

Stack Overflow is a community of 4.7 million programmers, just like you, helping each other.

Join them; it only takes a minute:

Sign up

Join the Stack Overflow community to:



Ask programming questions



Answer and help your peers



Get recognized for your expertise

## Flask logging - Cannot get it to write to a file

USE STACK OVERFLOW TO FIND THE BEST DEVELOPERS



Ok, here's the code where I setup everything:

```
if __name__ == '__main__':
    app.debug = False

    applogger = app.logger

    file_handler = FileHandler("error.log")
    file_handler.setLevel(logging.DEBUG)

    applogger.setLevel(logging.DEBUG)
    applogger.addHandler(file_handler)

    app.run(host='0.0.0.0')
```

What happens is

1. error.log gets created
2. Nothing is ever written to it
3. Despite not adding a StreamHandler and setting debug to false I still get everything to STDOUT (this might be correct, but still seems weird)

Am I totally off here somewhere or what is happening?

python logging python-2.7 flask

asked Jul 19 '13 at 9:46



fleshgolem

182 1 12

What happens when you set `app.debug = True` ? – dAnjou Jul 19 '13 at 13:53

1 Nothing changes – fleshgolem Jul 19 '13 at 14:29

## 5 Answers

Why not do it like this:

```
if __name__ == '__main__':
    init_db() # or whatever you need to do

    import logging
    logging.basicConfig(filename='error.log', level=logging.DEBUG)

    app.run(host="0.0.0.0")
```

If you now start you application, you'll see that error.log contains:

```
INFO:werkzeug: * Running on http://0.0.0.0:5000/
```

For more info, visit <http://docs.python.org/2/howto/logging.html>

Okay, as you insist that you cannot have two handler with the method I showed you, I'll add an example that makes this quite clear. First, add this logging code to your main:

```
import logging, logging.config, yaml
logging.config.dictConfig(yaml.load(open('logging.conf')))
```

Now also add some debug code, so that we see that our setup works:

```
logfile = logging.getLogger('file')
logconsole = logging.getLogger('console')
logfile.debug("Debug FILE")
logconsole.debug("Debug CONSOLE")
```

All what is left is the "logging.conf" program. Let's use that:

```
version: 1
formatters:
  hiformat:
    format: 'HI %(asctime)s - %(name)s - %(levelname)s - %(message)s'
  simple:
    format: '%(asctime)s - %(name)s - %(levelname)s - %(message)s'
handlers:
  console:
    class: logging.StreamHandler
    level: DEBUG
    formatter: hiformat
    stream: ext://sys.stdout
  file:
    class: logging.FileHandler
    level: DEBUG
    formatter: simple
    filename: errors.log
loggers:
  console:
    level: DEBUG
    handlers: [console]
    propagate: no
  file:
    level: DEBUG
    handlers: [file]
    propagate: no
root:
  level: DEBUG
  handlers: [console,file]
```

This config is more complicated than needed, but it also shows some features of the logging module.

Now, when we run our application, we see this output (werkzeug- and console-logger):


```
HI 2013-07-22 16:36:13,475 - console - DEBUG - Debug CONSOLE
HI 2013-07-22 16:36:13,477 - werkzeug - INFO - * Running on http://0.0.0.0:5000/
```

Also note that the custom formatter with the "HI" was used.

Now look at the "errors.log" file. It contains:

```
2013-07-22 16:36:13,475 - file - DEBUG - Debug FILE
2013-07-22 16:36:13,477 - werkzeug - INFO - * Running on http://0.0.0.0:5000/
```

edited Jul 22 '13 at 14:38

 **HolgerSchurig**  
900 ● 5 ● 8

answered Jul 22 '13 at 10:15

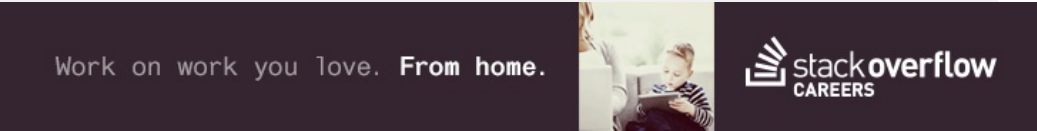
Because I actually aim to implement several handlers that log at different levels and this approach wouldn't get me far – [fleshgolem](#) Jul 22 '13 at 10:21

@fleshgolem: you still can create several handlers. Just do that directly with Python's logging. – [HolgerSchurig](#) Jul 22 '13 at 10:34

@fleshgolem: look at the URL I have you and search for logging.conf. There you'll see how to create a second logger. There you have the loggers 'root' and 'simpleExample', both can have different log-levels, for example. – [HolgerSchurig](#) Jul 22 '13 at 10:41

