nider Documentation

Release 0.1.0

Vladyslav Ovchynnykov

Aug 16, 2017

Contents

Contents:

CHAPTER 1

nider

Python package to add text to images, textures and different backgrounds

- Free software: MIT license
- Documentation: https://nider.readthedocs.io.

`nider` is an approach to make generation of text based images simple yet flexible. Creating of an image is as simple as describing units you want to be rendered to the image and choosing a method that will be used for drawing.

Installation

```
$ pip install nider
```

Features

Drawing on a texture

```
from nider.models import Content
from nider.models import Header
from nider.models import Linkback
from nider.models import Paragraph
from nider.models import TwitterPost

# TODO: change this fontpath to the fontpath on your machine
roboto = '/home/ovd/.local/share/fonts/Roboto/'

header = Header(text='Your super interesting title!',
                fontfullpath=roboto + 'Roboto-Bold.ttf',
                fontsize=30,
                text_width=40,
```

```
                align='left',
                color='#ededed'
                )

para = Paragraph(text='Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed␣
↪do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim␣
↪veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo␣
↪consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum␣
↪dolore eu fugiat nulla pariatur.',
                 fontfullpath=roboto + 'Roboto-Medium.ttf',
                 fontsize=29,
                 text_width=65,
                 align='left',
                 color='#ededed'
                 )

linkback = Linkback(text='foo.com | @username',
                    fontfullpath=roboto + 'Roboto-Bold.ttf',
                    fontsize=24,
                    color='#ededed'
                    )

content = Content(para, header=header, linkback=linkback, padding=60)

img = TwitterPost(content,
                  fullpath='result.png',
                  )

# TODO: change this texture path to the texture path on your machine
img.draw_on_texture('texture.png')
```

Drawing on a solid color

Drawing on an image

Code used to generate featured images can be found here

CHAPTER 2

Installation

Stable release

To install nider, run this command in your terminal:

```
$ pip install nider
```

This is the preferred method to install nider, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

From sources

The sources for nider can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/pythad/nider
```

Or download the tarball:

```
$ curl  -OL https://github.com/pythad/nider/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

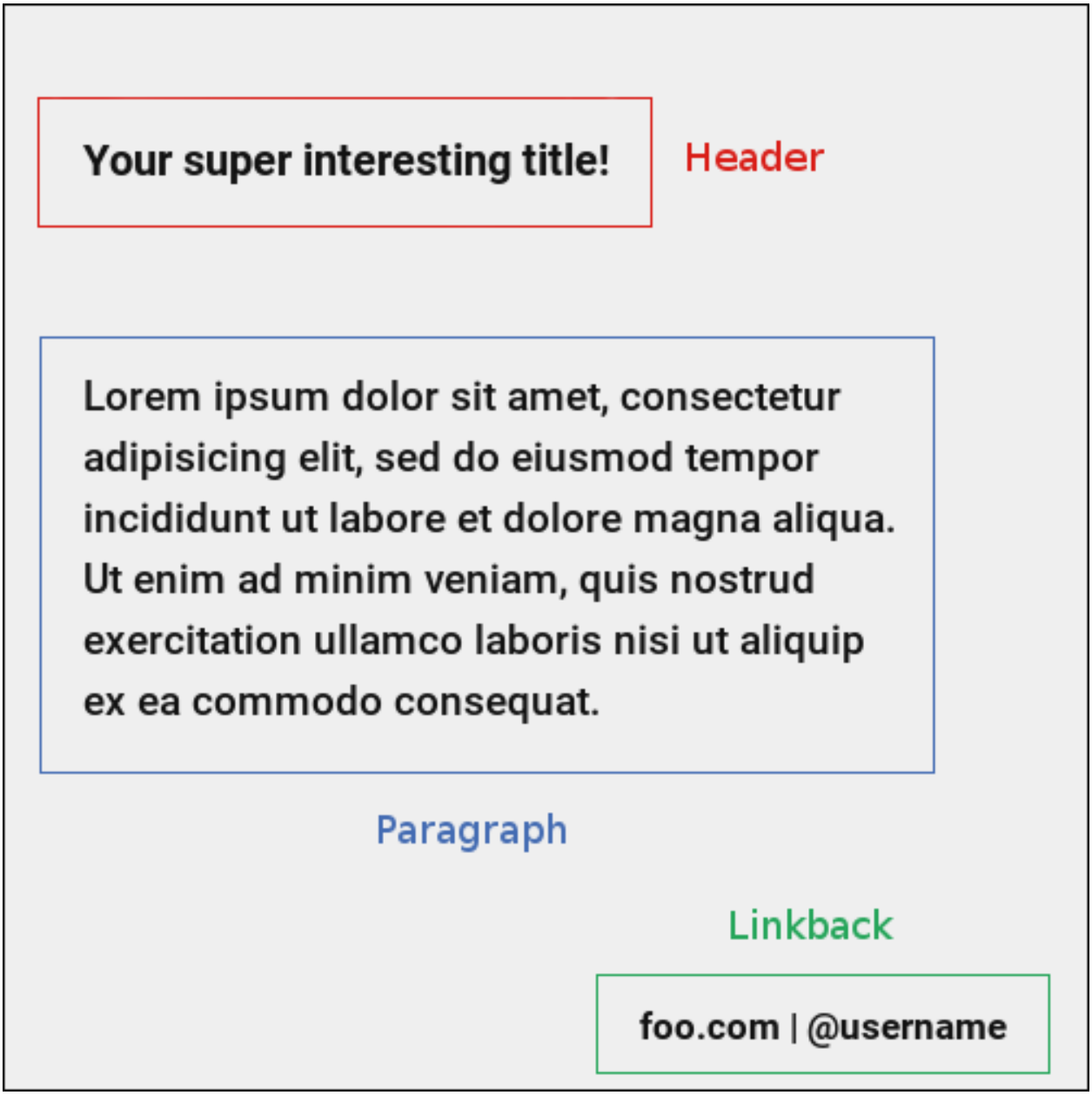
TODO: say that content.padding isn't always took into consideration

