
PyonFX Documentation

Release 0.9.2

Antonio Strippoli (CoffeeStraw/YellowFlash)

Jul 18, 2020

Contents

1	Quick Start Guide	2
1.1	Windows	2
1.2	Ubuntu/Debian	3
1.3	Fedora	3
1.4	Arch Linux	3
1.5	openSUSE	3
1.6	macOS	3
1.7	Installation - Extra Step	4
1.8	Starting	4
1.9	Tips	4
2	The PyonFX Library Reference	5
2.1	Ass Core	5
2.2	Convert Functions	17
2.3	Shape Functions	21
2.4	Utils	24
	Python Module Index	28
	Index	29

“PyonFX is an easy way to do KFX (Karaoke Effects) and complex typesetting based on subtitle format ASS (Advanced Substation Alpha).”

PyonFX is a Python library that helps you to combine tags, text and shapes in ASS format, eventually changing times and layers, with the help of all the automatically extracted data from an input file in ASS format.

That’s not all, it also offers some special functions to help you in some tricky tasks, for example frame per frame operations, shape manipulations or working pixel by pixel, so it’s easier to create **impressive visual 2D effects**.

In comparison to other karaoke effect programs, PyonFX is written in [Python3](#), which means that you will have all the advantages provided by this modern scripting language, constantly updated and perfect for both beginner and advanced users.

You can learn on how to start using the library on the [Quick Start Guide](#) section.

CHAPTER 1

Quick Start Guide

First of all, you need to know how you're creating what. You will need to learn (if you've not already) the following:

- **ASS format.** PyonFX is still an advanced tool for typesetter and karaokers, it is meant to be used by experienced typesetters that knows all the tags rendered by Libass. Check the footnote¹ for a complete list of all the tags.
- **Python3 scripting language.** A programming language like Python offers you to define what you want in which case, how often, attended to this or that... Basically you're more free. You're not limited to buttons, sliders or text fields with just a few commands in a completely graphical interface. **The basics are enough.** Variables, functions, conditions, loops, comparisons, string formatting, list and dictionaries... You can find the link to some good tutorials in the footnote².

To start generating, you will only have to write a script in Python3, which will describes the process of your KFX or advanced typesetting creation.

If you have trouble with the installation of Python, you can check some online guide, like <https://realpython.com/installing-python/>.

1.1 Windows

So, if you've not installed it before, you will have to **install Python3**. You can **download** it from the [official website](#). Just be sure to check the box that says "Add Python 3.x to PATH". This is really important to avoid some extra steps that would make Python callable in every directory from the command prompt.

Run this command below, which will use pip to install and eventually update the library:

```
pip install --upgrade https://github.com/CoffeeStraw/PyonFX/zipball/master
```

That's all. Nothing else is needed, every time you will have to update, just run again this command.

¹ List of all ASS tags with usage explanation: http://docs.aegisub.org/3.2/ASS_Tags/

² Suggested tutorials for learning Python3:

- Italian: https://github.com/AllenDowney/ThinkPythonItalian/blob/master/thinkpython_italian.pdf
- English: <http://greenteapress.com/thinkpython2/thinkpython2.pdf>

1.2 Ubuntu/Debian

Warning: The first of the following commands is not well tested. If you run into any problems, please create an issue or refer to the [official installation guide](#).

```
sudo apt install python3 python3-pip libgirepository1.0-dev gcc libcairo2-dev pkg-
↳config python3-dev gir1.2-gtk-3.0 python3-gi python3-gi-cairo
python3 -m pip install --upgrade https://github.com/CoffeeStraw/PyonFX/zipball/master
```

1.3 Fedora

Warning: The first of the following commands is not well tested. If you run into any problems, please create an issue or refer to the [official installation guide](#).

```
sudo dnf install python3 python3-pip gcc gobject-introspection-devel cairo-devel pkg-
↳config python3-devel python3-gobject gtk3
python3 -m pip install --upgrade https://github.com/CoffeeStraw/PyonFX/zipball/master
```

1.4 Arch Linux

Warning: The first of the following commands is not well tested. If you run into any problems, please create an issue or refer to the [official installation guide](#).

```
sudo pacman -S python python-pip cairo pkgconf gobject-introspection python-gobject_
↳gtk3
python3 -m pip install --upgrade https://github.com/CoffeeStraw/PyonFX/zipball/master
```

1.5 openSUSE

Warning: The first of the following commands is not well tested. If you run into any problems, please create an issue or refer to the [official installation guide](#).

```
sudo zypper install python3 python3-pip cairo-devel pkg-config python3-devel gcc_
↳gobject-introspection-devel python3-gobject python3-gobject-Gdk typelib-1_0-Gtk-3_0_
↳libgtk-3-0
python3 -m pip install --upgrade https://github.com/CoffeeStraw/PyonFX/zipball/master
```

1.6 macOS

You may need to install [Homebrew](#) first.

Warning: The first of the following commands is not well tested. If you run into any problems, please create an issue or refer to the [official installation guide](#).

```
brew install pygobject3 gtk+3 cairo py3cairo pkg-config
python3 -m pip install --upgrade https://github.com/CoffeeStraw/PyonFX/zipball/master
```

Warning: If you experience output not rendered correctly, you might need to change the PangoCairo backend to fontconfig.

```
PANGOCAIRO_BACKEND=fc python3 namefile.py
```

1.7 Installation - Extra Step

This step is not needed to start working with the library, but personally I consider Aegisub quite old and heavy, so I needed a more comfortable way to work.

That's why PyonFX integrates an additional way to reproduce your works in softsub faster after each generation, using the [MPV player](#). Installing it should be enough to make everything work if you're NOT on Windows.

If you're on Windows, all you need to do once you have installed it (check the website for that), is to add it to the PATH, so that the library will be able to utilize it. There are several guide for that, [here you can find one](#).

You need to add the folder that contains the .exe of mpv, generally C:\Program Files\mpv.

1.8 Starting

You may want to check if everything is working nicely now. For that, I suggest you to try running some of the examples in the [GitHub official repository of the project](#).

To run a script in python, all you need to do is run the following command:

```
python namefile.py
```

Or if this is not working for some reason (like you're not on Windows and both Python2 and Python3 are installed):

```
python3 namefile.py
```

I highly suggest you to generate and study every single example in this examples folder (download always up-to-date [here](#)). These are meant for absolute beginners until advanced users and explain in detail the usage of all the relevant functions of the library.

