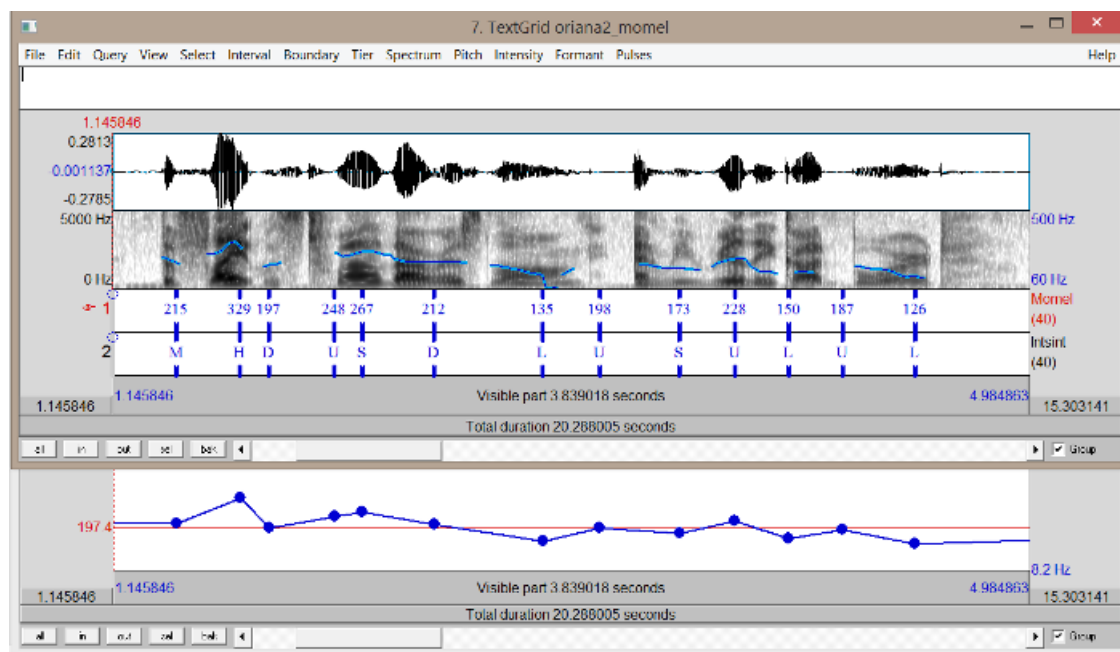


Figure 3.20: INTSINT example

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.

3.8.3 Perform Momel and INTSINT with the GUI

Click on the Momel activation button, select the language and click on the “Configure...” blue text to fix options. And click on the INTSINT activation button.

