pylog Documentation

Release 0.1.4

nir0s

August 03, 2014

Contents

Contents:

CHAPTER 1

Quick Start

Coming soon...

CHAPTER 2

Installation

Coming soon...

CHAPTER 3

CLI

Coming soon...

CHAPTER 4

Configuration

Coming soon...

CHAPTER 5

Advanced Configuration

Coming soon...

CHAPTER 6

Formatters

Formatters format the logs.

Contents:

pylog.formatters.**fake_data**(*data_type*)

class pylog.formatters.**BaseFormatter**(*config*)
Bases: object

generate_data()

class pylog.formatters.**CustomFormatter**(*config*)
Bases: pylog.formatters.BaseFormatter

generates log strings in a custom format

this is also the the formatter other formatters can rely on to generate application specific logs. see the ApacheAccessFormatter class for reference.

generate_data()
returns a log string

for every item in the format list, if an item in the data dict corresponds with it and the field's data equals "$RAND", use faker to fake an item for it. else, choose one item from the list randomly. if there no item in the data to correspond with the format, it will just append to format's field name to the log.

```
example:
 'CustomFormatter': {
     'format': ['name', ' - ', 'level'],
     'data': {
         'name': $RAND,
         'level': ['ERROR', 'DEBUG', 'INFO', 'CRITICAL'],
    }
}
```

the output of the above example might be:

```
Sally Fields - ERROR
or
Jason Banks - DEBUG
or
Danny Milwee - ERROR
or
...
```

class pylog.formatters.**JsonFormatter**(*config*)
Bases: pylog.formatters.BaseFormatter

generates log strings in json format

generate_data()
returns a json string

all fields in the data dict will be iterated over. if $RAND is set in one of the fields, random data will be generate_data for that field. If not, data will be chosen from the list.

```
example:
'JsonFormatter': {
    'data': {
        'date_time': '$RAND',
        'level': ['ERROR', 'DEBUG'],
        'address': '$RAND',
    }
},
```

the output of the above example might be:

```
{'date_time': '2006-11-05 13:31:09', 'name': 'Miss Nona Breitenberg DVM', 'level': 'ERROR'}
or
{'date_time': '1985-01-20 11:41:16', 'name': 'Almeda Lindgren', 'level': 'DEBUG'}  # NOQA
or
{'date_time': '1973-05-21 01:06:04', 'name': 'Jase Heaney', 'level': 'DEBUG'}  # NOQA
or
...
```

class pylog.formatters.**ApacheAccessFormatter**(*config*)
Bases: pylog.formatters.CustomFormatter

class pylog.formatters.**ApacheErrorFormatter**(*config*)
Bases: pylog.formatters.CustomFormatter

CHAPTER 7

Transports

Transports are the methods in which logs are sent.

Contents:

class pylog.transports.**BaseTransport**(*config*)
Bases: object

configure()

send(*client*, *log*)

class pylog.transports.**UDPTransport**(*config*)
Bases: pylog.transports.BaseTransport

configure()

send(*client*, *log*)

close()

class pylog.transports.**StreamTransport**(*config*)
Bases: pylog.transports.BaseTransport

configure()

send(*client*, *log*)

close()

class pylog.transports.**FileTransport**(*config*)
Bases: pylog.transports.BaseTransport

configure()

send(*client*, *log*)

close()

get_data()

class pylog.transports.**AmqpTransport**(*config*)
Bases: pylog.transports.BaseTransport

configure()

send(*client*, *log*)

close()

CHAPTER 8

API

Contents:

pylog.pylog.**init_logger**(*base_level=20*, *verbose_level=10*, *logging_config=None*)
initializes a base logger

you can use this to init a logger in any of your files. this will use config.py's LOGGER param and logging.dictConfig to configure the logger for you.

Parameters

- **base_level** (*int\|logging.LEVEL*) – desired base logging level
- **verbose_level** (*int\|logging.LEVEL*) – desired verbose logging level
- **logging_dict** (*dict*) – dictConfig based configuration. used to override the default configuration from config.py

Return type *python logger*

pylog.pylog.**set_global_verbosity_level**(*is_verbose_output=False*)
sets the global verbosity level for console and the lgr logger.

Parameters **is_verbose_output** (*bool*) – should be output be verbose

pylog.pylog.**get_current_time**()
returns the current time

pylog.pylog.**calculate_throughput**(*elapsed_time*, *messages*)
calculates throughput and extracts the number of seconds for the run from the elapsed time

Parameters

- **elapsed_time** – run time
- **messages** (*int*) – number of messages to write

Returns throughput and seconds

Return type tuple

pylog.pylog.**send**(*instance*, *client*, *format*, *format_config*, *messages*, *gap*, *batch*)
sends logs and prints the time it took to send all logs

Parameters

- **instance** – transport class instance
- **client** – client to use to send logs
- **format** (*string*) – formatter to use

- **format_config** (*dict*) – formatter configuration to use
- **messages** (*int*) – number of messages to send
- **gap** (*float*) – gap in seconds between 2 messages
- **batch** (*int*) – number of messages per batch

pylog.pylog.**config_transport**(*transports*, *transport*, *transport_config*)
returns a configured instance and client for the transport

Parameters

- **transport** (*string*) – transport to use
- **transport_config** (*dict*) – transport configuration

pylog.pylog.**generator**(*config=None*, *transport=None*, *formatter=None*, *gap=None*, *messages=None*, *batch=False*, *verbose=False*)
generates log messages

this will generate log message in the requested format and protocol.

Parameters

- **config** (*string*) – path to config file path
- **transport** (*string*) – transport type to use
- **formatter** (*string*) – formatter to use
- **gap** (*float*) – gap in seconds between 2 messages
- **messages** (*int*) – number of messages to send

pylog.pylog.**list_fake_types**()
prints a list of random data types with an example

exception pylog.pylog.**PylogError**
Bases: exceptions.Exception

CHAPTER 9

Indices and tables

- *genindex*
- *modindex*
- *search*

p