# errorhandler Documentation

***Release 2.0.1***

**Simplistix Ltd**

**Jan 02, 2018**

# Contents

CHAPTER 1

# Installation Instructions

If you want to experiment with errorhandler, the easiest way to install it is to do the following in a virtualenv:

```
pip install errorhandler
```

If your package uses setuptools and you decide to use errorhandler, then you should add it as a requirement by adding an `install_requires` parameter in your call to `setup` as follows:

```
setup(
    # other stuff here
    install_requires=['errorhandler'],
    )
```

CHAPTER 2

# Using ErrorHandler

This is a handler for the python standard logging framework that can be used to tell whether messages have been logged at or above a certain level.

This can be useful when wanting to ensure that no errors have been logged before committing data back to a database.

## 2.1 Basic usage

First, you set up the error handler:

```
>>> from logging import getLogger
>>> from errorhandler import ErrorHandler
>>> logger = getLogger()
>>> e = ErrorHandler()
```

The handler starts off being un-fired:

```
>>> e.fired
False
```

Then you do whatever else you need to do, which may involve logging:

```
>>> logger.info('some information')
>>> e.fired
False
```

However, if any logging occurs at an error level or above:

```
>>> logger.error('an error')
```

Then the error handler becomes fired:

```
>>> e.fired
True
```

You can use this as a condition to only perform certain actions when no errors have been logged:

```
>>> if e.fired:
...   print("Not updating files as errors have occurred")
Not updating files as errors have occurred
```

## 2.2 Resetting

If your code does work in batches, you may wish to reset the error handler at the start of each batch:

```
>>> e.fired
True
>>> e.reset()
>>> e.fired
False
```

## 2.3 Registering for a particular logger

The error handler can be set to only trigger on a certain logger and its children:

```
>>> from logging import getLogger
>>> e = ErrorHandler(logger='b')
```

Using these three loggers as an example:

```
>>> a = getLogger()
>>> b = getLogger('b')
>>> c = getLogger('b.c')
```

Logging to *a* won't trigger the handler:

```
>>> a.critical('message')
>>> e.fired
False
```

Logging to *b* will trigger the handler:

```
>>> b.critical('message')
>>> e.fired
True
>>> e.reset()
>>> e.fired
False
```

Logging to *c* will also trigger the handler:

```
>>> c.critical('message')
>>> e.fired
True
```

## 2.4 Using a different log level

The logging level at which the *ErrorHandler* is fired can also be configured:

```
>>> from logging import INFO
>>> e = ErrorHandler(INFO)
```

Debugging messages still don't trigger:

```
>>> logger.debug('debugging')
>>> e.fired
False
```

But now informational messages do:

```
>>> logger.info('some information')
>>> e.fired
True
```

## 2.5 Installing and removing the handler

By default, the *ErrorHandler* is installed when it is created, but this doesn't have to be the case:

```
>>> e = ErrorHandler(install=False)
>>> logger.error('an error')
>>> e.fired
False
```

When you create an *ErrorHandler* like this, you have to install it before log messages will cause it to become fired:

```
>>> e.install()
>>> logger.error('an error')
>>> e.fired
True
```

However, it's always good practice to remove the handler when you're done, like this:

```
>>> e.remove()
```

# CHAPTER 3

# API Reference

**class** errorhandler.**ErrorHandler**(*level=logging.ERROR*, *logger=''*, *install=True*)

This constructs an ErrorHandler.

**Parameters**

- **level** – This specifies the logging level at which the error handler will fire. Any message logged at or above this level will trigger the error handler.
- **logger** – This specifies the logger on which the error handler will be installed. The default is the root logger.
- **install** – If True, the handler is automatically installed. If False, the handler has to be manually installed by calling its *install()* method

**install**()

Installs this *ErrorHandler* object in the logger specified during instantiation.

**reset**()

Resets this *ErrorHandler* object.

CHAPTER 4

# Development

This package is developed using continuous integration which can be found here:

https://travis-ci.org/Simplistix/errorhandler

The latest development version of the documentation can be found here:

http://errorhandler.readthedocs.org/en/latest/

If you wish to contribute to this project, then you should fork the repository found here:

https://github.com/Simplistix/errorhandler

Once that has been done and you have a checkout, you can follow these instructions to perform various development tasks:

## 4.1 Setting up a virtualenv

The recommended way to set up a development environment is to turn your checkout into a virtualenv and then install the package in editable form as follows:

```
$ virtualenv .
$ bin/pip install -U -e .[test,build]
```

## 4.2 Running the tests

Once you've set up a virtualenv, the tests can be run as follows:

```
$ bin/nosetests
```

## 4.3 Building the documentation

The Sphinx documentation is built by doing the following from the directory containing setup.py:

```
$ source bin/activate
$ cd docs
$ make html
```

To check that the description that will be used on PyPI renders properly, do the following:

```
$ python setup.py --long-description | rst2html.py > desc.html
```

The resulting `desc.html` should be checked by opening in a browser.

## 4.4 Making a release

To make a release, just update `versions.txt`, update the change log, tag it and push to https://github.com/Simplistix/errorhandler and Travis CI should take care of the rest.

Once Travis CI is done, make sure to go to https://readthedocs.org/projects/testfixtures/versions/ and make sure the new release is marked as an Active Version.

CHAPTER 5

# Changes

## 5.1 2.0.1 (6 Jun 2016)

- Package as a universal wheel.

## 5.2 2.0.0 (6 Jun 2016)

- Support for Python 3
- Documentation on Read The Docs
- Continuous testing using Travis CI
- Code coverage reporting through Coveralls

## 5.3 1.1.0 (7 Nov 2009)

- Switched to Sphinx documentation

## 5.4 1.0.0 (3 Dec 2008)

- Initial Release

CHAPTER 6

# License

CHAPTER 7

# Indices and tables

- genindex
- modindex
- search

# Python Module Index

## e

# Index