3.8.4 Perform Momel and INTSINT with the CLI

`momel-intsint.py` is the program perform both 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
```


Resources for Automatic Annotations

4.1 What are SPPAS resources and where they come from?

4.1.1 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 some available languages in SPPAS resources:

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

- Mandarin Chinese: cmn
- Southern Min (or Min Nan): nan
- Cantonese: yue
- Polish: pol
- Catalan: cat
- Portuguese: por

SPPAS can deal with a new language by simply adding the language resources to the appropriate sub-directories. Of course, file formats must corresponds 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.

4.1.2 About the phonemes

Most of the dictionaries are using the international standard X-SAMPA to represent phonemes. See <https://en.wikipedia.org/wiki/X-SAMPA> for details. The list of all phonemes for each language is available in this documentation, in the section related to a given language.

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

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

4.1.3 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.

4.2 French Resources

List of phonemes

SPPAS	IPA	Examples
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
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~	œ̃	un parfum
o~	õ	son nom
b	b	beau
d	d	doux
f	f	fête pharmacie
g	g	gain guerre second
k	k	cabas psychologie quatre kelvin
l	l	le loup elle
m	m	mou femme
n	n	nous bonne
p	p	passé
R	ʁ	roue rhume

SPPAS	IPA	Examples
s	s	sa hausse ce garçon option scie
S	ʃ	choux schème shampoing
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
fp		euh (filled pause)
@@		laughter item
gb		(garbage: for noises)
dummy		(dummy: for untranscribed speech)

Pronunciation Dictionary

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

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 manually.

Acoustic Model

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

Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International Public License

The French acoustic model was created by Brigitte Bigi from various corpora mainly recorded at Laboratoire Parole et Langage. For example, it includes:

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

Syllabification configuration

The syllabification configuration file corresponds to the rules defined in the paper (Bigi et al. 2010).

4.3 Italian resources

List of phonemes

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	j	gnocco ogni
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

SPPAS	- IPA -	Examples
@@		laughter item
fp		eh, ah (filled pause)
gb		(garbage: for noises)
dummy		(dummy: for untranscribed speech)

Pronunciation dictionary

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

GNU Public License

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

Acoustic Model

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

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), and updated during the Evalita 2014 evaluation campaign.

Syllabification configuration

The syllabification configuration file corresponds to the rules defined in the paper (Bigi and Petrone, 2014).

4.4 Spanish resources

List of phonemes

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

SPPAS	- IPA -	Examples
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	ʁ	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
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

Pronunciation Dictionary

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

Acoustic Model

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

Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International Public License

The acoustic model was trained from Glissando corpus. We address special thanks to Juan-Maria Garrido for giving us access to this corpus.

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.

4.5 Catalan resources

List of phonemes

SPPAS	- IPA -	Examples
@	ə	sec
D	ð	
E	ε	sec
J	ɲ	
L	ʎ	
N	ɳ	
O	ɔ	
S	ʃ	
T	θ	
U	ʊ	
Z	ʒ	
a	a	sac
b	b	
b	β	
d	d	
d	ð	
dZ	dʒ	
e	e	séc
f	f	
g	g	
g	ɟ	
i	ɪ	
i	i	sic
j	j	

SPPAS	- IPA -	Examples
k	k	
l	l	
m	m	
n	n	
o	o	sóc
p	p	
rr	r	
r	R	
s	s	si
t	t	
tS	tʃ	
u	u	suc
v	v	
w	w	
x	x	
x	ɣ	
z	z	

Pronunciation dictionary

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 (thank you Eva!). New entries were then added from observed pronunciations in Glissando corpus.

Acoustic Model

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

Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International Public License

The acoustic model was trained from Glissando corpus. We address special thanks to Juan-Maria Garrido for giving us access to this corpus.

