speach

Le Tuan Anh

Apr 12, 2022

CONTENTS

Welcome to Speach's documentation! Speach, formerly texttaglib, is a Python 3 library for managing, annotating, and converting natural language corpuses using popular formats (CoNLL, ELAN, Praat, CSV, JSON, SQLite, VTT, Audacity, TTL, TIG, ISF, etc.)

Main functions:

- Text corpus management
- Manipulating ELAN transcription files directly in ELAN Annotation Format (eaf)
- TIG - A human-friendly intelinear gloss format for linguistic documentation
- Multiple storage formats (text files, JSON files, SQLite databases)

Contributors are welcome! If you want to help developing `speach`, please visit *Contributing* page.

CHAPTER

ONE

INSTALLATION

Speach is availble on PyPI.

```
pip install speach
```

CHAPTER

TWO

ELAN SUPPORT

Speach can be used to extract annotations as well as metadata from ELAN transcripts, for example:

```
from speach import elan

# Test ELAN reader function in speach
eaf = elan.read_eaf('./test/data/test.eaf')

# accessing tiers & annotations
for tier in eaf:
    print(f"{tier.ID} | Participant: {tier.participant} | Type: {tier.type_ref}")
    for ann in tier:
        print(f"{ann.ID.rjust(4, ' ')}. [{ann.from_ts} :: {ann.to_ts}] {ann.text}")
```

Speach also provides command line tools for processing EAF files.

```
# this command converts an eaf file into csv
python -m speach eaf2csv input_elan_file.eaf -o output_file_name.csv
```

More information:

2.1 Tutorials

Note: this section is under construction. Please see *Common Recipes* section for now.

Tutorials for getting started with speach

2.1.1 Installation

speach is available on PyPI and can be installed using *pip*.

```
pip install --user speach
```

2.1.2 Module tutorials

Media processing tutorial

Media module can be used to process audio and video files (converting, merging, cutting, et cetera). `speach.media` use ffmpeg underneath.

Installing ffmpeg

Linux

It is very likely that your Linux distribution comes with a default `ffmpeg` package under `/usr/bin/ffmpeg`.

Windows

Note: To be updated (download and install to `C:\\Users\\<account_name>\\local\\ffmpeg\\ffmpeg.exe`)

Mac OS

You can download ffmpeg binary file from https://ffmpeg.org and copy it to one of these folders

- /Users/<account_name>/ffmpeg
- /Users/<account_name>/bin/ffmpeg
- /Users/<account_name>/local/ffmpeg
- /Users/<account_name>/local/ffmpeg/ffmpeg

and `speach` will find it automatically.

See *Common Recipes* for code samples.

2.2 Common Recipes

Here are code snippets for common usecases of `speach`.

2.2.1 ELAN Recipes

Common snippets for processing ELAN transcriptions with `speach`.

For in-depth API reference, see *ELAN module* page.

Open an ELAN file

```
>>> from speach import elan
>>> eaf = elan.read_eaf('./data/test.eaf')
>>> eaf
<speach.elan.Doc object at 0x7f67790593d0>
```

Save an ELAN transcription to a file

After edited an *speach.elan.Doc* object, its content can be saved to an EAF file like this

```
>>> eaf.save("test_edited.eaf")
```

Parse an existing text stream

If you have an input stream ready, you can parse its content with *speach.elan.parse_eaf_stream()* method.

```
>>> from speach import elan
>>> with open('./data/test.eaf', encoding='utf-8') as eaf_stream:
>>> ...  eaf = elan.parse_eaf_stream(eaf_stream)
>>> ...
>>> eaf
<speach.elan.Doc object at 0x7f6778f7a9d0>
```

Accessing tiers & annotations

You can loop through all tiers in an *speach.elan.Doc* object (i.e. an eaf file) and all annotations in each tier using Python's `for ... in ...` loops. For example:

```
for tier in eaf:
    print(f"{tier.ID} | Participant: {tier.participant} | Type: {tier.type_ref}")
    for ann in tier:
        print(f"{ann.ID.rjust(4, ' ')}. [{ann.from_ts.ts} -- {ann.to_ts.ts}] {ann.text}")
```

