
EasyProcess Documentation

Release 0.1.4

ponty

February 16, 2013

CONTENTS

1	Basic usage	2
2	Installation	3
2.1	General	3
2.2	Ubuntu	3
2.3	Uninstall	3
3	Usage	4
3.1	Simple example	4
3.2	General	4
3.3	Shell commands	5
3.4	return_code	5
3.5	With	5
3.6	Timeout	6
3.7	Logging	6
3.8	Alias	7
3.9	Replacing existing functions	7
3.10	extract_version	8
4	API	9
5	Indices and tables	12
	Python Module Index	13
	Index	14

EasyProcess

Date February 16, 2013

PDF EasyProcess.pdf

Contents:

EasyProcess is an easy to use python subprocess interface.

Links:

- home: <https://github.com/ponty/EasyProcess>
- documentation: <http://ponty.github.com/EasyProcess>

Features:

- layer on top of `subprocess` module
- easy to start, stop programs
- easy to get standard output/error, return code of programs
- command can be list or string
- logging
- timeout
- unit-tests
- cross-platform, development on linux
- global config file with program aliases
- shell is not supported
- pipes are not supported
- stdout/stderr is set only after the subprocess has finished
- stop() does not kill whole subprocess tree
- unicode support
- supported python versions: 2.5, 2.6, 2.7, 3.1, 3.2, PyPy

Known problems:

- none

Similar projects:

- execute (<http://pypi.python.org/pypi/execute>)
- commandwrapper (<http://pypi.python.org/pypi/commandwrapper>)
- extcmd (<http://pypi.python.org/pypi/extcmd>)
- sh (<https://github.com/amoffat/sh>)
- envoy (<https://github.com/kennethreitz/envoy>)
- plumbum (<https://github.com/tomerfiliba/plumbum>)

BASIC USAGE

```
>>> from easyprocess import EasyProcess
>>> EasyProcess('python --version').call().stderr
u'Python 2.6.6'
```

INSTALLATION

2.1 General

- install `pip`
- install the program:

```
# as root
pip install EasyProcess
```

2.2 Ubuntu

```
sudo apt-get install python-pip
sudo pip install EasyProcess
```

2.3 Uninstall

```
# as root
pip uninstall EasyProcess
```

USAGE

3.1 Simple example

Example program:

```
from easyprocess import EasyProcess
import sys

v = EasyProcess([sys.executable, '--version']).call().stderr
print('your python version:%s' % v)
```

Output:

```
$ python -m easyprocess.examples.ver
your python version:Python 2.7.3
```

3.2 General

```
>>> from easyprocess import EasyProcess
>>>
>>> # Run program, wait for it to complete, get stdout (command is string):
>>> EasyProcess('python -c "print 3"]').call().stdout
u'3'
>>>
>>> # Run program, wait for it to complete, get stdout (command is list):
>>> EasyProcess(['python', '-c', 'print 3']).call().stdout
u'3'
>>>
>>> # Run program, wait for it to complete, get stderr:
>>> EasyProcess('python --version').call().stderr
u'Python 2.7.3'
>>>
>>> # Run program, wait for it to complete, get return code:
>>> EasyProcess('python --version').call().return_code
0
>>>
>>> # Run program, wait 1 second, stop it, get stdout:
>>> print EasyProcess('ping localhost').start().sleep(1).stop().stdout
PING localhost (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_req=1 ttl=64 time=0.031 ms
>>>
>>> # Unicode support
>>> EasyProcess(['python', '-c', 'print unichr(0x03A9).encode("utf-8")']).call().stdout
u'\u03a9'
```

3.3 Shell commands

Shell commands are not supported.

Warning: `echo` is a shell command on Windows (there is no `echo.exe`), but it is a program on Linux

3.4 return_code

`EasyProcess.return_code` is `None` until `EasyProcess.stop()` or `EasyProcess.wait()` is called.

```
>>> from easyprocess import EasyProcess
>>>
>>> # process has finished but no stop() or wait() was called
>>> print EasyProcess('echo hello').start().sleep(0.5).return_code
None
>>>
>>> # wait()
>>> print EasyProcess('echo hello').start().wait().return_code
0
>>>
>>> # stop() after process has finished
>>> print EasyProcess('echo hello').start().sleep(0.5).stop().return_code
0
>>>
>>> # stop() before process has finished
>>> print EasyProcess('sleep 2').start().stop().return_code
-15
>>>
>>> # same as start().wait().stop()
>>> print EasyProcess('echo hello').call().return_code
0
```

3.5 With

By using `with` statement the process is started and stopped automatically:

```
from easyprocess import EasyProcess
with EasyProcess('ping 127.0.0.1') as proc: # start()
    # communicate with proc
    pass
# stopped
```

Equivalent with:

```
from easyprocess import EasyProcess
proc = EasyProcess('ping 127.0.0.1').start()
try:
    # communicate with proc
    pass
finally:
    proc.stop()
```


3.6 Timeout

This was implemented with “daemon thread”.