1.9 Tips

- Don't make a KFX in one go. Make pauses, go for a walk, collect ideas from your surroundings;
- Pick elements of the video. Your effect should merge with the background in some manner;
- Consider human recognition. Mostly we notice motion, then contrasts, then colors. Too much can give a headache, too less is boring;
- Use modern styles to impress (light, curves, particles, gradients) and old ones for readability (solid colors, thick borders, static positions);
- When background is too flashy, try to insert a panel shape to put your text on 'safe terrain';
- Adjust to karaoke times and voice. Fast sung lines haven't syllable durations for effects which need some time to get seen.

The PyonFX Library Reference

This reference manual describes all the classes and functions provided by the library. It is terse, but attempts to be exact and complete.

For ASS parsing functions and object's classes, you can go on *Ass Core* section.

For Convert functions usefull to convert everything based on ASS format to something more comfortable (and the other way around), you can go on *Convert Functions* section.

For Shape functions that will let you do complex calculations with shapes in ASS format, you can go on *Shape Functions* section.

For general utility functions, you can go on *Utils* section.

2.1 Ass Core

class `pyonfx.ass_core.Meta`

Meta object contains informations about the Ass.

More info about each of them can be found on <http://docs.aegisub.org/manual/Styles>

wrap_style

Determines how line breaking is applied to the subtitle line

Type int

scaled_border_and_shadow

Determines if it has to be used script resolution (*True*) or video resolution (*False*) to scale border and shadow

Type bool

play_res_x

Video Width

Type int

play_res_y
Video Height

Type int

audio
Loaded audio path (absolute)

Type str

video
Loaded video path (absolute)

Type str

class pyonfx.ass_core.**Style**
Style object contains a set of typographic formatting rules that is applied to dialogue lines.
More info about styles can be found on http://docs.aegisub.org/3.2/ASS_Tags/.

fontname
Font name

Type str

fontsize
Font size in points

Type float

color1
Primary color (fill)

Type str

alpha1
Trasparency of color1

Type str

color2
Secondary color (secondary fill, for karaoke effect)

Type str

alpha2
Trasparency of color2

Type str

color3
Outline (border) color

Type str

alpha3
Trasparency of color3

Type str

color4
Shadow color

Type str

alpha4

Trasparency of color4

Type str

bold

Font with bold

Type bool

italic

Font with italic

Type bool

underline

Font with underline

Type bool

strikeout

Font with strikeout

Type bool

scale_x

Text stretching in the horizontal direction

Type float

scale_y

Text stretching in the vertical direction

Type float

spacing

Horizontal spacing between letters

Type float

angle

Rotation of the text

Type float

border_style

True for opaque box, *False* for standard outline

Type bool

outline

Border thickness value

Type float

shadow

How far downwards and to the right a shadow is drawn

Type float

alignment

Alignment of the text

Type int

margin_l

Distance from the left of the video frame

Type int

margin_r
Distance from the right of the video frame

Type int

margin_v
Distance from the bottom (or top if alignment >= 7) of the video frame

Type int

encoding
Codepage used to map codepoints to glyphs

Type int

class `pyonfx.ass_core.Char`
Char object contains informations about a single char of a line in the Ass.
A char is defined by some text between two karaoke tags (k, ko, kf).

i
Char index number

Type int

word_i
Char word index (e.g.: In line text Hello PyonFX users!, letter “u” will have word_i=2).

Type int

syl_i
Char syl index (e.g.: In line text {\k0}Hel{\k0}lo {\k0}Pyon{\k0}FX {\k0}users!, letter “F” will have syl_i=3).

Type int

syl_char_i
Char individual syl index (e.g.: In line text {\k0}Hel{\k0}lo {\k0}Pyon{\k0}FX {\k0}users!, letter “e” of “users” will have syl_char_i=2).

Type int

start_time
Char start time (in milliseconds).

Type int

end_time
Char end time (in milliseconds).

Type int

duration
Char duration (in milliseconds).

Type int

styleref
Reference to the Style object of this object original line.

Type obj

text
Char text.

Type str

inline_fx

Char inline effect (marked as -EFFECT in karaoke-time).

Type str

prespace

Char free space before text.

Type int

postspace

Char free space after text.

Type int

width

Char text width.

Type float

height

Char text height.

Type float

x

Char text position horizontal (depends on alignment).

Type float

y

Char text position vertical (depends on alignment).

Type float

left

Char text position left.

Type float

center

Char text position center.

Type float

right

Char text position right.

Type float

top

Char text position top.

Type float

middle

Char text position middle.

Type float

bottom

Char text position bottom.

Type float

class pyonfx.ass_core.Syllable

Syllable object contains informations about a single syl of a line in the Ass.

A syl can be defined as some text after a karaoke tag (k, ko, kf) (e.g.: In {\k0}Hel{\k0}lo {\k0}Pyon{\k0}FX {\k0}users!, “Pyon” and “FX” are distinct syllables),

i

Syllable index number

Type int

word_i

Syllable word index (e.g.: In line text {\k0}Hel{\k0}lo {\k0}Pyon{\k0}FX {\k0}users!, syl “Pyon” will have word_i=1).

Type int

start_time

Syllable start time (in milliseconds).

Type int

end_time

Syllable end time (in milliseconds).

Type int

duration

Syllable duration (in milliseconds).

Type int

styleref

Reference to the Style object of this object original line.

Type obj

text

Syllable text.

Type str

tags

All the remaining tags before syl text apart k ones.

Type str

inline_fx

Syllable inline effect (marked as -EFFECT in karaoke-time).

Type str

prespace

Syllable free space before text.

Type int

postspace

Syllable free space after text.

Type int

width

Syllable text width.

Type float

height

Syllable text height.

Type float

x

Syllable text position horizontal (depends on alignment).

Type float

y

Syllable text position vertical (depends on alignment).

Type float

left

Syllable text position left.

Type float

center

Syllable text position center.

Type float

right

Syllable text position right.

Type float

top

Syllable text position top.

Type float

middle

Syllable text position middle.

Type float

bottom

Syllable text position bottom.

Type float

class pyonfx.ass_core.**Word**

Word object contains informations about a single word of a line in the Ass.

A word can be defined as some text with some optional space before or after (e.g.: In the string “What a beautiful world!”, “beautiful” and “world” are both distinct words).

i

Word index number

Type int

start_time

Word start time (same as line start time) (in milliseconds).

Type int

end_time

Word end time (same as line end time) (in milliseconds).

Type int

duration

Word duration (same as line duration) (in milliseconds).

Type int

styleref

Reference to the Style object of this object original line.

Type obj

text

Word text.

