

---

# **procrunner Documentation**

*Release 2.3.3*

**Diamond Light Source - Scientific Software**

**Jan 09, 2023**



---

## Contents:

---

<b>1</b>	<b>ProcRunner</b>	<b>1</b>
1.1	Features . . . . .	1
1.2	Credits . . . . .	2
<b>2</b>	<b>Installation</b>	<b>3</b>
2.1	Stable release . . . . .	3
2.2	From sources . . . . .	3
<b>3</b>	<b>Usage</b>	<b>5</b>
<b>4</b>	<b>API</b>	<b>7</b>
<b>5</b>	<b>Contributing</b>	<b>9</b>
5.1	Types of Contributions . . . . .	9
5.2	Get Started! . . . . .	10
5.3	Pull Request Guidelines . . . . .	11
5.4	Tips . . . . .	11
5.5	Deploying . . . . .	11
<b>6</b>	<b>Credits</b>	<b>13</b>
6.1	Contributors . . . . .	13
<b>7</b>	<b>History</b>	<b>15</b>
7.1	3.0.0 (2022-01-??) . . . . .	15
7.2	2.3.3 (2022-03-23) . . . . .	15
7.3	2.3.2 (2022-01-28) . . . . .	15
7.4	2.3.1 (2021-10-25) . . . . .	16
7.5	2.3.0 (2020-10-29) . . . . .	16
7.6	2.2.0 (2020-09-07) . . . . .	16
7.7	2.1.0 (2020-09-05) . . . . .	16
7.8	2.0.0 (2020-06-24) . . . . .	16
7.9	1.1.0 (2019-11-04) . . . . .	16
7.10	1.0.2 (2019-05-20) . . . . .	17
7.11	1.0.1 (2019-04-16) . . . . .	17
7.12	1.0.0 (2019-03-25) . . . . .	17
7.13	0.9.1 (2019-02-22) . . . . .	17
7.14	0.9.0 (2018-12-07) . . . . .	17

7.15	0.8.1 (2018-12-04)	17
7.16	0.8.0 (2018-10-09)	17
7.17	0.7.2 (2018-10-05)	17
7.18	0.7.1 (2018-09-03)	17
7.19	0.7.0 (2018-05-13)	18
7.20	0.6.1 (2018-05-02)	18
7.21	0.6.0 (2018-05-02)	18
7.22	0.5.1 (2018-04-27)	18
7.23	0.5.0 (2018-04-26)	18
7.24	0.4.0 (2018-04-23)	18
7.25	0.3.0 (2018-04-17)	18
7.26	0.2.0 (2018-03-12)	18
7.27	0.1.0 (2018-03-12)	19
<b>8</b>	<b>Indices and tables</b>	<b>21</b>
	<b>Python Module Index</b>	<b>23</b>
	<b>Index</b>	<b>25</b>



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.

### 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
```





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')
```



```
procrunner.run(command, *, timeout: Optional[float] = None, callback_stderr: Optional[Callable] = None, callback_stdout: Optional[Callable] = None, creationflags: int = 0, environment: Optional[dict[str, str]] = None, environment_override: Optional[dict[str, str]] = None, preexec_fn: Optional[Callable] = None, print_stderr: bool = True, print_stdout: bool = True, raise_timeout_exception: Any = Ellipsis, stdin: Optional[Union[bytes, int]] = None, win32resolve: bool = True, working_directory: Optional[str] = None) → subprocess.CompletedProcess
```

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.
- **stdin** – Optional bytestring that is passed to command stdin, or subprocess.DEVNULL to disable 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.
- **creationflags** – flags that will be passed to subprocess call
- **environment** (*dict*) – The full execution environment for the command.
- **environment\_override** (*dict*) – Change environment variables from the current values for command execution.
- **preexec\_fn** – pre-execution function, will be passed to subprocess call

- **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*) – Deprecated compatibility flag.

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

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.





- Markus Gerstel

## 6.1 Contributors

None yet. Why not be the first?



### 7.1 3.0.0 (2022-01-??)

- Drop Python 3.6, 3.7 support
- The `run()` function now returns a `subprocess.CompletedProcess` object, which no longer allows array access operations (those were deprecated in #60)
- The `run()` argument `'raise_timeout_exception'` is now set by default, a `'False'` value will lead to a `UserWarning` and a behavioural change. The argument is now deprecated and will be removed in a future version. (previously introduced in #61)
- Calling the `run()` function with multiple unnamed arguments is no longer supported (previously deprecated in #62)
- The `run()` function no longer accepts a `'debug'` argument (previously deprecated in #63)

### 7.2 2.3.3 (2022-03-23)

- Allow specifying `'preexec_fn'` and `'creationflags'` keywords, which will be passed through to the subprocess call

### 7.3 2.3.2 (2022-01-28)

- The `run()` function now understands `stdin=subprocess.DEVNULL` to close the subprocess `stdin`, rather than to connect through the existing `stdin`, which is the current default

## 7.4 2.3.1 (2021-10-25)

- Add Python 3.10 support

## 7.5 2.3.0 (2020-10-29)

- Add Python 3.9 support, drop Python 3.5 support
- Fix a file descriptor leak on subprocess execution

## 7.6 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

## 7.7 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_exception'` (#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.

## 7.8 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

## 7.9 1.1.0 (2019-11-04)

- Add Python 3.8 support, drop Python 3.4 support

## 7.10 1.0.2 (2019-05-20)

- Stop environment override variables leaking into the process environment

## 7.11 1.0.1 (2019-04-16)

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

## 7.12 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+

## 7.13 0.9.1 (2019-02-22)

- Have deprecation warnings point to correct code locations

## 7.14 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`.

## 7.15 0.8.1 (2018-12-04)

- Fix a few deprecation warnings

## 7.16 0.8.0 (2018-10-09)

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

## 7.17 0.7.2 (2018-10-05)

- Officially support Python 3.7

## 7.18 0.7.1 (2018-09-03)

- Accept environment variable overriding with numeric values.

## 7.19 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 uffd (unicode replacement character). Same applies to printing of output.
- Mark stdin broken on Windows.

## 7.20 0.6.1 (2018-05-02)

- Maintenance release to add some tests for executable resolution.

## 7.21 0.6.0 (2018-05-02)

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

## 7.22 0.5.1 (2018-04-27)

- Fix Win32API dependency installation on Windows.

## 7.23 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+.

## 7.24 0.4.0 (2018-04-23)

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

## 7.25 0.3.0 (2018-04-17)

- run\_process() renamed to run()
- Python3 compatibility fixes

## 7.26 0.2.0 (2018-03-12)

- Procrunner is now Python3 3.3-3.6 compatible.

## 7.27 0.1.0 (2018-03-12)

- First release on PyPI.





## CHAPTER 8

---

### Indices and tables

---

- search



**p**

procrunner, 7



## P

`procrunner` (*module*), 7

## R

`run()` (*in module procrunner*), 7