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.

4.6 English

List of phonemes

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
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

SPPAS	- IPA -	Examples
@	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	r	Adam atom coda

Pronunciation 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.

Brigitte Bigi converted the original CMUdict into X-SAMPA.

Acoustic Model

The acoustic model distributed in SPPAS resources were downloaded (in 2014) from the VoxForge project at <http://www.voxforge.org/>. It was then converted to X-SAMPA by Brigitte Bigi.

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

4.7 Mandarin Chinese

List of phonemes

SPPAS	IPA	Examples
N	ŋ	
S	ʃ	
a	a	
e	e	
f	f	?
i	i	? ?
i_d		??
k	k	?
k_h		
l	l	?
m	m	
n	n	
o	o	?
p	p	?
p_h		
s	s	?
ts	ts	?
ts_h		? ?
S	ʃ	?
ss		?
t	t	? ?
tss		? ?
u	u	?
x	x	? ?
y	y	? ?
ts_hs		? ?
t_h		?? ??
@‘		
i‘		???
s‘		? ?
ts_h‘		? ?
ts‘		? ?
z‘		
@@		laughter item

SPPAS	IPA	Examples
gb		(garbage: for noises)
dummy		(dummy: for untranscribed speech)

Pronunciation dictionary

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

GNU Public License

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

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! Because more data implies a better acoustic model then better alignments...

4.8 Southern Min (or Min Nan) resources

List of phonemes

SPPAS	- IPA -	Examples
@	ə	
N	ŋ	
O	ɔ	
O~	õ	
S	ʃ	
a	a	
a~	ã	
b	b	

SPPAS	- IPA -	Examples
d	d	
dz	dz	
e	e	
e~		
f	f	
g	g	
h	h	
i	i	
i~		
k	k	
k_h		
l	l	
m	m	
n	n	
o	o	
o~		
p	p	
p_h		
s	s	
ss		
t	t	
t_h		
ts	ts	
ts_h		
u	u	
u~		
v	v	
w	w	
x	x	
y	y	
z	z	
@@		laughter item
gb		(garbage: for noises)
dummy		(dummy: for untranscribed speech)

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

Pronunciation Dictionary

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

GNU Public License

Acoustic Model

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

Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International Public License

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

4.9 Cantonese resources

List of phonemes

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

p

p_h

m

f

t

t_h

n

l

k

k_h

N

h

k_w

k_h_w

w

SPPAS	- IPA -	Examples
<hr/>		
	ts	
	ts_h	
	s	
	j	
	a:	
	6	
	E:	
	e	
	i:	
	I	
	O:	
	o	
	u:	
	U	
	9:	
	y:	
	8	
	@	
	S	
	tS	
	tS_h	

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

4.10 Polish Resources

List of phonemes

	SPPAS
	a
	b
	c
	x
	d
	dz
	dZ
	E
	E~
	f
	g
	x
	i
	j
	n
	k
	l
	m
	n
	N
	O
	O

SPPAS	
	o~
	p
	Q
	r
	r
	s
s ɣ S ʃ t t v v w w x x y y z z	
	u
	z'
	Z

Pronunciation Dictionary

GNU Public License

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 <http://katarzyna.klessa.pl/>.

Acoustic Model

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

Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International Public License

The acoustic model was created by Brigitte Bigi. We address special thanks to Katarzyna Klessa for giving us access to a corpus.

4.11 Portuguese Resources

List of phonemes

SPPAS	- IPA -	Examples
a	a	
a~	ã	
b	b	
d	d	
e	e	

SPPAS	- IPA -	Examples
E	ɛ	
f	f	
g	g	
i	i	
I		
j	j	
J	ɲ	
k	k	
l	l	
L	ʎ	
m	m	
n	n	
N	ŋ	
o	o	
O	ɔ	
p	p	
r	r	
R	ʀ	
s	s	
S	ʃ	
t	t	
u	u	
u~		
U	ʊ	
v	v	
w	w	
x	x	
y	y	
z	z	
Z	ʒ	
@@		laughter item
gb		(garbage: for noises)
dummy		(dummy: for untranscribed speech)

Pronunciation Dictionary

GNU Public License

The Portuguese 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.

Acoustic Model

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

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, mainly the Spanish and French models.

Portuguese data are welcome! Because data implies a better acoustic model than better alignments...

Analyses

5.1 Introduction

The analysis tools are useful for the analyses 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 analysis tool, select file(s) in the file explorer of the main frame, then click on the button of the tool. It will open the frame, and add the file(s) in the file manager of the tool.

Six tools 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. `AudioRoamer` allows to play and manage audio files: extract a channel, see clipping rates, change framerate, etc.
3. `IPUseriber` is useful to perform manual orthographic transcription.
4. `Vizualizer` displays audio 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 such tools share the same frame with:

- a menu (left),
- a toolbar (top),
- a list of files,
- a notebook to open files in tabs.

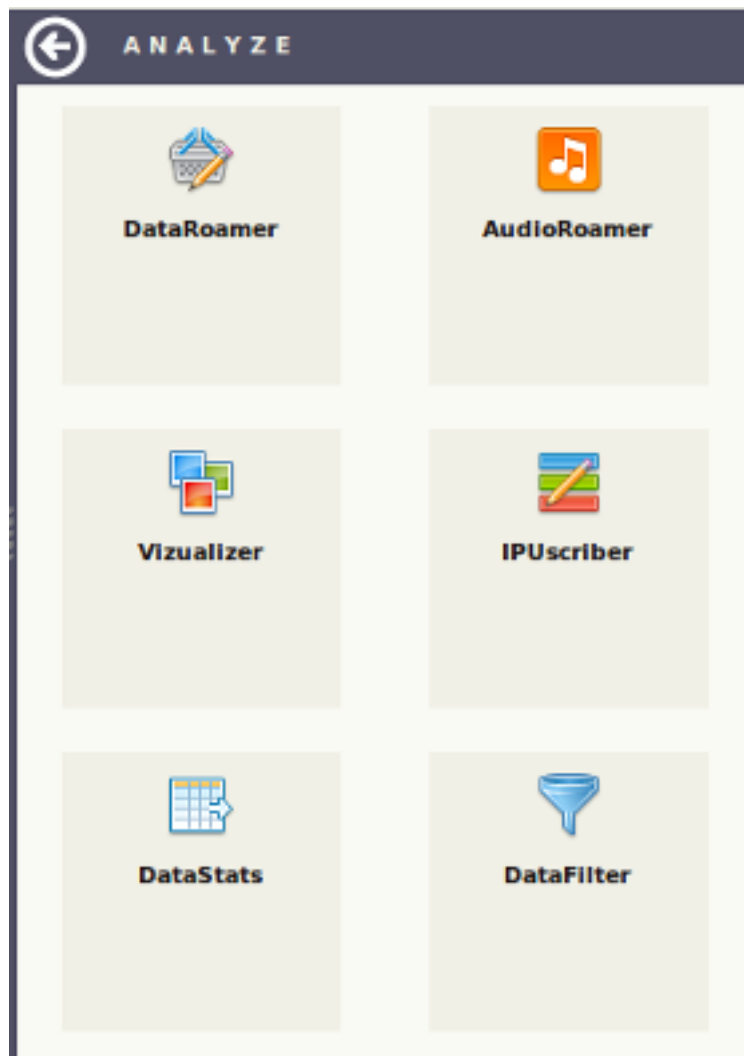


Figure 5.1: The analysis tools

5.2 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.

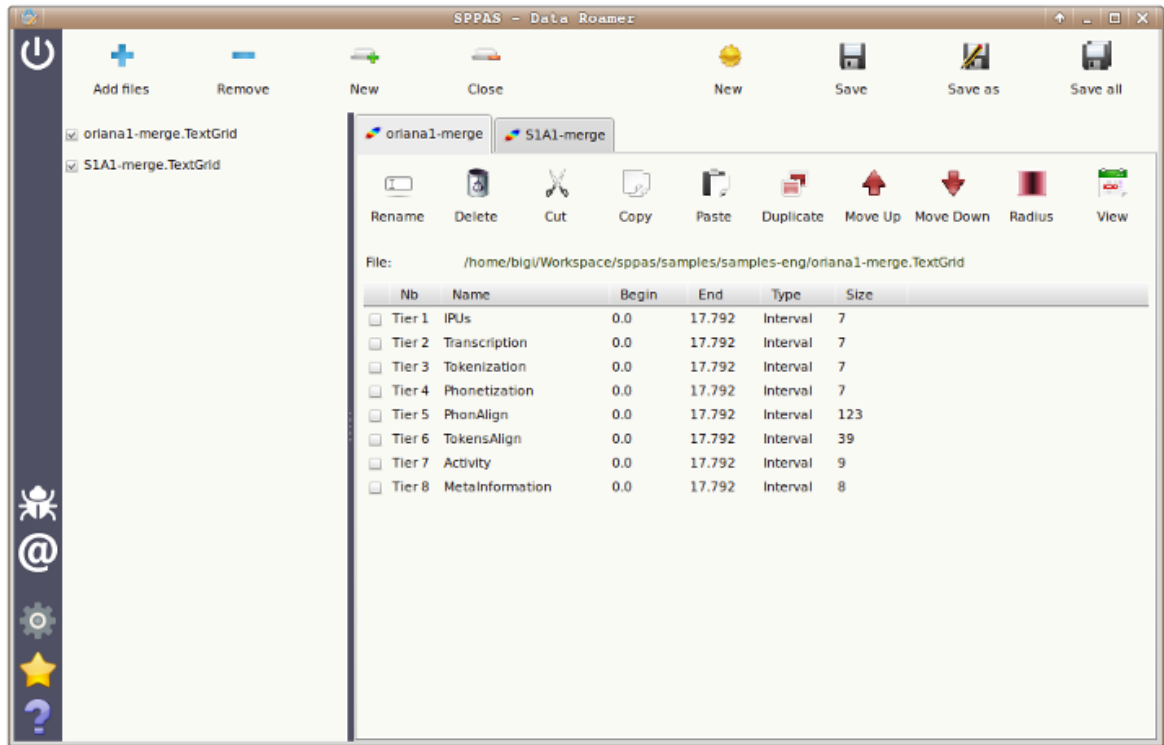