Type str

prespace

Word free space before text.

Type int

postspace

Word free space after text.

Type int

width

Word text width.

Type float

height

Word text height.

Type float

x

Word text position horizontal (depends on alignment).

Type float

y

Word text position vertical (depends on alignment).

Type float

left

Word text position left.

Type float

center

Word text position center.

Type float

right

Word text position right.

Type float

top

Word text position top.

Type float

middle

Word text position middle.

Type float

bottom

Word text position bottom.

Type float

class pyonfx.ass_core.Line

Line object contains informations about a single line in the Ass.

Note: (*) = This field is available only if *extended* = True

i

Line index number

Type int

comment

If *True*, this line will not be displayed on the screen.

Type bool

layer

Layer for the line. Higher layer numbers are drawn on top of lower ones.

Type int

start_time

Line start time (in milliseconds).

Type int

end_time

Line end time (in milliseconds).

Type int

duration

Line duration (in milliseconds) (*).

Type int

leadin

Time between this line and the previous one (in milliseconds; first line = 1000.1) (*).

Type float

leadout

Time between this line and the next one (in milliseconds; first line = 1000.1) (*).

Type float

style

Style name used for this line.

Type str

styleref

Reference to the Style object of this line (*).

Type obj

actor

Actor field.

Type str

margin_l
Left margin for this line.

Type int

margin_r
Right margin for this line.

Type int

margin_v
Vertical margin for this line.

Type int

effect
Effect field.

Type str

raw_text
Line raw text.

Type str

text
Line stripped text (no tags).

Type str

width
Line text width (*).

Type float

height
Line text height (*).

Type float

ascent
Line font ascent (*).

Type float

descent
Line font descent (*).

Type float

internal_leading
Line font internal lead (*).

Type float

external_leading
Line font external lead (*).

Type float

x
Line text position horizontal (depends on alignment) (*).

Type float

y

Line text position vertical (depends on alignment) (*).

Type float**left**

Line text position left (*).

Type float**center**

Line text position center (*).

Type float**right**

Line text position right (*).

Type float**top**

Line text position top (*).

Type float**middle**

Line text position middle (*).

Type float**bottom**

Line text position bottom (*).

Type float**words**List containing objects *Word* in this line (*).**Type** list**sylys**List containing objects *Syllable* in this line (if available) (*).**Type** list**chars**List containing objects *Char* in this line (*).**Type** list**copy()****Returns** A deep copy of this object (line)

class pyonfx.ass_core.**Ass** (*path_input*=", *path_output*='Output.ass', *keep_original*=True, *extended*=True, *vertical_kanji*=True)

Contains all the informations about a file in the ASS format and the methods to work with it for both input and output.

Usually you will create an Ass object and use it for input and output (see *example* section).

PyonFX set automatically an absolute path for all the info in the output, so that wherever you will put your generated file, it should always load correctly video and audio.

Parameters

- **path_input** (*str*) – Path for the input file (either relative to your .py file or absolute).
- **path_output** (*str*) – Path for the output file (either relative to your .py file or absolute) (DEFAULT: “Output.ass”).
- **keep_original** (*bool*) – If True, you will find all the lines of the input file commented before the new lines generated.
- **extended** (*bool*) – Calculate more informations from lines (usually you will not have to touch this).
- **vertical_kanji** (*bool*) – If True, line text with alignment 4, 5 or 6 will be positioned vertically.

path_input

Path for input file (absolute).

Type *str*

path_output

Path for output file (absolute).

Type *str*

meta

Contains informations about the ASS given.

Type *Meta*

styles

Contains all the styles in the ASS given.

Type list of *Style*

lines

Contains all the lines (events) in the ASS given.

Type list of *Line*

Example

```
io = Ass("in.ass")
meta, styles, lines = io.get_data()
```

get_data()

Utility function to retrieve easily meta styles and lines.

Returns *meta*, *styles* and *lines*

write_line(line)

Appends a line to the output list (which is private) that later on will be written to the output file when calling save().

Use it whenever you’ve prepared a line, it will not impact performance since you will not actually write anything until *save()* will be called.

Parameters *line* (*Line*) – A line object. If not valid, TypeError is raised.

save(quiet=False)

Write everything inside the private output list to a file.

Parameters `quiet` (*bool*) – If True, you will not get printed any message.

open_aegisub ()

Open the output (specified in `self.path_output`) with Aegisub.

This can be usefull if you don't have MPV installed or you want to look at your output in detailed.

Returns 0 if success, -1 if the output couldn't be opened.

open_mpv (*video_path*=", *video_start*", *full_screen*=False)

Open the output (specified in `self.path_output`) in softsub with the MPV player. To utilize this function, MPV player is required. Additionally if you're on Windows, MPV must be in the PATH (check <https://pyonfx.readthedocs.io/en/latest/quick%20start.html#installation-extra-step>).

This is one of the fastest way to reproduce your output in a comfortable way.

Parameters

- **video_path** (*string*) – The video file path (absolute) to reproduce. If not specified, `meta.video` is automatically taken.
- **video_start** (*string*) – The start time for the video (more info: <https://mpv.io/manual/master/#options-start>). If not specified, 0 is automatically taken.
- **full_screen** (*bool*) – If True, it will reproduce the output in full screen. If not specified, False is automatically taken.

2.2 Convert Functions

class `pyonfx.convert.Convert`

This class is a collection of static methods that will help the user to convert everything needed to the ASS format.

static time (*ass_ms*)

Converts between milliseconds and ASS timestamp.

You can probably ignore that function, you will not make use of it for KFX or typesetting generation.

Parameters `ass_ms` (*either int or str*) – If int, than milliseconds are expected, else ASS timestamp as str is expected.

Returns If milliseconds -> ASS timestamp, else if ASS timestamp -> milliseconds, else `ValueError` will be raised.

static coloralpha (*ass_r_a*, *g*=", *b*=", *a*=")

Converts between rgb color &/+ alpha numeric and ASS color &/+ alpha.

- Passing a string to this function, you want a conversion from ASS color+alpha, ASS color or ASS alpha to integer values;
- Passing a single number, you want a conversion from ASS alpha value to ASS alpha string;
- Passing 3 or 4 numbers, you want a conversion from rgb (or rgba) values to ASS color (or ASS color+alpha) string.

Parameters

- **ass_r_a** (*int or str*) – If a str is given, either an ASS color+alpha, ASS color, an ASS alpha string is expected; if an int is given, a value between 0 and 255 (inclusive) is expected.
- **g** (*int, optional*) – If given, a value between 0 and 255 (inclusive) is expected.

