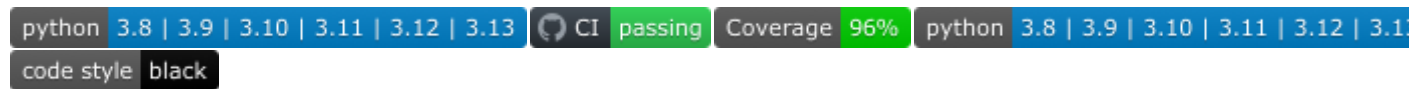

aiofiles: file support for asyncio



aiofiles is an Apache2 licensed library, written in Python, for handling local disk files in asyncio applications.

Ordinary local file IO is blocking, and cannot easily and portably be made asynchronous. This means doing file IO may interfere with asyncio applications, which shouldn't block the executing thread. aiofiles helps with this by introducing asynchronous versions of files that support delegating operations to a separate thread pool.

```
1 async with aiofiles.open('filename', mode='r') as f:
2     contents = await f.read()
3     print(contents)
4     'My file contents'
```

Asynchronous iteration is also supported.

```
1 async with aiofiles.open('filename') as f:
2     async for line in f:
3         ...
```

Asynchronous interface to tempfile module.

```
1 async with aiofiles.tempfile.TemporaryFile('wb') as f:
2     await f.write(b'Hello, World!')
```

Features

- a file API very similar to Python's standard, blocking API
- support for buffered and unbuffered binary files, and buffered text files
- support for `async/await` (PEP 492) constructs
- async interface to tempfile module

Installation

To install aiofiles, simply:

```
1 $ pip install aiofiles
```

Usage

Files are opened using the `aiofiles.open()` coroutine, which in addition to mirroring the builtin `open` accepts optional `loop` and `executor` arguments. If `loop` is absent, the default loop will be used, as per the set asyncio policy. If `executor` is not specified, the default event loop executor will be used.

In case of success, an asynchronous file object is returned with an API identical to an ordinary file, except the following methods are coroutines and delegate to an executor:

- `close`
- `flush`
- `isatty`
- `read`
- `readall`
- `read1`
- `readinto`
- `readline`
- `readlines`
- `seek`
- `seekable`
- `tell`
- `truncate`
- `writable`
- `write`
- `writelines`

In case of failure, one of the usual exceptions will be raised.

`aiofiles.stdin`, `aiofiles.stdout`, `aiofiles.stderr`, `aiofiles.stdin_bytes`, `aiofiles.stdout_bytes`, and `aiofiles.stderr_bytes` provide async access to `sys.stdin`, `sys.stdout`, `sys.stderr`, and their corresponding `.buffer` properties.

The `aiofiles.os` module contains executor-enabled coroutine versions of several useful `os` functions that deal with files:

- `stat`
- `statvfs`
- `sendfile`
- `rename`
- `renames`

-
- `replace`
 - `remove`
 - `unlink`
 - `mkdir`
 - `makedirs`
 - `rmdir`
 - `removedirs`
 - `link`
 - `symlink`
 - `readlink`
 - `listdir`
 - `scandir`
 - `access`
 - `path.exists`
 - `path.isfile`
 - `path.isdir`
 - `path.islink`
 - `path.ismount`
 - `path.getsize`
 - `path.getatime`
 - `path.getctime`
 - `path.samefile`
 - `path.sameopenfile`

Tempfile

aiofiles.tempfile implements the following interfaces:

- `TemporaryFile`
- `NamedTemporaryFile`
- `SpooledTemporaryFile`
- `TemporaryDirectory`

Results return wrapped with a context manager allowing use with `async with` and `async for`.

```
1  async with aiofiles.tempfile.NamedTemporaryFile('wb+') as f:
2      await f.write(b'Line1\n Line2')
3      await f.seek(0)
4      async for line in f:
5          print(line)
```

```
6
7 async with aiofiles.tempfile.TemporaryDirectory() as d:
8     filename = os.path.join(d, "file.ext")
```