This article is a tutorial for `nider` package and at the same time it is a full reference of all `nider` models and possibilities.

Image units

There are three main units each `nider.Image` can consist of:

- header
- paragraph
- linkback



Paragraph unit is required for each image whereas header and linkback are optional.

Each of the units is represented by a class in `nider.models`:

- `nider.models.Header`
- `nider.models.Paragraph`
- `nider.models.Linkback`

`nider.models.Header`

class **Header**(*text*, *fontfullpath*, *fontsize=18*, *text_width=21*, *line_padding=6*, *color='#000'*, *drop_shadow=False*, *shadowcolor='#646464'*, *align='center'*)

Base class for the header unit

Parameters

- **text** (*str*) – Text used in the header
- **fontfullpath** (*str*) – Path to the font used in the header
- **text_width** (*int*) – Header's text width - number of characters in a line
- **line_padding** (*int*) – Header's line padding - padding (in pixels) between the lines
- **fontsize** (*int*) – Size of the font
- **color** (*str or tuple(int, int, int)*) – Either hex or rgb representation of text color
- **drop_shadow** (*bool*) – Boolean flag that indicates if text has to drop shadow
- **shadowcolor** (*str or tuple(int, int, int)*) – Either hex or rgb representation of shadowcolor color
- **align** (*'left' or 'center' or 'right'*) – Side with respect to which the text will be aligned

Raises

- **nider.exceptions.InvalidAlignException** – if `align` is not one of 'left' or 'center' or 'right'
- **nider.exceptions.DefaultFontWarning** – if `fontfullpath` is `None`
- **nider.exceptions.FontNotFoundWarning** – if `fontfullpath` does not exist

Example

```
from nider.models import Header

header = Header(text='Your super interesting title!',
                fontfullpath='/home/me/.local/share/fonts/Roboto/Roboto-Bold.ttf',
                fontsize=30,
                text_width=40,
                align='left',
                color='#ededed'
                )
```

nider.models.Paragraph

The class has the same attribures and behaviour as `nider.models.Header`.

class Paragraph(*text*, *fontfullpath*, *fontsize=18*, *text_width=21*, *line_padding=6*, *color='#000'*, *drop_shadow=False*, *shadowcolor='#646464'*, *align='center'*)

Base class for the paragraph unit

Parameters

- **text** (*str*) – Text used in the paragraph
- **fontfullpath** (*str*) – Path to the font used in the paragraph
- **text_width** (*int*) – Paragraph's text width - number of characters in a line
- **line_padding** (*int*) – Paragraph's line padding - padding (in pixels) between the lines