- **b** (*int*, *optional*) – If given, a value between 0 and 255 (inclusive) is expected.
- **a** (*int*, *optional*) – If given, a value between 0 and 255 (inclusive) is expected.

Returns According to the parameters, either a tuple containing rgb (or rgba) integer values or a str containing an ASS color+alpha, an ASS color or an ASS alpha.

Examples

```
print( Convert.coloralpha(0) )
print( Convert.coloralpha("&HFF&") )

print( Convert.coloralpha("&H0000FF&") )
print( Convert.coloralpha(255, 0, 0) )

print( Convert.coloralpha("&HFF00FF00") )
print( Convert.coloralpha(0, 255, 0, 255) )
```

```
>>> &H00&
>>> 255
>>> (255, 0, 0)
>>> &H0000FF&
>>> (0, 255, 0, 255)
>>> &HFF00FF00
```

static text_to_shape (*obj*, *fscx=None*, *fscy=None*)

Converts text with given style information to an ASS shape.

Tips: *You can easily create impressive deforming effects.*

Parameters

- **obj** (*Line*, *Word*, *Syllable* or *Char*) – An object of class Line, Word, Syllable or Char.
- **fscx** (*float*, *optional*) – The scale_x value for the shape.
- **fscy** (*float*, *optional*) – The scale_y value for the shape.

Returns A Shape object, representing the text with the style format values of the object.

Examples

```
line = Line.copy(lines[1])
line.text = "{\\an7\\pos(%.3f,%.3f)\\p1}%s" % (line.left, line.top, convert.
->text_to_shape(line))
io.write_line(line)
```

static text_to_clip (*obj*, *an=5*, *fscx=None*, *fscy=None*)

Converts text with given style information to an ASS shape, applying some translation/scaling to it since it is not possible to position a shape with pos() once it is in a clip.

This is an high level function since it does some additional operations, check text_to_shape for further informations.

Tips: *You can easily create text masks even for growing/shrinking text without too much effort.*

Parameters

- **obj** (*Line, Word, Syllable or Char*) – An object of class Line, Word, Syllable or Char.
- **an** (*integer, optional*) – The alignment wanted for the shape.
- **fscx** (*float, optional*) – The scale_x value for the shape.
- **fscy** (*float, optional*) – The scale_y value for the shape.

Returns A Shape object, representing the text with the style format values of the object.

Examples

```
line = Line.copy(lines[1])
line.text = "{\\an5\\pos(%.3f,%.3f)\\clip(%s)}%s" % (line.center, line.middle,
↪ convert.text_to_clip(line), line.text)
io.write_line(line)
```

static text_to_pixels (*obj, supersampling=8*)

Converts text with given style information to a list of pixel data.

A pixel data is a dictionary containing 'x' (horizontal position), 'y' (vertical position) and 'alpha' (alpha/transparency).

It is highly suggested to create a dedicated style for pixels, because you will write less tags for line in your pixels, which means less size for your .ass file.

The style suggested is:

- **an=7 (very important!);**
- **bord=0;**
- **shad=0;**
- For Font informations leave whatever the default is;

Tips: *It allows easy creation of text decaying or light effects.*

Parameters

- **obj** (*Line, Word, Syllable or Char*) – An object of class Line, Word, Syllable or Char.
- **supersampling** (*int*) – Value used for supersampling. Higher value means smoother and more precise anti-aliasing (and more computational time for generation).

Returns A list of dictionaries representing each individual pixel of the input text styled.

Examples

```
line = lines[2].copy()
line.style = "p"
p_sh = Shape.rectangle()
for pixel in Convert.text_to_pixels(line):
    x, y = math.floor(line.left) + pixel['x'], math.floor(line.top) + pixel['y']
    ↪ alpha = "\\alpha" + Convert.coloralpha(pixel['alpha']) if pixel['alpha'] !
    ↪ 255 else ""
```

(continues on next page)

(continued from previous page)

```
line.text = "{\p1\pos(%d,%d)%s}%s" % (x, y, alpha, p_sh)
io.write_line(line)
```

static shape_to_pixels (*shape*, *supersampling*=8)

Converts a Shape object to a list of pixel data.

A pixel data is a dictionary containing ‘x’ (horizontal position), ‘y’ (vertical position) and ‘alpha’ (alpha/transparency).

It is highly suggested to create a dedicated style for pixels, because you will write less tags for line in your pixels, which means less size for your .ass file.

The style suggested is:

- **an=7 (very important!);**
- bord=0;
- shad=0;
- For Font informations leave whatever the default is;

Tips: *As for text, even shapes can decay!*

Parameters

- **shape** (*Shape*) – An object of class Shape.
- **supersampling** (*int*) – Value used for supersampling. Higher value means smoother and more precise anti-aliasing (and more computational time for generation).

Returns A list of dictionaries representing each individual pixel of the input shape.

Examples

```
line = lines[2].copy()
line.style = "p"
p_sh = Shape.rectangle()
for pixel in Convert.shape_to_pixels(Shape.heart(100)):
    # Random circle to pixel effect just to show
    x, y = math.floor(line.left) + pixel['x'], math.floor(line.top) + pixel['y']
    alpha = "\alpha" + Convert.coloralpha(pixel['alpha']) if pixel['alpha'] != 255 else ""
    line.text = "{\p1\pos(%d,%d)%s\fad(0,%d)%s" % (x, y, alpha, 1.dur/4, p_sh)
    io.write_line(line)
```

2.3 Shape Functions

class `pyonfx.shape.Shape` (*drawing_cmds*)

This class can be used to define a Shape object (by passing its drawing commands) and then apply functions to it in order to accomplish some tasks, like analyzing its bounding box, apply transformations, splitting curves into segments...

Parameters `drawing_cmds` (*str*) – The shape's drawing commands in ASS format as a string.

has_error ()

Utility function that checks if the shape is valid.

Returns False if no error has been found, else a string with the first error encountered.

map (*fun*)

Sends every point of a shape through given transformation function to change them.

Tips: *Working with outline points can be used to deform the whole shape and make f.e. a wobble effect.*

Parameters `fun` (*function*) – A function with two (or optionally three) parameters. It will define how each coordinate will be changed. The first two parameters represent the x and y coordinates of each point. The third optional it represents the type of each point (move, line, bezier...).

Returns A pointer to the current object.