Writing tests for aiofiles

Real file IO can be mocked by patching `aiofiles.threadpool.sync_open` as desired. The return type also needs to be registered with the `aiofiles.threadpool.wrap` dispatcher:

```
1 aiofiles.threadpool.wrap.register(mock.MagicMock)(
2     lambda *args, **kwargs: aiofiles.threadpool.AsyncBufferedIOBase(*
3         args, **kwargs)
4 )
5 async def test_stuff():
6     write_data = 'data'
7     read_file_chunks = [
8         b'file chunks 1',
9         b'file chunks 2',
10        b'file chunks 3',
11        b'',
12    ]
13    file_chunks_iter = iter(read_file_chunks)
14
15    mock_file_stream = mock.MagicMock(
16        read=lambda *args, **kwargs: next(file_chunks_iter)
17    )
18
19    with mock.patch('aiofiles.threadpool.sync_open', return_value=
20        mock_file_stream) as mock_open:
21        async with aiofiles.open('filename', 'w') as f:
22            await f.write(write_data)
23            assert f.read() == b'file chunks 1'
24
25        mock_file_stream.write.assert_called_once_with(write_data)
```

History

23.2.2 (UNRELEASED)

- Import `os.link` conditionally to fix importing on android. #175
- Remove spurious items from `aiofiles.os.__all__` when running on Windows.
- Switch to more modern async idioms: Remove `types.coroutine` and make `AiofilesContextManager` an awaitable instead a coroutine.

23.2.1 (2023-08-09)

- Import `os.statvfs` conditionally to fix importing on non-UNIX systems. #171 #172
- `aiofiles` is now also tested on Windows.

23.2.0 (2023-08-09)

- `aiofiles` is now tested on Python 3.12 too. #166 #168
- On Python 3.12, `aiofiles.tempfile.NamedTemporaryFile` now accepts a `delete_on_close` argument, just like the `stdlib` version.
- On Python 3.12, `aiofiles.tempfile.NamedTemporaryFile` no longer exposes a `delete` attribute, just like the `stdlib` version.
- Added `aiofiles.os.statvfs` and `aiofiles.os.path.ismount`. #162
- Use PDM instead of Poetry. #169

23.1.0 (2023-02-09)

- Added `aiofiles.os.access`. #146
- Removed `aiofiles.tempfile.temptypes.AsyncSpooledTemporaryFile.softspace`. #151
- Added `aiofiles.stdin`, `aiofiles.stdin_bytes`, and other `stdio` streams. #154
- Transition to `asyncio.get_running_loop` (vs `asyncio.get_event_loop`) internally.

22.1.0 (2022-09-04)

- Added `aiofiles.os.path.islink`. #126
- Added `aiofiles.os.readlink`. #125
- Added `aiofiles.os.symlink`. #124
- Added `aiofiles.os.unlink`. #123
- Added `aiofiles.os.link`. #121
- Added `aiofiles.os.rename`. #120
- Added `aiofiles.os.{listdir, scandir}`. #143
- Switched to CalVer.
- Dropped Python 3.6 support. If you require it, use version 0.8.0.
- `aiofiles` is now tested on Python 3.11.

0.8.0 (2021-11-27)

- `aiofiles` is now tested on Python 3.10.

-
- Added `aiofiles.os.replace`. #107
 - Added `aiofiles.os.{makedirs, removedirs}`.
 - Added `aiofiles.os.path.{exists, isfile, isdir, getsize, getatime, getctime, samefile, sameopenfile}`. #63
 - Added `suffix, prefix, dir` args to `aiofiles.tempfile.TemporaryDirectory`. #116

0.7.0 (2021-05-17)

- Added the `aiofiles.tempfile` module for async temporary files. #56
- Switched to Poetry and GitHub actions.
- Dropped 3.5 support.

0.6.0 (2020-10-27)

- `aiofiles` is now tested on ppc64le.
- Added `name` and `mode` properties to async file objects. #82
- Fixed a DeprecationWarning internally. #75
- Python 3.9 support and tests.

0.5.0 (2020-04-12)

- Python 3.8 support. Code base modernization (using `async/await` instead of `asyncio.coroutine/yield from`).
- Added `aiofiles.os.remove`, `aiofiles.os.rename`, `aiofiles.os.mkdir`, `aiofiles.os.rmdir`. #62

0.4.0 (2018-08-11)

- Python 3.7 support.
- Removed Python 3.3/3.4 support. If you use these versions, stick to aiofiles 0.3.x.

0.3.2 (2017-09-23)

- The LICENSE is now included in the sdist. #31

0.3.1 (2017-03-10)

- Introduced a changelog.

-
- `aiofiles.os.sendfile` will now work if the standard `os` module contains a `sendfile` function.

Contributing

Contributions are very welcome. Tests can be run with `tox`, please ensure the coverage at least stays the same before you submit a pull request.