color$_{b}ucket_{l}ogger Documentation$

Release 0.2.0

Adrian Likins

May 31, 2020

Contents

Contents:

CHAPTER 1

color_bucket_logger

Python logging Formatter for colorizing logs per thread, process, logger name, or any record attribute

Using this logging formatter to make log records that share a common attribute share a color automatically.

For example, a process with three threads could show the log entries for each thread in a different color. The same can be done per process, or per logger name. Any log record attribute can be used to choose the color used for the log entry.

The entire log record, the particular log field ('level' or 'process' for ex.), or a group of fields can be colorized based on an attribute value.

For example, the fields for 'thread', 'threadName', 'process', 'processName' could be colorized based on the thread id.

1.1 Usage

1.2 Examples

Basic config colorized by logger name:

```
import logging

import color_bucket_logger

log = logging.getLogger('example')
log.setLevel(logging.DEBUG)
```

(continues on next page)

(continued from previous page)

```
log_format = '%(asctime)s %(process)s %(levelname)s %(name)s %(funcName)s --
↪%(message)s'

# Use logger name for the primary color of each entry
formatter = color_bucket_logger.ColorFormatter(fmt=log_format, default_color_by_attr=
↪'name')

handler = logging.StreamHandler()
handler.setLevel(logging.DEBUG)
handler.setFormatter(formatter)

# basicConfig will add our handler to the root logger
# Note 'handlers' arg is py3 only
logging.basicConfig(level=logging.DEBUG, handlers=[handler])
```

1.3 Color Group Examples

Example uses of color_groups:

```
# color almost everything by logger name
color_groups = [('name', ['filename', 'module', 'lineno', 'funcName', 'pathname'])]

color_groups = [
    ('process', ['default', 'message']),
    ('process', ['processName', 'process']),
    ('thread', ['default', 'threadName', 'message', 'unset', 'processName', 'exc_text
↪']),
    ('thread', ['threadName', 'thread']),
    ]

# color logger name, filename and lineno same as the funcName
color_groups = [('funcName', ['default', 'message', 'unset', 'name', 'filename',
↪'lineno'])]

# color message same as debug level
color_groups = [('levelname', ['levelname', 'levelno'])]

# color funcName, filename, lineno same as logger name
color_groups = [('name', ['funcName', 'filename', 'lineno'])]

# color groups can be based on non standard 'extra' attributes or log record
# attibutes created from filters. In this example, a 'task' attributes.
color_groups = [('task', ['task_uuid', 'task'])]

# color default, msg and playbook/play/task by the play
color_groups = [('play', ['default','message', 'unset', 'play', 'task'])]
```

1.4 License

- Free software: MIT license

1.5 Features

- TODO

CHAPTER 2

Installation

2.1 Stable release

To install color_bucket_logger, run this command in your terminal:

```
$ pip install color_bucket_logger
```

This is the preferred method to install color_bucket_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 color_bucket_logger can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/alikins/color_bucket_logger
```

Or download the tarball:

```
$ curl  -OL https://github.com/alikins/color_bucket_logger/tarball/master
```

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

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

CHAPTER 3

Usage

To use color_bucket_logger in a project:

```
import color_bucket_logger
```

CHAPTER 4

color_bucket_logger

4.1 color_bucket_logger package

Top-level package for color_bucket_logger.

class color_bucket_logger.**ColorFormatter**(*fmt=None*, *default_color_by_attr=None*, *color_groups=None*, *auto_color=False*, *datefmt=None*, *style=None*)

Bases: logging.Formatter

Base color bucket formatter

format(*record*)

Format the specified record as text.

The record's attribute dictionary is used as the operand to a string formatting operation which yields the returned string. Before formatting the dictionary, a couple of preparatory steps are carried out. The message attribute of the record is computed using LogRecord.getMessage(). If the formatting string uses the time (as determined by a call to usesTime(), formatTime() is called to format the event time. If there is exception information, it is formatted using formatException() and appended to the message.

class color_bucket_logger.**TermFormatter**(*fmt=None*, *default_color_by_attr=None*, *color_groups=None*, *auto_color=False*, *datefmt=None*, *color_mapper=None*)

Bases: *color_bucket_logger.formatter.ColorFormatter*

Formatter for terminals that colorizes attributes based on their value

Parameters

- **fmt** (*str*) – A logging.Formatter style log format string
- **default_color_by_attr** (*str, optional*) – The name of the log record attribute whose color will used by default for other records if they do not have a color set (by either 'auto_color' or by 'color_groups')
- **color_groups** (*iterable, optional*) – Define when a attribute should use the same color as another attribute.

For example, to make the 'processName' and 'message' attributes use the same color as 'process'.

color_groups is a list of tuples. The first element of each tuple is the "leader" attribute, and the second element is a list of "follower" attributes.

For the 'process' and 'processName' example:

```
color_groups=[('process', ['processName', 'message']]
```

- **auto_color** (*boolean, optional*) – If true, automatically calculate and use colors for each attribute. Defaults to False
- **datefmt** (*str, optional*) – Date format string as used by `logging.Formatter`

color_bucket_logger.**get_default_record_attrs**(*record_context*, *attr_list*)

Add default values to a log record_context for each attr in attr_list

Return a dict of {record_attr_name: some_default_value}. For example, a record_context with no 'stack_info' item would cause the return to be {'stack_info': None}.

For now, the default value is always None.

4.1.1 Submodules

color_bucket_logger.formatter module

The logging Formatters

class color_bucket_logger.formatter.**ColorFormatter**(*fmt=None*, *default_color_by_attr=None*, *color_groups=None*, *auto_color=False*, *datefmt=None*, *style=None*)

Bases: `logging.Formatter`

Base color bucket formatter

format(*record*)

Format the specified record as text.

The record's attribute dictionary is used as the operand to a string formatting operation which yields the returned string. Before formatting the dictionary, a couple of preparatory steps are carried out. The message attribute of the record is computed using LogRecord.getMessage(). If the formatting string uses the time (as determined by a call to usesTime(), formatTime() is called to format the event time. If there is exception information, it is formatted using formatException() and appended to the message.

class color_bucket_logger.formatter.**TermFormatter**(*fmt=None*, *default_color_by_attr=None*, *color_groups=None*, *auto_color=False*, *datefmt=None*, *color_mapper=None*)

Bases: *color_bucket_logger.formatter.ColorFormatter*

Formatter for terminals that colorizes attributes based on their value

Parameters

- **fmt** (*str*) – A `logging.Formatter` style log format string

- **default_color_by_attr** (*str, optional*) – The name of the log record attribute whose color will used by default for other records if they do not have a color set (by either 'auto_color' or by 'color_groups')
- **color_groups** (*iterable, optional*) – Define when a attribute should use the same color as another attribute.

  For example, to make the 'processName' and 'message' attributes use the same color as 'process'.

  color_groups is a list of tuples. The first element of each tuple is the "leader" attribute, and the second element is a list of "follower" attributes.

  For the 'process' and 'processName' example:

```
color_groups=[('process', ['processName', 'message']]
```

- **auto_color** (*boolean, optional*) – If true, automatically calculate and use colors for each attribute. Defaults to False
- **datefmt** (*str, optional*) – Date format string as used by `logging.Formatter`

color_bucket_logger.formatter.**default_attr_value**(*attr*)

color_bucket_logger.formatter.**find_format_attrs**(*format_string*)

color_bucket_logger.formatter.**get_default_record_attrs**(*record_context*, *attr_list*)

Add default values to a log record_context for each attr in attr_list

Return a dict of {record_attr_name: some_default_value}. For example, a record_context with no 'stack_info' item would cause the return to be {'stack_info': None}.

For now, the default value is always None.

color_bucket_logger.mapper module

class color_bucket_logger.mapper.**BaseColorMapper**(*fmt=None*, *default_color_by_attr=None*, *color_groups=None*, *format_attrs=None*, *auto_color=False*)

Bases: `object`

get_colors_for_attr(*record*)

get_level_color(*levelname*, *levelno*)

return color idx for logging levelname and levelno

get_name_color(*name*, *perturb=None*)

return color idx for 'name' string

get_process_colors(*record*)

return a tuple of pname_color, pid_color, tname_color, tid_color idx for process record

get_thread_color(*thread_id*)

return color idx for thread_id

custom_attrs = ['levelname', 'levelno', 'process', 'processName', 'thread', 'threadNam

default_color_groups = []

high_cardinality = {'args', 'asctime', 'created', 'message', 'msecs', 'relativeCreated

color_bucket_logger.styles module

class color_bucket_logger.styles.**PercentStyle**(*fmt*)
Bases: object

context_color_format_string(*format_string*, *format_attrs*)
For extending a format string for logging.Formatter to include attributes with color info.

ie:

```
'%(process)d %(threadName)s - %(msg)'
```

becomes:

```
'%(_cdl_process)s%(process)d%(_cdl_reset)s %(_cdl_threadName)%(threadName)s%(_
↪cdl_reset)s - %(msg)'
```

find_format_attrs(*format_string*)

format(*record*)

usesTime()

validate()
Validate the input format, ensure it matches the correct style

asctime_format = '%(asctime)s'

asctime_search = '%(asctime)'

attrs_pattern = re.compile('(?P<full_attr>%\\((?P<attr_name>.*?)\\).*?[dsf])')

attrs_pattern_str = '(?P<full_attr>%\\((?P<attr_name>.*?)\\).*?[dsf])'

color_fmt

default_format = '%(message)s'

named_fields_pattern = '(?P<full_attr>%\\((?P<attr_name>.*?)\\)(?P<conversion_flag>[#0

validation_pattern = re.compile('%\\(\\w+\\)[#0+ -]*(*|\\d+)?(\\.(*|\\d+))?[diouxe

class color_bucket_logger.styles.**StrFormatStyle**(*fmt*)
Bases: *color_bucket_logger.styles.PercentStyle*

context_color_format_string(*format_string*, *format_attrs*)
For extending a format string for logging.Formatter to include attributes with color info.

ie:

```
'%(process)d %(threadName)s - %(msg)'
```

becomes:

```
'%(_cdl_process)s%(process)d%(_cdl_reset)s %(_cdl_threadName)%(threadName)s%(_
↪cdl_reset)s - %(msg)'
```

find_format_attrs(*format_string*)

validate()
Validate the input format, ensure it is the correct string formatting style

asctime_format = '{asctime}'

asctime_search = '{asctime'

```
default_format = '{message}'
field_spec = re.compile('^(\\d+|\\w+)(\\.\\w+|\\[[^]]+\\])*$')
fmt_spec = re.compile('^(.?[<>=^])?[+ -]?#?0?(\\d+|{\\w+})?[,_]?(\\.(\\d+|{\\w+}))?[bc
uber_format_pattern = '(?P<full_attr>{(?P<attr_name>\\d+|\\w+)[:]?(?P<modifiers>(?P<al
```

color_bucket_logger.term_colors module

color_bucket_logger.term_colors.**DEFAULT_COLOR = 7**
The default color (white)

color_bucket_logger.term_colors.**DEFAULT_COLOR_IDX = 257**
The index for the default 'default' color

color_bucket_logger.term_colors.**NAMED_COLOR_IDXS = {'BLACK': 0, 'BLUE': 4, 'CYAN': 6, 'GRE**
Some named colors that map to the first 8 terminal colors

color_bucket_logger.term_colors.**NUMBER_OF_ALL_COLORS = 203**
The number of total colors when excluded and skipped colors are considered. The color mappers use this to know what number to modulus (%) by to figure out the color bucket.

color_bucket_logger.term_colors.**RESET_SEQ_IDX = 256**
The index for a 'reset'

color_bucket_logger.term_colors.**RGB_COLOR_OFFSET = 18**
Used to determine what color index we start from.

The xterm256 color indexes are:

0-7 normal term colors

8-15 bright term colors

16-231 are the rest from a 6x6x6 (216 color) rgb cube

16, 17 are black and very dark blue so they are skipped since they are hard to read.

232-255 are the grays (white to gray to black) and are skipped and why END_OF_THREAD_COLORS is 231.

color_bucket_logger.term_mapper module

class color_bucket_logger.term_mapper.**TermColorMapper**(*fmt=None*, *default_color_by_attr=None*, *color_groups=None*, *format_attrs=None*, *auto_color=False*)

Bases: *color_bucket_logger.mapper.BaseColorMapper*

get_colors_for_record(*record_context*, *format_attrs*)
For a record_context dict, compute color for each field and return a color dict

get_level_color(*levelname*, *levelno*)
return color idx for logging levelname and levelno

get_name_color(*name*, *perturb=None*)
return color idx for 'name' string

get_process_colors(*record_context*)
Given process/thread info, return reasonable colors for them.

Roughly:

- attempts to get a unique color per 'processName'
- **attempts to get a unique color per 'process' (pid)**
  - attempt to make those the same for "MainProcess"
  - for any other 'processName', the pname color and the pid color can be different
- if 'threadName' is "MainThread", make tname_color and tid_color match "MainProcess" pname_color and pid_color
- other 'threadName' values get a new color and new tid_color

Notes

This doesn't track any state so there is no ordering or prefence to the colors given out.

Parameters **record_context** (dict) – The log record_context to calculate process colors for

Returns The color indexes for 'processName', 'process', 'threadName', and 'thread'

Return type int, int, int, int

get_thread_color(*threadid*)
Calculate the color index for a threadid (tid) number

Parameters **threadid** (*int*) – The value of the LogRecord.threadid attribute (the tid)

Returns

- *int*
- *The color index to use*

LEVEL_COLORS = {'CRITICAL': 1, 'DEBUG': 4, 'ERROR': 1, 'INFO': 2, 'SUBDEBUG': 4, 'SUBW
A fixed map of logging level to color used by the default get_level_color()

NUMBER_OF_COLORS = 203

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/alikins/color_bucket_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

color_bucket_logger could always use more documentation, whether as part of the official color_bucket_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/alikins/color_bucket_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 *color_bucket_logger* for local development.

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

```
$ git clone git@github.com:your_name_here/color_bucket_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 color_bucket_logger
$ cd color_bucket_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 color_bucket_logger 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.

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.
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/alikins/color_bucket_logger/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ py.test tests.test_color_bucket_logger
```

CHAPTER 6

Credits

6.1 Development Lead

- Adrian Likins <adrian@likins.com>

6.2 Contributors

None yet. Why not be the first?

CHAPTER 7

History

7.1 0.2.0 (2019-05-30)

- Prep for release

7.2 0.1.0 (2017-06-15)

- First release on PyPI.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

Index

H

L

N

P

R

S

T

U

V