Examples

```
original = Shape("m 0 0 1 20 0 20 10 0 10")
print ( original.map(lambda x, y: (x+10, y+5) ) )
```

```
>>> m 10 5 1 30 5 30 15 10 15
```

bounding ()

Calculates shape bounding box.

Tips: *Using this you can get more precise information about a shape (width, height, position).*

Returns A tuple (x0, y0, x1, y1) containing coordinates of the bounding box.

Examples

```
print ("Left-top: %d %d\nRight-bottom: %d %d" % ( Shape("m 10 5 1 25 5 25 42 ↵
↵10 42").bounding() ) )
```

```
>>> Left-top: 10 5
>>> Right-bottom: 25 42
```

move (*x=None, y=None*)

Moves shape coordinates in given direction.

If neither x and y are passed, it will automatically center the shape to the origin (0,0).

This function is an high level function, it just uses Shape.map, which is more advanced. Additionally, it is an easy way to center a shape.

Parameters

- **x** (*int or float*) – Displacement along the x-axis.
- **y** (*int or float*) – Displacement along the y-axis.

Returns A pointer to the current object.

Examples

```
print( Shape("m 0 0 1 30 0 30 20 0 20") .move(-5, 10) )
```

```
>>> m -5 10 1 25 10 25 30 -5 30
```

flatten (*tolerance=1.0*)

Splits shape's bezier curves into lines.

This is a low level function. Instead, you should use `split()` which already calls this function.

Parameters **tolerance** (*float*) – Angle in degree to define a curve as flat (increasing it will boost performance during reproduction, but lower accuracy)

Returns A pointer to the current object.

Returns The shape as a string, with bezier curves converted to lines.

split (*max_len=16, tolerance=1.0*)

Splits shape bezier curves into lines and splits lines into shorter segments with maximum given length.

Tips: You can call this before using `:func: 'map'` to work with more outline points for smoother deforming.

Parameters

- **tolerance** (*float*) – Angle in degree to define a bezier curve as flat (increasing it will boost performance during reproduction, but lower accuracy)
- **max_len** (*int or float*) – The max length that you want all the lines to be

Returns A pointer to the current object.

Examples

```
print( Shape("m -100.5 0 1 100 0 b 100 100 -100 100 -100.5 0 c") .split() )
```

```
>>> m -100.5 0 1 -100 0 -90 0 -80 0 -70 0 -60 0 -50 0 -40 0 -30 0 -20 0 -10 0
→ 0 0 10 0 20 0 30 0 40 0 50 0 60 0 70 0 80 0 90 0 100 0 1 99.964 2.325 99.
→ 855 4.614 99.676 6.866 99.426 9.082 99.108 11.261 98.723 13.403 98.271 15.
→ 509 97.754 17.578 97.173 19.611 96.528 21.606 95.822 23.566 95.056 25.488
→ 94.23 27.374 93.345 29.224 92.403 31.036 91.405 32.812 90.352 34.552 89.246
→ 36.255 88.086 37.921 86.876 39.551 85.614 41.144 84.304 42.7 82.945 44.22
→ 81.54 45.703 80.088 47.15 78.592 48.56 77.053 49.933 75.471 51.27 73.848 52.
→ 57 72.184 53.833 70.482 55.06 68.742 56.25 66.965 57.404 65.153 58.521 63.
→ 307 59.601 61.427 60.645 59.515 61.652 57.572 62.622 55.599 63.556 53.598
→ 64.453 51.569 65.314 49.514 66.138 47.433 66.925 45.329 67.676 43.201 68.39
→ 41.052 69.067 38.882 69.708 36.692 70.312 34.484 70.88 32.259 71.411 27.762
→ 72.363 23.209 73.169 18.61 73.828 13.975 74.341 9.311 74.707 4.629 74.927
→ 0.062 75 -4.755 74.927 -9.438 74.707 -14.103 74.341 -18.741 73.828 -23.345
→ 73.169 -27.9 72.363 -32.402 71.411 -34.63 70.88 -36.841 70.312 -39.033 69.
→ 708 -41.207 69.067 -43.359 68.39 -45.49 67.676 -47.599 66.925 -49.683 66.
→ 138 -51.743 65.314 -53.776 64.453 -55.782 63.556 -57.759 62.622 -59.707 61.
→ 652 -61.624 60.645 -63.509 59.601 -65.361 58.521 -67.178 57.404 -68.961 56.
→ 25 -70.707 55.06 -72.415 53.833 -74.085 52.57 -75.714 51.27 -77.303 49.933 -
→ 78.85 48.56 -80.353 47.15 -81.811 45.703 -83.224 44.22 -84.59 42.7 -85.909
```

(continues on next page)

(continued from previous page)

static ring (*out_r, in_r*)

Returns a shape object of a ring with given inner and outer radius, centered around (0,0).

Tips: *A ring with increasing inner radius, starting from 0, can look like an outfading point.***Parameters**

- **out_r** (*int or float*) – The outer radius for the ring.
- **in_r** (*int or float*) – The inner radius for the ring.

Returns A shape object representing a ring.**static ellipse** (*w, h*)

Returns a shape object of an ellipse with given width and height, centered around (0,0).

Tips: *You could use that to create rounded stripes or arcs in combination with blurring for light effects.***Parameters**

- **w** (*int or float*) – The width for the ellipse.
- **h** (*int or float*) – The height for the ellipse.

Returns A shape object representing an ellipse.**static heart** (*size, offset=0*)

Returns a shape object of a heart object with given size (width&height) and vertical offset of center point, centered around (0,0).

Tips: *An offset=size*(2/3) results in a splitted heart.***Parameters**

- **size** (*int or float*) – The width&height for the heart.
- **offset** (*int or float*) – The vertical offset of center point.

Returns A shape object representing an heart.**static star** (*edges, inner_size, outer_size*)

Returns a shape object of a star object with given number of outer edges and sizes, centered around (0,0).

Tips: *Different numbers of edges and edge distances allow individual n-angles.***Parameters**

- **edges** (*int*) – The number of edges of the star.
- **inner_size** (*int or float*) – The inner edges distance from center.
- **outer_size** (*int or float*) – The outer edges distance from center.

Returns A shape object as a string representing a star.**static glance** (*edges, inner_size, outer_size*)

Returns a shape object of a glance object with given number of outer edges and sizes, centered around (0,0).

Tips: *Glance is similar to Star, but with curves instead of inner edges between the outer edges.***Parameters**

- **edges** (*int*) – The number of edges of the star.

- **inner_size** (*int or float*) – The inner edges distance from center.
- **outer_size** (*int or float*) – The control points for bezier curves between edges distance from center.

