
sprockets.logging

Release 1.3.2

December 11, 2015

1	Installation	3
2	Documentation	5
3	Requirements	7
4	Example	9
5	Source	11
6	License	13
6.1	API Reference	13
6.2	Examples	14
6.3	Version History	17
7	Indices and tables	19
	Python Module Index	21

Making logs nicer since 2015!

Installation

`sprockets.logging` is available on the [Python Package Index](#) and can be installed via `pip` or `easy_install`:

```
pip install sprockets.logging
```

Documentation

<https://sprocketslogging.readthedocs.org>

Requirements

- No external requirements

Example

This examples demonstrates the most basic usage of `sprockets.logging`

```
import logging
import sys

import sprockets.logging

formatter = logging.Formatter('%(levelname)s %(message)s {%(context)s}')
handler = logging.StreamHandler(sys.stdout)
handler.setFormatter(formatter)
handler.addFilter(sprockets.logging.ContextFilter(properties=['context']))
logging.Logger.root.addHandler(handler)
logging.Logger.root.setLevel(logging.DEBUG)

# Outputs: INFO Hi there {None}
logging.info('Hi there')

# Outputs: INFO No KeyError {bah}
logging.info('No KeyError', extra={'context': 'bah'})

# Outputs: INFO Now with context! {foo}
adapted = logging.LoggerAdapter(logging.Logger.root, extra={'context': 'foo'})
adapted.info('Now with context!')
```

Source

`sprockets.logging` source is available on Github at <https://github.com/sprockets/sprockets.logging>

sprockets.logging is released under the [3-Clause BSD license](#).

6.1 API Reference

Make good log output easier.

- *ContextFilter* adds fixed properties to a log record
- *JSONRequestFormatter* formats log records as JSON output
- **method: 'tornado_log_function' is for use as the** `:class'tornado.web.Application.log_function'` in conjunction with *JSONRequestFormatter* to output log lines as JSON.

class sprockets.logging.**ContextFilter** (*name=''*, *properties=None*)
Ensures that properties exist on a LogRecord.

Parameters **properties** (*list|None*) – optional list of properties that will be added to LogRecord instances if they are missing

This filter implementation will ensure that a set of properties exists on every log record which means that you can always refer to custom properties in a format string. Without this, referring to a property that is not explicitly passed in will result in an ugly `KeyError` exception.

class sprockets.logging.**JSONRequestFormatter** (*fmt=None*, *datefmt=None*)
Instead of spitting out a “human readable” log line, this outputs the log data as JSON.

extract_exc_record (*typ, val, tb*)

Create a JSON representation of the traceback given the records `exc_info`

Parameters

- **typ** (*Exception*) – Exception type of the exception being handled
- **instance val** (*Exception*) – instance of the Exception class
- **tb** (*traceback*) – traceback object with the call stack

Return type `dict`

format (*record*)

Return the log data as JSON

Parameters **logging.LogRecord** (*record*) – The record to format

Return type `str`

`sprockets.logging.currentframe()`

Return the frame object for the caller's stack frame.

`sprockets.logging.tornado_log_function(handler)`

Assigned when creating a `tornado.web.Application` instance by passing the method as the `log_function` argument:

```
app = tornado.web.Application([('/', RequestHandler)],
                             log_function=tornado_log_function)
```

6.2 Examples

6.2.1 Simple Usage

The following snippet uses `sprockets.logging.ContextFilter` to insert context information into a message using a `logging.LoggerAdapter` instance.

```
import logging
import sys

import sprockets.logging

formatter = logging.Formatter('%(levelname)s %(message)s %(context)s')
handler = logging.StreamHandler(sys.stdout)
handler.setFormatter(formatter)
handler.addFilter(sprockets.logging.ContextFilter(properties=['context']))
logging.Logger.root.addHandler(handler)
logging.Logger.root.setLevel(logging.DEBUG)

# Outputs: INFO Hi there {None}
logging.info('Hi there')

# Outputs: INFO No KeyError {bah}
logging.info('No KeyError', extra={'context': 'bah'})

# Outputs: INFO Now with context! {foo}
adapted = logging.LoggerAdapter(logging.Logger.root, extra={'context': 'foo'})
adapted.info('Now with context!')
```

6.2.2 Dictionary-based Configuration

This package begins to shine if you use the dictionary-based logging configuration offered by `logging.config.dictConfig()`. You can insert the custom filter and format string into the logging infrastructure and insert context easily with `logging.LoggerAdapter`.

```
import logging.config
import signal
import uuid

from tornado import ioloop, web
import sprockets.logging
```

```

LOG_CONFIG = {
  'version': 1,
  'handlers': {
    'console': {
      'class': 'logging.StreamHandler',
      'stream': 'ext://sys.stdout',
      'formatter': 'simple',
      'filters': ['context'],
    },
  },
  'formatters': {
    'simple': {
      'class': 'logging.Formatter',
      'format': '%(levelname)s %(name)s: %(message)s [%s]',
    },
  },
  'filters': {
    'context': {
      '()': 'sprockets.logging.ContextFilter',
      'properties': ['context'],
    },
  },
  'loggers': {
    'tornado': {
      'level': 'DEBUG',
    },
  },
  'root': {
    'handlers': ['console'],
    'level': 'DEBUG',
  },
  'incremental': False,
}

