
S3Fs Documentation

Release 2021.11.1

Continuum Analytics

Nov 26, 2021

Contents

1	Examples	3
2	Integration	5
3	Async	7
4	Limitations	9
5	Logging	11
6	Credentials	13
7	Self-hosted S3	15
8	Requester Pays Buckets	17
9	Serverside Encryption	19
10	Bucket Version Awareness	21
11	Contents	23
	11.1 Installation	23
	11.2 API	23
	11.3 Changelog	33
12	Indices and tables	37
	Index	39

S3Fs is a Pythonic file interface to S3. It builds on top of [botocore](#).

The top-level class `S3FileSystem` holds connection information and allows typical file-system style operations like `cp`, `mv`, `ls`, `du`, `glob`, etc., as well as put/get of local files to/from S3.

The connection can be anonymous - in which case only publicly-available, read-only buckets are accessible - or via credentials explicitly supplied or in configuration files.

Calling `open()` on a `S3FileSystem` (typically using a context manager) provides an `S3File` for read or write access to a particular key. The object emulates the standard `File` protocol (`read`, `write`, `tell`, `seek`), such that functions expecting a file can access S3. Only binary read and write modes are implemented, with blocked caching.

S3Fs uses and is based upon [fsspec](#).

CHAPTER 1

Examples

Simple locate and read a file:

```
>>> import s3fs
>>> fs = s3fs.S3FileSystem(anon=True)
>>> fs.ls('my-bucket')
['my-file.txt']
>>> with fs.open('my-bucket/my-file.txt', 'rb') as f:
...     print(f.read())
b'Hello, world'
```

(see also `walk` and `glob`)

Reading with delimited blocks:

```
>>> s3.read_block(path, offset=1000, length=10, delimiter=b'\n')
b'A whole line of text\n'
```

Writing with blocked caching:

```
>>> s3 = s3fs.S3FileSystem(anon=False) # uses default credentials
>>> with s3.open('mybucket/new-file', 'wb') as f:
...     f.write(2*2**20 * b'a')
...     f.write(2*2**20 * b'a') # data is flushed and file closed
>>> s3.du('mybucket/new-file')
{'mybucket/new-file': 4194304}
```

Because S3Fs faithfully copies the Python file interface it can be used smoothly with other projects that consume the file interface like `gzip` or `pandas`.

```
>>> with s3.open('mybucket/my-file.csv.gz', 'rb') as f:
...     g = gzip.GzipFile(fileobj=f) # Decompress data with gzip
...     df = pd.read_csv(g)         # Read CSV file with Pandas
```


CHAPTER 2

Integration

The libraries `intake`, `pandas` and `dask` accept URLs with the prefix `s3://`, and will use `s3fs` to complete the IO operation in question. The IO functions take an argument `storage_options`, which will be passed to `S3FileSystem`, for example:

```
df = pd.read_excel("s3://bucket/path/file.xls",  
                  storage_options={"anon": True})
```

This gives the chance to pass any credentials or other necessary arguments needed to `s3fs`.

`s3fs` is implemented using `aiobotocore`, and offers async functionality. A number of methods of `S3FileSystem` are `async`, for each of these, there is also a synchronous version with the same name and lack of a `_` prefix.

If you wish to call `s3fs` from async code, then you should pass `asynchronous=True`, `loop=` to the constructor (the latter is optional, if you wish to use both async and sync methods). You must also explicitly await the client creation before making any S3 call.

```
async def run_program():
    s3 = S3FileSystem(..., asynchronous=True)
    session = await s3.set_session()
    ... # perform work
    await session.close()

asyncio.run(run_program()) # or call from your async code
```

Concurrent async operations are also used internally for bulk operations such as `pipe/cat`, `get/put`, `cp/mv/rm`. The async calls are hidden behind a synchronisation layer, so are designed to be called from normal code. If you are *not* using async-style programming, you do not need to know about how this works, but you might find the implementation interesting.

CHAPTER 4

Limitations

This project is meant for convenience, rather than feature completeness. The following are known current omissions:

- file access is always binary (although `readline` and iterating by line are possible)
- no permissions/access-control (i.e., no `chmod/chown` methods)

CHAPTER 5

Logging

The logger named `s3fs` provides information about the operations of the file system. To quickly see all messages, you can set the environment variable `S3FS_LOGGING_LEVEL=DEBUG`. The presence of this environment variable will install a handler for the logger that prints messages to `stderr` and set the log level to the given value. More advance logging configuration is possible using Python's standard [logging framework](#).