Accessing nested tiers in ELAN

If you want to loop through the root tiers only, you can use the `roots` list of an *speach.elan.Doc*:

```
eaf = elan.read_eaf('./data/test_nested.eaf')
# accessing nested tiers
for tier in eaf.roots:
    print(f"{tier.ID} | Participant: {tier.participant} | Type: {tier.type_ref}")
    for child_tier in tier.children:
        print(f"    | {child_tier.ID} | Participant: {child_tier.participant} | Type:
↪{child_tier.type_ref}")
        for ann in child_tier.annotations:
            print(f"    |- {ann.ID.rjust(4, ' ')}. [{ann.from_ts} -- {ann.to_ts}] {ann.
↪text}")
```

Retrieving a tier by name

All tiers are indexed in *speach.elan.Doc* and can be accessed using Python indexer operator. For example, the following code loop through all annotations in the tier `Person1 (Utterance)` and print out their text values:

```
>>> p1_tier = eaf["Person1 (Utterance)"]
>>> for ann in p1_tier:
>>>     print(ann.text)
```

Cutting annotations to separate audio files

Annotations can be cut and stored into separate audio files using `speach.elan.ELANDoc.cut()` method.

```
eaf = elan.read_eaf(ELAN_DIR / "test.eaf")
for idx, ann in enumerate(eaf["Person1 (Utterance)"], start=1):
    eaf.cut(ann, ELAN_DIR / f"test_person1_{idx}.ogg")
```

Converting ELAN files to CSV

`speach` includes a command line tool to convert an EAF file into CSV.

```
python -m speach eaf2csv path/to/my_transcript.eaf -o path/to/my_transcript.csv
```

By default, speach generate output using `utf-8` and this should be useful for general uses. However in some situations users may want to customize the output encoding. For example Microsoft Excel on Windows may require a file to be encoded in `utf-8-sig` (UTF-8 file with explicit BOM signature in the beginning of the file) to recognize it as an UTF-8 file. It is possible to specify output encoding using the keyword `encoding`, as in the example below:

```
python -m speach eaf2csv my_transcript.eaf -o my_transcript.csv  --encoding=utf-8-sig
```

2.2.2 Media processing recipes

Media module can be used to process audio and video files (converting, merging, cutting, et cetera). `speach.media` use ffmpeg underneath.

Media module requires `ffmpeg`, for installation guide please refer to *Installing ffmpeg*.

Basic

Just import media module from speach library to start using it

```
>>> from speach import media
```

Convert media files

converting the wave file `test.wav` in Documents folder into OGG format `test.ogg`

```
>>> media.convert("~/Documents/test.wav", "~/Documents/test.ogg")
```

Cutting media files

cutting `test.wav` from the beginning to 00:00:10 and write output to `test_before10.ogg`

```
>>> media.cut("test.wav", "test_before10.ogg", to_ts="00:00:10")
```

cutting `test.wav` from 00:00:15 to the end of the file and write output to `test_after15.ogg`

```
>>> media.cut("test.wav", "test_after15.ogg", from_ts="00:00:15")
```

cutting `test.wav` from 00:00:15 to 00:00:15 and write output to `test_10-15.ogg`

```
>>> media.cut("test.wav", "test_10-15.ogg", from_ts="00:00:10", to_ts="00:00:15")
```

Using extra arguments

When you process audio files using ffmpeg, sometimes you may want to use extra arguments, such as codec information or filters, you may add the extra arguments **after** the standard arguments of *speach.media* function calls. For example:

Setting async flag and audio codec

```
>>> media.convert("recording.wav", "recording.ogg", "-async", 1, "-c:a", "pcm_s16le")
```

Or in using ffmpeg demuxer commands

```
>>> concat_str = "file './recording.wav'\ninpoint 00:07:03\noutpoint 00:15:23.124"
>>> media.concat(concat_str, "outfile.ogg", "-segment_time_metadata",  1, "-af" ,
↪"asetnsamples=32,aselect=concatdec_select")
```

Querying ffmpeg information

```
>>> from speach import media
>>> media.version()
'4.2.4-1ubuntu0.1'
>>> media.locate_ffmpeg()
'/usr/bin/ffmpeg'
```

Others

For in-depth information and a complete API reference, please refer to *speach.media* API page.