- **fontsize** (*int*) – Size of the font
- **color** (*str or tuple(int, int, int)*) – Either hex or rgb representation of text color
- **drop_shadow** (*bool*) – Boolean flag that indicates if text has to drop shadow
- **shadowcolor** (*str or tuple(int, int, int)*) – Either hex or rgb representation of shadowcolor color
- **align** (*'left' or 'center' or 'right'*) – Side with respect to which the text will be aligned

Raises

- **nider.exceptions.InvalidAlignException** – if `align` is not one of 'left' or 'center' or 'right'
- **nider.exceptions.DefaultFontWarning** – if `fontfullpath` is `None`
- **nider.exceptions.FontNotFoundWarning** – if `fontfullpath` does not exist

Example

```
from nider.models import Paragraph

para = Paragraph(text='Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed␣
↪do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim␣
↪veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo␣
↪consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum␣
↪dolore eu fugiat nulla pariatur.',
                 fontfullpath='/home/me/.local/share/fonts/Roboto/Roboto-Medium.ttf',
                  fontsize=29,
                  text_width=65,
                  align='left',
                  color='#ededed'
                  )
```

nider.models.Linkback

class Linkback (*text*, *fontfullpath*, *fontsize=18*, *color='#000'*, *drop_shadow=False*, *shadowcolor='#646464'*, *align='center'*, *bottom_padding=20*)

Base class for the linkback unit

Parameters

- **text** (*str*) – Text used in the linkback
- **fontfullpath** (*str*) – Path to the font used in the linkback
- **fontsize** (*int*) – Size of the font
- **color** (*str or tuple(int, int, int)*) – Either hex or rgb representation of text color
- **drop_shadow** (*bool*) – Boolean flag that indicates if text has to drop shadow
- **shadowcolor** (*str or tuple(int, int, int)*) – Either hex or rgb representation of shadowcolor color

- **align** (*'left' or 'center' or 'right'*) – Side with respect to which the text will be aligned
- **bottom_padding** (*int*) – Linkback's bottom padding - padding (in pixels) between the bottom of the image and the linkback itself

Raises

- **nider.exceptions.InvalidAlignException** – if `align` is not one of 'left' or 'center' or 'right'
- **nider.exceptions.DefaultFontWarning** – if `fontfullpath` is `None`
- **nider.exceptions.FontNotFoundWarning** – if `fontfullpath` does not exist

Example

```
from nider.models import Linkback

linkback = Linkback(text='foo.com | @username',
                    fontfullpath='/home/me/.local/share/fonts/Roboto/Roboto-Bold.ttf',
                    fontsize=24,
                    color='#ededed'
                    )
```

Image content

In order to aggregate all of the units together you need to create an instance of `nider.models.Content` class.

nider.models.Content

class Content (*paragraph*, *header=None*, *linkback=None*, *padding=45*)

Class that aggregates different units into a sigle object

Parameters

- **paragraph** (*nider.models.Paragraph*) – Paragraph that will be used
- **header** (*nider.models.Header*) – Header that will be used
- **linkback** (*nider.models.Linkback*) – Linkback that will be used
- **padding** (*int*) – Content's padding - padding (in pixels) between the units

Example

```
from nider.models import Content
from nider.models import Linkback
from nider.models import Paragraph

para = Paragraph(...)

linkback = Linkback(...)

content = Content(para, linkback=linkback, padding=60)
```

Initializing an image

After the content is prepared it's the right time to initialize an image. In `nider` a basic image is represented by `nider.models.Image`

`nider.models.Image`

class **`Image`**(*content*, *fullpath*, *width=1080*, *height=1080*)

Base class for a text based image

Parameters

- **`content`** (*`nider.models.Content`*) – Content object that has units to be rendered
- **`fullpath`** (*`str`*) – Path where the image has to be saved
- **`width`** (*`int`*) – Width of the image
- **`height`** (*`int`*) – Height of the image

Raises

- **`AttributeError`** – if it's impossible to create a file at `fullpath` path
- **`AttributeError`** – if width <= 0 or height <= 0

Example

```
from nider.models import Content
from nider.models import Image

content = Content(...)

img = Image(content,
            fullpath='example.png',
            width=500,
            height=500
            )
```

