pep3143daemon Documentation

Release 0.0.5

Stephan Schultchen

July 13, 2015

Contents

Contents:

CHAPTER 1

Introduction

The main reason for the development of pep3143daemon was that the original implementation of PEP 3143 python-daemon seemed to be unmaintained, and failed to run under python 3.

Also the bundled Pidfile implementation was not working anymore, because of incompatibilities with newer version of external libraries.

The goal of pep3143daemon is to create a complete, and working PEP 3143 library. For this reason, also a working PidFile class is bundled with this Package.

The package only depend on Python build in modules.

This package is tested withPython 3.3/3.4 and Python 2.7, using the unittest framework, and additionally for python 2.7 with the mock library, which is part of unittest in Python 3.

1.1 Differences

pep3143daemon mostly sticks to the PEP, but does not implement some details that seem to be wrong. The main difference is the DaemonContext.close() method.

The close method of DeamonContext is implemented as a dummy method. According to the PEP, this method should mark the instance as closed. It would also call the __exit__ method of a PidFile context manager, if one is attached.

The close method is not implemented, because it is not possible to close a daemon in a sane way. But worse, removing the PidFile before the daemon is terminated, would allow a second instance to start. Which can lead to undefined behaviour.

The removal of the PidFile is now implemented in the PidFile class, distributed with the pep3143daemon package. The file will be removed via an atexit call.

The rest of the implementation sticks to the PEP 3143.

CHAPTER 2

Package Documentation

2.1 DaemonContext

class pep3143daemon.**DaemonContext**(*chroot_directory=None*, *working_directory='/'*, *umask=0*, *uid=None*, *gid=None*, *prevent_core=True*, *detach_process=None*, *files_preserve=None*, *pidfile=None*, *stdin=None*, *stdout=None*, *stderr=None*, *signal_map=None*)

Implementation of PEP 3143 DaemonContext class

This class should be instantiated only once in every program that has to become a Unix Daemon. Typically you should call its open method after you have done everything that may require root privileges. For example opening port <= 1024.

Each option can be passed as a keyword argument to the constructor, but can also be changed by assigning a new value to the corresponding attribute on the instance.

Altering attributes after open() is called, will have no effect. In future versions, trying to do so, will may raise a DaemonError.

Parameters

- **chroot_directory** (*str*) – Full path to the directory that should be set as effective root directory. If None, the root directory is not changed.
- **working_directory** (*str.*) – Full Path to the working directory to which to change to. If chroot_directory is not None, and working_directory is not starting with chroot_directory, working directory is prefixed with chroot_directory.
- **umask** (*int.*) – File access creation mask for this daemon after start
- **uid** (*int.*) – Effective user id after daemon start.
- **gid** (*int.*) – Effective group id after daemon start.
- **prevent_core** (*bool.*) – Prevent core file generation.
- **detach_process** (*bool.*) – If True, do the double fork magic. If the process was started by inet or an init like program, you may don´t need to detach. If not set, we try to figure out if forking is needed.
- **files_preserve** (*list*) – List of integers, or objects with a fileno method, that represent files that should not be closed while daemoninzing.
- **pidfile** (*Instance of Class that implements a pidfile behaviour*) – Instance that implements a pidfile, while daemonizing its acquire method will be called.
- **stdin** (*file object.*) – Redirect stdin to this file, if None, redirect to /dev/null.

- **stdout** (*file object.*) – Redirect stdout to this file, if None, redirect to /dev/null.
- **stderr** (*file object.*) – Redirect stderr to this file, if None, redirect to /dev/null.
- **signal_map** (*instance of dict*) – Mapping from operating system signal to callback actions.

close()
Dummy function

is_open
True when this instances open method was called

Returns bool

open()
Daemonize this process

Do everything that is needed to become a Unix daemon.

Returns None

Raise DaemonError

terminate(*signal_number*, *stack_frame*)
Terminate this process

Simply terminate this process by raising SystemExit. This method is called if signal.SIGTERM was received.

Check carefully if this really is what you want!

Most likely it is not!

You should implement a function/method that is able to cleanly shutdown you daemon. Like gracefully terminating child processes, threads. or closing files.

You can create a custom handler by overriding this method, ot setting a custom handler via the signal_map. It is also possible to set the signal handlers directly via signal.signal().

Returns None

Raise SystemExit

working_directory
The working_directory property

Returns str

2.2 DaemonError

class pep3143daemon.**DaemonError**
Exception raised by DaemonContext

2.3 PidFile

class pep3143daemon.**PidFile**(*pidfile*)
PidFile implementation for PEP 3143 Daemon.

This Class can also be used with pythons 'with' statement.

Parameters **pidfile** (*str*) – filename to be used as pidfile, including path

acquire()
Acquire the pidfile.

Create the pidfile, lock it, write the pid into it and register the release with atexit.

Returns None

Raise SystemExit

release()
Release the pidfile.

Close and delete the Pidfile.

Returns None

See also:

pep3143daemon´s source code

CHAPTER 3

Usage Example

3.1 Example Daemon

Simple Daemon that sends a syslog message every 2 seconds, and terminates after two minutes:

```
from pep3143daemon import DaemonContext, PidFile
import syslog, time


counter = 60
pid='/tmp/pep3134daemon_example.pid'

pidfile = PidFile(pid)
daemon = DaemonContext(pidfile=pidfile)
# we could have written this also as:
# daemon.pidfile = pidfile

print('pidfile is: {0}'.format(pid))
print('daemonizing...')

daemon.open()

syslog.syslog('pep3134daemon_example: daemonized')

while counter > 0:
    syslog.syslog('pep3134daemon_example: still running')
    counter -= 1
    syslog.syslog('pep3134daemon_example: counter: {0}'.format(counter))
    time.sleep(2)

syslog.syslog('pep3134daemon_example: terminating...')
```

CHAPTER 4

Indices and tables

- genindex
- modindex
- search