2.3 API Reference

An overview of speach modules.

2.3.1 Contents

ELAN module

speach supports reading and manipulating multi-tier transcriptions from ELAN directly.

Note: For better security, speach will use the package defusedxml automatically if available to parse XML streams (instead of Python's default parser). When defusedxml is available, the flag speach.elan.SAFE_MODE will be set to True.

For common code samples to processing ELAN, see *ELAN Recipes* page.

Table of Contents

ELAN module functions

ELAN module - manipulating ELAN transcript files (*.eaf, *.pfsx)

speach.elan.**read_eaf**(*eaf_path*, *encoding='utf-8'*, **args*, ***kwargs*)

Read an EAF file and return an elan.Doc object

```
>>> from speach import elan
>>> eaf = elan.read_eaf("myfile.eaf")
```

Parameters

- **eaf_path** (*str or Path-like object*) – Path to existing EAF file
- **encoding** (*str*) – Encoding of the eaf stream, defaulted to UTF-8

Return type *speach.elan.Doc*

speach.elan.**parse_eaf_stream**(*eaf_stream*, **args*, ***kwargs*)

Parse an EAF input stream and return an elan.Doc object

```
>>> with open('test/data/test.eaf').read() as eaf_stream:
>>>     eaf = elan.parse_eaf_stream(eaf_stream)
```

Parameters **eaf_stream** – EAF text input stream

Return type *speach.elan.Doc*

speach.elan.**parse_string**(*eaf_string*, **args*, ***kwargs*)

Parse EAF content in a string and return an elan.Doc object

```
>>> with open('test/data/test.eaf').read() as eaf_stream:
>>>     eaf_content = eaf_stream.read()
>>>     eaf = elan.parse_string(eaf_content)
```

Parameters **eaf_string** (*str*) – EAF content stored in a string

Return type *speach.elan.Doc*

ELAN Document model

class speach.elan.**Doc**(***kwargs*)

This class represents an ELAN file (*.eaf)

annotation(*ID*)

Get annotation by ID

clone(**args*, ***kwargs*)

Clone this ELAN object by using the save() action

classmethod **create**(*media_file='audio.wav'*, *media_url=None*, *relative_media_url=None*, *author=''*, **args*, ***kwargs*)

Create a new blank ELAN doc

```
>>> from speach import elan
>>> eaf = elan.create()
```

Parameters **encoding** (*str*) – Encoding of the eaf stream, defaulted to UTF-8

Return type *speach.elan.Doc*

cut(*section*, *outfile*, *media_file=None*, *use_concat=False*, **args*, ***kwargs*)

Cut the source media with timestamps defined in section object

For example, the following code cut all annotations in tier "Tier 1" into appopriate audio files

```
>>> for idx, ann in enumerate(eaf["Tier 1"], start=1):
>>>     eaf.cut(ann, f"tier1_ann{idx}.wav")
```

Parameters

- **section** – Any object with `from_ts` and `to_ts` attributes which return TimeSlot objects

- **outfile** – Path to output media file, must not exist or a FileExistsError will be raised
- **media_file** – Use to specify source media file. This will override the value specified in source EAF file

Raises FileExistsError, ValueError

get_linguistic_type(*type_id*)

Get linguistic type by ID. Return None if can not be found

get_participant_map()

Map participants to tiers Return a map from participant name to a list of corresponding tiers

get_vocab(*vocab_id*)

Get controlled vocab list by ID

media_path()

Try to determine the best path to source media file

new_timeslot(*value*)

Create a new timeslot object

Parameters **value** (*int or str*) – Timeslot value (in milliseconds)

classmethod parse_string(*eaf_string*, **args*, ***kwargs*)

Parse EAF content in a string and return an elan.Doc object

```
>>> with open('test/data/test.eaf').read() as eaf_stream:
>>>     eaf_content = eaf_stream.read()
>>>     eaf = elan.parse_string(eaf_content)
```

