Global Logger Documentation

Release 0.3.30

Alexey Rubasheff

Jun 18, 2022

Contents:

CHAPTER 1

Global Logger

Global Logger

Based on Python built-in logger, expands it, and provides a global logger to your system.

- Free software: MIT license
- **Documentation:**
  - https://alertua.github.io/global_logger
  - https://global-logger.readthedocs.io / https://global-logger.rtfd.io
- PyPi modules: https://pypi.org/project/global-logger

1.1 Features

- Easy on-screen logging setup without pain for Python newcomers
- Easy .log files output setup without pain for Python newcomers
- Python 2 and 3 compatible

CHAPTER 2

Installation

2.1 Stable release

To install Global Logger, run this command in your terminal:

```
$ pip install global_logger
```

This is the preferred method to install Global Logger, 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.

2.2 From sources

The sources for Global Logger can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/alertua/global_logger
```

Or download the tarball:

```
$ curl -OJL https://github.com/alertua/global_logger/tarball/master
```

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

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

CHAPTER 3

Usage

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
""" Global Logger Examples """
from global_logger import Log

# create and/or reuse a global logger, choosing its name dynamicaly
# with screen-only output and the default logging level INFO
log = Log.get_logger()

# this forcec ALL loggers to lower their logging level to DEBUG
log.verbose = True

log.debug("debug text: level: %s" % log.Levels.DEBUG)
log.info("info text: level: %s" % log.Levels.INFO)
log.warning("warning text: level: %s" % log.Levels.WARNING)
log.error("error text: level: %s" % log.Levels.ERROR)
log.critical("critical text: level: %s" % log.Levels.CRITICAL)
# 2020-06-28 14:18:42.004  14:source.examples DEBUG  debug text: level: '10'
# 2020-06-28 14:18:42.004  15:source.examples INFO   info text: level: '20'
# 2020-06-28 14:18:42.004  16:source.examples WARNING warning text: level: '30'
# 2020-06-28 14:18:42.005  17:source.examples ERROR  error text: level: '40'
# 2020-06-28 14:18:42.005  18:source.examples CRITICAL critical text: level: '50'

# log text in purple color without a newline, clearing all the ANSI sylbols from the␣
↪message
log.printer('always printed text....', color='blue', end='', clear=True)
# can also be simplified to:
log.green('green text', clear=False)
log.yellow('yellow text', end='\t\t\t\t')
log.red('red text')

# create and/or reuses a global logger, choosing its name dynamicaly
# with screen and .log files output at the relative folder 'logs' and the default␣
↪logging level INFO
```

(continues on next page)

(continued from previous page)

```
log = Log.get_logger(logs_dir='logs')

# force ALL loggers to lower their logging level to WARNING
# Note: file output will always remain on logging level DEBUG
log.level = log.Levels.WARNING


# log a function call including all the arguments
def func(number, other_number):
    log.trace()
    return number * other_number


func(1, 2)
# 2020-06-28 14:30:55.194 322:source.examples DEBUG  source.examples.func[('number',␣
↪1), ('other_number', 2)]
```

CHAPTER 4

global_logger

4.1 global_logger package

4.1.1 Submodules

4.1.2 global_logger.global_logger module

Main Global Logger Module

class global_logger.global_logger.**InfoFilter**(*name=”*)

Bases: logging.Filter

filter(*record*)

Determine if the specified record is to be logged.

Is the specified record to be logged? Returns 0 for no, nonzero for yes. If deemed appropriate, the record may be modified in-place.

class global_logger.global_logger.**Log**(*name*, *level=None*, *global_level=True*, *logs_dir=None*, *log_session_filename=None*, *max_log_files=None*, *file_message_format=None*, *screen_message_format=None*, *date_format_full=None*, *date_format=None*, *use_colors=True*, *direct=True*)

Bases: object

DEFAULT_LOGS_DIR = 'logs'

GLOBAL_LOG_LEVEL = 20

LOGGER_DATE_FORMAT = '%H:%M:%S'

LOGGER_DATE_FORMAT_FULL = '%Y-%m-%d %H:%M:%S'

LOGGER_FILE_MESSAGE_FORMAT = '%(asctime)s.%(msecs)03d %(lineno)3s:%(name)-22s %(levelna

LOGGER_SCREEN_MESSAGE_FORMAT = '%(log_color)s%(message)s'

class Levels

Bases: `enum.IntEnum`

An enumeration.

CRITICAL = 50

DEBUG = 10

ERROR = 40

FATAL = 50

INFO = 20

NOTSET = 0

WARN = 30

WARNING = 30

MAX_LOG_FILES = 50

static add_handler_to_all_loggers(*handler*)

auto_added_handlers = []

classmethod get_logger(*name=None*, *level=None*, *global_level=True*, *logs_dir=None*, *log_session_filename=None*, *max_log_files=None*, *file_message_format=None*, *screen_message_format=None*, *date_format_full=None*, *date_format=None*, *use_colors=True*)

Main instantiating method for the class. Use it to instantiate global logger.