```

```

class RequestHandler(web.RequestHandler):

    def __init__(self, *args, **kwargs):
        self.parent_log = kwargs.pop('parent_log')
        super(RequestHandler, self).__init__(*args, **kwargs)

    def prepare(self):
        uniq_id = self.request.headers.get('X-UniqID', uuid.uuid4().hex)
        self.logger = logging.LoggerAdapter(
            self.parent_log.getChild('RequestHandler'),
            extra={'context': uniq_id})

    def get(self, object_id):
        self.logger.debug('fetchin %s', object_id)
        self.set_status(200)
        return self.finish()

    def sig_handler(signo, frame):
        logging.info('caught signal %d, stopping IO loop', signo)
        iol = ioloop.IOLoop.instance()
        iol.add_callback_from_signal(iol.stop)

```

```

if __name__ == '__main__':
    logging.config.dictConfig(LOG_CONFIG)
    logger = logging.getLogger('app')
    app = web.Application([
        web.url('/(?P<object_id>\w+)', RequestHandler,
            kwargs={'parent_log': logger}),
    ])
    app.listen(8000)
    signal.signal(signal.SIGINT, sig_handler)
    signal.signal(signal.SIGTERM, sig_handler)
    ioloop.IOLoop.instance().start()
    logger.info('IO loop stopped, exiting')

```

6.2.3 Tornado Application JSON Logging

If you're looking to log Tornado requests as JSON, the `sprockets.logging.JSONRequestFormatter` class works in conjunction with the `tornado_log_function()` method to output all Tornado log entries as JSON objects. In the following example, the dictionary-based configuration is expanded upon to include specify the `sprockets.logging.JSONRequestFormatter` as the formatter and passes `tornado_log_function()` in as the `log_function` when creating the Tornado application.

```

import logging.config
import signal
import uuid

from tornado import ioloop, web
import sprockets.logging

LOG_CONFIG = {
    'version': 1,
    'handlers': {
        'console': {
            'class': 'logging.StreamHandler',
            'stream': 'ext://sys.stdout',
            'formatter': 'simple',
            'filters': ['context']
        }
    },
    'formatters': {
        'simple': {
            '()': sprockets.logging.JSONRequestFormatter
        }
    },
    'filters': {
        'context': {
            '()': 'sprockets.logging.ContextFilter',
            'properties': ['context']
        }
    },
    'loggers': {
        'tornado': {
            'level': 'DEBUG'
        }
    },
    'root': {

```

```

    'handlers': ['console'],
    'level': 'DEBUG'
},
'incremental': False
}

class RequestHandler(web.RequestHandler):

    def __init__(self, *args, **kwargs):
        self.parent_log = kwargs.pop('parent_log')
        super(RequestHandler, self).__init__(*args, **kwargs)

    def prepare(self):
        uniq_id = self.request.headers.get('X-UniqID', uuid.uuid4().hex)
        self.logger = logging.LoggerAdapter(
            self.parent_log.getChild('RequestHandler'),
            extra={'context': uniq_id})

    def get(self, object_id):
        self.logger.debug('fetchin %s', object_id)
        self.set_status(200)
        return self.finish()

def sig_handler(signo, frame):
    logging.info('caught signal %d, stopping IO loop', signo)
    iol = ioloop.IOLoop.instance()
    iol.add_callback_from_signal(iol.stop)

if __name__ == '__main__':
    logging.config.dictConfig(LOG_CONFIG)
    logger = logging.getLogger('app')
    app = web.Application([
        web.url('/(?P<object_id>\w+)', RequestHandler,
            kwargs={'parent_log': logger}),
    ], log_function=sprockets.logging.tornado_log_function)
    app.listen(8000)
    signal.signal(signal.SIGINT, sig_handler)
    signal.signal(signal.SIGTERM, sig_handler)
    ioloop.IOLoop.instance().start()
    logger.info('IO loop stopped, exiting')

```

6.3 Version History

6.3.1 Next Release

- Updated `tornado_log_function` to work with Tornado 4.3.

6.3.2 1.3.2 Oct 2, 2015

- Switch to packaging as a package instead of a `py_module`.

6.3.3 1.3.1 Sep 14, 2015

- Fix `query_arguments` handling in Python 3

6.3.4 1.3.0 Aug 28, 2015

- Add the `traceback` and `environment` if set

6.3.5 1.2.1 Jun 24, 2015

- Fix a potential `KeyError` when a `HTTP` request object is not present.

6.3.6 1.2.0 Jun 23, 2015

- Monkeypatch `logging.currentframe`
- Include a logging message if it's there

6.3.7 1.1.0 Jun 18, 2015

- Added `sprockets.logging.JSONRequestFormatter`
- Added `sprockets.logging.tornado_log_function()`
- Added convenience constants and methods as a pass through to Python's logging package:
- `sprockets.logging.DEBUG` to `logging.DEBUG`
- `sprockets.logging.ERROR` to `logging.ERROR`
- `sprockets.logging.INFO` to `logging.INFO`
- `sprockets.logging.WARN` to `logging.WARN`
- `sprockets.logging.WARNING` to `logging.WARNING`
- `sprockets.logging.dictConfig()` to `logging.config.dictConfig()`
- `sprockets.logging.getLogger()` to `logging.getLogger()`

6.3.8 1.0.0 Jun 09, 2015

- Added `sprockets.logging.ContextFilter`

Indices and tables

- `genindex`
- `modindex`
- `search`

S

`sprockets.logging`, 13

C

ContextFilter (class in sprockets.logging), 13
currentframe() (in module sprockets.logging), 13

E

extract_exc_record() (sprockets.logging.JSONRequestFormatter method), 13

F

format() (sprockets.logging.JSONRequestFormatter method), 13

J

JSONRequestFormatter (class in sprockets.logging), 13

S

sprockets.logging (module), 13

T

tornado_log_function() (in module sprockets.logging), 14