Parameters **eaf_string** (*str*) – EAF content stored in a string

Return type *speach.elan.Doc*

save(*path*, *encoding='utf-8'*, *xml_declaration=None*, *default_namespace=None*, *short_empty_elements=True*, **args*, ***kwargs*)

Write ELAN Doc to an EAF file

tiers() → Tuple[*speach.elan.Tier*]

Collect all existing Tier in this ELAN file

to_csv_rows() → List[List[str]]

Convert this ELAN Doc into a CSV-friendly structure (i.e. list of list of strings)

Returns A list of list of strings

Return type List[List[str]]

to_xml_bin(*encoding='utf-8'*, *default_namespace=None*, *short_empty_elements=True*, **args*, ***kwargs*)

Generate EAF content (bytes) in XML format

Returns EAF content

Return type bytes

to_xml_str(*encoding='utf-8'*, **args*, ***kwargs*)

Generate EAF content string in XML format

property constraints: Tuple[speach.elan.Constraint]

A tuple of all existing constraints in this ELAN file

property external_refs: Tuple[speach.elan.ExternalRef]

Get all external references

property languages: Tuple[speach.elan.Language]

Get all languages

property licenses: Tuple[speach.elan.License]

Get all licenses

property linguistic_types: Tuple[speach.elan.LinguisticType]

A tuple of all existing linguistic types in this ELAN file

property roots: Tuple[*speach.elan.Tier*]

All root-level tiers in this ELAN doc

property vocabs: Tuple[speach.elan.ControlledVocab]

A tuple of all existing controlled vocabulary objects in this ELAN file

ELAN Tier model

class speach.elan.**Tier**(*doc=None*, *xml_node=None*, ***kwargs*)

Represents an ELAN annotation tier

filter(*from_ts=None*, *to_ts=None*)

Filter utterances by from_ts or to_ts or both If this tier is not a time-based tier everything will be returned

get_child(*ID*)

Get a child tier by ID, return None if nothing is found

new_annotation(*value*, *from_ts=None*, *to_ts=None*, *ann_ref_id=None*, *values=None*, *timeslots=None*, *check_cv=True*)

Create new annotation(s) in this current tier ELAN provides 5 different tier stereotypes.

To create a new standard annotation (in a tier with no constraints), a text value and a pair of from-to timestamp must be provided.

```
>>> from speach import elan
>>> eaf = elan.create()  # create a new ELAN transcript
>>> # create a new utterance tier
>>> tier = eaf.new_tier('Person1 (Utterance)')
>>> # create a new annotation between 00:00:01.000 and 00:00:02.000
>>> a1 = tier.new_annotation('Xin chào', 1000, 2000)
```

Included-In tiers

```
>>> eaf.new_linguistic_type('Phoneme', 'Included_In')
>>> tp = eaf.new_tier('Person1 (Phoneme)', 'Phoneme', 'Person1 (Utterance)')
>>> # string-based timestamps can also be used with the helper function elan.
↪ts2msec()
>>> tt.new_annotation('ch', elan.ts2msec("00:00:01.500"),
                      elan.ts2msec("00:00:01.600"),
                      ann_ref_id=a1.ID)
```

Annotations in Symbolic-Association tiers:

```
>>> eaf.new_linguistic_type('Translate', 'Symbolic_Association')
>>> tt = eaf.new_tier('Person1 (Translate)', 'Translate', 'Person1 (Utterance)')
>>> tt.new_annotation('Hello', ann_ref_id=a1.ID)
```

Symbolic-Subdivision tiers:

```
>>> eaf.new_linguistic_type('Tokens', 'Symbolic_Subdivision')
>>> tto = eaf.new_tier('Person1 (Tokens)', 'Tokens', 'Person1 (Utterance)')
>>> # extra annotations can be provided with the argument values
>>> tto.new_annotation('Xin', values=['chào'], ann_ref_id=a1.ID)
>>> # alternative method (set value to None and provide everything with values)
>>> tto.new_annotation(None, values=['Xin', 'chào'], ann_ref_id=a1.ID)
```