Figure 5.2: DataRoamer: explore annotated files

5.3 AudioRoamer

AudioRoamer allows to play audio files, to display general information about a digitalized audio-PCM file and to manipulate the file.

Pulse-code modulation (PCM) is a method used to digitally represent sampled analog signals. In a PCM stream, the amplitude of the analog signal is sampled regularly at uniform intervals. A PCM stream has two basic properties that determine the stream's fidelity to the original analog signal:

1. the sampling rate, which is the number of times per second that samples are taken;
2. the bit depth, which determines the number of possible digital values that can be used to represent each sample.

Common sampling frequencies are 48 kHz as used with DVD format videos, or 44.1 kHz as used in Compact discs. Common sample depths are 8, 16 or 24 bits per sample. 16 is the most frequent one: this allow values to range from -32768 to 32767.

Support for multichannel audio depends on file format and relies on interweaving or synchronization of streams.

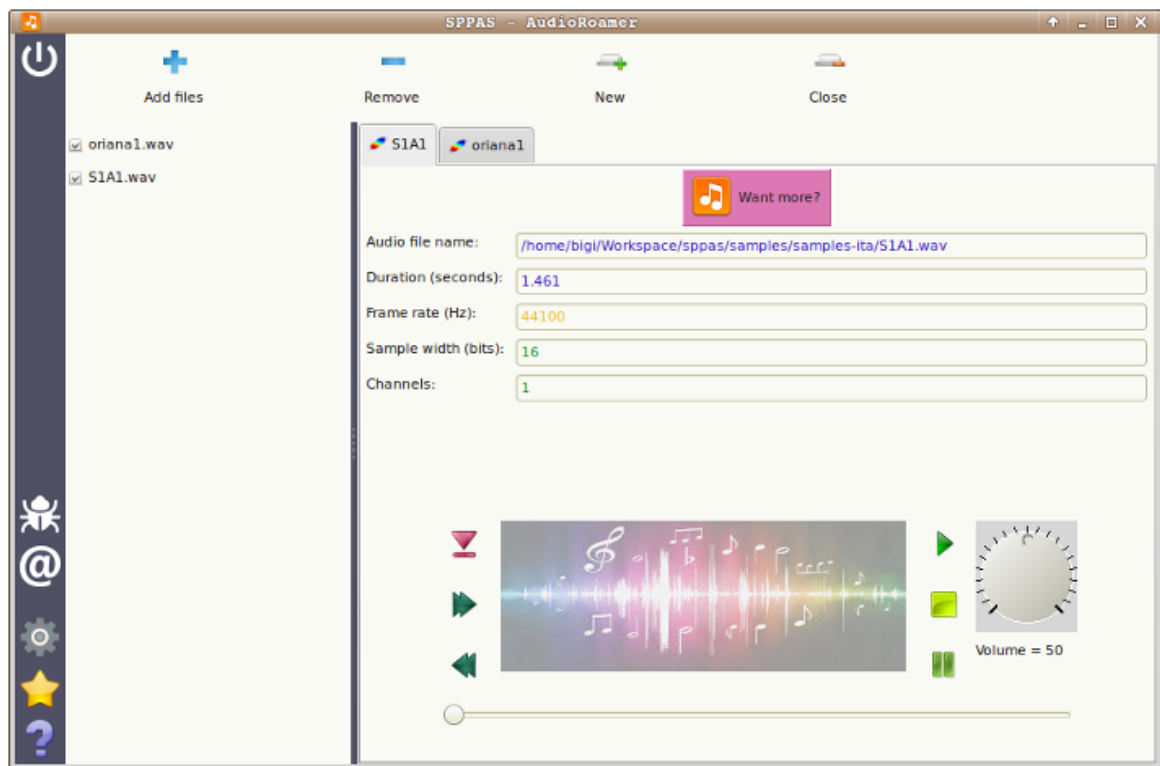


Figure 5.3: AudioRoamer: play and manage audio files

When an audio file is checked in the list of files, a new page is opened in the notebook. The main information about the audio file are then displayed in the panel at the middle and a player is displayed at bottom. The audio file is not loaded in memory, so event very long audio files can be displayed. At the top of the page, a button “Want more?” can be clicked: the audio file will then be fully read and analyzed, then a new window will be opened to display detailed information. It will also allow to change some properties of the audio file, extract a channel, etc.

5.3.1 Properties of the audio file

The main properties of any audio file is displayed. SPPAS can open audio files of type:

1. Waveform Audio File Format (WAVE), a Microsoft and IBM audio file format standard;
2. Audio Interchange File Format (AIFF), an audio file format standard developped by Apple Inc;
3. Au file format, a simple audio file format introduced by Sun Microsystems.

The following information are extracted or deducted of the header of the audio file:

- the duration given in seconds with a maximum of 3 digits;
- the frame rate given in Hz;
- the sample width in bits (one of 8, 16, 24 or 32);
- the number of channels.

A color code is used to indicate if the file is compatible with SPPAS automatic annotations:

- Green color indicates that the value is perfectly matching the expectations;
- Orange color indicates that the value is not the expected one but automatic annotations will be able to convert it to the right one;
- Red color indicates that none of the automatic annotations will work with the given value. The file must be modified.

5.3.2 Playing audio files

The following shortcuts can be used:

- TAB: Play
- ESC: Stop
- F6: Rewind
- F7: Pause
- F8: Next

5.3.3 Want more?

AudioRoamer allows to display a large set of information for each channel. For a large file, it can take a while to estimate such information... For example, an mono-audio file of 243 seconds (21.5Mb) is loaded in 35 seconds. **So be patient! It's worth it!**

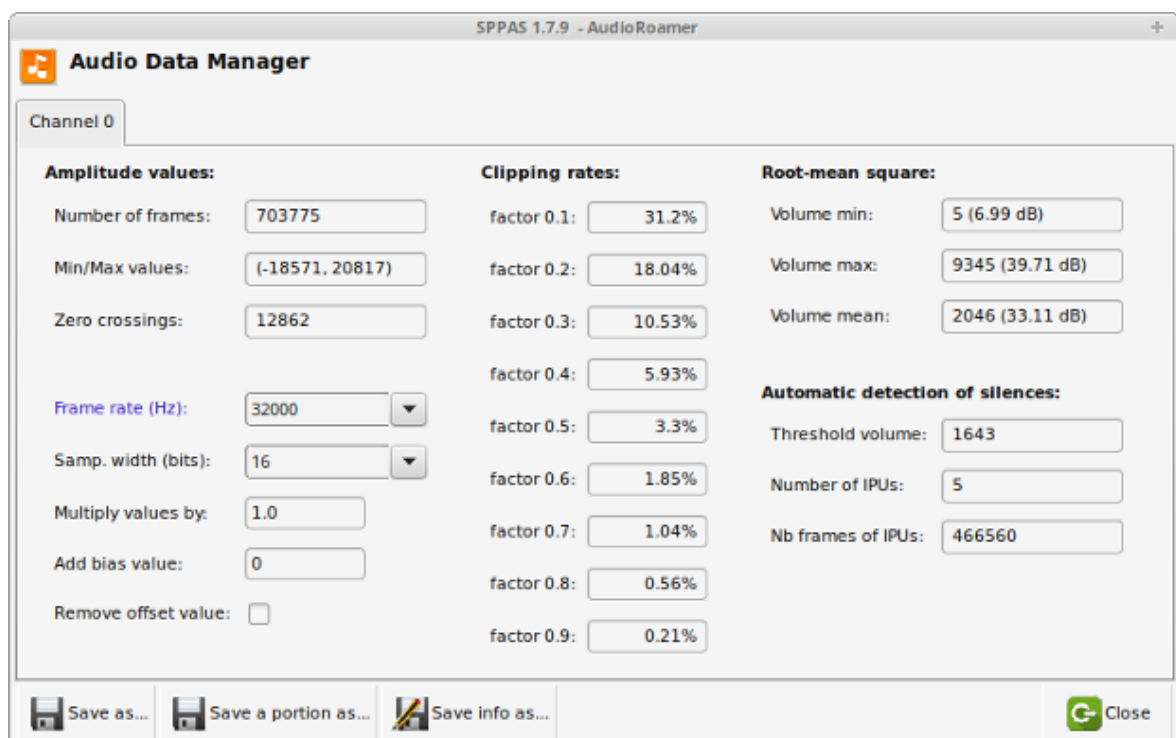


Figure 5.4: AudioRoamer if you want more

At the bottom of the window, it is possible to click on buttons to:

1. Save the channel currently displayed;

2. Save a portion (from...to...) of the channel;
3. Save the displayed information in a text file;
4. Close the window.

There are 3 main columns of information, related to amplitude, clipping and volume.

Amplitude

The amplitude is a variable characterizing the sinusoidal oscillation of the digital-audio recording. It gives the deflection of a physical quantity from its neutral position (zero point) up to a positive or negative value. With sound waves, it is the extent to which air particles are displaced, and this amplitude of sound or sound amplitude is experienced as the loudness of sound. The loudness perception of a sound is determined by the amplitude of the sound waves – the higher the amplitude, the louder the sound or the noise.

The first column of *AudioRoamer* Data Manager gives amplitude values of each channel of the PCM file:

- Number of frames: the number of samples, i.e. the number of amplitude values;
- Min/Max values: minimum and maximum amplitude values in the channel;
- Zero crossings

The 5 information below can be modified and modifications can be saved separately for each channel:

- Frame rate: the sampling frequency observed in the channel included in a list of possible values. If it is modified, the color is changed.
- Samp. width: the bit depth observed in the channel included in a list of possible values. If it is modified, the color is changed.
