Dot Configs Documentation

Release 0.1.0

Magnus "Loxosceles" Henkel

Apr 22, 2019

Contents

1 Features **3**
1.1 Contents: . . . 3
1.2 Feedback . . . 6

pypi package ?.?.?

Import and handle project configurations as dot-separated configs object

Full documentation can be found here: https://dot-configs.readthedocs.io/en/latest/

CHAPTER 1

Features

- Easy parsing and handling of project options by using a modifiable configuration object
- Simple setup
- Splittable configs
- Backlinks: Every end-node can trace back its connection to the root node
- Dot-seperated object chain, convenient for addressing nodes and properties
- Configurations can quickly be inspected inside a command line interpreter, like Jupyter/iPython

1.1 Contents:

1.1.1 Installation

It is recommended to use a virtual environment to install **dot-configs**. Once your virtual environment is activated, install via pip:

```
$ pip install dot-configs
```

1.1.2 Usage

An example of a JSON configuration file could look like this[1] :

```
{
  "architecture": {
    "convNet": {
      "spectrograms": {
        "frame_sizes": [
```

(continues on next page)

[1] You can find this example JSON under `examples` in the repository.

(continued from previous page)

```
          1024,
          2048,
          4096
        ],
        "hop_size": 441,
        "num_bands": 40
      },
      "training_data": {
        "test_size": 0.1,
        "val_size": 0.2,
        "random_state": 42
      },
      "paths": {
        "files": {
          "train_logs": "train_logs.csv",
          "state": "state.json",
          "model": "model.hdf5"
        },
        "dirs": {
          "store": "/path/to/data_store",
          "sandbox": "/path/to/sandbox",
          "tests": "/path/to/test_dir"
        }
      },
      "overwrite": false,
      "default_params": null
    }
  }
}
```

Note that these configurations can be nested. You can use *strings*, *floats*, *integers*, *lists*, *booleans* and *null* values. The JSON format does not support *None* directly but, once imported, *null* will be converted to *None*. The same goes for *false/true* which will be converted to their upper case equivalents *False* and *True*.

To use **dot-configs** in your project:

```
import dot_configs as dc
config_filepath = '/path/to/json_configuration_file'
cfg = dc.Configurations(config_filepath).get_configurations()
```

Now you can address any of the values in the following way:

```
cfg.architecture.convNet.paths.files.train_logs     # "train_logs.csv"
```

1.1.3 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://gitlab.com/loxosceles/dot_configs/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 GitLab issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

Implement Features

Look through the GitLab issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

Write Documentation

Dot Configs could always use more documentation, whether as part of the official Dot Configs 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://gitlab.com/loxosceles/dot_configs/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 *dot_configs* for local development.

1. Fork the *dot_configs* repo on GitLab.
2. Clone your fork locally:

```
$ git clone git@gitlab.com:your_name_here/dot_configs.git
```

3. Create a branch for local development:

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

Now you can make your changes locally.

4. When you're done making changes, check that your changes pass style and unit tests, including testing other Python versions with tox:

```
$ tox
```

To get tox, just pip install it.

5. Commit your changes and push your branch to GitLab:

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

6. Submit a pull request through the GitLab 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. Run the `tox` command and make sure that the tests pass for all supported Python versions.

Tips

To run a subset of tests:

```
$ py.test test/test_dot_configs.py
```

1.1.4 Credits

Development Lead

- Magnus “Loxosceles” Henkel <loxosceles@gmx.de>

Contributors

None yet. Why not be the first?

1.1.5 History

0.1.0 (2019-04-30)

- First release on PyPI.

1.2 Feedback

If you have any suggestions or questions about **Dot Configs** feel free to email me at loxosceles@gmx.de.

If you encounter any errors or problems with **Dot Configs**, please let me know! Open an Issue at the GitLab http://gitlab.com/loxosceles/dot_configs main repository.