property linguistic_type: speach.elan.LinguisticType

Linguistic type object of this Tier (alias of type_ref

property name

An alias to tier's ID

property parent_ref

ID of the parent tier. Return None if this is a root tier

property time_alignable

Check if this tier contains time alignable annotations

property type_ref: speach.elan.LinguisticType

Tier type object

property type_ref_id

ID of the tier type ref

ELAN Annotation model

There are two different annotation types in ELAN: *TimeAnnotation* and *RefAnnotation*. TimeAnnotation objects are time-alignable annotations and contain timestamp pairs `from_ts, to_ts` to refer back to specific chunks in the source media. On the other hand, RefAnnotation objects are annotations that link to something else, such as another annotation or an annotation sequence in the case of symbolic subdivision tiers.

class speach.elan.**TimeAnnotation**(*ID*, *from_ts*, *to_ts*, *value*, *xml_node=None*, ***kwargs*)

An ELAN time-alignable annotation

overlap(*other*)

Calculate overlap score between two time annotations Score = 0 means adjacent, score > 0 means overlapped, score < 0 means no overlap (the distance between the two)

property duration: float

Duration of this annotation (in seconds)

property from_ts: *speach.elan.TimeSlot*

Start timestamp of this annotation

property to_ts: *speach.elan.TimeSlot*

End timestamp of this annotation

class speach.elan.**RefAnnotation**(*ID*, *ref_id*, *previous*, *value*, *xml_node=None*, ***kwargs*)

An ELAN ref annotation (not time alignable)

property ref_id

ID of the referenced annotation

class speach.elan.**Annotation**(*ID*, *value*, *cve_ref=None*, *xml_node=None*, ***kwargs*)

An ELAN abstract annotation (for both alignable and non-alignable annotations)

property text

An alias to ELANAnnotation.value

property value: str

Annotated text value.

It is possible to change value of an annotation

```
>>> ann.value
'Old value'
>>> ann.value = "New value"
>>> ann.value
'New value'
```

class speach.elan.**TimeSlot**(*xml_node=None*, *ID=None*, *value=None*, **args*, ***kwargs*)

property sec

Get TimeSlot value in seconds

property ts: str

Return timestamp of this annotation in vtt format (00:01:02.345)

Returns An empty string will be returned if TimeSlot value is None

property value

TimeSlot value (in milliseconds)

Media module

This is a module for media processing.

For common media code samples, please refer to *Media processing recipes*.

Media processor module for cutting, converting media contents (audio, video, etc.)

speach.media.**concat**(*text*, *outfile*, *dir=None*, **args*, ***kwargs*)

Process a ffmpeg demuxer file and write result to outfile

Read more: https://trac.ffmpeg.org/wiki/Concatenate

Parameters

- **text** (*str*) – demuxer content string, which will be written to a temporary file before calling
- **outfile** – path to an output file
- **dir** – The directory to create the temp demuxer file, leave as None to use Python default temp dir

speach.media.**convert**(*infile*, *outfile*, **args*, *ffmpeg_path=None*)

Convert an audio/video file into another format

To convert the file `test.wav` in Music folder under current user's home directory into `output.ogg`

```
>>> from speach import media
>>> media.convert("~/Music/test.wav", "~/Music/output.ogg")
```

speach.media.**cut**(*infile*, *outfile*, *from_ts=None*, *to_ts=None*, *use_concat=False*, **args*, ***kwargs*)

Cut a media file from a timestamp to another timestamp

To cut myfile.wav from `00:03:12` to the end of the file and write output to outfile.ogg

```
>>> media.cut("myfile.wav", "outfile.ogg", from_ts="00:03:12")
```

To cut myfile.wav from the beginning to `00:04:27` and then write output to outfile.ogg

```
>>> media.cut("myfile.wav", "outfile.ogg", to_ts="00:04:27")
```

When use_concat is set to True, both from_ts and to_ts must be specified

```
>>> media.cut("myfile.wav", "outfile.ogg", from_ts="00:03:12", to_ts="00:04:27",␣
↪use_concat=True)
```

