verboselogs

Release 1.2

June 22, 2016

Contents

1 verboselogs: Verbose logging level for Python's logging module **3**
- 1.1 Usage . . . 3
- 1.2 PyLint plugin . . . 5
- 1.3 Contact . . . 5
- 1.4 License . . . 5

2 API documentation **7**
- 2.1 `verboselogs` . . . 7
- 2.2 `verboselogs.pylint` . . . 8

Python Module Index **9**

Welcome to the documentation of *verboselogs* version 1.2!

The first part of the documentation is the readme which gives you a general overview of the *verboselogs* package:

CHAPTER 1

verboselogs: Verbose logging level for Python's logging module

The `verboselogs.VerboseLogger` class extends logging.Logger to add the log levels `VERBOSE` and `SPAM`. The `VERBOSE` level sits between the predefined `INFO` and `DEBUG` levels and `SPAM` sits between `DEBUG` and `NOTSET`. The code to do this is simple and short, but I still don't want to copy/paste it to every project I'm working on, hence this package. The table below shows the names, numeric values and descriptions of the predefined log levels and the `VERBOSE` and `SPAM` levels defined by this module. I've added some notes to the other levels, these are marked in bold:

Level	Numeric value	Description
NOTSET	0	When a logger is created, the level is set to NOTSET (note that the root logger is created with level WARNING). **In practice this level is never explicitly used; it's mentioned here only for completeness.**
SPAM	5	**Way too verbose for regular debugging, but nice to have when someone is getting desperate in a late night debugging session and decides that they want as much instrumentation as possible! :-)**
DEBUG	10	Detailed information, typically of interest only when diagnosing problems. **Usually at this level the logging output is so low level that it's not useful to users who are not familiar with the software's internals.**
VERBOSE	15	**Detailed information that should be understandable to experienced users to provide insight in the software's behavior; a sort of high level debugging information.**
INFO	20	Confirmation that things are working as expected.
WARNING	30	An indication that something unexpected happened, or indicative of some problem in the near future (e.g. 'disk space low'). The software is still working as expected.
ERROR	40	Due to a more serious problem, the software has not been able to perform some function.
CRITICAL	50	A serious error, indicating that the program itself may be unable to continue running.

1.1 Usage

It's very simple to start using the `verboselogs` package:

```
>>> import logging, verboselogs
>>> logger = verboselogs.VerboseLogger('verbose-demo')
>>> logger.addHandler(logging.StreamHandler())
>>> logger.setLevel(logging.VERBOSE)
>>> logger.verbose("Can we have verbose logging? %s", "Yes we can!")
```

Here's a skeleton of a very simple Python program with a command line interface and configurable logging:

```
"""
Usage: demo.py [OPTIONS]

This is the usage message of demo.py. Usually
this text explains how to use the program.

Supported options:
  -v, --verbose  make more noise
  -h, --help     show this message and exit
"""

import getopt
import logging
import sys
import verboselogs

logger = verboselogs.VerboseLogger('demo')
logger.addHandler(logging.StreamHandler())
logger.setLevel(logging.INFO)

# Command line option defaults.
verbosity = 0

# Parse command line options.
opts, args = getopt.getopt(sys.argv[1:], 'vqh', ['verbose', 'quiet', 'help'])

# Map command line options to variables.
for option, argument in opts:
    if option in ('-v', '--verbose'):
        verbosity += 1
    elif option in ('-q', '--quiet'):
        verbosity -= 1
    elif option in ('-h', '--help'):
        print __doc__.strip()
        sys.exit(0)
    else:
        assert False, "Unhandled option!"

# Configure logger for requested verbosity.
if verbosity >= 3:
    logger.setLevel(logging.SPAM)
elif verbosity >= 2:
    logger.setLevel(logging.DEBUG)
elif verbosity >= 1:
    logger.setLevel(logging.VERBOSE)
elif verbosity < 0:
    logger.setLevel(logging.WARNING)

# Your code goes here.
...
```