- Multiply values by: all samples in the original channel are multiplied by the floating-point value factor. Sample values are truncated in case of overflow.
- Add bias value: a bias value is added to each sample. Sample values wrap around in case of overflow.
- Remove offset value: the average over all values is deduced to each sample. Sample values wrap around in case of overflow.

Clipping

The second column displays the clipping rates of the sample values given a factor. It will consider that all values outside the interval are clipped. The clipping rate is given as a percentage related to the total number of values.

Volume

RMS: Volume is estimated from root-mean-square (RMS) of the samples, i.e. $\sqrt{\sum(S_i^2)/n}$. This is a measure of the power in an audio signal. Between parenthesis, the volume in dB is estimated as: $10 \cdot \log_{10}(\text{rms}/\text{ref})$, where ref is a reference factor (energy quantity) $\text{ref} = 1 \equiv 0 \text{ dB}$ (power).

Illustration: The level of -3 dB is equivalent to 50% (factor = 0.5) and the level of -6 dB is equivalent to 25% (factor = $1/4 = 0.25$) of the initial power. For example, a RMS value of 350 gives a power value equal to 25.45 dB compared to a RMS value of 700 that gives a power value of 28.45 dB.

Automatic detection of silences: Finally, the result of an automatic detection of silences is given. Of course, this information is given for information only. It is estimated with default parameters which should be adapted.

5.4 IPUScriber

IPUScriber is useful to perform manual orthographic transcription.

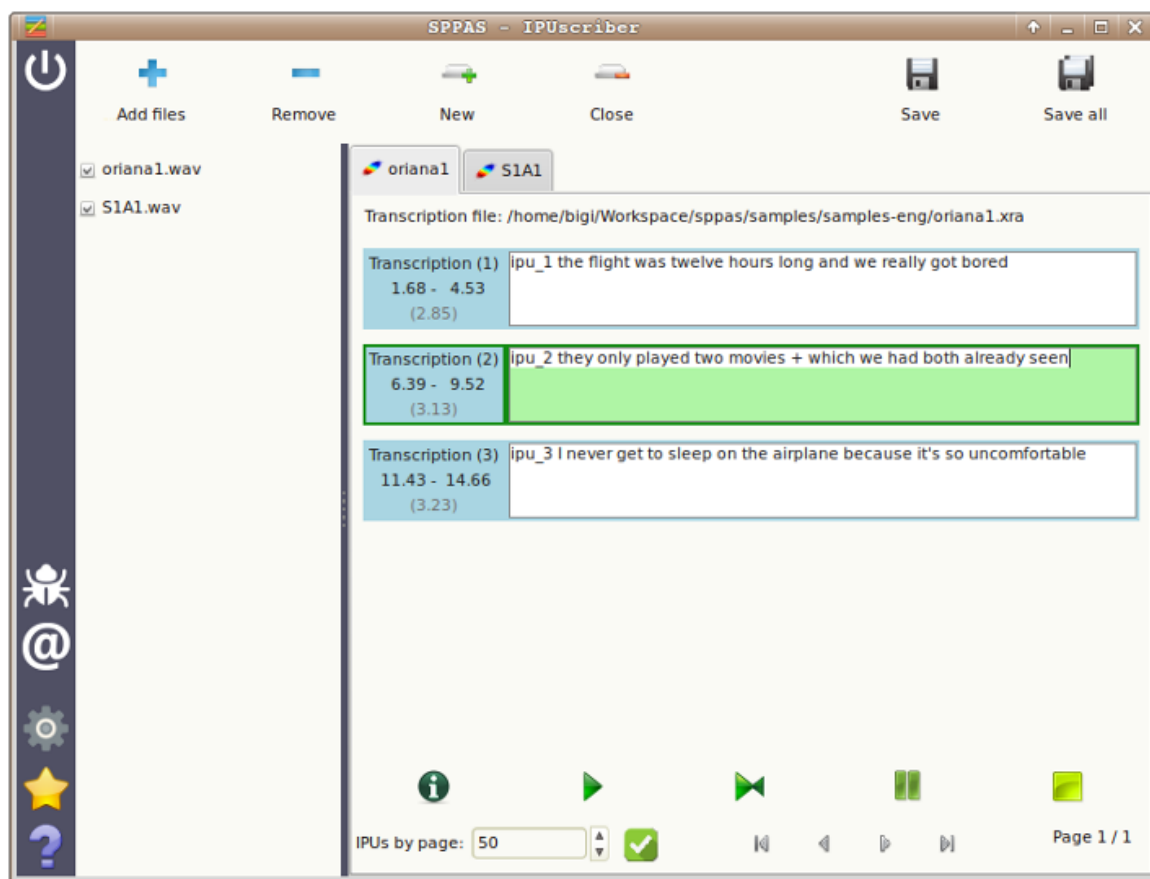


Figure 5.5: IPUScriber: for manual orthographic transcription

To transcribe an IPU, click on the IPU box, play sound and write the corresponding text: refer to the transcription convention of this document. To manage sound, use green buttons just at the bottom of the IPUs list (from left to right):

- information about the file
- play sound
- auto-play sound
- pause
- stop

The following keyboard shortcuts can also be used:

- TAB: Play

- ESC: Stop
- F6: Rewind
- F7: Pause
- F8: Next

5.5 Vizualizer

Vizualizer displays speech files and annotated files, and is very useful to take a nice screenshot.

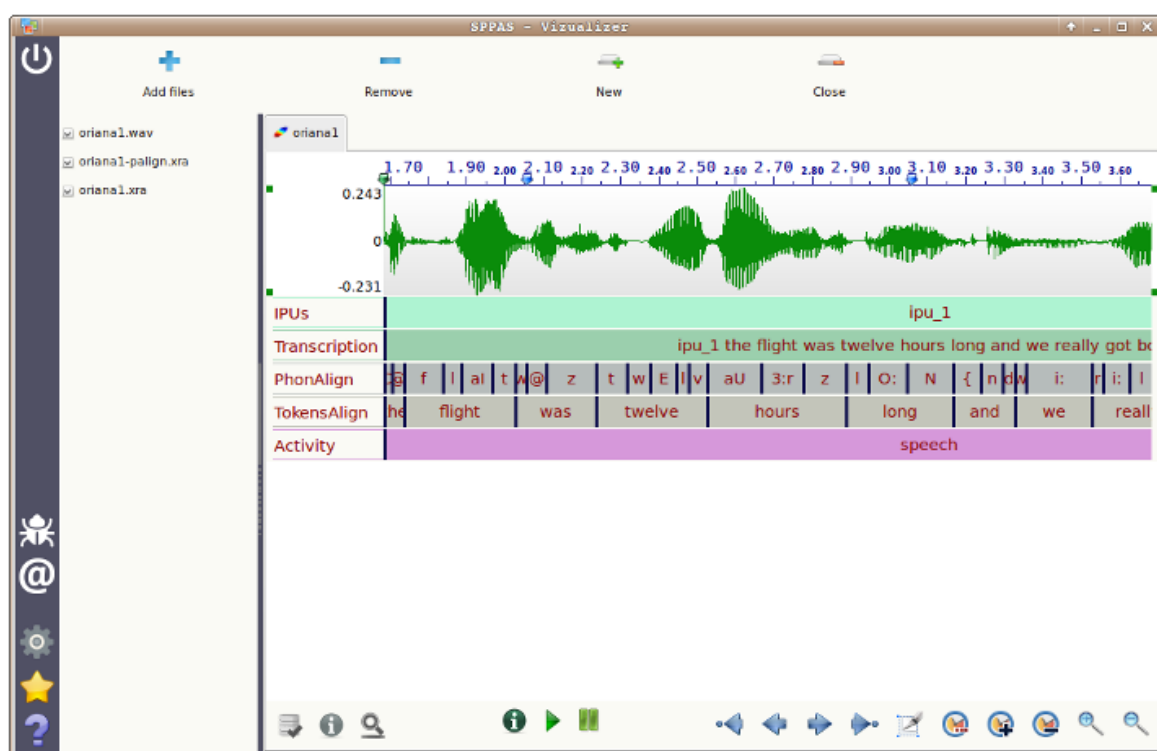


Figure 5.6: Vizualizer: displays audio files and annotated files

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

5.6 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.

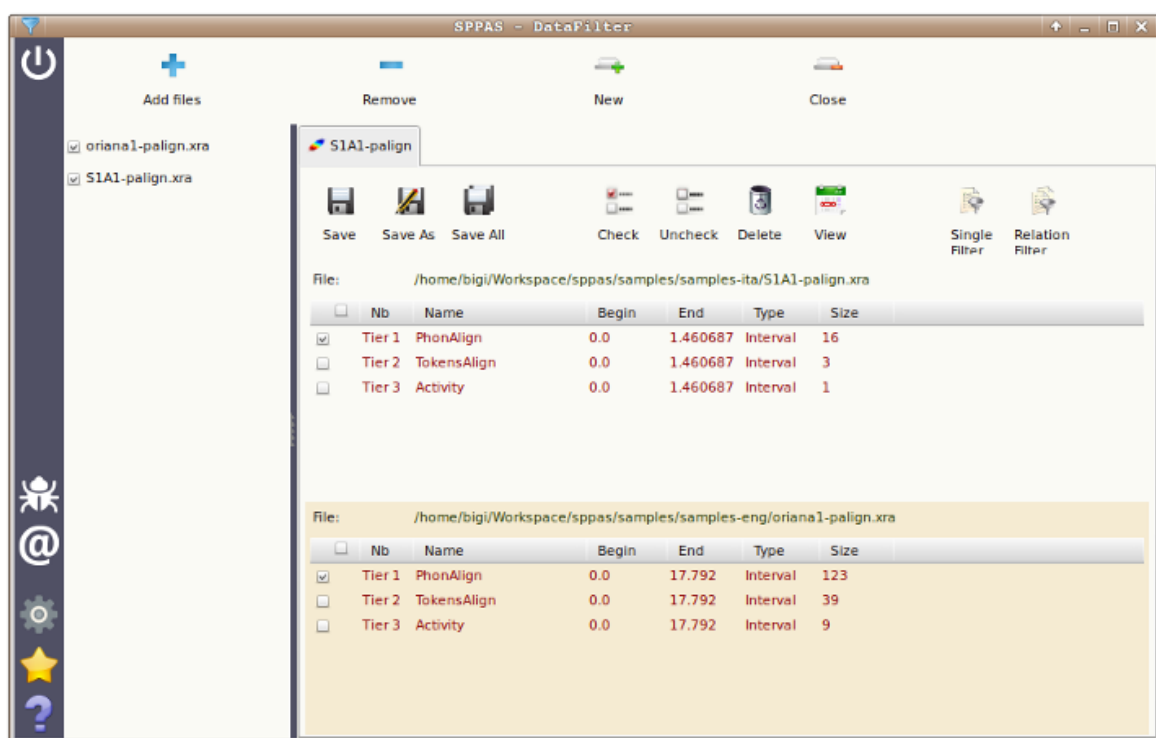


Figure 5.7: DataFilter: select annotations

5.6.1 Filtering annotations of a tier: SingleFilter

Pattern selection is an important part to extract data of a corpus and is obviously an 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”.

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 start and/or end at specific time values in a tier.

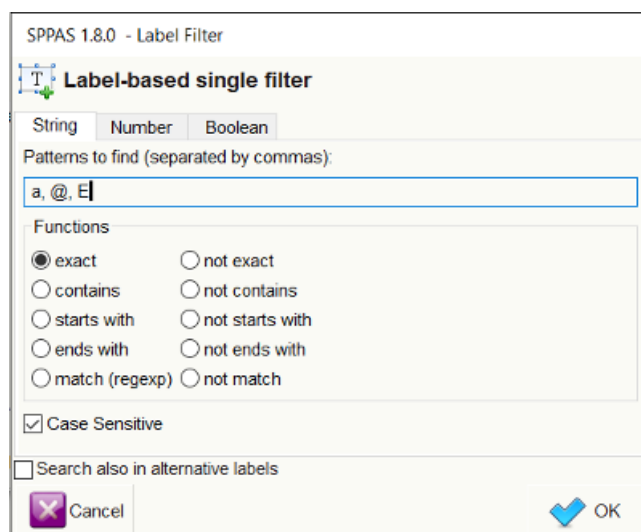


Figure 5.8: Frame to create a filter on annotation labels. In that case, filtering annotations that exactly match either a, @ or E

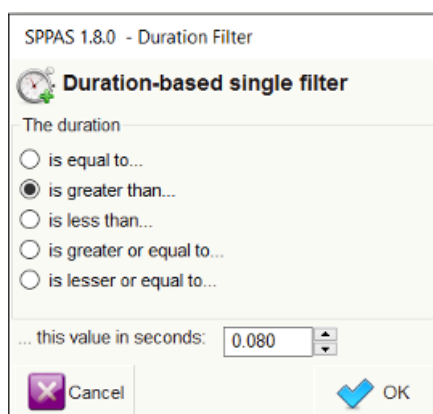


Figure 5.9: Frame to create a filter on annotation durations. In that case, filtering annotations that are during more that 80 ms

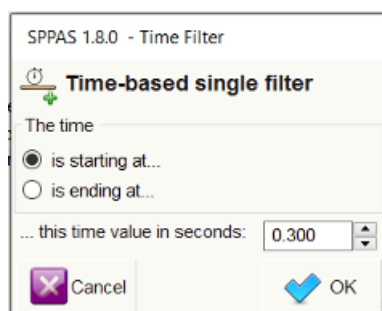


Figure 5.10: Frame to create a filter on annotation time values. In that case, filtering annotations that are starting after the 5th minute.

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.

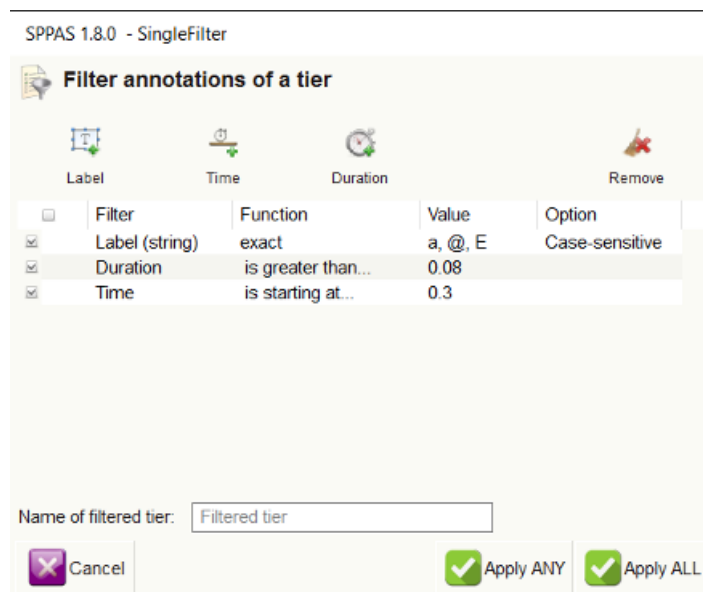


Figure 5.11: DataFilter: SingleFilter frame

5.6.2 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.

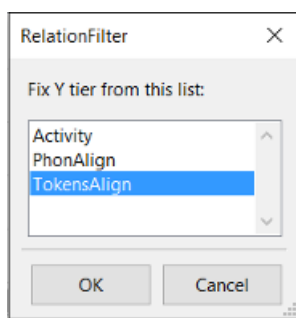


Figure 5.12: Fix time-relation tier name

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

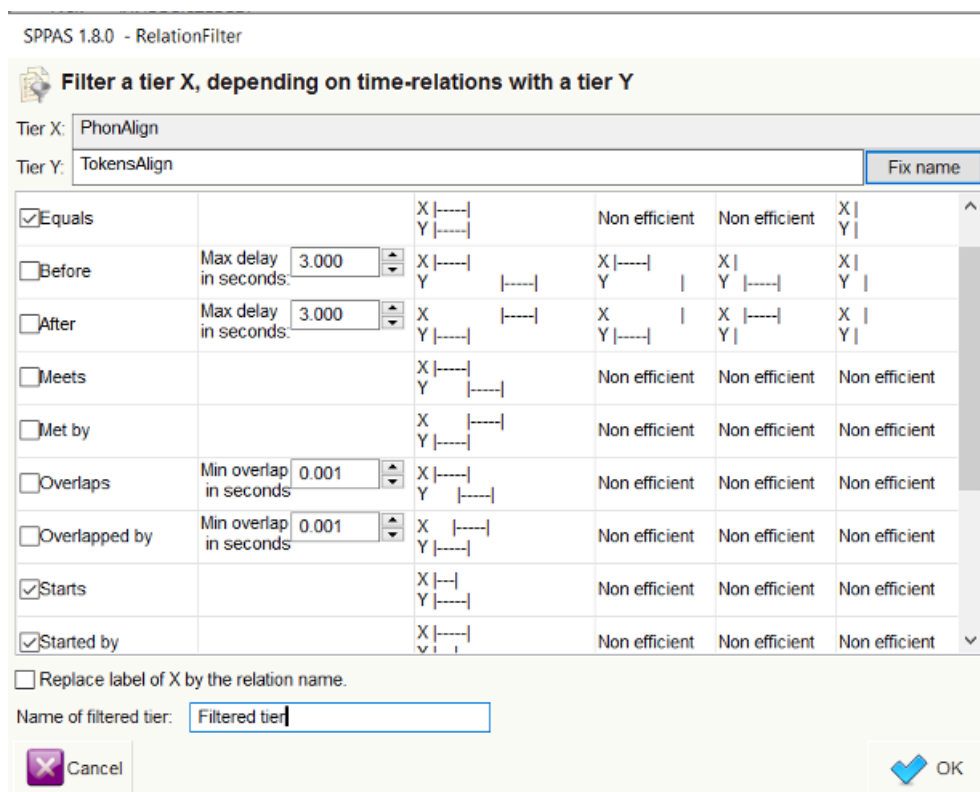


Figure 5.13: DataFilter: RelationFilter frame

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).

5.7 Statistics

Statistics allows to get descriptives statistics about a set of selected tiers 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. It also allows to estimate a user agreement rate (Kappa as a first stage).

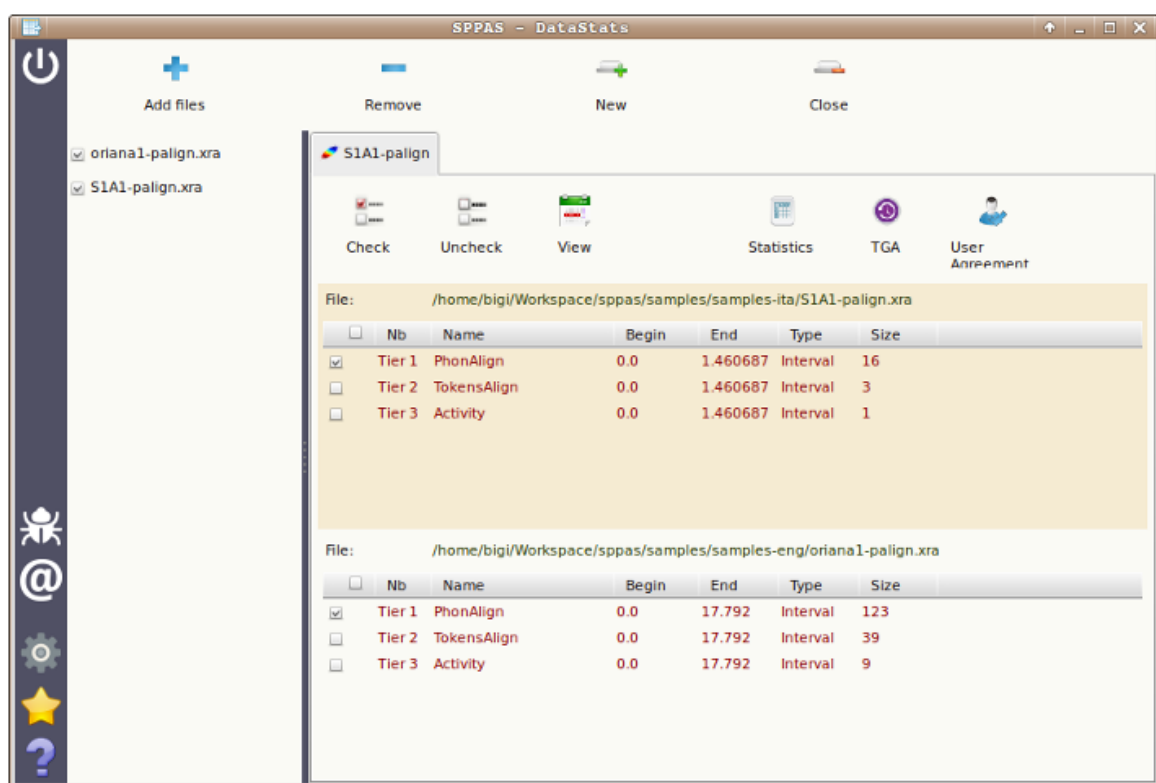


Figure 5.14: Statistics: descriptive statistics and TGA

5.7.1 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 series 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).

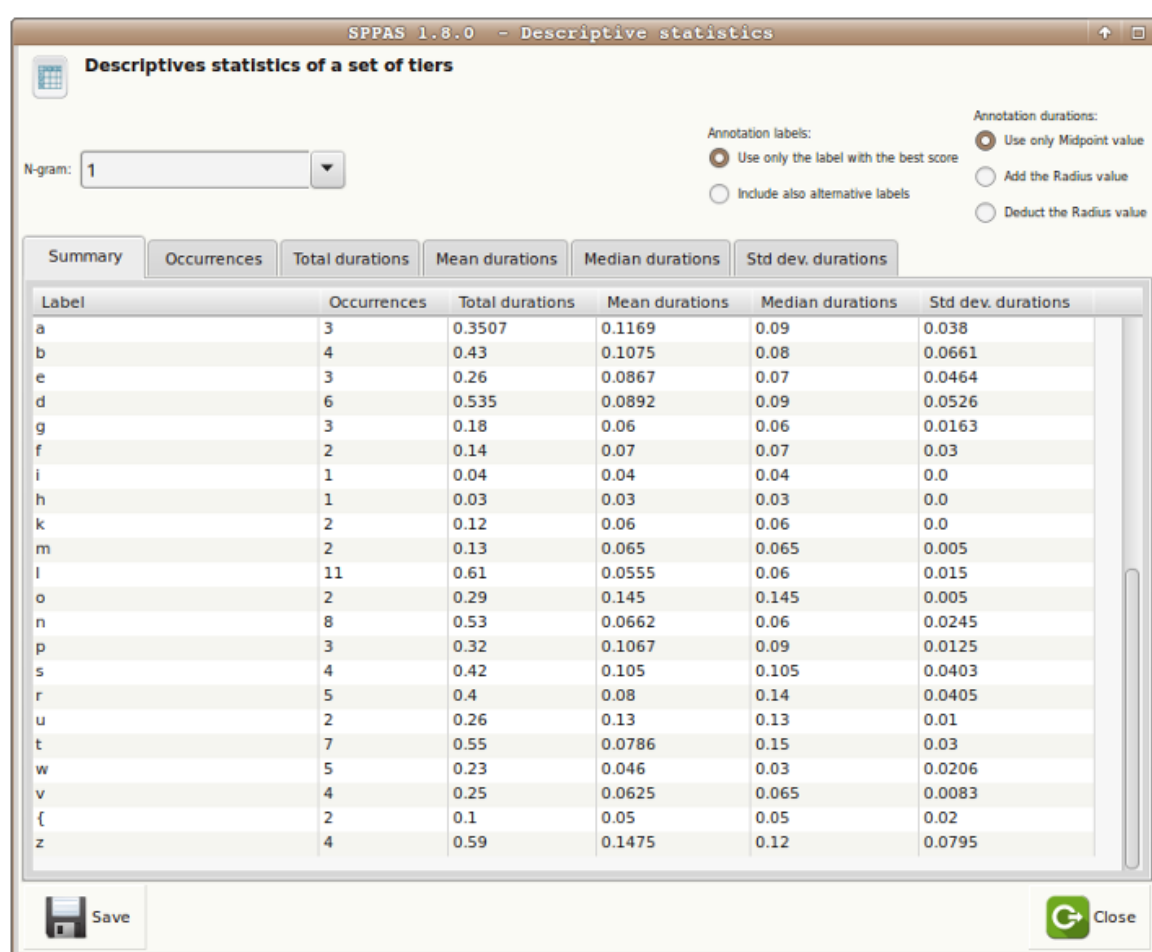


Figure 5.15: Descriptive statistics

5.7.2 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 estimate 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

5.7.3 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.

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 a 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!'
```

