Trafaret Config Documentation

Release 2.0.2

paul@colomiets.name

Sep 13, 2018

Contents

Trafaret-config is a wrapper that loads yaml and checks config using trafaret while keeping track of actual lines of file where error has happened.

Additionally, it can pretty print the error.

CHAPTER 1

Contents

1.1 Simple API

There are just two functions:

read_and_validate (*filename*, *trafaret*)
Reads the file at `filename` and validates it using trafaret. Returns config when configuration is fine.

Example usage:

```
try:
    config = read_and_validate('config.yaml', TRAFARET)
except ConfigError as e:
    e.output()
    sys.exit(1)
```

parse_and_validate (*data*, *trafaret*, *filename='<config.yaml>'*)
Parses a string and validates it using trafaret. Returns config when configuration is fine. For having adequate filename in error messages you can either pass *filename* here or you can implement your own error printer.

Example usage:

```
with open("config.yaml") as f:
    text = f.read()
try:
    config = parse_and_validate(text, TRAFARET, filename='config.yaml')
except ConfigError as e:
    e.output()
    sys.exit(1)
```

1.2 Error API

class ConfigError

Error returned from configuration validator. Contains filenames, line numbers, error messages and other info needed to pretty-print error messages.

We don't provide programmatic API to access the data yet, because we're not sure about the details yet.

output (*stream=None*)

Output the error to a stream.

Parameters **stream** – A text stream (a file open in text mode) to write output to. If not specified error is printed to `sys.stderr`. You can use `io.StringIO` to collect output to an in-memory buffer.

Example:

```
try:
    config = read_and_validate(filename, trafaret)
except ConfigError as err:
    err.output(stream=sys.stderr)
```

1.3 Command-Line

Usually you want to accept filename of a configuration file from the command-line. While it's easy to define command-line arguments yourself, there are two helpers, which allow to define options with the standard names, so all of your applications are configured in the same way.

Usage:

```
from trafaret_config import read_and_validate, ConfigError
from trafaret_config import commandline
from your_config_module import CONFIG_TRAFARET

def main():
    ap = argparse.ArgumentParser()
    commandline.standard_argparse_options(ap, default_config='config.yaml')
    #
    # define your command-line arguments here
    #
    options = ap.parse_args()
    config = commandline.config_from_options(options, CONFIG_TRAFARET)
    pprint.pprint(config)
```

You can find full example in the repository.

The `--help` looks like:

```
usage: example.py [-h] [-c CONFIG] [--print-config] [--print-config-vars] [-C]

optional arguments:
  -h, --help            show this help message and exit
  -c CONFIG, --config CONFIG
                        Configuration file (default: 'config.yaml')
  --print-config        Print config as it is read after parsing and exit
  --print-config-vars   Print variables used in configuration file
  -C, --check-config    Check configuration and exit
```

Alternatively you can put configuration parameters into it's own option group:

```
def main():
    ap = argparse.ArgumentParser()
    commandline.standard_argparse_options(
        ap.add_argument_group('configuration'),
        default_config='config.yaml')

    ap.add_argument('--verbose', action='store_true')
```

Output looks like:

```
usage: example-cli.py [-h] [-c CONFIG] [--print-config] [-C] [--verbose]

optional arguments:
  -h, --help            show this help message and exit
  --verbose

configuration:
  -c CONFIG, --config CONFIG
                        Configuration file (default: 'config.yaml')
  --print-config        Print config as it is read after parsing and exit
  -C, --check-config    Check configuration and exit
```

1.4 Command-Line with *click*

click is another popular package for creating beautiful CLI.

One option to use *trafaret_config* is to define new *click* argument type based, which expects path to an existing configuration file plus trafaret rules.

Create *cli.py*:

```
import click
import trafaret_config as traf_cfg
import trafaret as t
import time

CONFIG_TRAFARET = t.Dict({t.Key("host"): t.String(), t.Key("port"): t.Int()})


class TrafaretYaml(click.Path):
    """Configuration read from YAML file checked against trafaret rules."""
    name = "trafaret yaml file"

    def __init__(self, trafaret):
        self.trafaret = trafaret
        super().__init__(
            exists=True, file_okay=True, dir_okay=False, readable=True)

    def convert(self, value, param, ctx):
        cfg_file = super().convert(value, param, ctx)
        try:
            return traf_cfg.read_and_validate(cfg_file, self.trafaret)
        except traf_cfg.ConfigError as e:
            msg = "\n".join(str(err) for err in e.errors)
```

(continues on next page)

