ProcRunner Documentation

Release 2.2.0

Markus Gerstel

Sep 07, 2020

Contents:

CHAPTER 1

ProcRunner

Versatile utility function to run external processes

- Free software: BSD license
- Documentation: https://procrunner.readthedocs.io.

1.1 Features

- runs an external process and waits for it to finish
- does not deadlock, no matter the process stdout/stderr output behaviour
- returns the exit code, stdout, stderr (separately, both as bytestrings), as a subprocess.CompletedProcess object
- process can run in a custom environment, either as a modification of the current environment or in a new environment from scratch
- stdin can be fed to the process
- stdout and stderr is printed by default, can be disabled
- stdout and stderr can be passed to any arbitrary function for live processing (separately, both as unicode strings)
- optionally enforces a time limit on the process, raising a subprocess.TimeoutExpired exception if it is exceeded.

1.2 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

2.1 Stable release

To install ProcRunner, run this command in your terminal:

```
$ pip install procrunner
```

This is the preferred method to install ProcRunner, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for ProcRunner can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/DiamondLightSource/python-procrunner
```

Or download the tarball:

```
$ curl  -OL https://github.com/DiamondLightSource/python-procrunner/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

To use ProcRunner in a project:

```
import procrunner
result = procrunner.run(['/bin/ls', '/some/path/containing spaces'])
```

To test for successful completion:

```
assert not result.returncode
assert result.returncode == 0  # alternatively
result.check_returncode()  # raises subprocess.CalledProcessError()
```

To test for no STDERR output:

```
assert not result.stderr
assert result.stderr == b''  # alternatively
```

To run with a specific environment variable set:

```
result = procrunner.run(..., environment_override={ 'VARIABLE': 'value' })
```

To run with a specific environment:

```
result = procrunner.run(..., environment={ 'VARIABLE': 'value' })
```

To run in a specific directory:

```
result = procrunner.run(..., working_directory='/some/path')
```

CHAPTER 4

API

class procrunner.**ReturnObject**(*exitcode=None*, *command=None*, *stdout=None*, *stderr=None*, ***kw*)

Bases: subprocess.CompletedProcess

A subprocess.CompletedProcess-like object containing the executed command, stdout and stderr (both as bytestrings), and the exitcode. The check_returncode() function raises an exception if the process exited with a non-zero exit code.

procrunner.**run**(*command*, *timeout=None*, *debug=None*, *stdin=None*, *print_stdout=True*, *print_stderr=True*, *callback_stdout=None*, *callback_stderr=None*, *environment=None*, *environment_override=None*, *win32resolve=True*, *working_directory=None*, *raise_timeout_exception=False*)

Run an external process.

File system path objects (PEP-519) are accepted in the command, environment, and working directory arguments.

Parameters

- **command** (*array*) – Command line to be run, specified as array.
- **timeout** – Terminate program execution after this many seconds.
- **debug** (*boolean*) – Enable further debug messages. (deprecated)
- **stdin** – Optional bytestring that is passed to command stdin.
- **print_stdout** (*boolean*) – Pass stdout through to sys.stdout.
- **print_stderr** (*boolean*) – Pass stderr through to sys.stderr.
- **callback_stdout** – Optional function which is called for each stdout line.
- **callback_stderr** – Optional function which is called for each stderr line.
- **environment** (*dict*) – The full execution environment for the command.
- **environment_override** (*dict*) – Change environment variables from the current values for command execution.

- **win32resolve** (*boolean*) – If on Windows, find the appropriate executable first. This allows running of .bat, .cmd, etc. files without explicitly specifying their extension.
- **working_directory** (*string*) – If specified, run the executable from within this working directory.
- **raise_timeout_exception** (*boolean*) – Forward compatibility flag. If set then a subprocess.TimeoutExpired exception is raised instead of returning an object that can be checked for a timeout condition. Defaults to False, will be changed to True in a future release.

Returns The exit code, stdout, stderr (separately, as byte strings) as a subprocess.CompletedProcess object.

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/DiamondLightSource/python-procrunner/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

5.1.4 Write Documentation

ProcRunner could always use more documentation, whether as part of the official ProcRunner docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/DiamondLightSource/python-procrunner/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *procrunner* for local development.

