

---

# **atomicwrites Documentation**

*Release 1.4.0*

**Markus Unterwaditzer**

**Oct 03, 2020**



---

## Contents

---

<b>1</b>	<b>How it works</b>	<b>3</b>
1.1	fsync . . . . .	3
<b>2</b>	<b>Alternatives and Credit</b>	<b>5</b>
<b>3</b>	<b>License</b>	<b>7</b>
<b>4</b>	<b>API</b>	<b>9</b>
4.1	Errorhandling . . . . .	9
4.2	Low-level API . . . . .	9
<b>5</b>	<b>License</b>	<b>11</b>
	<b>Python Module Index</b>	<b>13</b>
	<b>Index</b>	<b>15</b>



Atomic file writes.

```
from atomicwrites import atomic_write

with atomic_write('foo.txt', overwrite=True) as f:
    f.write('Hello world.')
    # "foo.txt" doesn't exist yet.

# Now it does.
```

See [API documentation](#) for more low-level interfaces.

Features that distinguish it from other similar libraries (see *Alternatives and Credit*):

- Race-free assertion that the target file doesn't yet exist. This can be controlled with the `overwrite` parameter.
- Windows support, although not well-tested. The MSDN resources are not very explicit about which operations are atomic. I'm basing my assumptions off a [comment](#) by [Doug Crook](#), who appears to be a Microsoft employee:

Question: Is MoveFileEx atomic if the existing and new files are both on the same drive?

The simple answer is “usually, but in some cases it will silently fall-back to a non-atomic method, so don't count on it”.

The implementation of MoveFileEx looks something like this: [...]

The problem is if the rename fails, you might end up with a CopyFile, which is definitely not atomic.

If you really need atomic-or-nothing, you can try calling NtSetInformationFile, which is unsupported but is much more likely to be atomic.

- Simple high-level API that wraps a very flexible class-based API.
- Consistent error handling across platforms.



It uses a temporary file in the same directory as the given path. This ensures that the temporary file resides on the same filesystem.

The temporary file will then be atomically moved to the target location: On POSIX, it will use `rename` if files should be overwritten, otherwise a combination of `link` and `unlink`. On Windows, it uses `MoveFileEx` through `stdlib`'s `ctypes` with the appropriate flags.

Note that with `link` and `unlink`, there's a timewindow where the file might be available under two entries in the filesystem: The name of the temporary file, and the name of the target file.

Also note that the permissions of the target file may change this way. In some situations a `chmod` can be issued without any concurrency problems, but since that is not always the case, this library doesn't do it by itself.

## 1.1 `fsync`

On POSIX, `fsync` is invoked on the temporary file after it is written (to flush file content and metadata), and on the parent directory after the file is moved (to flush filename).

`fsync` does not take care of disks' internal buffers, but there don't seem to be any standard POSIX APIs for that. On OS X, `fcntl` is used with `F_FULLFSYNC` instead of `fsync` for that reason.

On Windows, `_commit` is used, but there are no guarantees about disk internal buffers.





---

### Alternatives and Credit

---

Atomicwrites is directly inspired by the following libraries (and shares a minimal amount of code):

- The Trac project's [utility functions](#), also used in [Werkzeug](#) and [mitsuhiko/python-atomicfile](#). The idea to use `ctypes` instead of `PyWin32` originated there.
- [abarnert/fatomic](#). Windows support (based on `PyWin32`) was originally taken from there.

Other alternatives to atomicwrites include:

- [sashka/atomicfile](#). Originally I considered using that, but at the time it was lacking a lot of features I needed (Windows support, `overwrite-parameter`, overriding behavior through subclassing).
- The [Boltons library collection](#) features a class for atomic file writes, which seems to have a very similar `overwrite` parameter. It is lacking Windows support though.



## CHAPTER 3

---

License

---

Licensed under the MIT, see LICENSE.



`atomicwrites.atomic_write` (*path*, *writer\_cls*=<class 'atomicwrites.AtomicWriter'>, *\*\*cls\_kwargs*)  
Simple atomic writes. This wraps *AtomicWriter*:

```
with atomic_write(path) as f:  
    f.write(...)
```

#### Parameters

- **path** – The target path to write to.
- **writer\_cls** – The writer class to use. This parameter is useful if you subclassed *AtomicWriter* to change some behavior and want to use that new subclass.

Additional keyword arguments are passed to the writer class. See *AtomicWriter*.

## 4.1 Errorhandling

All filesystem errors are subclasses of `OSError`.

- On UNIX systems, errors from the Python `stdlib` calls are thrown.
- On Windows systems, errors from Python's `ctypes` are thrown.

In either case, the `errno` attribute on the thrown exception maps to an errorcode in the `errno` module.

## 4.2 Low-level API

`atomicwrites.replace_atomic` (*src*, *dst*)

Move *src* to *dst*. If *dst* exists, it will be silently overwritten.

Both paths must reside on the same filesystem for the operation to be atomic.

`atomicwrites.move_atomic(src, dst)`

Move `src` to `dst`. There might a timewindow where both filesystem entries exist. If `dst` already exists, `FileExistsError` will be raised.

Both paths must reside on the same filesystem for the operation to be atomic.

**class** `atomicwrites.AtomicWriter(path, mode='w', overwrite=False, **open_kwargs)`

A helper class for performing atomic writes. Usage:

```
with AtomicWriter(path).open() as f:
    f.write(...)
```

#### Parameters

- **path** – The destination filepath. May or may not exist.
- **mode** – The filemode for the temporary file. This defaults to `wb` in Python 2 and `w` in Python 3.
- **overwrite** – If set to `false`, an error is raised if `path` exists. Errors are only raised after the file has been written to. Either way, the operation is atomic.

If you need further control over the exact behavior, you are encouraged to subclass.

**commit** (*f*)

Move the temporary file to the target location.

**get\_fileobject** (*suffix=""*, *prefix='tmp'*, *dir=None*, *\*\*kwargs*)

Return the temporary file to use.

**open** ()

Open the temporary file.

**rollback** (*f*)

Clean up all temporary resources.

**sync** (*f*)

responsible for clearing as many file caches as possible before commit

## CHAPTER 5

---

### License

---

Copyright (c) 2015-2016 Markus Unterwaditzer

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.





**a**

`atomicwrites`, 7



## A

`atomic_write()` (*in module atomicwrites*), 9

`AtomicWriter` (*class in atomicwrites*), 10

`atomicwrites` (*module*), 7

## C

`commit()` (*atomicwrites.AtomicWriter method*), 10

## G

`get_fileobject()` (*atomicwrites.AtomicWriter method*), 10

## M

`move_atomic()` (*in module atomicwrites*), 9

## O

`open()` (*atomicwrites.AtomicWriter method*), 10

## R

`replace_atomic()` (*in module atomicwrites*), 9

`rollback()` (*atomicwrites.AtomicWriter method*), 10

## S

`sync()` (*atomicwrites.AtomicWriter method*), 10