

---

# **graypy Documentation**

*Release 1.2.0*

**Sever Băneşiu, Nathan Klapstein**

**Aug 05, 2019**



**CONTENTS:**

- 1 graypy** **3**
- 1.1 Description . . . . . 3
- 1.2 Installing . . . . . 3
- 1.3 Usage . . . . . 4
- 1.4 Contributors . . . . . 6
  
- 2 graypy.handler module** **9**
  
- 3 graypy.rabbitmq module** **13**
  
- 4 Indices and tables** **15**
  
- Python Module Index** **17**
  
- Index** **19**



This code is open source, and is available on [GitHub](#).



## 1.1 Description

Python logging handlers that send log messages in the Graylog Extended Log Format (GELF).  
graypy supports sending GELF logs to both Graylog2 and Graylog3 servers.

## 1.2 Installing

### 1.2.1 Using pip

Install the basic graypy python logging handlers:

```
pip install graypy
```

Install with requirements for GELFRabbitHandler:

```
pip install graypy[amqp]
```

### 1.2.2 Using easy\_install

Install the basic graypy python logging handlers:

```
easy_install graypy
```

Install with requirements for GELFRabbitHandler:

```
easy_install graypy[amqp]
```

## 1.3 Usage

graypy sends GELF logs to a Graylog server via subclasses of the python `logging.Handler` class.

Below is the list of ready to run GELF logging handlers defined by graypy:

- `GELFUDPHandler` - UDP log forwarding
- `GELFTCPHandler` - TCP log forwarding
- `GELFTLSHandler` - TCP log forwarding with TLS support
- `GELFHTTPHandler` - HTTP log forwarding
- `GELFRabbitHandler` - RabbitMQ log forwarding

### 1.3.1 UDP Logging

UDP Log forwarding to a locally hosted Graylog server can be easily done with the `GELFUDPHandler`:

```
import logging
import graypy

my_logger = logging.getLogger('test_logger')
my_logger.setLevel(logging.DEBUG)

handler = graypy.GELFUDPHandler('localhost', 12201)
my_logger.addHandler(handler)

my_logger.debug('Hello Graylog.')
```

### 1.3.2 RabbitMQ Logging

Alternately, use `GELFRabbitHandler` to send messages to RabbitMQ and configure your Graylog server to consume messages via AMQP. This prevents log messages from being lost due to dropped UDP packets (`GELFUDPHandler` sends messages to Graylog using UDP). You will need to configure RabbitMQ with a `gelf_log` queue and bind it to the `logging.gelf` exchange so messages are properly routed to a queue that can be consumed by Graylog (the queue and exchange names may be customized to your liking).

```
import logging
import graypy

my_logger = logging.getLogger('test_logger')
my_logger.setLevel(logging.DEBUG)

handler = graypy.GELFRabbitHandler('amqp://guest:guest@localhost/', exchange='logging.
↪gelf')
my_logger.addHandler(handler)

my_logger.debug('Hello Graylog.')
```

### 1.3.3 Django Logging

It's easy to integrate graypy with Django's logging settings. Just add a new handler in your `settings.py`:



```

LOGGING = {
    'version': 1,
    # other dictConfig keys here...
    'handlers': {
        'graypy': {
            'level': 'WARNING',
            'class': 'graypy.GELFUDPHandler',
            'host': 'localhost',
            'port': 12201,
        },
    },
    'loggers': {
        'django.request': {
            'handlers': ['graypy'],
            'level': 'ERROR',
            'propagate': True,
        },
    },
}

```

### 1.3.4 Traceback Logging

By default log captured exception tracebacks are added to the GELF log as `full_message` fields:

```

import logging
import graypy

my_logger = logging.getLogger('test_logger')
my_logger.setLevel(logging.DEBUG)

handler = graypy.GELFUDPHandler('localhost', 12201)
my_logger.addHandler(handler)

try:
    puff_the_magic_dragon()
except NameError:
    my_logger.debug('No dragons here.', exc_info=1)

```

### 1.3.5 Default Logging Fields

By default a number of debugging logging fields are automatically added to the GELF log if available:

- `function`
- `pid`
- `process_name`
- `thread_name`

You can disable automatically adding these debugging logging fields by specifying `debugging_fields=False` in the handler's constructor:

```

handler = graypy.GELFUDPHandler('localhost', 12201, debugging_fields=False)

```

### 1.3.6 Adding Custom Logging Fields

graypy also supports including custom fields in the GELF logs sent to Graylog. This can be done by using Python's `LoggerAdapter` and `Filter` classes.

