# proc

***Release 0.3***

September 25, 2015

Contents

Welcome to the online documentation for the Python package *proc*.

CHAPTER 1

# Introduction & usage

The first part of the documentation is the readme which contains general information about the *proc* package and getting started instructions. Here are the topics discussed in the readme:

## 1.1 proc: Linux process information interface

The Python package *proc* exposes process information available in the Linux process information pseudo-file system available at `/proc`. The *proc* package is currently tested on cPython 2.6, 2.7 and 3.4 and PyPy (2.7). The automated test suite regularly runs on Ubuntu Linux but other Linux variants (also those not based on Debian Linux) should work fine. For usage instructions please refer to the documentation.



### 1.1.1 Installation

The *proc* package is available on PyPI which means installation should be as simple as:

```
$ pip install proc
```

There's actually a multitude of ways to install Python packages (e.g. the per user site-packages directory, virtual environments or just installing system wide) and I have no intention of getting into that discussion here, so if this intimidates you then read up on your options before returning to these instructions ;-).

Once you've installed the *proc* package head over to the documentation for some examples of how the *proc* package can be used.

### 1.1.2 Design choices

The *proc* package was created with the following considerations in mind:

**Completely specialized to Linux** It parses `/proc` and nothing else ;-).

**Fully implemented in Python** No binary/compiled components, as opposed to psutil which is way more portable but requires a compiler for installation.

**Very well documented** The documentation should make it easy to get started (as opposed to procfs which I evaluated and eventually gave up on because I had to resort to reading through its source code just to be disappointed in its implementation).

**Robust implementation** Reading `/proc` is inherently sensitive to race conditions and the *proc* package takes this into account, in fact the test suite contains a test that creates race conditions in order to verify that they are handled correctly. The API of the *proc* package hides race conditions as much as possible and where this is not possible the consequences are clearly documented.

**A layered API design (where each layer is documented)** Builds higher level abstractions on top of lower level abstractions:

1. **The proc.core module:** A simple, fast and easy to use API for the process information available in `/proc`. If you're looking for a simple and/or fast interface to `/proc` that does the heavy lifting (parsing) for you then this is what you're looking for.
2. **The proc.tree module:** Builds on top of the `proc.core` module to provide an in-memory tree data structure that mimics the actual process tree, enabling easy searching and navigation through the process tree.
3. **The proc.cron module:** Implements the command line program `cron-graceful` which gracefully terminates cron daemons. This module builds on top of the `proc.tree` module as a demonstration of the possibilities of the *proc* package and as a practical tool that is ready to be used on any Linux system that has Python and cron installed.

### 1.1.3 History

I've been writing shell and Python scripts that parse `/proc` for years now (it seems so temptingly easy when you get started ;-). Sometimes I resorted to copy/pasting snippets of Python code between personal and work projects because the code was basically done, just not available in an easy to share form.

Once I started fixing bugs in diverging copies of that code I decided it was time to combine all of the features I'd grown to appreciate into a single well tested and well documented Python package with an easy to use API and share it with the world.

This means that, although I made my first commit on the *proc* package in March 2015, much of its code has existed for years in various forms.

### 1.1.4 Similar projects

Below are several other Python libraries that expose process information. If the *proc* package isn't working out for you consider trying one of these. The summaries are copied and/or paraphrased from the documentation of each package:

**psutil** A cross-platform library for retrieving information on running processes and system utilization (CPU, memory, disks, network) in Python.

**procpy** A Python wrapper for the procps library and a module containing higher level classes (with some extensions compared to procps).

**procfs** Python API for the Linux `/proc` virtual filesystem.

### 1.1.5 Contact

The latest version of *proc* is available on PyPI and GitHub. The documentation is hosted on Read the Docs. For bug reports please create an issue on GitHub. If you have questions, suggestions, etc. feel free to send me an e-mail at peter@peterodding.com.

### 1.1.6 License

This software is licensed under the MIT license.

© 2015 Peter Odding.

CHAPTER 2

# API documentation

The second part of the documentation is the API documentation, based on version 0.3 of the *proc* package. Here are the contents of the API documentation:

## 2.1 API documentation

This API documentation is based on the source code of version 0.3 of the *proc* package. The following generic modules are available:

- *proc*
- `proc.core`
- `proc.tree`

The following modules are based on the modules above:

- `proc.apache`
- `proc.cron`

These modules are meant to actually be used (for those who are interested in them) but they also function as examples of how to use the `proc.core` and `proc.tree` modules.

### 2.1.1 The `proc` module

The top level *proc* module contains only a version number.

`proc.__version__`
The version number of the *proc* package (a string).

### 2.1.2 The `proc.core` module

### 2.1.3 The `proc.tree` module

### 2.1.4 The `proc.apache` module

### 2.1.5 The `proc.cron` module

## p

## Symbols



## P