Parameters

- **infile** – Path to an existing file (in str or Path-like object)
- **outfile** – Path to output file (must not exist, or else a FileExistsError will be raised)
- **from_ts** (*a timestamp string or a TimeSlot object*) – Leave as None to start cutting from the beginning
- **to_ts** (*a timestamp string or a TimeSlot object*) – Timestamp to end cutting. Leave as None to cut to the end of the file
- **use_concat** – Set to True to use demuxer to cut audio file. Both from_ts and to_ts must be specified when use. Defaulted to None

speach.media.**locate_ffmpeg**()

locate the binary file of ffmpeg program (i.e. ffmpeg.exe)

```
>>> from speach import media
>>> media.locate_ffmpeg()
'/usr/bin/ffmpeg'
```

speach.media.**metadata**(*infile*, **args*, *ffmpeg_path=None*)

Read metadata of a given media file

speach.media.**version**(*ffmpeg_path=None*)

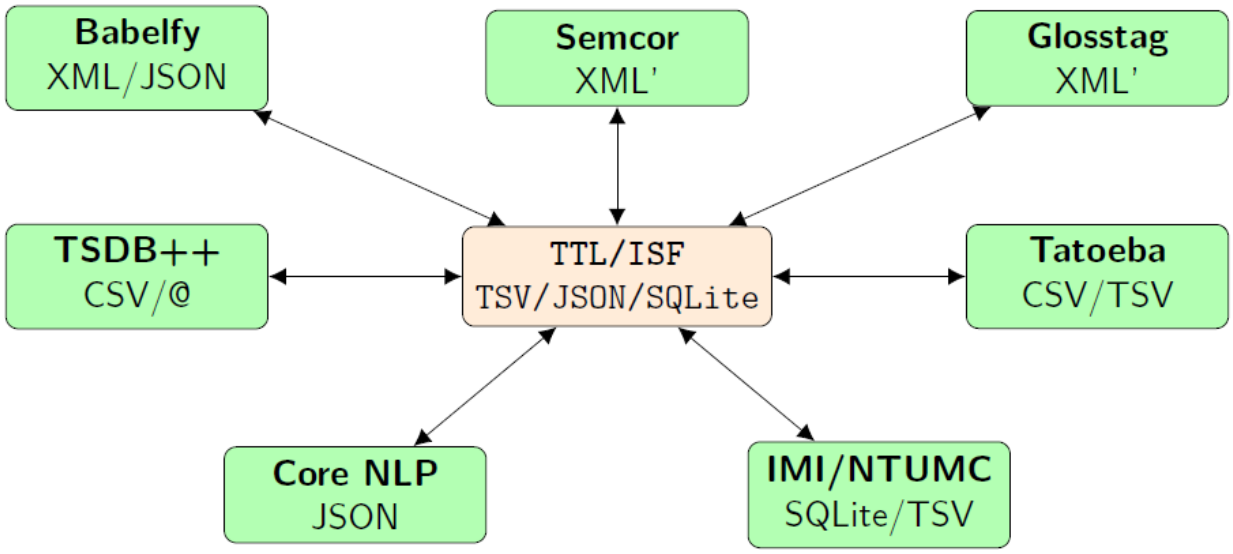
Determine using ffmpeg version

```
>>> from speach import media
>>> media.version()
'4.2.4-1ubuntu0.1'
```

texttaglib module

TTL (abbreviated from `texttaglib`) is a Python implementation of the corpus linguistic method described in *Tuan Anh (2019)*. TTL was designed to be a robust linguistic documentation framework which is flexible enough to handle linguistic data from different sources (Core NLP, ELAN, CoNLL, Semcor, Babelfy, Glosstag Wordnet, Tatoeba project, TSDB++, to name a few).

TTL can be used as a data interchange format for converting to and from different data formats.