Returns A shape object as a string representing a glance.

static rectangle (*w=1, h=1*)

Returns a shape object of a rectangle with given width and height, centered around (0,0).

Tips: *A rectangle with width=1 and height=1 is a pixel.*

Parameters

- **w** (*int or float*) – The width for the rectangle.
- **h** (*int or float*) – The height for the rectangle.

Returns A shape object representing an rectangle.

static triangle (*size*)

Returns a shape object of an equilateral triangle with given side length, centered around (0,0).

Parameters **size** (*int or float*) – The side length for the triangle.

Returns A shape object representing an triangle.

2.4 Utils

class `pyonfx.utils.Utils`

This class is a collection of static methods that will help the user in some tasks.

static all_non_empty (*lines_chars_syls_or_words*)

Helps to not check everytime for text containing only spaces or object's duration equals to zero.

Parameters **lines_chars_syls_or_words** (list of Line, Char, Syllable or Word)

–

Returns A list containing lines_chars_syls_or_words without objects with duration equals to zero or blank text (no text or only spaces).

static interpolate (*pct, val1, val2, acc=1.0*)

Interpolates 2 given values (ASS colors, ASS alpha channels or numbers) by percent value as decimal number.

You can also provide a <http://cubic-bezier.com> to accelerate based on bezier curves. (TO DO)

You could use that for the calculation of color/alpha gradients.

Parameters

- **pct** (*float*) – Percent value of the interpolation.
- **val1** (*int, float or str*) – First value to interpolate (either string or number).
- **val2** (*int, float or str*) – Second value to interpolate (either string or number).
- **acc** (*float, optional*) – Optional acceleration that influences final percent value.

Returns Interpolated value of given 2 values (so either a string or a number).

Examples

```
print( Utils.interpolate(0.5, 10, 20) )
print( Utils.interpolate(0.9, "&HFFFFFF&", "&H000000&") )
```

```
>>> 15
>>> &HE5E5E5&
```

class pyonfx.utils.**FrameUtility**(*start_time, end_time, fr=41.71*)

This class helps in the stressfull calculation of frames per frame.

Parameters

- **start_time** (*positive float*) – Initial time
- **end_time** (*positive float*) – Final time
- **fr** (*positive float, optional*) – Frame Duration

Returns Returns a Generator containing start_time, end_time, index and total number of frames for each step.

Examples

```
FU = FrameUtility(0, 100)
for s, e, i, n in FU:
    print(f"Frame {i}/{n}: {s} - {e}")
```

```
>>> Frame 1/3: 0 - 41.71
>>> Frame 2/3: 41.71 - 83.42
>>> Frame 3/3: 83.42 - 100
```

add(*start_time, end_time, end_value, accelerator=1.0*)

This function makes a lot easier the calculation of tags value.

You can see this as a “t” tag usable in frame per frame operations.

Use it in a for loop which iterates a FrameUtility object, as you can see in the example.

Examples

```
FU = FrameUtility(0, 105, 40)
for s, e, i, n in FU:
    fsc = 100
    fsc += FU.add(0, 50, 50)
    fsc += FU.add(50, 100, -50)
    print(f"Frame {i}/{n}: {s} - {e}; fsc: {fsc}")
```

```
>>> Frame 1/3: 0 - 40; fsc: 140.0
>>> Frame 2/3: 40 - 80; fsc: 120.0
>>> Frame 3/3: 80 - 105; fsc: 100
```

class pyonfx.utils.ColorUtility (lines, offset=0)

This class helps to obtain all the color transformations written in a list of lines (usually all the lines of your input .ass) to later retrieve all of those transformations that fit between the start_time and end_time of a line passed, without having to worry about interpolating times or other stressful tasks.

It is highly suggested to create this object just one time in your script, for performance reasons.

Note: A few notes about the color transformations in your lines:

- Every color-tag has to be in the format of `c&Hxxxxxx&`, do not forget the last `&`;
- You can put color changes without using transformations, like `{\1c&HFFFFFF&\3c&H000000&}Test`, but those will be interpreted as `{\t(0,0,\1c&HFFFFFF&\3c&H000000&) }Test`;
- For an example of how color changes should be put in your lines, check [this](#).

Also, it is important to remember that **color changes in your lines are treated as if they were continuous**.

For example, let's assume we have two lines:

1. `{\1c&HFFFFFF&\t(100,150,\1c&H000000&) }Line1`, starting at 0ms, ending at 100ms;
2. `{ }Line2`, starting at 100ms, ending at 200ms.

Even if the second line **doesn't have any color changes** and you would expect to have the style's colors, **it will be treated as it has** `\1c&H000000&`. That could seem strange at first, but thinking about your generated lines, **the majority** will have **start_time and end_time different** from the ones of your original file.

Treating transformations as if they were continuous, **ColorUtility will always know the right colors** to pick for you. Also, remember that even if you can't always see them directly on Aegisub, you can use transformations with negative times or with times that exceed line total duration.

Parameters

- **lines** (*list of Line*) – List of lines to be parsed
- **offset** (*integer, optional*) – Milliseconds you may want to shift all the color changes

Returns Returns a ColorUtility object.

Examples

```
# Parsing all the lines in the file
CU = ColorUtility(lines)
# Parsing just a single line (the first in this case) in the file
CU = ColorUtility([ line[0] ])
```

get_color_change (line, c1=None, c3=None, c4=None)

Returns all the color_changes in the object that fit (in terms of time) between line.start_time and line.end_time.

Parameters

- **line** (*Line object*) – The line of which you want to get the color changes
- **c1** (*bool, optional*) – If False, you will not get color values containing primary color

- **c3** (*bool, optional*) – If False, you will not get color values containing border color
- **c4** (*bool, optional*) – If False, you will not get color values containing shadow color

Returns A string containing color changes interpolated.

Note: If c1, c3 or c4 is/are None, the script will automatically recognize what you used in the color changes in the lines and put only the ones considered essential.

Examples

```
# Assume that we have l as a copy of line and we're iterating over all the
↪syl in the current line
# All the fun stuff of the effect creation...
l.start_time = line.start_time + syl.start_time
l.end_time   = line.start_time + syl.end_time

l.text = "{\\an5\\pos(%.3f,%.3f)\\fscx120\\fscy120%s}%s" % (syl.center, syl.
↪middle, CU.get_color_change(l), syl.text)
```

get_fr_color_change (*line, c1=None, c3=None, c4=None*)

Returns the single color(s) in the color_changes that fit the current frame (line.start_time) in your frame loop.