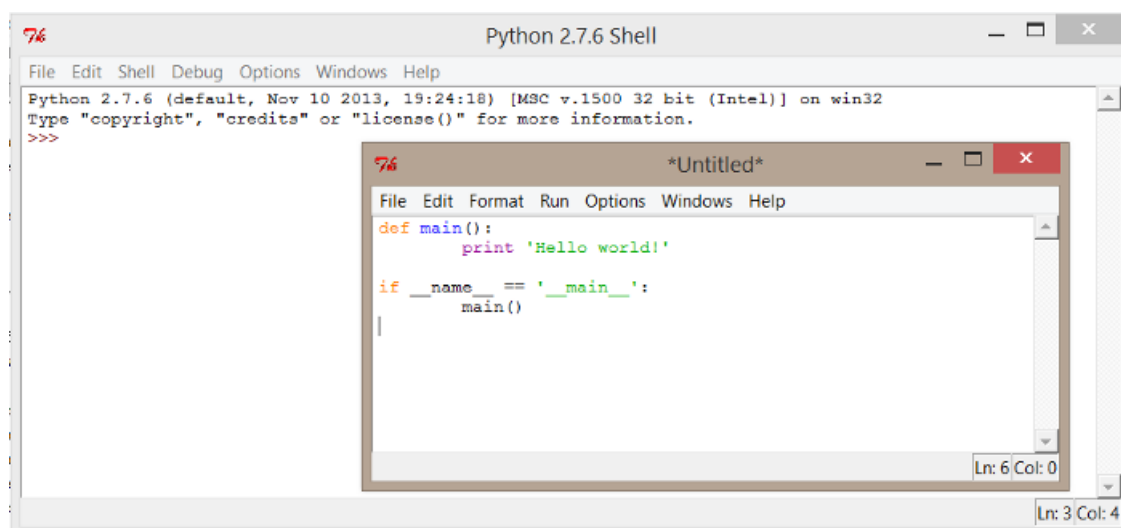


Figure 6.2: Hello world! in a Python script

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:

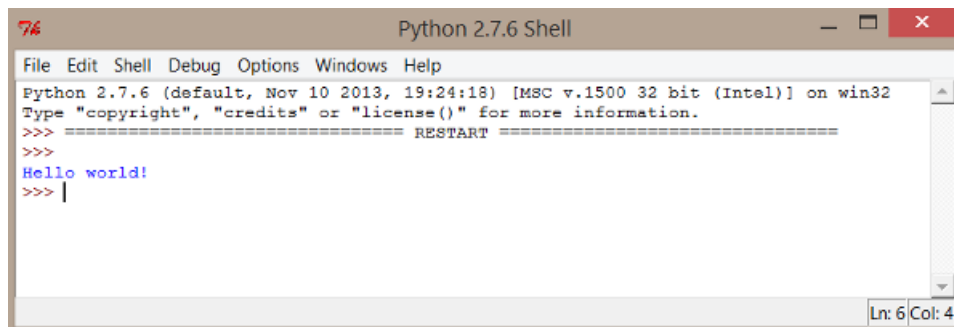


Figure 6.3: Output of the first script

```

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()`.

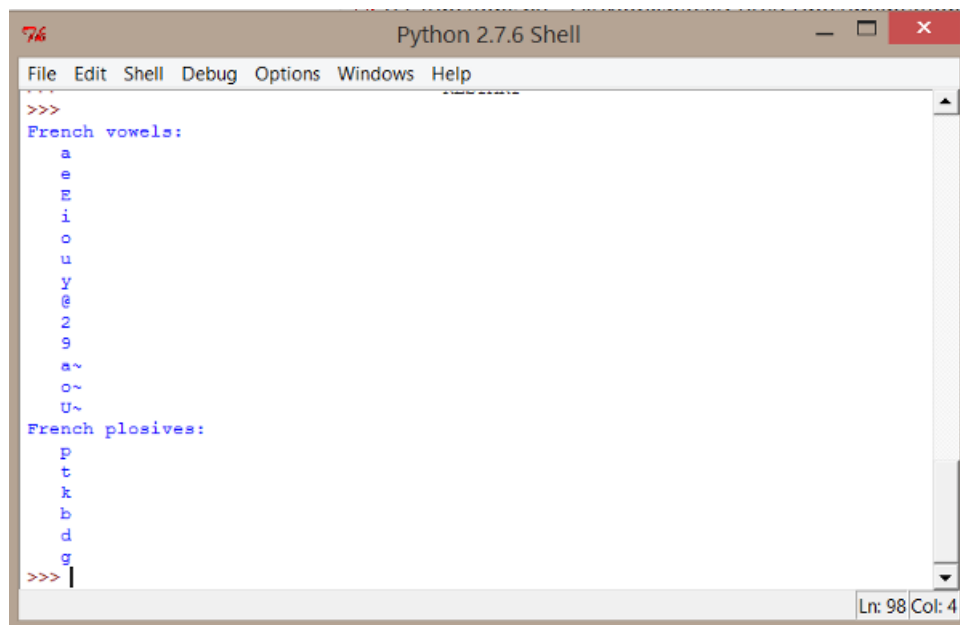
```

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

```

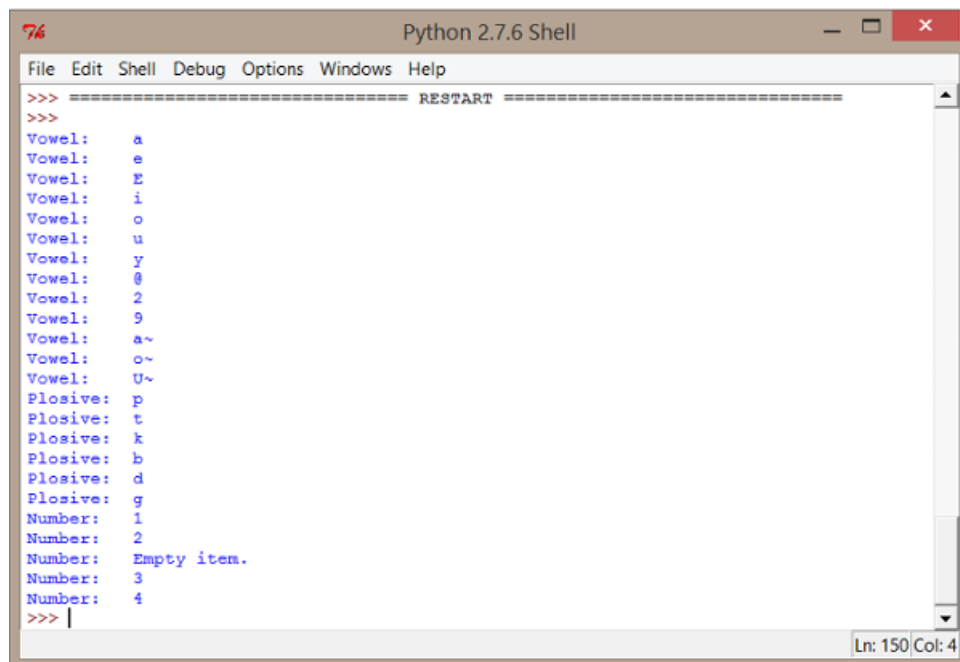


Figure 6.5: Expected output of the 3rd script

```

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://www.sppas.org/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).