Text corpus

```
>>> from speach import ttl
>>> doc = ttl.Document('mydoc')
>>> sent = doc.new_sent("I am a sentence.")
>>> sent
#1: I am a sentence.
>>> sent.ID
1
>>> sent.text
'I am a sentence.'
>>> sent.import_tokens(["I", "am", "a", "sentence", "."])
>>> >>> sent.tokens
[`I`<0:1>, `am`<2:4>, `a`<5:6>, `sentence`<7:15>, `.`<15:16>]
>>> doc.write_ttl()
```

The script above will generate this corpus

```
-rw-rw-r--.  1 tuananh tuananh       0  3 29 13:10 mydoc_concepts.txt
-rw-rw-r--.  1 tuananh tuananh       0  3 29 13:10 mydoc_links.txt
-rw-rw-r--.  1 tuananh tuananh      20  3 29 13:10 mydoc_sents.txt
-rw-rw-r--.  1 tuananh tuananh       0  3 29 13:10 mydoc_tags.txt
-rw-rw-r--.  1 tuananh tuananh      58  3 29 13:10 mydoc_tokens.txt
```

TIG - TTL Interlinear Gloss format

TIG is a human friendly interlinear gloss format that can be edited using any text editor.

TTL SQLite

TTL supports SQLite storage format to manage large scale corpuses.

References

- Le, T. A. (2019). *Developing and applying an integrated semantic framework for natural language understanding (pp. 69-78)*. DOI:10.32657/10220/49370

Web VTT module

Speach supports Web VTT - The Web Video Text Tracks Format. Read more about it at: https://www.w3.org/2013/07/webvtt.html

Web Video Text Tracks format (WebVTT) support

More information:

WebVTT: The Web Video Text Tracks Format https://www.w3.org/2013/07/webvtt.html

`speach.vtt.`**`sec2ts`**(*seconds: float*) → str

Convert duration in seconds in seconds to VTT format (e.g. 01:53:47.262)

```
>>> from speach import vtt
>>> vtt.sec2ts(25.403)
'00:00:25.403'
>>> vtt.sec2ts(14532.768)
'04:02:12.768'
```

Parameters **`seconds`** – Timestamp in seconds

Returns Web VTT formatted timestamp

Return type str

Raises `ValueError`: when seconds is not a number or cannot be casted into a number

`speach.vtt.`**`ts2sec`**(*ts: str*) → float

Convert VTT timestamp to duration (float, in seconds)

```
>>> from speach import vtt
>>> vtt.ts2sec("01:00:41.231")
3641.231
>>> print(vtt.ts2sec("00:00:00.000"))
0.0
```

Parameters **`ts`** (*str*) – Timestamp (hh:mm:ss.nnn)

Returns timestamp in seconds

Return type float

Raises `ValueError`: if timestamp is not formatted correctly

2.4 Contributing

There are many ways to contribute to `speach`. Currently development team are focusing on:

- Improving *documentation*
- Improving ELAN support
- Fixing bugs *existing bugs*

If you have some suggestions or bug reports, please share on speach issues tracker.

2.4.1 Code of conduct

Please read our *contributor code of conduct* for more information.

2.4.2 Fixing bugs

If you found a bug please report at https://github.com/neocl/speach/issues

When it is possible, please also share how to reproduce the bugs to help with the bug finding process.

Pull requests are welcome.

2.4.3 Updating Documentation

1. Fork speach repository to your own Github account.
2. Clone *speach* repository to your local machine.

```
git clone https://github.com/<your-account-name>/speach
```

3. Create a virtual environment (optional, but highly recommended)

```
# if you use virtualenvwrapper
mkvirtualenv speach
workon speach

# if you use Python venv
python3 -m venv .env
. .env/bin/activate
python3 -m pip install --upgrade pip wheel Sphinx
```

4. Build the docs

```
cd docs
# compile the docs
make dirhtml
# serve the docs using Python3 built-in development server
```

(continues on next page)

(continued from previous page)

```
# Note: this requires Python >= 3.7 to support --directory
python3 -m http.server 7000 --directory _build/dirhtml

# if you use earlier Python 3, you may use
cd _build/dirhtml
python3 -m http.server 7000

# if you use Windows, you may use
python -m http.server 7000 --directory _build/dirhtml
```

5. Now the docs should be ready to view at http://localhost:7000 . You can visit that URL on your browser to view the docs.
6. More information:
   - Sphinx tutorial: https://sphinx-tutorial.readthedocs.io/start/
   - Using *virtualenv*: https://virtualenvwrapper.readthedocs.io/en/latest/install.html
   - Using *venv*: https://docs.python.org/3/library/venv.html

2.4.4 Development

Development contributions are welcome. Setting up development environment for speach should be similar to *Updating Documentation*.

