ManimPango

Release 1.0.0a2

The Manim Community Dev Team

Apr 30, 2023

CONTENTS

ManimPango is a library for rendering text to images (SVGs also supported) using Pango. It also provides various other utilities like registering fonts in system library for use with Pango (could also be used with GTK apps for example).

CHAPTER

ONE

INSTALLATION

1.1 Installation

You can install **ManimPango** using **pip**:

```
pip install manimpango
```

There are prebuild wheels for both Windows and macOS.

For **Linux Users**, there are no Wheels. You must have a C compiler as well as **Pango** and its dependencies along with the **Pango** development headers. See *Building ManimPango* for more information.

1.2 Building ManimPango

1.2.1 Linux/MacOS

For building **ManimPango**, you need

- a C compiler
- Python's development headers
- pkg-config
- Pango along with its development headers and its dependencies.

If you are on MacOS, you can use brew to install those. Using MacPorts is also possible.

```
brew install pango pkg-config
```

If you are on Linux, you can use a system package manager to do so. For example, if you are on Debian based system, you can use `apt`

```
apt install libpango1.0-dev pkg-config python3-dev
```

Arch Linux: `pacman -S pango pkgconf`

Fedora: `dnf install pango-devel pkg-config python3-devel`

Or similar in your system's package manager.

Using `tar` archives

If you don't want to contribute to this repository, you can use the tar archives published on PyPi, or just use `pip` to install using

```
pip install manimpango --no-binary :all:
```

Note: `pip` by default uses wheels, so make sure to pass the `--no-binary` parameter.

Using `git` clones / Contributing

Please remember to do this inside your virtual environment, if you want to use your **Manimpango** with **Manim**.

```
python -m venv ./venv
source venv/bin/activate # Linux/macOS
venv\Scripts\activate # Windows
```

If you are using a clone of this repository, you will need Cython which can be easily installed using `pip`:

```
pip install Cython
```

After that you can use `pip` to install the clone with the following command:

```
pip install -e .
pip install -r requirements-dev.txt .
```

Next, build the library inplace using:

```
python setup.py build_ext -i
```

After installation is complete, you should be able to run pytest:

```
pytest
```

1.2.2 Windows

Note: If you are a normal user, don't read this, you have wheels which you can just install directly using pip.

If you want to contribute to **ManimPango** and you are on Windows, this section is for you.

As Windows does not include a C compiler by default, you will first need to install one. You have two choices:

1. *MinGW/Msys2*
2. *Visual Studio*

MinGW/Msys2

1. Download **MSYS2** from the download link provided on their page https://www.msys2.org/#installation and install it according to their instructions.
2. Once you have **MSYS2** installed, it offers you three different shells: the **MinGW32** shell, the **MinGW64** shell and **MSYS** shell. In order for the following steps to work, you have to open the **MSYS2 MinGW64** shell (you can search for this). Small hint: it has a blue color logo.
3. Run the following commands to install Python, Pango, Cython, Numpy, Scipy, Pillow, Pycairo and ffmpeg

```
pacman -S mingw-w64-x86_64-python
pacman -S mingw-w64-x86_64-python-pip
pacman -S mingw-w64-x86_64-pango
pacman -S mingw-w64-x86_64-cython
pacman -S mingw-w64-x86_64-python-numpy
pacman -S mingw-w64-x86_64-python-scipy
pacman -S mingw-w64-x86_64-python-pillow
pacman -S mingw-w64-x86_64-python-cairo
pacman -S mingw-w64-x86_64-ffmpeg
```

4. Still in the same shell, install **Manim** using `pip install manim`.
5. Finally, get your clone of **ManimPango**, `cd` into that directory and then run `pip install -e .`.

Note: You can't use it with your regular Python version. It will cause weird errors if you do so. For working with **ManimPango**, you must be inside the *MSYS2 MINGW64 shell*.

---

6. You can then use `manim` inside that shell, to run **Manim**.

Note: If you want to try out Python interactively, you can open *idle* using the command `python -m idlelib` inside that shell.

Visual Studio

First, install Visual Studio as specified in https://wiki.python.org/moin/WindowsCompilers. Possibly Visual Studio Build Tools 2022 with Windows11 SDK.