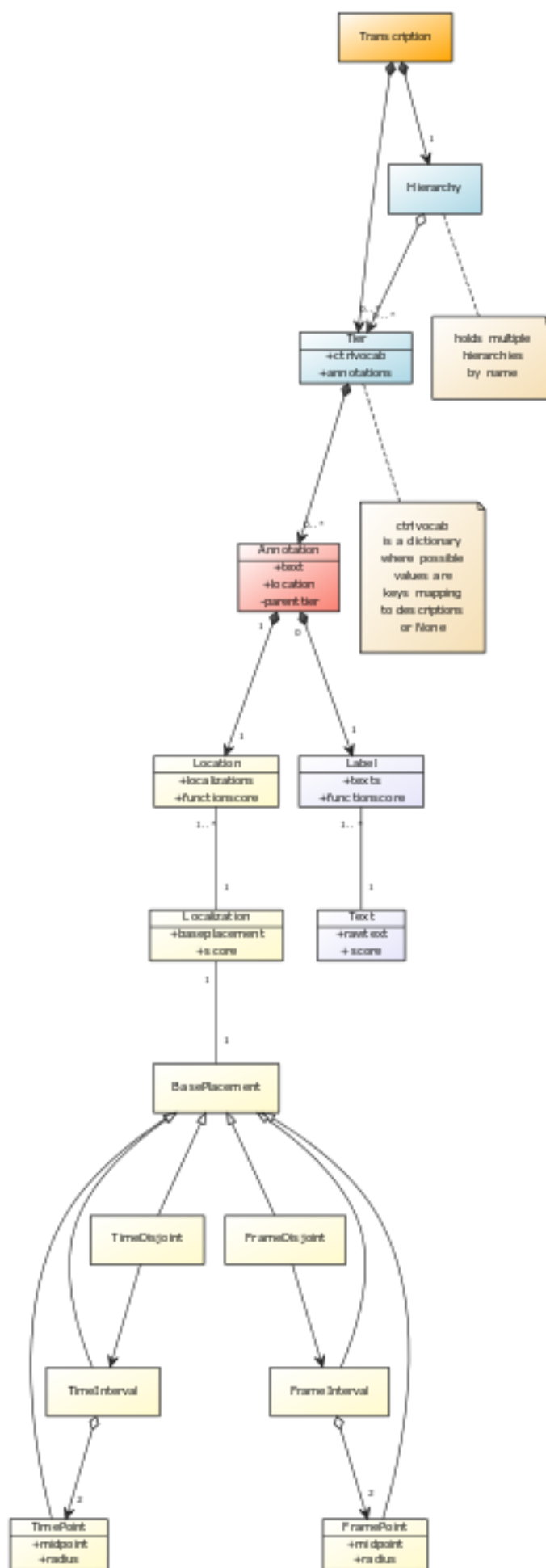
6.5 Creating scripts with the SPPAS API

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.



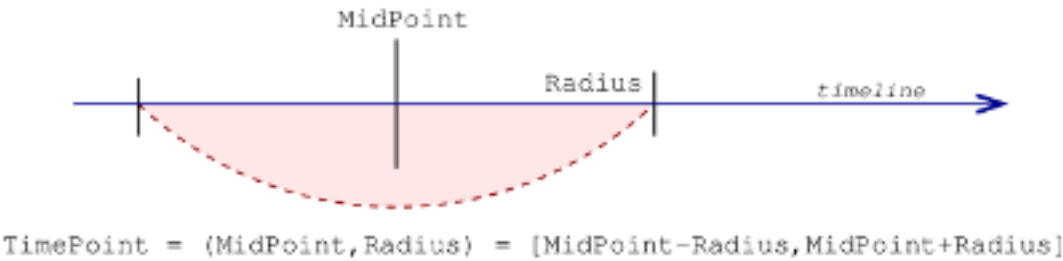


Figure 6.7: Representation of a TimePoint

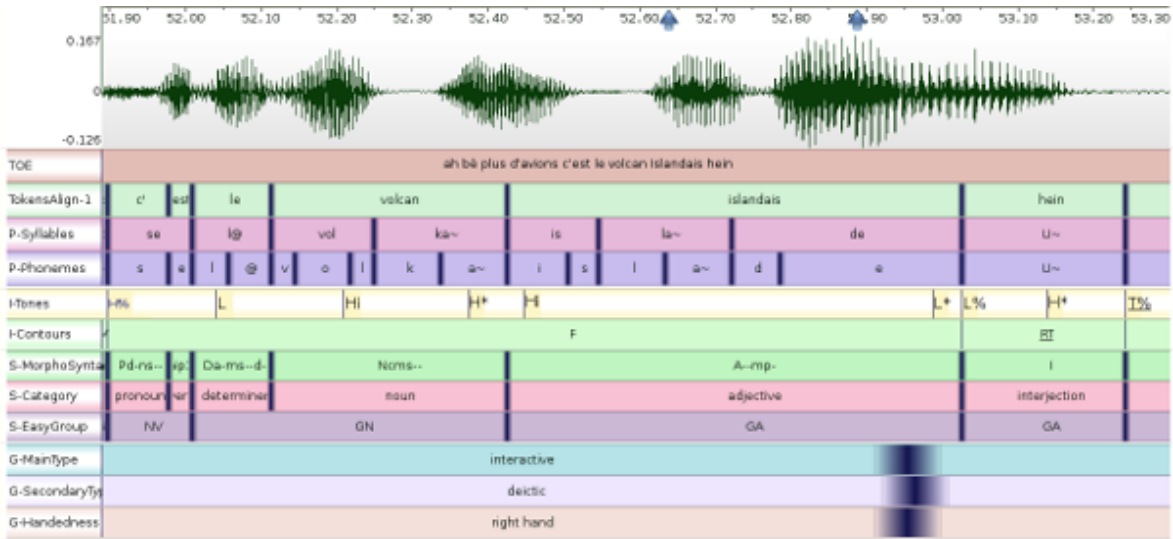


Figure 6.8: Example of multimodal data

```

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, ...) 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
- tdf

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(regexp=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

By using SPPAS, you agree to cite any of these references. For a general citation of SPPAS, simply use the main reference below. For a specific purpose of SPPAS, like for example automatic syllabification, or forced-alignment, or the request system, etc, please refer to the related specific publication.

Notice that a PDF version of the main publication is available in the folder `documentation` of the SPPAS package. All the publications are available on the author website, at the following URL: <http://www.lpl-aix.fr/~bigi/publications.html>

7.1.1 Main reference

How to cite SPPAS?

Brigitte Bigi (2015). SPPAS - Multi-lingual Approaches to the Automatic Annotation of Speech. In “the Phonetician” - International Society of Phonetic Sciences, ISSN 0741-6164, Number 111-112 / 2015-I-II, pages 54-69.

Summary

The first step of most acoustic analyses unavoidably involves the alignment of recorded speech sounds with their phonetic annotation. This step is very labor-intensive and cost-ineffective since it has to be performed manually by experienced phoneticians during many hours of work.

This paper describes the main features of SPPAS, a software tool designed for the needs of automatically producing annotations of speech at the level of utterance, word, syllable and phoneme based on the recorded speech sound and its orthographic transcription. In other words, it can automatize the phonetic transcription task for speech materials, as well as the alignment task of transcription with speech recordings for further acoustic analyses.

Special attention will be given to the methodology implemented in SPPAS, based on algorithms which are as language-and-task-independent as possible. This procedure allows for the addition of new languages quickly and for the adaptation of this tool to the user’s specific needs. Consequently, the quality of the automatic

annotations is largely influenced by external resources, and the users can modify the process as needed. In that sense, phoneticians need automatic tools and these tools can be significantly improved by phonetician input.

7.1.2 Specific references

SPPAS software description...

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.

Brigitte Bigi, Daniel Hirst (2012) **SPeech Phonetization Alignment and Syllabification (SP-PAS): 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).

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.

About Tokenization...

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

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.

Roxana Fung, Brigitte Bigi* (2015). **Automatic Word Segmentation for Spoken Cantonese**, The Oriental Chapter of COCOSDA (International Committee for the Co-ordination and Standardization of Speech Databases and Assessment Techniques).

About Phonetization...

Brigitte Bigi, Pauline Péri, Roxane Bertrand (2012). **Orthographic Transcription: Which Enrichment is required for Phonetization?**, Language Resources and Evaluation Conference, Istanbul (Turkey), pages 1756-1763, ISBN 978-2-9517408-7-7.

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

Brigitte Bigi (to appear). **A phonetization approach for the forced-alignment task in SPPAS**, Human Language Technologies Challenges for Computer Science and Linguistics LNAI 8387, Springer, Heidelberg. ISBN: 978-3-319-14120-6.

About Forced-Alignment...

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

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

Brigitte Bigi (2014). **The SPPAS participation to Evalita 2014**, Proceedings of the First Italian Conference on Computational Linguistics CLiC-it 2014 and the Fourth International Workshop EVALITA 2014, Pisa (Italy). Editors R. Basili, A. Lenci, B. Magnini. ISBN 978-886741-472-7. Volume 2. Pages 127-130.

About Syllabification...

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.

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

Brigitte Bigi, Caterina Petrone (2014). **A generic tool for the automatic syllabification of Italian**, Proceedings of the First Italian Conference on Computational Linguistics CLiC-it 2014 and the Fourth International Workshop EVALITA 2014, Pisa (Italy). Editors R. Basili, A. Lenci, B. Magnini. ISBN 978-886741-472-7. Volume 1. Pages 73-77.

About Repetitions...

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.

About the components...

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.

Brigitte Bigi, Jorane Saubesty (2015). **Searching and retrieving multi-levels annotated data**, Proceedings of Gesture and Speech in Interaction, Nantes (France).

About the data...

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.

7.1.3 Related references

Some results and analyses...

Marion Tellier, Gale Stam, Brigitte Bigi (2012). **Same speech, different gestures?**, 5th International Society for Gesture Studies (ISGS), Lund, Sweden.

Marion Tellier, Gale Stam, Brigitte Bigi (2013). **Gesturing While Pausing In Conversation: Self-oriented Or Partner-oriented?**, TIGER, Tillburg.

Laurent Prévot, Brigitte Bigi, Roxane Bertrand (2013). **A quantitative view of feedback lexical markers in conversational French**, 14th Annual Meeting of the Special Interest Group on Discourse and Dialogue, Metz, France.

Brigitte Bigi, Katarzyna Klessa, Laurianne Georgeton, Christine Meunier (2015). **A syllable-based analysis of speech temporal organization: a comparison between speaking styles in dysarthric and healthy populations**. Interspeech2015, Dresden (Germany), pages 2977-2981.

Laurent Prevot, Jan Gorisch, Roxane Bertrand, Emilien Gorene and Brigitte Bigi (2015). **A SIP of CoFee: A Sample of Interesting Productions of Conversational Feedback**, 16th Annual Meeting of the Special Interest Group on Discourse and Dialogue, Prague (Czech Republic), pp. 149–153.

Some corpora...

Sophie Herment, Anastasia Loukina, Anne Tortel, Daniel Hirst, Brigitte Bigi (2012). **AixOx, a multi-layered learners corpus: automatic annotation**, Proceedings of international conference on corpus linguistics, Jaën (Spain).

Ellen Gurman Bard, Corine Astésano, Alice Turk, Mariapaola D'imperio, Noel Nguyen, Laurent Prévot, Brigitte Bigi (2013). **Aix MapTask: A (rather) new French resource for prosodic and discourse studies**, 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. 15-19.

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.

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), pages 2648-2652. ISBN: 978-2-9517408-8-4.

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 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).

7.2.5 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.

SPPAS Release notes

8.1 the Prehistory

SPPAS 1.0 to 1.3 was made only of tcsh and gawk scripts. It was developped under Linux and tested under Windows with Cygwin.

Version 1.0

The first release was published on 2011, 9th March.

It was able to perform speech segmentation of English read speech.

Version 1.1

Published on 2011, 7th June.

Version 1.2:

Published on 2011, 23th July.

Support of English, French and Italian.

Version 1.3

Published on 2011, 12th December.

This is a transitional version, from tcsh/bash scripts to python. The main fixes and improvements are:

Programs:

- MacOS support
- bugs corrected in many scripts.
- check.csh -> check.py

- sppas.csh -> sppas.py
- GUI: zenity -> pyGtk
- sox now only used for resampling
- a wav python library is added.

Resources:

- Italian dictionary improved
- Italian acoustic model changed (triphones trained from map-task dialogues. see doc/publications)
- French acoustic model changed

8.2 the Middle Ages

Since version 1.4, SPPAS has a name!

It's python made only. It uses python 2.7.x and wxpython 2.8.x or 2.9.x.

Remark for MacOS users: wxpython must be 32 bits. SPPAS requires sox and julius free software to be installed.

SPPAS 1.4 (2012-06-14)

This is the first real version of SPPAS.

Automatic annotations main changes:

- add momel: implements momel in python. Momel (MOdelling MELody) is an algorithm developed by Daniel Hirst and Robert Espesser for the analysis and synthesis of intonation patterns. Momel need a .hz files with pitch values: ASCII file with one value each 10ms. This Momel implementation can be used directly, with a large set of options:

```
> python $SPPAS/scripts/lib/momel.py -h
```

or in SPPAS, using the GUI, or in text-mode (only with default options):

```
> $SPPAS/sppas.py -i INPUT --momel
```

- add the first version of INTSINT, proposed by Daniel Hirst. INTSINT is an acronym for INternational Transcription System for INTonation. INTSINT codes the intonation of an utterance by means of an alphabet of 8 discrete symbols constituting a surface phonological representation of the intonation:
 - T (Top),
 - H (Higher),
 - U (Upstepped),
 - S (Same),
 - M (mid),

- D (Downstepped),
- L (Lower),
- B (Bottom).

Package:

- merge AUTHORS, VERSION and README files in the README file
- create packages in the lib directory (2012-03-20)
- ipu-segmentation in python (2012-02-16)
- improve ipu segmentation algorithm (2012-05-24)
- phonetization in python (2012-02-27), with an unknown word phonetization.
- alignment entirely in python (2012-03-29)
- syllabification implemented with python, and tool/syllabify.py
- add a simple terminal controller (not needed for Unix, just for “DOS”)
- Momel can read PitchTier (from Praat) files
- Momel can deal with long sounds (not only one IPU)
- manage source files, comments, exceptions...
- SPPAS works as a single instance by mean of a lock file.
- SPPAS can be installed in a directory containing spaces and can deal with file names with spaces (bug fixed: 2012-06-11)

GUI:

- pyGTK GUI: select steps in the main window
- pyGTK GUI: add ‘None’ in the language list for language-independent modules
- GUI with the wxpython library (2012-03-27)
- Manage a list of selected files
- Add a “File information”
- Add a “Wav player”
- Fix options to each annotation step with the menu
- Open the log file after each process

Resources:

- French dictionary is using UTF-8 (instead of iso8859-1 in previous versions)
- French dictionary is SAMPA
- Chinese: work with chinese characters instead of pinyin.

Known Bugs:

- The wav player can not play wav files if the filename contains '#’.
- The first line of the Italian dictionary must be changed to "# [#] #”.

SPPAS 1.4.1 (2012-07-13)

Resources:

- Updated English models (from voxforge, 2012-06-25)
- English acoustic models updated, and converted to SAMPA

Annotations:

- IPU's Segmentation annotation performs a simple silence detection if no transcription is available (the volume is automatically adjusted)
- A specific language can be selected for each annotation depending on available resources
- Updated transcription conventions: truncated words: a '-' at the end of the token string (an ex- example) liaisons: the 'letter' between '=' (an =n= example) noises: * (only for FR and IT) short pauses: + (a + example) silences: # (a # example)

GUI:

- Create (systematically) a merged annotations file.

Development:

- Package's management

SPPAS 1.4.2 (2012-08-02)

GUI:

- add a panel with a Transcription editor
- IPU segmentation can split a way into tracks
- IPU segmentation can fix a shift value to boundaries
- IPU segmentation: min-volume option is removed (because the min-volume value is automatically adjusted)
- "File Information" button adds some tier manipulation tools: cut/copy/paste/rename/duplicate/move/preview/filter

Known bug:

- The filter frame is not working under Windows XP.

SPPAS 1.4.3 (2012-10-10)

This is primarily a bug-fix release. Many thanks to all users who send their comments!

GUI:

- Frames and Dialog design is more uniform.

- User's preferences changed. Themes and colors introduced.
- Help is available.

Annotations:

- Bug fixed for phonetization/alignment if the input transcription contains series of silence intervals or series of speech intervals. Previous versions was supposing a **strict IPU**s input transcription.
- Tokenization is done.

Development:

- Code cleaning in the package wxGUI.
- Debug...

SPPAS 1.4.4 (2012-12-06)

GUI:

- add information/options when "Request" a file
- debug Request/Filter (for windows)

Annotations:

- New Italian acoustic model
- New Chinese acoustic model, and some minor changes in the dictionary

Development:

- ".trs" files support (transcriptions from Transcriber)
- debug (alignment, tokenization)
- add ".lab" files export (HTK format)

Known bugs:

- Alignment: it fails under Windows, if julius is not installed properly.
- Syllabification: the last syllable of each file is "broken".
- Alignment error if unknown words during phonetization.

SPPAS 1.4.5 (2013-01-15)

Development:

- Correct a few bugs of the previous version (phonetization, alignment, syllabification)
- ".eaf" files support (transcriptions from Elan)

Annotations:

- Experimental version of Vietnamese
- add Tokenization as a specific annotation level
- add Phonetization of PinYin

GUI:

- Request/Filter a tier: multiple patterns filtering
- Request: add a “New File” button

Tools/scripts:

- tierfilter.py
- tiercombine.py

SPPAS 1.4.6 (2013-02-12)

GUI:

- Improved Request/Filter a tier:
 - add new modes: not contains, not starts with, not ends with
 - add time constraints: minimum duration, maximum duration, meets
 - multiple modes selection (replace radio buttons by check buttons)
- Add Requests/Stats to obtain basic statistics
- Requests Copy/Cut/Paste/Duplicate/Save debugged

Annotations:

- IPU segmentation: can take a “Name” tier to fix names of tracks (if the “Split into tracks” option is checked)

SPPAS 1.4.7 (2013-03-25)

Requests/Filter:

- “Starts search at...” and “Ends search at...”
- remove negative search (because of a bug...). Replaced by a “reverse” option.

Development:

- re-organization of lib, except for the wxGUI library.
- Cancelled the sppas.lock file creation when SPPAS is running.

Resources:

- add experimental version of Taiwanese automatic segmentations (from romanized transcriptions), thanks to Sheng-Fu Wang for the corpus.

Annotations:

- New version of INTSINT, based on the algorithm proposed in (Hirst 2011) and implemented in the last version of Momel/INTSINT Praat plugin!

SPPAS 1.4.8 (2013-05-30)**Development:**

- reorganization of the GUI code. SPPAS is made of:
 - automatic annotations;
 - 3 components (wavplayer, transcriber, requests).
- sppas.py removed. sppas.bat created (for Windows).
- Input/Output library entirely changed

GUI:

- components (wavplayer, transcriber, requests) in separate frames
- new design
- Help and documentation changed, expanded and improved

SPPAS 1.4.9 (2013-07-03)**New Logo!****Development:**

- bug correction:
 - Bug fixed in IPU segmentation with option “Split into tracks”
 - Bug fixed in Momel with some rare files
 - Bug fixed to create a merged file
 - Bug fixed in align and IPU seg.
 - Bug fixed in Transcriber
- library organization revised

GUI:

- Add an export button to the FLP
- Migrate wav infos from the Requests component to the Wav player component
- Volume control removed in the Wav Player component
- Save As improved in the Requests component

SPPAS 1.5 (2013-08-02)

Development:

- bug correction

Annotation:

- Tokenization: TOE support finalized

GUI:

- Transcribe: debug of IPU player, for MacOS

Resources:

- New French acoustic model. Attention: phoneset changed.
- New French dictionary: use the new phoneset!
- New Taiwanese acoustic model. Attention: phoneset changed: accept some chinese...
- New Taiwanese dictionary: use the new phoneset + some chinese syllables added.
- Vietnamese removed: due to the lack of data, the model can't be improved.

SPPAS 1.5.1 (2013-08-29)

Development:

- bug correction in annotations
- help system debugged

Annotation:

- Phonetization of unknown words improved

Resources:

- French dict modified

SPPAS 1.5.2 (2013-09-27)

ALL RESOURCES ARE MOVED IN A 'RESOURCES' DIRECTORY.

Development:

- bug correction in annotations and GUI

Resources:

- French dict modified
- New resources for the tokenization

Components:

- Statistics: an avanced component to estimate (and save!) statistics on multiple annotated files

SPPAS 1.5.3 (2013-10-25)

Resources:

- Add Spanish support, thanks to JuanMaria Garrido and the “Glissando” corpus

Components:

- improved Statistics component
- Add a “Filter” component, first version with a basic GUI

Help updated.

SPPAS 1.5.4 (2013-12-03)

The most stable release of this period.

Components:

- Wav Player changed.
- Information and Requests removed. Replaced by Data Roamer.
- Transcriber removed. Replaced by IPU Transcriber.
- Statistics updated.
- Filter updated.

Annotations:

- Add Repetitions (detection of sources only)

SPPAS 1.5.5 (2013-12-23)

Development:

- improved annotationdata (add methods: Find, Index; add uncertain label; debug radius).

Components:

- debug

GUI:

- Tips at start-up
- New theme: Christmast

SPPAS 1.5.6 (2014-01-28)

Development:

- improved annotationdata (add methods: Near, Search).

Components:

- debug

Resources:

- New Italian dictionary (including germination)

Package cleaning!

SPPAS 1.5.7 (2014-02-18)

Plugins management.

Resources:

- Japanese support, thanks to the resources available on the Julius website.

SPPAS 1.5.8 (2014-03-18)

Development:

- annotationdata: more flexibility while adding annotations, add sub-divisions, export csv modified, doc-strings, import/export in a native format (xra, version 1.0).

Documentation:

- add a PDF file of slides: “SPPAS for dummies”

SPPAS 1.5.9 (2014-04-15)

Components:

- new: DataViewer, experimental version

Annotations:

- syllabification rules: accept 2 types of vowels (V and W)
- syllabification: faster!

Development:

- Export Elan
- Import/Export xra (SPPAS native format. See etc/xra for details)

Resources:

- New French acoustic model

SPPAS 1.6 (2014-05-22)

Package:

- Rename folder Doc to documentation
- Documentation created, SPPAS-for-dummies updated.

Development:

- helpsystem removed
- gui: package re-organization, re-implementation.
- Alignment: choice of the aligner (julius, hvite or basic)

Resources:

- new acoustic model: FR-Read and FR
- new acoustic model: ZH
- new acoustic model: TW
- new acoustic model: SP

SPPAS 1.6.1 (2014-09-26)

Development:

- bug correction (export, syllabification, alignment)
- DataViewer, major bugs corrected.

Resources:

- new pronunciation dictionary and new acoustic model for Italian
- add resources for Catalan

GUI:

- add a new Export button

SPPAS 1.6.2 (2014-10-21)

Resources:

- language names changed. They are now corresponding to the international standard ISO639-3. See <http://www-01.sil.org/iso639-3/>
- Mandarin Chinese dictionary changed.
- Catalan dictionary changed.

SPPAS 1.6.3 (2014-11-02)

Resources:

- Add resources for Cantonese

Documentation:

- It's now (more or less) finished!

Development:

- Support of Elan improved (add Controlled vocabulary)

GUI:

- Change the organization of the main frame: annotations above components

8.3 the Renaissance

Since version 1.6.4, SPPAS is still using python 2.7.x, but SPPAS requires wxpython > 3.0.

For MacOS users: wxpython must be installed in 64 bits (cocoa).

SPPAS 1.6.4 (2014-12-05)

Package re-organized!

Development:

- Phonetization of unknown words improved
- Support of upper/lower of the extension of speech files (wav, WAV)
- Tokenization of languages with dictionaries in upper case (eng, ita): bug fixed.
- Creates systematically a dump file of resources for a faster load at the next use
- Read TextGrid files exported by Elan: bug fixed.
- `sppas.command` checks the system and run either in 32 or 64 bits (MacOS)

Components:

- IPUscribe replaces IPUTranscriber mainly for the support large files: tested with a file of 1 hour speech (143 Go) and 800 IPUs.
- SndRoamer replaces WavPlayer
- Dataroamer has also a new version

SPPAS 1.6.5 (2014-12-17)

This is primarily a bug-fix release.

Development:

- all programs in “bin” and “scripts” sub-directories were revised and tested, or removed.

Annotation:

- Tokenization: code cleaning and re-organisation.

GUI:

- Procedure outcome report: print a warning message in the log file if no file is matching the expected extension

SPPAS 1.6.6 (2015-01-19)

Documentation completed and updated. Now, only the documentation of all the components is missing.

Annotations:

- log messages more explicit and status with colors:
 - OK is green
 - WARNING is orange
 - IGNORED is violet
 - INFO is blue
 - ERROR is red
- management of input/output file format redone: now easier for the user.

Development:

- package architecture revised: mainly “sppasgui” and “components” merged in “wxgui”, and many other changes.
- thread removed in automatic annotation process.
- debug of alignment: if too short units.
- radius value properly fixed in most of the automatic annotations.

GUI:

- GUI is more homogeneous and pretty (hope!)
- Show the date in the status bar
- New Settings frame:
 - 4 icon themes available
 - Choice of foreground and background colours
 - Choice of the font
 - Choice of the input/output file format of annotations
- New in Help menu:
 - access to the project homepage
 - access to the online documentation
 - New Feedback
- New Help browser
- Add Keyboard shortcuts:
 - ALT+F4 to exit,
 - CTRL+A to add files in FLP
 - SHIFT+CTRL+A to add a directory in FLP

- Del to remove selected files of the FLP
- SHIFT+Del to erase selected files of the FLP
- CTRL+C to copy files
- CTRL+E to export files
- F5 to refresh the FLP
- F1 to open the help browser
- F2 to open the “About” frame

Components:

- GUI design unified for DataRoamer, SndPlayer, IPUscribe and SppasEdit
- New Tier Preview frame (still under development)
- SndPlayer print information about a sound file with colors:
 - Green: the information corresponds to the SPPAS requirements
 - Orange: the information does not exactly corresponds to the requirements however SPPAS is able to deal with (after conversion)
 - Red: SPPAS does not support. It must be converted before using it!

Web site host has changed: <http://sldr.org/sldr00800/preview/>

SPPAS-1.6.7 (2015-02-16)

Automatic Annotations:

- By default, tokenization produces only one tier. Check an option to get TokensStd and TokensFaked, in case of EOT.
- radius value properly fixed in most of the automatic annotations.

GUI:

- Tested and debugged on MacOS (version 10.9.5, with wxpython 3.0.2)

Development:

- Tier hierarchy partly implemented: TimeAlignement and TimeAssociation are two “links” that can be fixed between tiers.

Annotations:

- Add Polish support

SPPAS-1.6.8 (2015-04-09)

Resources:

- new French acoustic model
- new English acoustic model (VoxForge nightly build of March, 15th, 2015)
- add phoneset mapping tables

Development:

- Add a phoneme mapping in models, to allow both the dictionary to include real SAMPA symbols and the acoustic model to be compatible with Hvite requirements.
- annotationdata bug correction with min and max values
- IPU's Segmentation:
 - bug correction when split into tracks with a tier “Name”
 - add the ipu number in speech segments if silence/speech segmentation
- Self-repetitions debug in finding the repetition interval

GUI:

- DataRoamer: “New” button debugged
- DataRoamer: Add a button “Radius” to adjust manually the vagueness of each bounday of a tier

8.4 the Early modern period

Since version 1.6.9, the installation is simplified: sox is unnecessary. Python 2.7.something, wxpython 3.something and Julius are the only dependencies.

SPPAS-1.6.9 (2015-05-14)

Development:

- package annotationdata.filter updated to support last changes of annotationdata: multiple labels and numerical labels.
- package annotationdata.io:
 - praat.py newly created. Support of TextGrid, PitchTier and IntensityTier files completely re-written
 - htk.py newly created to support .lab and .mlf files
 - sclite.py newly created to support .stm and .ctm files
 - signaix.py newly created to support .hz files
 - SPPAS native format XRA upgraded to version 1.1 to support improvements of the library.

- package annotationdata:
 - updated object Transcription to support more/different input data
 - updated hierarchy
 - updated meta-data
- package signal: partially re-written. As a consequence, sox won't be neither used, but the file conversion (if required) is slower.

GUI:

- DataFilter component partially re-written to facilitate its use
- Preview frame modified

SPPAS-1.7.0 (2015-07-03)

Development:

- package annotationdata.io:
 - add support of subtitles: sub, srt
 - elan.py created to replace eaf.py: allows a full support of Elan annotations
 - transcriber.py created to replace trs.py for a better support of Transcriber files
 - anvil.py allows to import anvil files
- package annotationdata:
 - updated hierarchy
 - updated meta-data
- package signal:
 - support of audio files re-written: can open/save wav, aiff and au files
 - add a lot of possibilities to manage one channel of an audio file

GUI:

- DataFilter component finalized: can deal with alternative labels, and typed labels (string, number, boolean)
- Statistics component fully re-written to facilitate its use
- SndRoamer displays more properties of audio files (min, mean and max added)

Annotations:

- IPU's Segmentation produces 2 tiers if a transcription is given: the IPU's segmentation itself, and the Transcription time-aligned at the IPU's level.

SPPAS-1.7.1 (2015-08-05)

Development:

- package re-organization:
 - package signal is now standalone
 - package resources is now standalone
 - package presenters created
- updated statistics estimations
- package annotationdata:
 - add Media object
 - add CtrlVocab object
 - add inheritance of MetaObject for Annotation
- package annotationdata.io:
 - complete debug of all file formats
 - add comments and documentation
 - add also some tests
 - add Media/CtrlVocab in some file formats
 - add Annotation Pro support of antx files (in API)

Components:

- add Time Group Analyser (TGA) in Statistics
- add Kappa estimation on labels of 2 tiers with the same number of intervals

Annotations:

- New version of XRA: 1.2
- Make XRA the default input/output file format

SPPAS-1.7.2 (2015-09-03)

Development:

- updated XRA reader/writer to solve problems with upper/lower cases of elements
- updated Elan reader to be compatible with format 2.8
- updated Praat reader/writer for single/double quotes
- support of antx files in the GUI
- add the julius score in PhonAlign annotation (can be seen only if XRA)
- remove tk dependencies

SPPAS-1.7.3 (2015-10-09)

Resources:

- New word-based vocab for Cantonese
- New word-based dict for Cantonese
- New phoneme-based acoustic model for Cantonese

Development:

- Descriptive statistics debugged for detailed panels.

SPPAS-1.7.4 (2015-11-06)

Resources:

- Add a vocab for Portuguese
- Add a dict for Portuguese
- Add an acoustic model for Portuguese. It has to be noticed that it was constructed from French/Spanish/Italian models, not trained from data.

Samples:

- Add Portuguese
- Change some samples in various languages

Development:

- Debug of the Tokenizer.

SPPAS-1.7.5 (2015-12-11)

Development:

- Add vagueness to the annotation duration estimation
- Bug correction while saving the procedure outcome report in the GUI

GUI:

- Changed all pictures to remove the problem with colorset of some of them
- Add a christmas theme for the pictures of tips
- IPUscribe improvements:
 - add keyboard shortcuts to play/pause/stop the sound
 - add a new button to auto-play the sound

SPPAS-1.7.6 (2016-01-28)

Web site host has changed: <http://www.sppas.org/>

Development:

- IPU segmentation:
 - bug correction with option “split into tracks”
 - new option to add or not add the IPU index of each IPU into the transcription tier
- Tokenization:
 - bug correction in input tier selection
 - bug correction for replacements

Resources:

- add a vocabulary of Korean
- add an acoustic model for Korean (made from the English and the Taiwanese ones).

GUI:

- DataRoamer: bug correction of “Duplicate”

Others:

- Add a sample in Korean
- Updated references to cite SPPAS in publications

SPPAS-1.7.7 (2016-03-30)

Resources:

- Correction of errors in most of the acoustics models, for /@@/ and /dummy/ models
- New vocabulary and pronunciation dictionary for Mandarin Chinese.

GUI:

- Dialogs are customized
- DataRoamer debug and changes: save buttons moved in the main toolbar.
- HelpSystem is (welcome) back!

Development:

- Add the API and a script to train acoustic models with HTK.
- Add Kullback-Leibler distance estimator.
- Add a script to compare 2 segmentations (of 2 tiers).
- Classes of the package resources are debugged, improved and extended.
- Automatic annotation alignment: bug correction if input starts by an empty interval.
- `sppas.command` modified for MacOS-X (use python if python2 not available)
- GUIs in `bin` directory are updated (test of python, wxpython, and so on).

SPPAS-1.7.8 (2016-05-05)

Resources:

- Add a pronunciation mapping table `eng-fra.map`: to be used for the speech segmentation for French speakers reading an English text.

Development:

- Phonetization is extended: it can take into account a mapping table of phones to generate new pronunciation variants.
- Phonetization: use of minus instead of dots to separate phones as recommended in SAMPA standard.
- Alignment is restructured and extended: it can take into account two acoustics models and mix them.
- DataFilter is extended: Relations can be based on a delay between the intervals.
- annotationdata: add Xtrans reader (extension: `.tdf`)
- Code cleaning in several packages.

GUI:

- Automatic annotations: improved log messages.

SPPAS-1.7.9 (2016-06-03)

GUI: - Some debug in SppasEdit - SndPlayer renamed AudioRoamer because new fonctionnalités were added:

- See detailed information about each channel of an audio file
- Can extract/save a channel or a fragment of channel
- Can modify the framerate and the sample width of a channel
- Can add a bias on amplitude values of a channel,
- Can multiply amplitude values of a channel,
- Can remove the offset of amplitude values of a channel.

Automatic annotations:

- IPUs Segmentation fully re-implemented.
- Silence/speech segmentation improved for both quality and fastness
- Package code cleaning and re-organization.

SPPAS-1.8.0 (2016-30-08)

GUI:

- Design fully revisited and tested under Linux Mint, Windows 10 and MacOS 10.9

Development:

- SLM package created: can estimate a statistical language model (without smooth method)

Automatic annotations:

- Add a diagnosis of files
- Tokenize extended: applied also on alternative labels
- Phonetize extended: applied also on alternative labels
- Alignment code cleaning and partly re-implemented
- Add "Chunk alignment"
- Use of a .ini file to configure each annotation instead of a sppas.conf file

SPPAS-1.8.1 (2016-28-11)**New:**

- A few tutorials are available on the web site.

Automatic annotations:

- Align: an ActivityDuration tier can be optionnally added.
- Support of 3-columns tab-delimited files with .txt extension. It allows the compatibility with Audacity.
- Acoustic models training validated.

Resources: - Catalan: new pronunciation dictionary and new acoustic model.

8.5 the Modern period

... it will be the case when SPPAS will be based on Python 3.x which is not currently the plan!