#### Using `LoggerAdapter`

`LoggerAdapter` makes it easy to add static information to your GELF log messages:

```
import logging
import graypy

my_logger = logging.getLogger('test_logger')
my_logger.setLevel(logging.DEBUG)

handler = graypy.GELFUDPHandler('localhost', 12201)
my_logger.addHandler(handler)

my_adapter = logging.LoggerAdapter(logging.getLogger('test_logger'),
                                   {'username': 'John'})

my_adapter.debug('Hello Graylog from John.')
```

#### Using `Filter`

`Filter` gives more flexibility and allows for dynamic information to be added to your GELF logs:

```
import logging
import graypy

class UsernameFilter(logging.Filter):
    def __init__(self):
        # In an actual use case would dynamically get this
        # (e.g. from memcache)
        self.username = 'John'

    def filter(self, record):
        record.username = self.username
        return True

my_logger = logging.getLogger('test_logger')
my_logger.setLevel(logging.DEBUG)

handler = graypy.GELFUDPHandler('localhost', 12201)
my_logger.addHandler(handler)

my_logger.addFilter(UsernameFilter())

my_logger.debug('Hello Graylog from John.')
```

## 1.4 Contributors

- Sever Banesiu

- Daniel Miller
- Tushar Makkar
- Nathan Klapstein



## GRAYPY.HANDLER MODULE

Logging Handlers that send messages in Graylog Extended Log Format (GELF)

```
class graypy.handler.BaseGELFHandler(chunk_size=1420, debugging_fields=True, extra_fields=True, fqdn=False, localname=None, facility=None, level_names=False, compress=True)
```

Bases: `logging.Handler`, `abc.ABC`

Abstract class defining the basic functionality of converting a `logging.LogRecord` into a GELF log. Provides the boilerplate for all GELF handlers defined within `graypy`.

```
__init__(chunk_size=1420, debugging_fields=True, extra_fields=True, fqdn=False, localname=None, facility=None, level_names=False, compress=True)
```

Initialize the BaseGELFHandler.

### Parameters

- **chunk\_size** (*int*) – Message chunk size. Messages larger than this size will be sent to Graylog in multiple chunks.
- **debugging\_fields** (*bool*) – If `True` add debug fields from the log record into the GELF logs to be sent to Graylog.
- **extra\_fields** (*bool*) – If `True` add extra fields from the log record into the GELF logs to be sent to Graylog.
- **fqdn** (*bool*) – If `True` use the fully qualified domain name of localhost to populate the `host` GELF field.
- **localname** (*str or None*) – If specified and `fqdn` is `False`, use the specified host-name to populate the `host` GELF field.
- **facility** (*str*) – If specified, replace the `facility` GELF field with the specified value. Also add a additional `_logger` GELF field containing the `LogRecord.name`.
- **level\_names** (*bool*) – If `True` use python logging error level name strings instead of syslog numerical values.
- **compress** (*bool*) – If `True` compress the GELF message before sending it to the Graylog server.

**makePickle** (*record*)

Convert a `logging.LogRecord` into bytes representing a GELF log

**Parameters** **record** (*logging.LogRecord*) – `logging.LogRecord` to convert into a GELF log.

**Returns** bytes representing a GELF log.

**Return type** `bytes`

**class** graypy.handler.ChunkedGELF (*message*, *size*)

Bases: `object`

Class that chunks a message into a GELF compatible chunks

**\_\_init\_\_** (*message*, *size*)

Initialize the ChunkedGELF message class

#### Parameters

- **message** (*bytes*) – The message to chunk.
- **size** (*int*) – The size of the chunks.