Then run the script at `packing/download_dlls.py`. This will get a **Pango** build along with `pkg-config` and install it at `C:\cibw\vendor`. Add `C:\cibw\vendor\bin` and `C:\cibw\vendor\pkg-config\bin` to PATH.

Note: You can change the install location by editing line 24 of the file `packing/download_dlls.py`.

Then set an environment variable `PKG_CONFIG_PATH=C:\cibw\vendor\lib\pkgconfig`.

Then you can install Cython using

```
pip install Cython
```

Finally, you can install your local **ManimPango** clone just like any other python package by typing:

```
pip install -e .
```

Important: You have to to use https://docs.python.org/3/library/os.html#os.add_dll_directory before running **ManimPango**. This is applicable for Python 3.8 and above.

```
import os
os.add_dll_directory('C:\cibw\vendor\bin')
```

Note that this is done automatically when running test suite.

CHAPTER

TWO

EXAMPLES

2.1 Examples

2.1.1 Simple Example

The simplest way to render a text into a image create a new instance of `Layout` and then a renderer and call the `Layout.render()`.

```
from manimpango import *
l = Layout("Hello World")
r = ImageRenderer(400, 400, l, "test.png")
r.render()
r.save()
```

This will create a 400x400 image with the text "Hello World" in it at the position (0, 0) in the image.

2.1.2 Calculating Bounding Box

The bounding box of the text can be obtained by calling the `Layout.get_bounding_box()` method. This will return a tuple of the form `(x, y, width, height)`.

```
>>> from manimpango import *
>>> l = Layout("Hello World")
>>> print(l.get_bounding_box())
(0, 0, 90, 19)
```

The bounding box is the smallest rectangle that contains all the glyphs of the text.

2.1.3 Changing the Font

The font can be changed by passing a `FontDescription` while creating the `Layout`.

```
>>> from manimpango import *
>>> l = Layout("Hello World", font_desc=FontDescription.from_string("Arial 60"))
>>> l.render('test.png')
```

The font description can also be changed after the `Layout` has been created by setting the `Layout.font_desc` attribute.

CHAPTER

THREE

INTEGRATION WITH OTHER LIBRARIES

This section contains how to use ManimPango with other libraries.

3.1 Integrations

ManimPango is designed to be used with other libraries. It provides utilities for rendering text to images, but only supports rendering PNG or SVG images. For renderering to other formats, you can use various other libraries such as PIL.

3.1.1 Integration with Pillow

The following example shows how to create a Pillow image from a `Layout` object.

```
import manimpango as mp
from PIL import Image

layout = mp.Layout(
    "Hello World",
    font_desc=mp.FontDescription.from_string("Georgia 80")
)
bbox = layout.get_bounding_box()
renderer = mp.ImageRenderer(*bbox[2:], layout)

renderer.render()
img = Image.frombuffer(
    "RGBA",
    (renderer.width, renderer.height),
    bytes(renderer.get_buffer()),
    "raw",
    "BGRa",
    renderer.stride,
)

# Now you can save the image or open it
img.show()
```

3.1.2 Integration with NumPy

The following example shows how to create a NumPy array from a `Layout` object.

```
import manimpango as mp
import numpy as np

layout = mp.Layout(
    "Hello World",
    font_desc=mp.FontDescription.from_string("Georgia 80")
)
bbox = layout.get_bounding_box()

renderer = mp.ImageRenderer(*bbox[2:], layout)
renderer.render()

# Create a numpy array from the buffer
arr = np.ndarray(
    shape=(renderer.height, renderer.width),
    dtype=np.uint32,
    buffer=renderer.get_buffer(),
)

print(arr)
```

3.1.3 Integration with ModernGL

The following example shows how to create a ModernGL texture from a `Layout` object.

```
import manimpango as mp
import moderngl

layout = mp.Layout(
    "Hello World",
    font_desc=mp.FontDescription.from_string("Georgia 80")
)
bbox = layout.get_bounding_box()

renderer = mp.ImageRenderer(*bbox[2:], layout)
renderer.render()

# Create a ModernGL texture from the buffer
ctx = moderngl.create_standalone_context(standalone=True)
texture = ctx.texture(
    (renderer.width, renderer.height),
    4,
    renderer.get_buffer(),
)
```

CHAPTER

FOUR

REFERENCE

4.1 Manimpango Reference

4.1.1 Text Attributes

manimpango.attributes.TextAttribute([...])	`TextAttribute` defines the properties/attributes of the text within a specific range of the text.

