Subsy

Release 1.0.0

John Hennig

Dec 20, 2022

CONTENTS

Access to subtitles from various file formats

This library is not fundamentally different from established ones, but offers some helpful abstractions that those others don't. First and foremost, subtitles loaded from a file are represented as a linked list. This makes it possible to implement search patterns and sanitization strategies that take the preceding or following subtitle into account, for example to recognize a running sentence.

```
>>> import subsy
>>> subtitles = subsy.load('subtitles.srt')
>>> first = subtitles[0]
>>> first.text
'How are you?'
>>> second = first.next
>>> second.text
'great, thanks.'
>>> second.text = 'Great, thanks.'
>>> subtitles.save()
```

Subtitles can be loaded from and saved to these file formats:

- Subrip (`.srt`)
- Advanced Substation Alpha (`.ass`)
- Substation Alpha (`.ssa`)
- WebVTT (`.vtt`)
- SubViewer (`.sub`)

The text encoding of input files is detected automatically.

CHAPTER

ONE

INSTALLATION

Subsy is available on PyPI and can be readily installed via

```
pip install subsy
```

Pip will automatically install the following dependencies:

- Srt3 — For reading subtitles in the SubRip format.
- Aeidon — For reading and writing various other formats.
- Chardet — To detect text encoding of input files.

Run `pip uninstall subsy` in order to remove the package from your system, though note that this will not uninstall the dependencies.

CHAPTER

TWO

TUTORIAL

You can download the subtitles file used in this tutorial from the library's source-code repository (where it is part of the automated test suite). If we start the Python interpreter in the same folder as the downloaded file, it can be loaded like so:

```
>>> import subsy
>>> subtitles = subsy.load('reference.srt')
```

The `load()` functions returns a `Subtitles` object. It is basically a list of the individual subtitles:

```
>>> len(subtitles)
46
>>> subtitle = subtitles[0]
>>> subtitle
Subtitle(00:00:00.000 → 00:00:01.234: "Just a single line of text.")
```

But it is a linked list. That is, it provides additional funtionality to go from one subtitle to the next one, or back to the previous one.

```
>>> subtitle = subtitle.next
>>> subtitle
Subtitle(00:00:02.000 → 00:00:02.900: "Text extending over", "two lines.")
>>> subtitle.previous
Subtitle(00:00:00.000 → 00:00:01.234: "Just a single line of text.")
```

This can be useful when cleaning up subtitles, for example to recognize running sentences when correcting improper capitalization.

The individual subtitles do of course have time stamps. These can be accessed, and also changed, in either milliseconds or in a text-based format.

```
>>> subtitle.start
2000
>>> subtitle.start_time
'00:00:02.000'
>>> subtitle.duration
900
>>> subtitle.end_time
'00:00:02.900'
>>> subtitle.end_time = 3000
>>> subtitle.duration
1000
>>> subtitle.start = 2500
```

(continues on next page)

(continued from previous page)

```
>>> subtitle.duration
1000
>>> subtitle.end
3500
```

Note how when we set the end time, the duration changes accordingly. But when we assign a new start time, the duration remains the same and the end time shifts along.

Text can either be accessed as individual lines or as a `\n`-separated string.

```
>>> subtitle.lines
['Text extending over', 'two lines.']
>>> subtitle.text
'Text extending over\ntwo lines.'
```

Changing one also changes the other.

```
>>> subtitle.text = subtitle.text.upper()
>>> subtitle.text
'TEXT EXTENDING OVER\nTWO LINES.'
>>> subtitle.lines
['TEXT EXTENDING OVER', 'TWO LINES.']
```

Text may contain markup, of the SubRip flavor familiar from `.srt` files.

```
>>> subtitle = subtitles[16]
>>> subtitle.text
'<i>Two lines of text,</i>\n<i>separately in italics.</i>'
```

Sometimes we want the plain text without the markup.

```
>>> subtitle.plain
'Two lines of text,\nseparately in italics.'
```

The character length of the plain text is also reported as the length of the subtitle.

```
>>> len(subtitle)
41
>>> len(subtitle.plain)
41
```

CHAPTER

THREE

API

Code documentation of the public application programming interface provided by this library.

load	Loads subtitles from a file.
Subtitles	Represents a sequence of subtitles.
Subtitle	Represents a single subtitle.