“The entire Python program exits when only daemon threads are left.”
<http://docs.python.org/library/threading.html>

```
>>> from easyprocess import EasyProcess
>>> # Run ping with timeout
>>> print EasyProcess('ping localhost').call(timeout=1).stdout
PING localhost (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_req=1 ttl=64 time=0.019 ms
64 bytes from localhost (127.0.0.1): icmp_req=2 ttl=64 time=0.030 ms
```

3.7 Logging

Example program:

```
from easyprocess import EasyProcess
import logging

# turn on logging
logging.basicConfig(level=logging.DEBUG)

EasyProcess('python --version').call()
EasyProcess('ping localhost').start().sleep(1).stop()
EasyProcess('python --version').check()
try:
    EasyProcess('bad_command').check()
except Exception, detail:
    print detail

try:
    EasyProcess('sh -c bad_command').check()
except Exception, detail:
    print detail
```

Output:

```
$ python -m easyprocess.examples.log
DEBUG:easyprocess:param: "python --version" command: ['python', '--version'] ("python --version")
DEBUG:easyprocess:reading config: /home/titi/.easyprocess.cfg
DEBUG:easyprocess:process was started (pid=21281)
DEBUG:easyprocess:process has ended
DEBUG:easyprocess:return code=0
DEBUG:easyprocess:stdout=
DEBUG:easyprocess:stderr=Python 2.7.3
DEBUG:easyprocess:param: "ping localhost" command: ['ping', 'localhost'] ("ping localhost")
DEBUG:easyprocess:process was started (pid=21282)
DEBUG:easyprocess:stopping process (pid=21282 cmd=["'ping', 'localhost'"])
DEBUG:easyprocess:process is active -> sending SIGTERM
DEBUG:easyprocess:process has ended
DEBUG:easyprocess:return code=-15
DEBUG:easyprocess:stdout=PING localhost (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_req=1 ttl=64 time=0.022 ms
64 bytes from localhost (127.0.0.1): icmp_req=2 ttl=64 time=0.025 ms
DEBUG:easyprocess:stderr=
DEBUG:easyprocess:param: "python --version" command: ['python', '--version'] ("python --version")
DEBUG:easyprocess:process was started (pid=21361)
DEBUG:easyprocess:process has ended
DEBUG:easyprocess:return code=0
```

```
DEBUG:easyprocess:stdout=
DEBUG:easyprocess:stderr=Python 2.7.3
DEBUG:easyprocess:param: "bad_command" command: ['bad_command'] ("bad_command")
DEBUG:easyprocess:OSError exception:[Errno 2] No such file or directory
DEBUG:easyprocess:param: "sh -c bad_command" command: ['sh', '-c', 'bad_command'] ("sh -c bad_com
DEBUG:easyprocess:process was started (pid=21364)
DEBUG:easyprocess:process has ended
DEBUG:easyprocess:return code=127
DEBUG:easyprocess:stdout=
DEBUG:easyprocess:stderr=sh: 1: bad_command: not found
start error <EasyProcess cmd_param=bad_command alias={alias} cmd=['bad_command'] ({scmd}) oserror:
check error, return code is not zero! <EasyProcess cmd_param=sh -c bad_command alias={alias} cmd=
```

3.8 Alias

You can define an alias for EasyProcess calls by editing your config file (\$HOME/.easyprocess.cfg) This can be used for:

- testing different version of the same program
- redirect calls
- program path can be defined here. (Installed programs are not in \$PATH on Windows)

start python and print python version:

```
>>> from easyprocess import EasyProcess
>>> EasyProcess('python --version').call().stderr
'Python 2.6.6'
```

edit the config file: \$HOME/.easyprocess.cfg:

```
[link]
python=/usr/bin/python2.7
```

restart python and print python version again:

```
>>> from easyprocess import EasyProcess
>>> EasyProcess('python --version').call().stderr
'Python 2.7.0+'
```