TextAttribute

Qualified name: `manimpango.attributes.TextAttribute`

class TextAttribute(*start_index=0*, *end_index=-1*, ***, *allow_breaks=None*, *background_alpha=None*, *background_color=None*, *foreground_alpha=None*, *foreground_color=None*, *fallback=None*, *family=None*, *weight=None*, *line_height=None*)

TextAttribute defines the properties/attributes of the text within a specific range of the text.

A *TextAttribute* object can define multiple properties at the same time, for example, it can change the *background_color*, as well as, *foreground_color*. Also, a *TextAttribute* can be used for multiple times for different texts. By default, an attribute has an inclusive range from `0` to the end of the text `-1`, ie. `[0, -1]`.

Initialize *TextAttribute*.

Parameters

- **start_index** (*int, optional*) – The start index of the range, by default 0 (start of the string).
- **end_index** (*int, optional*) – End index of the range. The character at this index is not included in the range, by default -1 (end of the string).
- **allow_breaks** (*bool | None*) –
- **background_alpha** (*float | None*) –
- **background_color** (*T.Union[str, T.Iterable[int]] | None*) –
- **foreground_alpha** (*float | None*) –
- **foreground_color** (*T.Union[str, T.Iterable[int]] | None*) –
- **fallback** (*bool | None*) –
- **family** (*str | None*) –
- **weight** (*Weight | None*) –

- **line_height** (*float | None*) –

Methods

property allow_breaks: Optional[bool]

Whether to break text or not.

If breaks are disabled, the range will be kept in a single run, as far as possible.

property background_alpha: Optional[float]

The background_alpha of the text.

Raises
ValueError – If the value is not between 0 and 1.

property background_color: Optional[Tuple[int]]

The background color of the region.

If the input is a `str` the value is considered as string representation of color from CSS Specification. The color is then parsed and `ValueError` is raised if the color is invalid.

If the input is a `collections.abc.Iterable` then the items in them are parsed in the order of `red, green, blue` and checked whether they are valid (between 0 and 65535).

Returns either `None` or a `tuple` with 3 elements representing red, green, blue respectively. The value of each items in that tuple ranges from 0 to 65535.

Raises
ValueError – If the value passed isn't a `collections.abc.Iterable` of 3 elements or a string. Another condition when *ValueError* is raised is when the color passed is invalid.

property end_index: int

It is the start of the range. The character at this index is not included in the range.

Raises
ValueError – If the value is not an `int`.

property fallback: bool

Enable or disable fallbacks.

If fallback is disabled, characters will only be used from the closest matching font on the system. No fallback will be done to other fonts on the system that might contain the characters in the text.

property family: Optional[str]

The font family the text should render. Can be a comma seperated list of fonts in a string.

Raises
ValueError – If value isn't a str.

property foreground_alpha: Optional[float]

The foreground_alpha of the text.

Raises
ValueError – If the value is not between 0 and 1.

property foreground_color: Optional[Tuple[int]]

The foreground color attribute.

If the input is a `str` the value is considered as string representation of color from CSS Specification. The color is then parsed and `ValueError` is raised if the color is invalid.

If the input is a `collections.abc.Iterable` then the items in them are parsed in the order of `red, green, blue` and checked whether they are valid (between 0 and 65535).

Returns either `None` or a `tuple` with 3 elements representing red, green, blue respectively. The value of each items in that tuple ranges from 0 to 65535.

Raises

ValueError – If the value passed isn't a `collections.abc.Iterable` of 3 elements or a string. Another condition when *ValueError* is raised is when the color passed is invalid.

property line_height: Optional[float]

The line height of the text.

Raises

ValueError – If value isn't a float.

property start_index: int

It is the end index of the range.

Raises

ValueError – If the value is not an `int`.

property weight: Optional[*Weight*]

The font weight of the text.

Raises

ValueError – If value isn't a str.