Note: If you get errors, try either modifying your \t values or set your **fr parameter** in FU object to **10**.

Parameters

- **line** (*Line object*) – The line of which you want to get the color changes
- **c1** (*bool, optional*) – If False, you will not get color values containing primary color.
- **c3** (*bool, optional*) – If False, you will not get color values containing border color.
- **c4** (*bool, optional*) – If False, you will not get color values containing shadow color.

Returns A string containing color changes interpolated.

Examples

```
# Assume that we have l as a copy of line and we're iterating over all the
↪syl in the current line and we're iterating over the frames
l.start_time = s
l.end_time   = e

l.text = "{\\an5\\pos(%.3f,%.3f)\\fscx120\\fscy120%s}%s" % (syl.center, syl.
↪middle, CU.get_fr_color_change(l), syl.text)
```

p

`pyonfx.ass_core`, 5
`pyonfx.convert`, 17
`pyonfx.shape`, 21
`pyonfx.utils`, 24

A

actor (*pyonfx.ass_core.Line attribute*), 13
 add() (*pyonfx.utils.FrameUtility method*), 25
 alignment (*pyonfx.ass_core.Style attribute*), 7
 all_non_empty() (*pyonfx.utils.Utils static method*), 24
 alpha1 (*pyonfx.ass_core.Style attribute*), 6
 alpha2 (*pyonfx.ass_core.Style attribute*), 6
 alpha3 (*pyonfx.ass_core.Style attribute*), 6
 alpha4 (*pyonfx.ass_core.Style attribute*), 6
 angle (*pyonfx.ass_core.Style attribute*), 7
 ascent (*pyonfx.ass_core.Line attribute*), 14
 Ass (*class in pyonfx.ass_core*), 15
 audio (*pyonfx.ass_core.Meta attribute*), 6

B

bold (*pyonfx.ass_core.Style attribute*), 7
 border_style (*pyonfx.ass_core.Style attribute*), 7
 bottom (*pyonfx.ass_core.Char attribute*), 9
 bottom (*pyonfx.ass_core.Line attribute*), 15
 bottom (*pyonfx.ass_core.Syllable attribute*), 11
 bottom (*pyonfx.ass_core.Word attribute*), 13
 bounding() (*pyonfx.shape.Shape method*), 21

C

center (*pyonfx.ass_core.Char attribute*), 9
 center (*pyonfx.ass_core.Line attribute*), 15
 center (*pyonfx.ass_core.Syllable attribute*), 11
 center (*pyonfx.ass_core.Word attribute*), 12
 Char (*class in pyonfx.ass_core*), 8
 chars (*pyonfx.ass_core.Line attribute*), 15
 color1 (*pyonfx.ass_core.Style attribute*), 6
 color2 (*pyonfx.ass_core.Style attribute*), 6
 color3 (*pyonfx.ass_core.Style attribute*), 6
 color4 (*pyonfx.ass_core.Style attribute*), 6
 coloralpha() (*pyonfx.convert.Convert static method*), 17
 ColorUtility (*class in pyonfx.utils*), 25
 comment (*pyonfx.ass_core.Line attribute*), 13

Convert (*class in pyonfx.convert*), 17
 copy() (*pyonfx.ass_core.Line method*), 15

D

descent (*pyonfx.ass_core.Line attribute*), 14
 duration (*pyonfx.ass_core.Char attribute*), 8
 duration (*pyonfx.ass_core.Line attribute*), 13
 duration (*pyonfx.ass_core.Syllable attribute*), 10
 duration (*pyonfx.ass_core.Word attribute*), 11

E

effect (*pyonfx.ass_core.Line attribute*), 14
 ellipse() (*pyonfx.shape.Shape static method*), 23
 encoding (*pyonfx.ass_core.Style attribute*), 8
 end_time (*pyonfx.ass_core.Char attribute*), 8
 end_time (*pyonfx.ass_core.Line attribute*), 13
 end_time (*pyonfx.ass_core.Syllable attribute*), 10
 end_time (*pyonfx.ass_core.Word attribute*), 11
 external_leading (*pyonfx.ass_core.Line attribute*), 14

F

flatten() (*pyonfx.shape.Shape method*), 22
 fontname (*pyonfx.ass_core.Style attribute*), 6
 fontsize (*pyonfx.ass_core.Style attribute*), 6
 FrameUtility (*class in pyonfx.utils*), 25

G

get_color_change() (*pyonfx.utils.ColorUtility method*), 26
 get_data() (*pyonfx.ass_core.Ass method*), 16
 get_fr_color_change() (*pyonfx.utils.ColorUtility method*), 27
 glance() (*pyonfx.shape.Shape static method*), 23

H

has_error() (*pyonfx.shape.Shape method*), 21
 heart() (*pyonfx.shape.Shape static method*), 23
 height (*pyonfx.ass_core.Char attribute*), 9

height (*pyonfx.ass_core.Line attribute*), 14
 height (*pyonfx.ass_core.Syllable attribute*), 10
 height (*pyonfx.ass_core.Word attribute*), 12

I

i (*pyonfx.ass_core.Char attribute*), 8
 i (*pyonfx.ass_core.Line attribute*), 13
 i (*pyonfx.ass_core.Syllable attribute*), 10
 i (*pyonfx.ass_core.Word attribute*), 11
 inline_fx (*pyonfx.ass_core.Char attribute*), 9
 inline_fx (*pyonfx.ass_core.Syllable attribute*), 10
 internal_leading (*pyonfx.ass_core.Line attribute*), 14
 interpolate() (*pyonfx.utils.Utils static method*), 24
 italic (*pyonfx.ass_core.Style attribute*), 7

L

layer (*pyonfx.ass_core.Line attribute*), 13
 leadin (*pyonfx.ass_core.Line attribute*), 13
 leadout (*pyonfx.ass_core.Line attribute*), 13
 left (*pyonfx.ass_core.Char attribute*), 9
 left (*pyonfx.ass_core.Line attribute*), 15
 left (*pyonfx.ass_core.Syllable attribute*), 11
 left (*pyonfx.ass_core.Word attribute*), 12
 Line (*class in pyonfx.ass_core*), 13
 lines (*pyonfx.ass_core.Ass attribute*), 16