Please contact the development team if you need more information: https://github.com/neocl/speach/issues

2.5 Contributor Covenant Code of Conduct

2.5.1 Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to make participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

2.5.2 Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances

- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

2.5.3 Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

2.5.4 Scope

This Code of Conduct applies within all project spaces, and it also applies when an individual is representing the project or its community in public spaces. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

2.5.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project team at <neocl@dakside.org>. All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

2.5.6 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 1.4, available at https://www.contributor-covenant.org/version/1/4/code-of-conduct.html

For answers to common questions about this code of conduct, see https://www.contributor-covenant.org/faq

2.6 Speach Changelog

2.6.1 Speach 0.1a15

- 2022-03-16
  - Use chirptext >= 0.2a6.post1
  - Infer timestamps and duratio of ref_annotations from linked annotations
  - Fixed Symbolic Association linking bug
  - Use explicit None checking for timestamps and duration info

  - Add `elan.Doc.clone()` function
  - Other bug fixes
- 2022-03-15
  - Use `chirptext` >= 0.2a6 for Python 3.10 and Python 3.11 support
  - Fixed missing lemmas bug in `ttl_to_igrow()`
  - Fixed `_xml_tostring()` method for Python < 3.8

2.6.2 Speach 0.1a14

- 20022-03-09
  - Cross-check with controlled vocabularies when creating annotations if possible
- 2022-03-07
  - Allow to add new annotations to all 5 tier stereotypes
    - None (default root tiers)
    - Included In
    - Time Subdivision
    - Symbolic Subdivision
    - Symbolic Association
  - Support `lxml` when available
  - Support `pretty_print` in `elan.Doc.to_xml_str()` and `elan.Doc.to_xml_bin()`
- 2022-03-02
  - Add `speach.elan.create()` function for creating a new ELAN file from scratch
- 2022-02-28
  - Add `LOCALE` support
  - Add Speach logo

2.6.3 Speach 0.1a13

- 2022-01-14
  - Use `defusedxml` automatically when available to parse XML for better security

2.6.4 Speach 0.1a12

- 2021-11-03
  - **Support controlled vocabularies editing**
    - Add new controlled vocabulary entries
    - Remove controlled vocabulary entries
    - Edit controlled vocabularies (values, desciptions, languages)

  - Add crc32 helper functions to `speach.media` module

2.6.5 Speach 0.1a11

- 2021-08-26
  - Add `encoding` option to `eaf2csv` command

2.6.6 Speach 0.1a10

- 2021-07-27
  - Support editing ELAN media fields
    - media_file
    - time_units
    - media_url
    - mime_type
    - relative_media_url

2.6.7 Speach 0.1a9

- 2021-05-27
  - Use TTLv2 API (chirptext >= 0.2a4.post1)

2.6.8 Speach 0.1a8

- 2021-05-27: Added some basic ELAN editing functions
  - Update participant codes
  - Update tier names
  - Update annotation texts

2.6.9 Speach 0.1a7

- 2021-04-30
  - Added `speach.elan.ELANDoc.cut()` function to cut annotations to separate audio files.
  - Expand user home directory automatically when using *speach.elan.read_eaf()* function.
  - Module *speach.media* supports converting media files, cutting them by timestamps, and demuxer concat.
  - Package *Victoria Chua*'s media processing code into `speach.media` module.
- 2021-04-28
  - Initial release on PyPI

CHAPTER

THREE

USEFUL LINKS

- Soure code: https://github.com/neocl/speach/
- Speach on PyPI: https://pypi.org/project/speach/
- Speach documentation: https://speach.readthedocs.io/

CHAPTER

FOUR

RELEASE NOTES

Release notes is available *here*.

CHAPTER

FIVE

CONTRIBUTORS

- Le Tuan Anh (Maintainer)
- Victoria Chua

5.1 Graphic materials

The Speach logo () was created by using the snake emoji (created by Selina Bauder) and the peach emoji (created by Marius Schnabel) from Openmoji project. License: CC BY-SA 4.0

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX

T

V