4.1.2 Font Description

manimpango.fonts.FontDescription([family, ...])	A `FontDescription` describes a font.
manimpango.fonts.enums	Contains Enums which defines text properties from Pango.

FontDescription

Qualified name: `manimpango.fonts.FontDescription`

class FontDescription(*family=None*, *size=None*, *style=None*, *weight=None*, *variant=None*)

A *FontDescription* describes a font.

This describes the characteristics of a font to load.

Parameters

- **family** (*str*) – Sets *family*.
- **size** (*int*) – Sets *size*.
- **style** (Style) – Sets *style*.
- **weight** (Weight) – Sets *weight*.

- **variant** (Variant) – Sets *variant*.

_font_desc

Reference to the C-implementation of font description.

Type
manimpango.fonts._font_desc._FontDescription

Methods

from_string	Parse a string and form *FontDescription* from it.

property family: str

The family name of the font.

The family name represents a family of related font styles, and will resolve to a particular family. It is also possible to use a comma separated list of family names for this field.

classmethod from_string(*string*)

Parse a string and form *FontDescription* from it.

See https://docs.gtk.org/Pango/type_func.FontDescription.from_string.html#description for the syntax of the string.

Parameters
string (*str*) – The string to be parsed.

Returns
The *FontDescription* that is based on the string.

Return type
FontDescription

property size: int

The size of the font of the text.

property style: *Style*

The style of the font of the text.

It should be one of *Style*. Most fonts will either have a italic style or an oblique style, but not both, and font matching in Pango will match italic specifications with oblique fonts and vice-versa if an exact match is not found.

property variant

The variant of the font of the text.

Should be one of *Variant*.

property weight: *Weight*

The weight of the font of the text.

The weight field specifies how bold or light the font should be. Should be one of *Weight*.

enums

Contains Enums which defines text properties from Pango.

Most of these are used in `FontDescription`.

Classes

Style	An enumeration specifying the various slant styles possible for a font.
Variant	An enumeration specifying capitalization variant of the font.
Weight	An enumeration specifying the weight (boldness) of a font.

Style

Qualified name: `manimpango.fonts.enums.Style`

class Style(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

An enumeration specifying the various slant styles possible for a font.

NORMAL

the font is upright.

ITALIC

the font is slanted, but in a roman style.

OBLIQUE

the font is slanted in an italic style.

Variant

Qualified name: `manimpango.fonts.enums.Variant`

class Variant(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

An enumeration specifying capitalization variant of the font.

NORMAL

A normal font.

SMALL_CAPS

A font with the lower case characters replaced by smaller variants of the capital characters.

Weight

Qualified name: `manimpango.fonts.enums.Weight`

class Weight(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

An enumeration specifying the weight (boldness) of a font. This is a numerical value ranging from 100 to 1000, but there are some predefined values Using numerical value other then that defined here is not supported.

NORMAL

the default weight (= 400)

BOLD

the bold weight(= 700)

THIN

the thin weight(= 100; Since: 1.24)

ULTRALIGHT

the ultralight weight(= 200)

LIGHT

the light weight(= 300)

BOOK

the book weight(= 380; Since: 1.24)

MEDIUM

the normal weight(= 500; Since: 1.24)

SEMIBOLD

the semibold weight(= 600)

ULTRABOLD

the ultrabold weight(= 800)

HEAVY

the heavy weight(= 900)

ULTRAHEAVY

the ultraheavy weight(= 1000; Since: 1.24)

4.1.3 Layout

manimpango.Layout([text, markup, font_desc, ...])	A Layout class represents an entire paragraph of text.

Layout

Qualified name: `manimpango.Layout`

class Layout(*text=None*, *markup=None*, *font_desc=None*, *attributes=None*, *width=None*, *height=None*, *alignment=None*, *justify=None*)

A *Layout* class represents an entire paragraph of text.

Layout provides a high-level driver for formatting entire paragraphs of text at once. This includes paragraph-level functionality such as line breaking, justification, alignment and ellipsization.

A *Layout* is initialized with a str. The layout can then be rendered. There are a number of parameters to adjust the formatting of a *Layout*.

When both *markup* and *text* is set the behavior is unknown.