3.9 Replacing existing functions

Replacing os.system:

```
retcode = os.system("ls -l")
==>
p = EasyProcess("ls -l").call()
retcode = p.return_code
print p.stdout
```

Replacing subprocess.call:

```
retcode = subprocess.call(["ls", "-l"])
==>
p = EasyProcess(["ls", "-l"]).call()
retcode = p.return_code
print p.stdout
```

3.10 extract_version

`easyprocess.extract_version(txt)`

This function tries to extract the version from the help text of any program.

```
>>> from easyprocess import EasyProcess, extract_version
>>> print extract_version(EasyProcess('python --version').call().stderr)
2.7.3
```

API

Easy to use python subprocess interface.

```
class easyprocess.EasyProcess (cmd,      ubuntu_package=None,      url=None,      cwd=None,
                                use_temp_files=False)
    simple interface for subprocess
    shell is not supported (shell=False)
```

Warning: unicode is supported only for string list command (Python2.x) (check `shlex` for more information)

Parameters

- **cmd** – string ('ls -l') or list of strings (['ls', '-l'])
- **cwd** – working directory
- **use_temp_files** – use temp files instead of pipes for stdout and stderr, pipes can cause deadlock in some cases (see unit tests)

call (*timeout=None*)

Run command with arguments. Wait for command to complete.

same as:

1. `start()`
2. `wait()`
3. `stop()`

Return type self

check (*return_code=0*)

Run command with arguments. Wait for command to complete. If the exit code was as expected and there is no exception then return, otherwise raise `EasyProcessError`.

Parameters **return_code** – int, expected return code

Return type self

check_installed()

Used for testing if program is installed.

Run command with arguments. Wait for command to complete. If `OSError` raised, then raise `EasyProcessCheckInstalledError` with information about program installation

Parameters **return_code** – int, expected return code

Return type self

is_alive()

poll process using `subprocess.Popen.poll()`

Return type bool

pid
PID(`subprocess.Popen.pid`)

Return type int

return_code
returncode(`subprocess.Popen.returncode`)

Return type int

sendstop()
Kill process(`subprocess.Popen.terminate()`). Do not wait for command to complete.

Return type self

sleep(sec)
sleeping (same as `time.sleep()`)

Return type self

start()
start command in background and does not wait for it

Return type self

stop()
Kill process and wait for command to complete.

same as:

1. `sendstop()`
2. `wait()`

Return type self

wait(timeout=None)
Wait for command to complete.

Timeout:

- discussion: <http://stackoverflow.com/questions/1191374/subprocess-with-timeout>
- implementation: threading

Return type self

wrap(func, delay=0)

returns a function which:

1. start process
2. call func, save result
3. stop process
4. returns result

similar to `with` statement

Return type

exception `easyprocess.EasyProcessCheckInstalledError(easy_process)`

This exception is raised when a process run by `check()` returns a non-zero exit status or `OSError` is raised.

exception `easyprocess.EasyProcessError(easy_process, msg='')`

`easyprocess.Proc`
alias of `EasyProcess`

`easyprocess.extract_version(txt)`

This function tries to extract the version from the help text of any program.

INDICES AND TABLES

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

PYTHON MODULE INDEX

e

`easyprocess`, 4

INDEX

C

[call\(\)](#) (easyprocess.EasyProcess method), 9
[check\(\)](#) (easyprocess.EasyProcess method), 9
[check_installed\(\)](#) (easyprocess.EasyProcess method), 9

E

[EasyProcess](#) (class in easyprocess), 9
[easyprocess](#) (module), 4, 9
[EasyProcessCheckInstalledError](#), 10
[EasyProcessError](#), 10
[extract_version\(\)](#) (in module easyprocess), 8, 11

I

[is_alive\(\)](#) (easyprocess.EasyProcess method), 9

P

[pid](#) (easyprocess.EasyProcess attribute), 10
[Proc](#) (in module easyprocess), 10

R

[return_code](#) (easyprocess.EasyProcess attribute), 10

S

[sendstop\(\)](#) (easyprocess.EasyProcess method), 10
[sleep\(\)](#) (easyprocess.EasyProcess method), 10
[start\(\)](#) (easyprocess.EasyProcess method), 10
[stop\(\)](#) (easyprocess.EasyProcess method), 10

W

[wait\(\)](#) (easyprocess.EasyProcess method), 10
[wrap\(\)](#) (easyprocess.EasyProcess method), 10