**encode** (*sequence*, *chunk*)

**message\_chunks** ()

**class** graypy.handler.GELFHTTPHandler (*host*, *port=12203*, *compress=True*, *path='/gelf'*, *timeout=5*, *\*\*kwargs*)

Bases: `graypy.handler.BaseGELFHandler`

GELF HTTP handler

**\_\_init\_\_** (*host*, *port=12203*, *compress=True*, *path='/gelf'*, *timeout=5*, *\*\*kwargs*)

Initialize the GELFHTTPHandler

#### Parameters

- **host** (*str*) – GELF HTTP input host.
- **port** (*int*) – GELF HTTP input port.
- **compress** (*bool*) – If `True` compress the GELF message before sending it to the Graylog server.
- **path** (*str*) – Path of the HTTP input. (see [http://docs.graylog.org/en/latest/pages/sending\\_data.html#gelf-via-http](http://docs.graylog.org/en/latest/pages/sending_data.html#gelf-via-http))
- **timeout** (*int*) – Number of seconds the HTTP client should wait before it discards the request if the Graylog server doesn't respond.

**emit** (*record*)

Convert a `logging.LogRecord` to GELF and emit it to Graylog via a HTTP POST request

**Parameters** **record** (`logging.LogRecord`) – `logging.LogRecord` to convert into a GELF log and emit to Graylog via a HTTP POST request.

**class** graypy.handler.GELFTCPHandler (*host*, *port=12201*, *\*\*kwargs*)

Bases: `graypy.handler.BaseGELFHandler`, `logging.handlers.SocketHandler`

GELF TCP handler

**\_\_init\_\_** (*host*, *port=12201*, *\*\*kwargs*)

Initialize the GELFTCPHandler

#### Parameters

- **host** (*str*) – GELF TCP input host.
- **port** (*int*) – GELF TCP input port.

**Attention:** GELF TCP does not support compression due to the use of the null byte (`\0`) as frame delimiter.

Thus, `handler.GELFTCPHandler` does not support setting `compress` to `True` and is locked to `False`.

**makePickle** (*record*)

Add a null terminator to generated pickles as TCP frame objects need to be null terminated

**Parameters** **record** (*logging.LogRecord*) – `logging.LogRecord` to create a null terminated GELF log.

**Returns** Null terminated bytes representing a GELF log.

**Return type** `bytes`

**class** `graypy.handler.GELFTLSHandler` (*host, port=12204, validate=False, ca\_certs=None, certfile=None, keyfile=None, \*\*kwargs*)

Bases: `graypy.handler.GELFTCPHandler`

GELF TCP handler with TLS support

**\_\_init\_\_** (*host, port=12204, validate=False, ca\_certs=None, certfile=None, keyfile=None, \*\*kwargs*)  
Initialize the GELFTLSHandler

**Parameters**

- **host** (*str*) – GELF TLS input host.
- **port** (*int*) – GELF TLS input port.
- **validate** (*bool*) – If `True`, validate the Graylog server's certificate. In this case specifying `ca_certs` is also required.
- **ca\_certs** (*str*) – Path to CA bundle file.
- **certfile** (*str*) – Path to the client certificate file.
- **keyfile** (*str*) – Path to the client private key. If the private key is stored with the certificate, this parameter can be ignored.

**makeSocket** (*timeout=1*)

Create a TLS wrapped socket

**class** `graypy.handler.GELFUDPHandler` (*host, port=12202, \*\*kwargs*)

Bases: `graypy.handler.BaseGELFHandler`, `logging.handlers.DatagramHandler`

GELF UDP handler

**\_\_init\_\_** (*host, port=12202, \*\*kwargs*)  
Initialize the GELFUDPHandler

**Parameters**

- **host** (*str*) – GELF UDP input host.
- **port** (*int*) – GELF UDP input port.

**send** (*s*)

Send a pickled string to a socket.

This function no longer allows for partial sends which can happen when the network is busy - UDP does not guarantee delivery and can deliver packets out of sequence.





## GRAYPY.RABBITMQ MODULE

Logging Handler integrating RabbitMQ and Graylog Extended Log Format (GELF)

**class** graypy.rabbitmq.**ExcludeFilter** (*name*)

Bases: `logging.Filter`

A subclass of `logging.Filter` which should be instantiated with the name of the logger which, together with its children, will have its events excluded (filtered out)

**\_\_init\_\_** (*name*)

Initialize the ExcludeFilter

**Parameters** *name* (*str*) – Name to match for within a `logging.LogRecord`'s name field for filtering.

**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** graypy.rabbitmq.**GELFRabbitHandler** (*url*, *exchange*='logging.gelf', *exchange\_type*='fanout', *virtual\_host*='/', *routing\_key*='', **\*\*kwargs**)