Parameters

- **text** (*str*) – The text to be set, by default None.
- **markup** (*str*) – The text encoded in PangoMarkup, by default None.
- **font_desc** (FontDescription) – The font description to be used while rendering.
- **attributes** (*list[TextAttribute]*) –
- **width** (*int*) –
- **height** (*int*) –
- **alignment** (*Alignment*) –
- **justify** (*bool*) –

Examples

```
>>> import manimpango as mp
>>> mp.Layout("hello world")
<Layout text='hello world' markup=None>
```

Raises

ValueError – If both `text` and `markup` is None.

Parameters

- **text** (*str*) –
- **markup** (*str*) –
- **font_desc** (FontDescription) –
- **attributes** (*list[TextAttribute]*) –
- **width** (*int*) –
- **height** (*int*) –
- **alignment** (*Alignment*) –
- **justify** (*bool*) –

Methods

get_bounding_box	Returns the bounding box of the layout.
render	Renders the layout into a PNG or SVG file depending on the filename.

get_bounding_box()

Returns the bounding box of the layout.

Note that it's heavy to calculate the bounding box of a layout, so it's better to cache the result.

Example

```
>>> import manimpango as mp
>>> layout = mp.Layout("hello world")
>>> layout.get_bounding_box()
(0, 0, 82, 19)
```

Returns
The bounding box of the layout in the form of `(x, y, width, height)`.

Return type
tuple

property height: int

The height to which the text should be ellipsized at.

Raises
TypeError – If `height` is not a `int`.

property justify: bool

Whether the text should be justified.

Raises
TypeError – If `justify` is not a `bool`.

property markup: str

The markup (in pango markup format) to render.

Raises

- **TypeError** – If `text` is not a `str`.
- ***MarkupParseError*** – If the passed markup is invalid.

render(*file_name*)

Renders the layout into a PNG or SVG file depending on the filename.

Parameters
file_name (*str*) – The filename to which the layout should be rendered.

Return type
None

property text: str

The text to render.

Raises
TypeError – If `text` is not a `str`.

property width: int

The width to which the text should be wrapped or ellipsized.

Raises
TypeError – If `width` is not a `int`.

4.1.4 Renderers

manimpango.renderer.SVGRenderer	SVGRenderer is a renderer which renders the *Layout* to an SVG file.
manimpango.renderer.ImageRenderer	ImageRenderer is a renderer which renders the *Layout* to an image buffer.

SVGRenderer

Qualified name: manimpango.renderer.SVGRenderer

class SVGRenderer

SVGRenderer is a renderer which renders the *Layout* to an SVG file. Note that unlike other renderers the *file_name* is a required parameter.

The *file_name* is opened when the class is initialised and only closed when the renderer is destroyed.

Parameters

- **width** (float) – The width of the SVG.
- **height** (float) – The height of the SVG.
- **layout** (Layout) – The *Layout* that needs to be rendered.
- **file_name** (str) – The path to SVG file.

Example

```
>>> import manimpango as mp
>>> a = mp.SVGRenderer(100, 100, mp.Layout('hello'), 'test.svg')
>>> a
<SVGRenderer file_name='test.svg' width=100.0 height=100.0 layout=<Layout text=
↪'hello' markup=None>
>>> a.render()
True
>>> a.save()
'test.svg'
```

Raises

Exception – Any error reported by cairo.

Methods

render	*render()* actually does the rendering.
save	This method save's the SVG file.

file_name

The file_name where the file is rendered onto

height

The height of the SVG.

`layout`

The *Layout* which is being rendered.

`render()`

render() actually does the rendering. Any error reported by Cairo is reported as an exception. If this method suceeds you can expect an valid SVG file at *file_name*.

Returns
`True` if the function worked, else `False`.

Return type
bool

`save()`

This method save's the SVG file. Note that this method is provided for compatibility with other renderers and does nothing.

Parameters
file_name – The filename to save the file. Note that this is not used and instead only *file_name* is used

Returns
The filename of the rendered SVG file.

Return type
str

`width`

The width of the SVG.

ImageRenderer

Qualified name: `manimpango.renderer.ImageRenderer`

`class ImageRenderer`

ImageRenderer is a renderer which renders the *Layout* to an image buffer. You can directly use the buffer or else you can save it to a file, as a png file, using the *save()* method.