If you want to set `verboselogs.VerboseLogger` as the default logging class for all subsequent logger instances, you can do so:

```
import logging
import verboselogs
```

```
verboselogs.install()
logger = logging.getLogger(__name__) # will be a VerboseLogger instance
```

1.2 PyLint plugin

If using the above `verboselogs.install()` approach, Pylint is not smart enough to recognize that `logging` is using `verboselogs`, resulting in errors like:

```
E:285,24: Module 'logging' has no 'VERBOSE' member (no-member)
E:375,12: Instance of 'RootLogger' has no 'verbose' member (no-member)
```

To fix this, `verboselogs` provides a Pylint plugin `verboselogs.pylint` which, when loaded with `pylint --load-plugins verboselogs.pylint`, adds the `verboselogs` methods and constants to Pylint's understanding of the `logging` module.

1.3 Contact

The latest version of `verboselogs` is available on PyPI and GitHub. For bug reports please create an issue on GitHub. If you have questions, suggestions, etc. feel free to send me an e-mail at peter@peterodding.com.

1.4 License

This software is licensed under the MIT license.

© 2016 Peter Odding.

The second part is the documentation of the available modules:

CHAPTER 2

API documentation

The following documentation is based on the source code of version 1.2 of the *verboselogs* package.

2.1 `verboselogs`

Verbose and spam log levels for Python's `logging` module.

The `verboselogs` module defines the `VERBOSE` and `SPAM` constants, the `VerboseLogger` class and the `add_log_level()` and `install()` functions. At import time `add_log_level()` is used to register the custom log levels `VERBOSE` and `SPAM` with Python's `logging` module.

`verboselogs.`**`VERBOSE`** **= 15**

The numeric value of the 'verbose' log level (a number).

The value of `VERBOSE` positions the verbose log level between the `INFO` and `DEBUG` levels.

See also The `verbose()` method of the `VerboseLogger` class.

`verboselogs.`**`SPAM`** **= 5**

The numeric value of the 'spam' log level (a number).

The value of `SPAM` positions the spam log level between the `DEBUG` and `NOTSET` levels.

See also The `spam()` method of the `VerboseLogger` class.

`verboselogs.`**`install`**`()`

Make `VerboseLogger` the default logger class.

The `install()` function uses `setLoggerClass()` to configure `VerboseLogger` as the default class for all loggers created by `logging.getLogger()` after `install()` has been called. Here's how it works:

```
import logging
import verboselogs

verboselogs.install()
logger = logging.getLogger(__name__) # will be a VerboseLogger instance
```

`verboselogs.`**`add_log_level`**`(`*value*, *name*`)`

Add a new log level to the `logging` module.

Parameters

- **`value`** – The log level's number (an integer).
- **`name`** – The name for the log level (a string).

class `verboselogs.`**`VerboseLogger`**`(`**args*, ***kw*`)`

Custom logger class to support the additional logging levels.

This subclass of `logging.Logger` adds support for the additional logging methods *`verbose()`* and *`spam()`*. You can use *`install()`* to make *`VerboseLogger`* the default logger class.

`verbose``(`**args*, ***kw*`)`

Log a message with level *`VERBOSE`*. The arguments are interpreted as for `logging.debug()`.

`spam``(`**args*, ***kw*`)`

Log a message with level *`SPAM`*. The arguments are interpreted as for `logging.debug()`.

2.2 `verboselogs.pylint`

Python Module Index

v

verboselogs, 7

A

add_log_level() (in module verboselogs), 7

I

install() (in module verboselogs), 7

S

SPAM (in module verboselogs), 7
spam() (verboselogs.VerboseLogger method), 8

V

VERBOSE (in module verboselogs), 7
verbose() (verboselogs.VerboseLogger method), 8
VerboseLogger (class in verboselogs), 7
verboselogs (module), 7