Social media images

`nider` comes with some pre-built models that can be used to generate images for some social networks. These are subclasses of `nider.models.Image` with changed size

Instagram

- `nider.models.InstagramSquarePost` - 1080x1080 image
- `nider.models.InstagramPortraitPost` - 1080x1350 image
- `nider.models.InstagramLandscapePost` - 1080x566 image

Facebook

- `nider.models.FacebookSquarePost` - 470x470 image
- `nider.models.FacebookLandscapePost` - 1024x512 image

Twitter

- `nider.models.TwitterPost` - 1024x512 image
- `nider.TwitterLargeCard` - 506x506 image

I highly recommend reading this post if you are curious about what are the right image sizes for social media images.

Drawing on the image

Having an instance of `nider.models.Image` we are ready to create a real image.

`nider` comes with 3 options of drawing your image:

- `Image.draw_on_texture` - draws preinitialized image and its attributes on a texture. **nider takes care of filling image of any size with textrure you privide. You don't need to create textured images by pasting texture mulpitle times in Photoshop or Gimp.**
- `Image.draw_on_bg` - Draws preinitialized image and its attributes on a colored background. nider uses a color you provide to fill the image and then draws the content.
- `Image.draw_on_image` - Draws preinitialized image and its attributes on an image. Content will be drawn directly on the image you provide.

`Image.draw_on_texture`

draw_on_texture(*texture_path=None*)

Draws preinitiated image and its attributes on a texture. If `texture_path` is set to `None`, takes random textures from `textures/`

Parameters **texture_path** (*str*) – Path of the texture to use

Raises

- **FileNotFoundError** – if the file at `texture_path` cannot be found
- **nider.exceptions.ImageSizeFixedWarning** – if the image size has to be adjusted to the provided content's size because the content takes much space

Example

```
from nider.models import Content
from nider.models import Image

content = Content(...)

img = Image(content,
```

```
            fullpath='example.png',
            width=500,
            height=500
            )

img.draw_on_texture('example_texture.png')
```

Check the full example here

---

nider comes with a huge bundle of textures. As for now you need to copy them to your machine if you want to use any of them.

Image.draw_on_bg

draw_on_bg(*bgcolor=None*)

Draws preinitiated image and its attributes on a colored background. If bgcolor is set to None, random nider.colors.colormap.FLAT_UI color is generated

Parameters **bgcolor** (*str or tuple(int, int, int)*) – Either hex or rgb representation of background color

Raises **nider.exceptions.ImageSizeFixedWarning** – if the image size has to be adjusted to the provided content's size because the content takes much space

Example

```
from nider.models import Content
from nider.models import Image

content = Content(...)

img = Image(content,
            fullpath='example.png',
            width=500,
            height=500
            )

img.draw_on_bg('#efefef')
```

Check the full example here

Image.draw_on_image

draw_on_image(*image_path*)

Draws preinitiated image and its attributes on an image. Image size will be changed to the size of provided image.

Parameters **image_path** (*str*) – Path of the image to draw on

Raises **FileNotFoundError** – if the file at image_path cannot be found

Example

```
from nider.models import Content
from nider.models import Image

content = Content(...)

img = Image(content,
            fullpath='example.png',
            width=500,
            height=500
            )

img.draw_on_image('example_bg.jpg')
```

Check the full example here

That's it. After any of draw methods has been called and successfully completed the new image will be saved to `Image.fullpath`.

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

Types of Contributions

Report Bugs

Report bugs at https://github.com/pythad/nider/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

Write Documentation

nider could always use more documentation, whether as part of the official nider docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/pythad/nider/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

Get Started!

Ready to contribute? Here's how to set up *nider* for local development.

1. Fork the *nider* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/nider.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv nider
$ cd nider/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 nider tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, 3.4 and 3.5, and for PyPy. Check https://travis-ci.org/pythad/nider/pull_requests and make sure that the tests pass for all supported Python versions.

Tips

To run a subset of tests:

```
$ python -m unittest tests.test_nider
```

CHAPTER 5

Credits

Development Lead

- Vladyslav Ovchynnykov <ovd4mail@gmail.com>

Contributors

None yet. Why not be the first?

CHAPTER 6

History

0.1.0 (2017-07-27)

- First release on PyPI.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Index