(continued from previous page)

```
            self.fail("\n" + msg)


@click.group()
def cli():
    pass


@cli.command()
@click.argument("config", type=TrafaretYaml(CONFIG_TRAFARET))
def validate(config):
    """Validate configuration file structure."""
    click.echo("OK: Configuration is valid.")


@cli.command()
@click.argument("config", type=TrafaretYaml(CONFIG_TRAFARET))
def run(config):
    """Run web application.
    """
    # Start the application
    host = config["host"]
    port = config["port"]
    print("Would like to run the app at {host}:{port}...".format(
        host=host, port=port))
    time.sleep(5)
    print("..done.")


if __name__ == "__main__":
    cli()
```

CONFIG_TRAFARET is sample trafaret rule for our config file, which may look like *config.yaml*:

```
host: localhost
port: 1234
```

class TrafaretYaml(click.Path) defines a class for new *click* type.

def cli(): defines top level command to run and it has two subcommands:

Subcommand validating the configuration file is really simple:

```
@cli.command()
@click.argument("config", type=TrafaretYaml(CONFIG_TRAFARET))
def validate(config):
    """Validate configuration file structure."""
    click.echo("OK: Configuration is valid.")
```

using *type=TrafaretYaml* it implicitly expects path to config file and at the same time prescribes trafaret rules for it's content.

def run(): goes one step further and uses the configuration values.

1.4.1 Sample usage

First explore the main command:

```
$ python cli.py
Usage: cli.py [OPTIONS] COMMAND [ARGS]...

Options:
--help  Show this message and exit.

Commands:
run       Run web application.
validate  Validate configuration file structure.
```

It provides two subcommands.

Subcommand *validate* allows configuration file validation:

```
$ python cli.py validate cfg.yaml
OK: Configuration is valid.
```

If the config file does not exist:

```
$ python cli.py run cfg-not-here.yaml
Usage: cli.py run [OPTIONS] CONFIG

Error: Invalid value for "config": Path "cfg-not-here.yaml" does not exist.
```

it reports this problem.

If port number has value *1234a*, it uses trafaret rules to report the problem:

```
$ python cli.py va lidate cfg.yaml
Usage: cli.py validate [OPTIONS] CONFIG

Error: Invalid value for "config":
cfg.yaml:2: port: value can't be converted to int
```

If all is fine, it allows running the applicaiton:

```
$ python cli.py run cfg.yaml
Would like to run the app at localhost:1234...
..done.
```

Hint: add subcommand *init* printing sample configuration file content.

1.5 Variable Substitution

Since trafaret-config 2.0 environment variables in the config are replaced by default, this means that config like this:

```
url: http://${HOST}:$PORT/
```

Will get `HOST` and `PORT` variables insert from the environment, and if variable does not exist, you will get the following error:

```
config.yaml:2: variable 'PORT' not found
  -> 'http://${HOST}:$PORT/'
```

To override variables that are subsituted pass `vars={'some': 'dict'}` to any of the functions:

- `config_from_options(..., vars=custom_vars)`

- `read_and_validate(..., vars=custom_vars)`
- `parse_and_validate(..., vars=custom_vars)`

To turn off variable substitution at all pass `vars=None`

Sometimes you might want to print variables used in configuration file, i.e. to make some configuration file inter. If you're using `trafaret_config.commandline` you can do it using default command-line argument:

```
$ ./run.py --print-config-vars
HOST
PORT
```

1.6 Trafaret-config Changes by Release

1.6.1 v2.0.1

- Package metadata update only

1.6.2 v2.0.0

- breaking: trafaret >= 1.2.0 is only supported (previous versions may work)
- breaking: PyYAML >= 4.1 is only supported (previous versions may work)
- breaking feature: Variables like `$this` or `${THIS}` are substituted in all scalar in yaml file, if you relied on this kind of values present in the config verbatim, pass `vars=None` to config parser
- feature: Add `--print-config-vars` command-line argument to print variables used in specific config

CHAPTER 2

Basic Usage

For easier real-life usage see *Command-Line* section.

```
import sys
import trafaret
from trafaret_config import read_and_validate

TRAFARET = trafaret.Dict({'x': trafaret.String()})

try:
    config = read_and_validate('config.yaml', TRAFARET)
except ConfigError as e:
    e.output()
    sys.exit(1)
```

Example output (from a *test.py* which has better trafaret than example above):

```
bad.yaml:2: smtp.port: value can't be converted to int
bad.yaml:3: smtp.ssl_port: value can't be converted to int
bad.yaml:4: port: value can't be converted to int
```

CHAPTER 3

Indices and tables

- genindex
- search

Index