M

map() (*pyonfx.shape.Shape method*), 21
 margin_l (*pyonfx.ass_core.Line attribute*), 14
 margin_l (*pyonfx.ass_core.Style attribute*), 7
 margin_r (*pyonfx.ass_core.Line attribute*), 14
 margin_r (*pyonfx.ass_core.Style attribute*), 8
 margin_v (*pyonfx.ass_core.Line attribute*), 14
 margin_v (*pyonfx.ass_core.Style attribute*), 8
 Meta (*class in pyonfx.ass_core*), 5
 meta (*pyonfx.ass_core.Ass attribute*), 16
 middle (*pyonfx.ass_core.Char attribute*), 9
 middle (*pyonfx.ass_core.Line attribute*), 15
 middle (*pyonfx.ass_core.Syllable attribute*), 11
 middle (*pyonfx.ass_core.Word attribute*), 12
 move() (*pyonfx.shape.Shape method*), 21

O

open_aegisub() (*pyonfx.ass_core.Ass method*), 17
 open_mpv() (*pyonfx.ass_core.Ass method*), 17
 outline (*pyonfx.ass_core.Style attribute*), 7

P

path_input (*pyonfx.ass_core.Ass attribute*), 16
 path_output (*pyonfx.ass_core.Ass attribute*), 16
 play_res_x (*pyonfx.ass_core.Meta attribute*), 5
 play_res_y (*pyonfx.ass_core.Meta attribute*), 6

postspace (*pyonfx.ass_core.Char attribute*), 9
 postspace (*pyonfx.ass_core.Syllable attribute*), 10
 postspace (*pyonfx.ass_core.Word attribute*), 12
 prespace (*pyonfx.ass_core.Char attribute*), 9
 prespace (*pyonfx.ass_core.Syllable attribute*), 10
 prespace (*pyonfx.ass_core.Word attribute*), 12
 pyonfx.ass_core (*module*), 5
 pyonfx.convert (*module*), 17
 pyonfx.shape (*module*), 21
 pyonfx.utils (*module*), 24

R

raw_text (*pyonfx.ass_core.Line attribute*), 14
 rectangle() (*pyonfx.shape.Shape static method*), 24
 right (*pyonfx.ass_core.Char attribute*), 9
 right (*pyonfx.ass_core.Line attribute*), 15
 right (*pyonfx.ass_core.Syllable attribute*), 11
 right (*pyonfx.ass_core.Word attribute*), 12
 ring() (*pyonfx.shape.Shape static method*), 23

S

save() (*pyonfx.ass_core.Ass method*), 16
 scale_x (*pyonfx.ass_core.Style attribute*), 7
 scale_y (*pyonfx.ass_core.Style attribute*), 7
 scaled_border_and_shadow (*pyonfx.ass_core.Meta attribute*), 5
 shadow (*pyonfx.ass_core.Style attribute*), 7
 Shape (*class in pyonfx.shape*), 21
 shape_to_pixels() (*pyonfx.convert.Convert static method*), 20
 spacing (*pyonfx.ass_core.Style attribute*), 7
 split() (*pyonfx.shape.Shape method*), 22
 star() (*pyonfx.shape.Shape static method*), 23
 start_time (*pyonfx.ass_core.Char attribute*), 8
 start_time (*pyonfx.ass_core.Line attribute*), 13
 start_time (*pyonfx.ass_core.Syllable attribute*), 10
 start_time (*pyonfx.ass_core.Word attribute*), 11
 strikeouts (*pyonfx.ass_core.Style attribute*), 7
 Style (*class in pyonfx.ass_core*), 6
 style (*pyonfx.ass_core.Line attribute*), 13
 styleref (*pyonfx.ass_core.Char attribute*), 8
 styleref (*pyonfx.ass_core.Line attribute*), 13
 styleref (*pyonfx.ass_core.Syllable attribute*), 10
 styleref (*pyonfx.ass_core.Word attribute*), 12
 styles (*pyonfx.ass_core.Ass attribute*), 16
 syl_char_i (*pyonfx.ass_core.Char attribute*), 8
 syl_i (*pyonfx.ass_core.Char attribute*), 8
 Syllable (*class in pyonfx.ass_core*), 9
 syls (*pyonfx.ass_core.Line attribute*), 15

T

tags (*pyonfx.ass_core.Syllable attribute*), 10
 text (*pyonfx.ass_core.Char attribute*), 8
 text (*pyonfx.ass_core.Line attribute*), 14

[text \(pyonfx.ass_core.Syllable attribute\), 10](#)
[text \(pyonfx.ass_core.Word attribute\), 12](#)
[text_to_clip\(\) \(pyonfx.convert.Convert static method\), 18](#)
[text_to_pixels\(\) \(pyonfx.convert.Convert static method\), 19](#)
[text_to_shape\(\) \(pyonfx.convert.Convert static method\), 18](#)
[time\(\) \(pyonfx.convert.Convert static method\), 17](#)
[top \(pyonfx.ass_core.Char attribute\), 9](#)
[top \(pyonfx.ass_core.Line attribute\), 15](#)
[top \(pyonfx.ass_core.Syllable attribute\), 11](#)
[top \(pyonfx.ass_core.Word attribute\), 12](#)
[triangle\(\) \(pyonfx.shape.Shape static method\), 24](#)

U

[underline \(pyonfx.ass_core.Style attribute\), 7](#)
[Utils \(class in pyonfx.utils\), 24](#)

V

[video \(pyonfx.ass_core.Meta attribute\), 6](#)

W

[width \(pyonfx.ass_core.Char attribute\), 9](#)
[width \(pyonfx.ass_core.Line attribute\), 14](#)
[width \(pyonfx.ass_core.Syllable attribute\), 10](#)
[width \(pyonfx.ass_core.Word attribute\), 12](#)
[Word \(class in pyonfx.ass_core\), 11](#)
[word_i \(pyonfx.ass_core.Char attribute\), 8](#)
[word_i \(pyonfx.ass_core.Syllable attribute\), 10](#)
[words \(pyonfx.ass_core.Line attribute\), 15](#)
[wrap_style \(pyonfx.ass_core.Meta attribute\), 5](#)
[write_line\(\) \(pyonfx.ass_core.Ass method\), 16](#)

X

[x \(pyonfx.ass_core.Char attribute\), 9](#)
[x \(pyonfx.ass_core.Line attribute\), 14](#)
[x \(pyonfx.ass_core.Syllable attribute\), 11](#)
[x \(pyonfx.ass_core.Word attribute\), 12](#)

Y

[y \(pyonfx.ass_core.Char attribute\), 9](#)
[y \(pyonfx.ass_core.Line attribute\), 14](#)
[y \(pyonfx.ass_core.Syllable attribute\), 11](#)
[y \(pyonfx.ass_core.Word attribute\), 12](#)