The *file_name* is optional, if you don't provide it then you can use the *get_buffer()* method to get the buffer.

Parameters

- **width** (`float`) – The width of the PNG.
- **height** (`float`) – The height of the PNG.
- **layout** (`Layout`) – The *Layout* that needs to be rendered.
- **file_name** (`str`) – The path to render the PNG file to.

Example

```
>>> import manimpango as mp
>>> a = mp.ImageRenderer(100, 100, mp.Layout('hello'), 'test.png')
>>> a
<ImageRenderer file_name='test.png' width=100.0 height=100.0 layout=<Layout text=
↪'hello' markup=None>
>>> a.render()
True
>>> a.save()
'test.png'
```

Raises
Exception – Any error reported by cairo.

Methods

get_buffer	This method returns the buffer of the image.
render	*render()* actually does the rendering.
save	This method is to save the image to an PNG image.

file_name

The file_name where the file is rendered onto

get_buffer()

This method returns the buffer of the image. This contains the image in the format of ARGB32.

Returns
The buffer of the image.

Return type
bytes

height

The height of the PNG.

layout

The *Layout* which is being rendered.

render()

render() actually does the rendering. Any error reported by Cairo is reported as an exception. If this method suceeds you can expect an valid image in the buffer.

Returns
True if the function worked, else False.

Return type
bool

save()

This method is to save the image to an PNG image. Note that only PNG image are supported, if you need other formats, use external libraries such as Pillow.

Parameters
file_name – The file_name to write the image to.

Raises
ValueError – Raised when the file_name parameter is None.

Returns
The filepath to the saved file.

Return type
str

stride
The stride of the PNG.

width
The width of the PNG.

4.1.5 Exceptions

manimpango.exceptions.MarkupParseError	MarkupParseError is raised when the markup passed in invalid.

manimpango.exceptions.MarkupParseError

exception MarkupParseError
MarkupParseError is raised when the markup passed in invalid.

4.1.6 Utilities

manimpango.register_font	This function registers the font file using fontconfig so that it is available for use by Pango.
manimpango.unregister_font	This function unregisters(removes) the font file using fontconfig.
manimpango.list_fonts	Lists the fonts available to Pango.

manimpango.register_font

register_font()
This function registers the font file using fontconfig so that it is available for use by Pango. On Linux it is aliased to *register_font()* and on Windows and macOS this would work only when using fontconfig backend.

Parameters
font_path (str) – Relative or absolute path to font file.

Returns
True means it worked without any error. False means there was an unknown error

Return type
bool

Examples

```
>>> register_font("/home/roboto.tff")
True
```

Raises
AssertionError – Font is missing.

manimpango.unregister_font

unregister_font()

This function unregisters(removes) the font file using fontconfig. It is mostly optional to call this. Mainly used in tests. On Linux it is aliased to *unregister_font()* and on Windows and macOS this would work only when using fontconfig backend.

Parameters
font_path (str) – For compatibility with the windows function.

Returns
True means it worked without any error. False means there was an unknown error

Return type
bool

manimpango.list_fonts

list_fonts()

Lists the fonts available to Pango. This is usually same as system fonts but it also includes the fonts added through *register_font()*.

Returns
List of fonts sorted alphabetically.

Return type
list

4.1.7 Deprecated API

manimpango.TextSetting(start, end, font, ...)	Formatting for slices of a manim.mobject.svg.text_mobject.Text object.
manimpango.PangoUtils()	
manimpango.text2svg	Render an SVG file from a manim.mobject.svg.text_mobject.Text object.
manimpango.MarkupUtils()	

TextSetting

Qualified name: `manimpango.TextSetting`

class TextSetting(*start*, *end*, *font*, *slant*, *weight*, *line_num=-1*, *color=None*)

Formatting for slices of a `manim.mobject.svg.text_mobject.Text` object.

Methods

Parameters

- **start** (*int*) –
- **end** (*int*) –
- **font** (*unicode*) –
- **color** (*unicode*) –

PangoUtils

Qualified name: `manimpango.PangoUtils`

class PangoUtils

Methods

remove_last_M	Remove element from the SVG file in order to allow comparison.
str2style	Internally used function.
str2weight	Internally used function.

static remove_last_M(*file_name*)

Remove element from the SVG file in order to allow comparison.