1. Fork the *procrunner* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/python-procrunner.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv procrunner
$ cd procrunner/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 procrunner tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in HISTORY.rst/README.rst.

5.4 Tips

To run a subset of tests:

```
$ py.test tests.test_procrunner
```

5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 6

Credits

- Markus Gerstel

6.1 Contributors

None yet. Why not be the first?

CHAPTER 7

History

7.1 2.2.0 (2020-09-07)

- Calling the run() function with unnamed arguments (other than the command list as the first argument) is now deprecated. As a number of arguments will be removed in a future version the use of unnamed arguments will cause future confusion. Use explicit keyword arguments instead (#62).
- The run() function debug argument has been deprecated (#63). This is only used to debug the NonBlockingStream* classes. Those are due to be replaced in a future release, so the argument will no longer serve a purpose. Debugging information remains available via standard logging mechanisms.
- Final version supporting Python 3.5

7.2 2.1.0 (2020-09-05)

- Deprecated array access on the return object (#60). The return object will become a subprocess.CompletedProcess in a future release, which no longer allows array-based access. For a translation table of array elements to attributes please see the pull request linked above.
- Add a new parameter 'raise_timeout_exceptions' (#61). When set to 'True' a subprocess.TimeoutExpired exception is raised when the process runtime exceeds the timeout threshold. This defaults to 'False' and will be set to 'True' in a future release.

7.3 2.0.0 (2020-06-24)

- Python 3.5+ only, support for Python 2.7 has been dropped
- Deprecated function alias run_process() has been removed
- Fixed a stability issue on Windows

7.4 1.1.0 (2019-11-04)

- Add Python 3.8 support, drop Python 3.4 support

7.5 1.0.2 (2019-05-20)

- Stop environment override variables leaking into the process environment

7.6 1.0.1 (2019-04-16)

- Minor fixes on the return object (implement equality, mark as unhashable)

7.7 1.0.0 (2019-03-25)

- Support file system path objects (PEP-519) in arguments
- Change the return object to make it similar to subprocess.CompletedProcess, introduced with Python 3.5+

7.8 0.9.1 (2019-02-22)

- Have deprecation warnings point to correct code locations

7.9 0.9.0 (2018-12-07)

- Trap UnicodeEncodeError when printing output. Offending characters are replaced and a warning is logged once. Hints at incorrectly set PYTHONIOENCODING.

7.10 0.8.1 (2018-12-04)

- Fix a few deprecation warnings

7.11 0.8.0 (2018-10-09)

- Add parameter working_directory to set the working directory of the subprocess

7.12 0.7.2 (2018-10-05)

- Officially support Python 3.7

7.13 0.7.1 (2018-09-03)

- Accept environment variable overriding with numeric values.

7.14 0.7.0 (2018-05-13)

- Unicode fixes. Fix crash on invalid UTF-8 input.
- Clarify that stdout/stderr values are returned as bytestrings.
- Callbacks receive the data decoded as UTF-8 unicode strings with unknown characters replaced by ufffd (unicode replacement character). Same applies to printing of output.
- Mark stdin broken on Windows.

7.15 0.6.1 (2018-05-02)

- Maintenance release to add some tests for executable resolution.

7.16 0.6.0 (2018-05-02)

- Fix Win32 API executable resolution for commands containing a dot ('.') in addition to a file extension (say '.bat').

7.17 0.5.1 (2018-04-27)

- Fix Win32API dependency installation on Windows.

7.18 0.5.0 (2018-04-26)

- New keyword 'win32resolve' which only takes effect on Windows and is enabled by default. This causes procrunner to call the Win32 API FindExecutable() function to try and lookup non-.exe files with the corresponding name. This means .bat/.cmd/etc.. files can now be run without explicitly specifying their extension. Only supported on Python 2.7 and 3.5+.

7.19 0.4.0 (2018-04-23)

- Python 2.7 support on Windows. Python3 not yet supported on Windows.

7.20 0.3.0 (2018-04-17)

- run_process() renamed to run()
- Python3 compatibility fixes

7.21 0.2.0 (2018-03-12)

- Procrunner is now Python3 3.3-3.6 compatible.

7.22 0.1.0 (2018-03-12)

- First release on PyPI.

CHAPTER 8

Indices and tables

- search

Python Module Index

p

Index

P

R