Bases: `graypy.handler.BaseGELFHandler`, `logging.handlers.SocketHandler`

RabbitMQ / GELF handler

---

**Note:** This handler ignores all messages logged by amqplib.

---

**\_\_init\_\_** (*url*, *exchange*='logging.gelf', *exchange\_type*='fanout', *virtual\_host*='/', *routing\_key*='', **\*\*kwargs**)

Initialize the GELFRabbitHandler

**Parameters**

- **url** (*str*) – RabbitMQ URL (ex: `amqp://guest:guest@localhost:5672/`)
- **exchange** (*str*) – RabbitMQ exchange. A queue binding must be defined on the server to prevent GELF logs from being dropped.
- **exchange\_type** (*str*) – RabbitMQ exchange type.
- **virtual\_host** (*str*) –
- **routing\_key** (*str*) –

**makePickle** (*record*)

Convert a `logging.LogRecord` into bytes representing a GELF log

**Parameters** `record` (*logging.LogRecord*) – `logging.LogRecord` to convert into a GELF log.

**Returns** bytes representing a GELF log.

**Return type** `bytes`

**makeSocket** (*timeout=1*)

A factory method which allows subclasses to define the precise type of socket they want.

**class** `graypy.rabbitmq.RabbitSocket` (*cn\_args, timeout, exchange, exchange\_type, routing\_key*)

Bases: `object`

**\_\_init\_\_** (*cn\_args, timeout, exchange, exchange\_type, routing\_key*)

Initialize self. See `help(type(self))` for accurate signature.

**close** ()

Close the connection to the RabbitMQ socket

**sendall** (*data*)

## INDICES AND TABLES

- genindex
- modindex



## PYTHON MODULE INDEX

### g

`graypy.handler`, 9  
`graypy.rabbitmq`, 13



## Symbols

`__init__()` (*graypy.handler.BaseGELFHandler method*), 9  
`__init__()` (*graypy.handler.ChunkedGELF method*), 10  
`__init__()` (*graypy.handler.GELFHTTPHandler method*), 10  
`__init__()` (*graypy.handler.GELFTCPHandler method*), 10  
`__init__()` (*graypy.handler.GELFTLSHandler method*), 11  
`__init__()` (*graypy.handler.GELFUDPHandler method*), 11  
`__init__()` (*graypy.rabbitmq.ExcludeFilter method*), 13  
`__init__()` (*graypy.rabbitmq.GELFRabbitHandler method*), 13  
`__init__()` (*graypy.rabbitmq.RabbitSocket method*), 14

## B

`BaseGELFHandler` (*class in graypy.handler*), 9

## C

`ChunkedGELF` (*class in graypy.handler*), 9  
`close()` (*graypy.rabbitmq.RabbitSocket method*), 14

## E

`emit()` (*graypy.handler.GELFHTTPHandler method*), 10  
`encode()` (*graypy.handler.ChunkedGELF method*), 10  
`ExcludeFilter` (*class in graypy.rabbitmq*), 13

## F

`filter()` (*graypy.rabbitmq.ExcludeFilter method*), 13

## G

`GELFHTTPHandler` (*class in graypy.handler*), 10  
`GELFRabbitHandler` (*class in graypy.rabbitmq*), 13  
`GELFTCPHandler` (*class in graypy.handler*), 10  
`GELFTLSHandler` (*class in graypy.handler*), 11

`GELFUDPHandler` (*class in graypy.handler*), 11  
`graypy.handler` (*module*), 9  
`graypy.rabbitmq` (*module*), 13

## M

`makePickle()` (*graypy.handler.BaseGELFHandler method*), 9  
`makePickle()` (*graypy.handler.GELFTCPHandler method*), 11  
`makePickle()` (*graypy.rabbitmq.GELFRabbitHandler method*), 13  
`makeSocket()` (*graypy.handler.GELFTLSHandler method*), 11  
`makeSocket()` (*graypy.rabbitmq.GELFRabbitHandler method*), 14  
`message_chunks()` (*graypy.handler.ChunkedGELF method*), 10

## R

`RabbitSocket` (*class in graypy.rabbitmq*), 14

## S

`send()` (*graypy.handler.GELFUDPHandler method*), 11  
`sendall()` (*graypy.rabbitmq.RabbitSocket method*), 14