Parameters
file_name (*str*) –

Return type
None

static str2style(*string*)

Internally used function. Converts text to Pango Understandable Styles.

Parameters
string (*str*) –

Return type
Style

static str2weight(*string*)

Internally used function. Convert text to Pango Understandable Weight

Parameters
string (*str*) –

Return type
Weight

manimpango.text2svg

text2svg()

Render an SVG file from a `manim.mobject.svg.text_mobject.Text` object.

MarkupUtils

Qualified name: `manimpango.MarkupUtils`

class MarkupUtils

Methods

text2svg	Render an SVG file from a `manim.mobject.svg.text_mobject.MarkupText` object.
validate	Validates whether markup is a valid Markup and return the error's if any.

static text2svg(*text*, *font*, *slant*, *weight*, *size*, *_*, *disable_liga*, *file_name*, *START_X*, *START_Y*, *width*, *height*, ***, *justify=None*, *indent=None*, *line_spacing=None*, *alignment=None*, *pango_width=None*)

Render an SVG file from a `manim.mobject.svg.text_mobject.MarkupText` object.

Parameters

- **text** (*unicode*) –
- **font** (*unicode*) –
- **slant** (*unicode*) –
- **weight** (*unicode*) –
- **size** (*int*) –
- **_** (*int*) –
- **disable_liga** (*bool*) –
- **file_name** (*unicode*) –
- **START_X** (*int*) –
- **START_Y** (*int*) –
- **width** (*int*) –
- **height** (*int*) –

- **justify** (*bool*) –
- **indent** (*float*) –
- **line_spacing** (*float*) –
- **alignment** (*Alignment*) –
- **pango_width** (*Optional[int]*) –

Return type
int

static validate(*markup*)

Validates whether markup is a valid Markup and return the error's if any.

Parameters
markup (str) – The markup which should be checked.

Returns
Returns empty string if markup is valid. If markup contains error it return the error message.

Return type
str

CHAPTER

FIVE

CHANGELOG

5.1 Release Notes

5.1.1 Manimpango 1.0.0a2 (2023-04-30)

Bugfixes

- Include `*.pxi` files in tarball. Without these the package cannot be installed from source.
- Raise on invalid `string` passed to `FontDescriptor` constructor. Previously, it silently crashed.

5.1.2 Manimpango 1.0.0a1 (2023-04-29)

Features

- The package has undergone a complete rewrite to offer an improved API. The new API provides a greater degree of consistency and flexibility compared to its predecessor. (#28)

CHAPTER

SIX

DEVELOPER DOCUMENTATION

6.1 Release Procedure

This is the **maintainer** note on how to Release. All versioning is in accordance to Semantic Versioning 2.0.0. This means older version would have a backport of bugs fixes.

1. Check whether the test suite passes on the main branch.
2. Revert any changes which seems to be not working, and check for the milestone if PR are merged accordingly.
3. Check whether the Wheels Build, against the main branch works as expected.
4. Clone the repository locally.
5. Bump the version in manimpango/_version accordingly.
6. Generate the changelog using towncrier.

```
towncrier
```

7. Make a commit with the changes done, as `Release v<version here>`
8. Create a tag, locally with

```
git tag -s v<version-number>
```

Note: Here, `-s` is used to sign the tag with gpg so that users can later verify it, and a tag shouldn't be created with signing because Github shows it unverified.

Important: The message should include the changelog of the release. There is a github actions which will creates a draft release with the changelog. You can edit them and copy it to the tag you create.

---

9. Push the tag to remote.
10. Go to Github, and draft a new release with the same tag pushed. You can copy the same changelog you copied when you created the tag.

Important: You should actually "draft a new release" instead of just publishing a previously present draft release created by the Github Action. This is important so that the wheels build workflow triggers.

---

11. Check whether the CI uploads the wheels and the `.tar.gz` file to PyPi.

12. Finally, test the `.tar.gz` which was uploaded to PyPi, and install it in a new virtual environment.

PYTHON MODULE INDEX

INDEX

T

U

V

W