Well, I'm not here to fight and that IS a very nice and detailed answer, so have your checkmark – [fleshgolem](#) Jul 22 '13 at 15:17

6 This isn't really using the app.logger that flask provides. – [Milimetric](#) Jul 25 '13 at 12:53



Ok, my failure stemmed from two misconceptions:

- 1) Flask will apparently ignore all your custom logging unless it is running in production mode
- 2) debug=False is not enough to let it run in production mode. You have to wrap the app in any sort of WSGI server to do so

After i started the app from gevent's WSGI server (and moving logging initialization to a more appropriate place) everything seems to work fine

```
answered Jul 22 '13 at 12:27
```



fleshgolem

182 ● 1 ● 1 ● 12

Your conclusion is wrong. The code I suggested to you runs a) with the built-in server, b) with Testing and/or debug mode, c) with several handlers. Additionally, your question was originally "Cannot get it to write to a file", and that question has been solved by me. Unfortunately, you enlarged your question midway and wanted something different ... next time, I'd suggest that you describe your problem way more precisely. – [HolgerSchurig](#) Jul 22 '13 at 13:25

- I don't entirely agree. I asked, why flask's internal logger wouldn't write to a file, you supplied a solution that circumvents that logger. Yes it works, but I don't see it as a precise answer. I agree though, that I stated my problem badly. I will just leave both open, so anyone who finds this might find help in any of the two – [fleshgolem](#) Jul 22 '13 at 14:14

There is no "flask internal logger". Flask is using (and configuring) the logger in Python's logging module. I actually found the Flask documentation on logging confusing, until I saw the statement "from logging import getLogger, ..." in flask/logging.py. From then on it was easy, because Python's logging module is very nicely documented. So no, I don't circumvent Flask's logger. If I would, how could Flask's log output possible end up in errors.log? – [HolgerSchurig](#) Jul 22 '13 at 14:59

The output you see in the console of your app is from the underlying Werkzeug logger that can be accessed through `logging.getLogger('werkzeug')`.

Your logging can function in both development and release by also adding handlers to that logger as well as the Flask one.

More information and example code: [Write Flask Requests to an Access Log](#).

answered Apr 19 '14 at 18:20



AsksAnyway

348 ● 1 ● 2 ● 14

Took me a while to discover this one my own, the 'werkzeug' logger is where all the actual http layed logging occurs. Unfortunately it doesn't format the logs and instead pushes '%s - - [%s] %s\n' % pre-formatted into the logger. So if you want to same style logs you'll need to recreate that formatting locally or call the `werkzeug._internal._log` method. – [Pyrce](#) Jan 26 '15 at 22:33

This works:

```
if __name__ == '__main__':
    import logging
    logFormatStr = '%(asctime)s p%(process)s %(pathname)s:%(lineno)d} %
(levelname)s - %(message)s'
    logging.basicConfig(format = logFormatStr, filename = "global.log",
level=logging.DEBUG)
    formatter = logging.Formatter(logFormatStr,'%m-%d %H:%M:%S')
    fileHandler = logging.FileHandler("summary.log")
    fileHandler.setLevel(logging.DEBUG)
    fileHandler.setFormatter(formatter)
    streamHandler = logging.StreamHandler()
    streamHandler.setLevel(logging.DEBUG)
    streamHandler.setFormatter(formatter)
    app.logger.addHandler(fileHandler)
    app.logger.addHandler(streamHandler)
    app.logger.info("Logging is set up.")
    app.run(host='0.0.0.0', port=8000, threaded=True)
```

edited Nov 17 '15 at 13:36

answered Nov 17 '15 at 13:13



nicodjimenez

196 ● 2 ● 8

I didn't like the other answers so I kept at it and it seems like I had to make my logging config AFTER Flask did it's own setup.

```
@app.before_first_request
def initialize():
    logger = logging.getLogger("your_package_name")
    logger.setLevel(logging.DEBUG)
    ch = logging.StreamHandler()
    ch.setLevel(logging.DEBUG)
    formatter = logging.Formatter(
        ""%(levelname)s in %(module)s [%(pathname)s:%(lineno)d]:\n%(message)s""
    )
    ch.setFormatter(formatter)
    logger.addHandler(ch)
```

My app is structured like

```
/package_name
__main__.py <- where I put my logging configuration
__init__.py <- convenience for myself, not necessary
/tests
/package_name <- Actual flask app
__init__.py
```

```
/views
/static
/templates
/lib
```

Following these directions <http://flask.pocoo.org/docs/0.10/patterns/packages/>

edited Mar 23 at 21:11

answered Mar 23 at 21:06



David

6,976 ● 4 ● 43 ● 77