3.1 load

load(*file*, *format=None*, *encoding=None*)

Loads subtitles from a file.

`file` is preferably a `pathlib.Path` object, but may also be a string denoting an absolute or relative file path.

The file `format`, if not explicitly specified, is deduced from the file ending. The file's text `encoding` can be given, but is otherwise detected automatically (which may fail in some rare cases).

Returns a `Subtitles` object.

3.2 Subtitles

class Subtitles(*subtitles=None*, *source=None*)

Represents a sequence of subtitles.

Example:

```
>>> import subsy
>>> subtitles = subsy.load('reference.srt')
>>> len(subtitles)
```

subtitles

List of `Subtitle` instances that make up the sequence.

source

Dictionary holding meta information about the subtitles's source.

This is usually a file, and the dictionary would contain a key named `'file'` with a `pathlib.Path` object for its value pointing to the file-system location that the subtitles were loaded from. A second key named `'encoding'` would store the text encoding, as automatically detected by `subsy.load()` or passed in to it specifically.

These two entries, if they exist, are used by the `save()` method to update the file (if called without arguments).

save(*file=None*, *encoding=None*, *format=None*)

Saves the subtitles to disk.

An optional `file` can be specified (as a `pathlib.Path` object or string) to indicate the path and file name. If not given, the `'file'` reference will be retrieved from the meta information stored in the `source` attribute. Failing that, a `ValueError` is raised.

An optional `encoding` can be specified. All valid encoding names are supported. If not given, the value for `'encoding'` will be retrieved from the `source` meta info. If the key is not found, it defaults to `'UTF-8-sig'`, i.e. variable-length Unicode with signature / byte-order mark.

An optional `format` can be specified, which would otherwise be deduced from the file ending.

3.3 Subtitle

class Subtitle(*lines=None*, *start=0*, *duration=1000*, *parent=None*, *index=None*, *next=None*, *previous=None*)

Represents a single subtitle.

Example:

```
>>> import subsy
>>> subtitle = subsy.Subtitle(['First line.', 'Line <i>with</i> markup.'])
>>> subtitle.lines
['First line.', 'Line <i>with</i> markup.']
>>> subtitle.text
'First line.\nLine <i>with</i> markup.'
>>> subtitle.plain
'First line.\nLine with markup.'
>>> subtitle.start
0
>>> subtitle.start_time
'00:00:00.000'
>>> subtitle.end_time
'00:00:01.000'
>>> subtitle.end_time = '00:00:02.000'
>>> subtitle.end
2000
```

lines

List of individual lines of the subtitle's text.

start

Start time in milliseconds.

duration

Duration in milliseconds.

parent

Sequence, if any, that this subtitle belongs to.

index

Index number if part of a sequence.

next

Next subtitle if part of a sequence.

previous

Previous subtitle if part of a sequence.

property prev

Alias for `previous`.

property start_time

Start time in time-stamp format `hh:mm:ss.ms`.

property end_time

End time in time-stamp format `hh:mm:ss.ms`.

property text

The entire text of the subtitle, all lines joined together.

property plain

The entire text, but any markup removed.

CHAPTER

FOUR

RELEASES

4.1 1.0.0

- Published on October 15, 2021.
- Fixed typos in meta information.
- Changed development status to “stable”.

4.2 0.9.2

- Published on October 3, 2021.
- Fixed some code smells.
- Changed development status to “beta”.

4.3 0.9.1

- Published on September 30, 2021.
- Cosmetic changes to code and documentation.
- Don’t refer to Windows-1252 encoding as ANSI.

4.4 0.9.0

- Published on September 29, 2021.
- Initial release.

INDEX