Parameters

- **name** (*str or unicode*) – a unique logger name that is re-/used if already exists, defaults to the function path.
- **level** (*int*) – Logging level for the current instance.
- **global_level** (*bool*) – Treat this level as a global (True) or as an individual (False) Individual loggers do not gain global logging level changes.
- **logs_dir** (*Path or str or None*) – Path where the .log files would be created, if provided.
- **log_session_filename** (*str or None*) – Log output filename.
- **max_log_files** (*int*) – Maximum .log files to store.
- **screen_message_format** (*str*) – Screen Logging message format.
- **file_message_format** (*str*) – File Logging message format.
- **date_format_full** (*str*) – Logging full date format.
- **date_format** (*str*) – Logging on-screen date format.
- **use_colors** (*bool*) – Use colored Stdout and Stderr output

Returns *Log* instance to work with.

Return type *Log*

green(**message*, ***kwargs*)

individual_loggers = {}

level
Returns current on-screen logging output level. File output is always DEBUG. :return: int or None

log_session_filename = None

loggers = {}

logs_dir = None

printer(**message*, ***kwargs*)

Parameters

- **message** (*str or list of str or unicode*) – a message to print: as a string or as a list of strings
- **end** (*str*) – line ending symbol, defaults to
- **color** (*AnsiFore*) – message color to use
- **clear** (*bool*) – Whether to clear message string from ANSI symbols, defaults to True

red(**message*, ***kwargs*)

static set_global_log_level(*level*)
Global Logging Level Setter Method. Sets Logging Level for all loggers of this type :type level: int :param level: Global Logging Level to set.

trace()

verbose

yellow(**message*, ***kwargs*)

global_logger.global_logger.**clear_message**(*msg*)

global_logger.global_logger.**get_prev_function_name**()

4.1.3 Module contents

Top-level package for Global Logger.

CHAPTER 5

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:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/alertua/global_logger/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.

5.1.2 Fix Bugs

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

5.1.3 Implement Features

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

5.1.4 Write Documentation

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

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/alertua/global_logger/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 :)

5.2 Get Started!

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

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

```
$ git clone git@github.com:your_name_here/global_logger.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 global_logger
$ cd global_logger/
$ 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 global_logger tests
$ python setup.py test or pytest
$ 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.

5.3 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.

5.4 Tips

To run a subset of tests:

```
$ python -m pytest tests.test_global_logger
```

5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bump2version patch # possible: major / minor / patch
$ git push
$ git push --tags
```

CHAPTER 6

Credits

6.1 Developer

- Alexey Rubasheff <alexey.rubasheff@gmail.com>

6.2 Contributors

None yet. Why not be the first?

CHAPTER 7

History

7.1 0.3.30 (2022-18-06)

- Bugfix: Proper emit to file handler

7.2 0.3.29 (2022-31-01)

- It is now possible to toggle Logger screen colored output

7.3 0.3.28 (2022-15-01)

- It is now possible to set the current global log_session_filename while instantiating the first Logger

7.4 0.3.27 (2021-22-11)

- Small precaution

7.5 0.3.26 (2021-15-10)

- GitHub Actions Python Version Bugfix

7.6 0.3.25 (2021-15-10)

- Bugfix

- tox bugfix

7.7 0.3.24 (2021-15-10)

- Removed dumping sensitive data

7.8 0.3.23 (2021-09-10)

- Pathlib prerequisite fix for Python 3.4+

7.9 0.3.22 (2021-08-17)

- Colorlog import fixed @ Python < 3.5

7.10 0.3.21 (2021-02-02)

- IntEnum import fixed @ Python < 3.6

7.11 0.3.20 (2020-11-19)

- win_unicode_console version tweak

7.12 0.3.17 (2020-10-19)

- Python 3.9 support
- Screen Logger Message Format argument added

7.13 0.3.16 (2020-09-02)

- FileHandler now adds to all previously existing loggers

7.14 0.3.14 (2020-08-20)

- Pylint removed
- Bugfix

7.15 0.3.12 (2020-07-27)

- Individual loggers that skip global logging level change
- Pendulum module version bumped

7.16 0.3.9 (2020-07-20)

- Enum module usage
- Code testing update

7.17 0.3.8 (2020-07-13)

- Bugfix

7.18 0.3.6 (2020-07-01)

- Minor tweaks
- Bugfix

7.19 0.3.4 (2020-06-29)

- Documentation update
- Github Pages setup
- ReadTheDocs documentation setup

7.20 0.2.11 (2020-06-28)

- Refactoring
- Bugfix

7.21 0.2.6 (2020-06-27)

- GitHub Actions stabilized.

7.22 0.1.0 (2020-06-23)

- First release on PyPI.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

g

Index

S

T

V

W

Y