The AWS key and secret may be provided explicitly when creating an `S3FileSystem`. A more secure way, not including the credentials directly in code, is to allow boto to establish the credentials automatically. Boto will try the following methods, in order:

- `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, and `AWS_SESSION_TOKEN` environment variables
- configuration files such as `~/.aws/credentials`
- for nodes on EC2, the IAM metadata provider

You can specify a profile using `s3fs.S3FileSystem(profile='PROFILE')`. Otherwise `sf3s` will use authentication via [boto environment variables](#).

In a distributed environment, it is not expected that raw credentials should be passed between machines. In the explicitly provided credentials case, the method `get_delegated_s3pars()` can be used to obtain temporary credentials. When not using explicit credentials, it should be expected that every machine also has the appropriate environment variables, config files or IAM roles available.

If none of the credential methods are available, only anonymous access will work, and `anon=True` must be passed to the constructor.

Furthermore, `S3FileSystem.current()` will return the most-recently created instance, so this method could be used in preference to the constructor in cases where the code must be agnostic of the credentials/config used.

CHAPTER 7

Self-hosted S3

To use `s3fs` against your self hosted S3-compatible storage, like [MinIO](#) or [Ceph Object Gateway](#), you can set your custom `endpoint_url` when creating the `s3fs` filesystem:

```
>>> s3 = s3fs.S3FileSystem(  
    anon=False,  
    client_kwargs={  
        'endpoint_url': 'https://...'  
    }  
)
```

Requester Pays Buckets

Some buckets, such as the [arXiv raw data](#), are configured so that the requester of the data pays any transfer fees. You must be authenticated to access these buckets and (because these charges may be unexpected) amazon requires an additional key on many of the API calls. To enable `RequesterPays` create your file system as

```
>>> s3 = s3fs.S3FileSystem(anon=False, requester_pays=True)
```

Serverside Encryption

For some buckets/files you may want to use some of s3's server side encryption features. `s3fs` supports these in a few ways

```
>>> s3 = s3fs.S3FileSystem(  
...     s3_additional_kwargs={'ServerSideEncryption': 'AES256'})
```

This will create an s3 filesystem instance that will append the `ServerSideEncryption` argument to all s3 calls (where applicable).

The same applies for `s3.open`. Most of the methods on the filesystem object will also accept and forward keyword arguments to the underlying calls. The most recently specified argument is applied last in the case where both `s3_additional_kwargs` and a method's `**kwargs` are used.

The `s3.utils.SSEParams` provides some convenient helpers for the serverside encryption parameters in particular. An instance can be passed instead of a regular python dictionary as the `s3_additional_kwargs` parameter.

Bucket Version Awareness

If your bucket has object versioning enabled then you can add version-aware support to `s3fs`. This ensures that if a file is opened at a particular point in time that version will be used for reading.

This mitigates the issue where more than one user is concurrently reading and writing to the same object.

```
>>> s3 = s3fs.S3FileSystem(version_aware=True)
# Open the file at the latest version
>>> fo = s3.open('versioned_bucket/object')
>>> versions = s3.object_version_info('versioned_bucket/object')
# Open the file at a particular version
>>> fo_old_version = s3.open('versioned_bucket/object', version_id='SOMEVERSIONID')
```

In order for this to function the user must have the necessary IAM permissions to perform a `GetObjectVersion`

11.1 Installation

11.1.1 Conda

The `s3fs` library and its dependencies can be installed from the [conda-forge](#) repository using `conda`:

```
$ conda install s3fs -c conda-forge
```

11.1.2 PyPI

You can install `s3fs` with `pip`:

```
pip install s3fs
```

11.1.3 Install from source

You can also download the `s3fs` library from Github and install normally:

```
git clone git@github.com:fsspec/s3fs
cd s3fs
python setup.py install
```

11.2 API

`S3FileSystem`([anon, key, secret, token, ...])

Access S3 as if it were a file system.

Continued on next page

Table 1 – continued from previous page

<code>S3FileSystem.cat(path[, recursive, on_error])</code>	Fetch (potentially multiple) paths' contents
<code>S3FileSystem.du(path[, total, maxdepth])</code>	Space used by files within a path
<code>S3FileSystem.exists(path)</code>	
<code>S3FileSystem.find(path[, maxdepth, ...])</code>	List all files below path.
<code>S3FileSystem.get(rpath, lpath[, recursive, ...])</code>	Copy file(s) to local.
<code>S3FileSystem.glob(path, **kwargs)</code>	Find files by glob-matching.
<code>S3FileSystem.info(path, **kwargs)</code>	Give details of entry at path
<code>S3FileSystem.ls(path[, detail])</code>	List objects at path.
<code>S3FileSystem.mkdir(path[, acl, create_parents])</code>	
<code>S3FileSystem.mv(path1, path2[, recursive, ...])</code>	Move file(s) from one location to another
<code>S3FileSystem.open(path[, mode, block_size, ...])</code>	Return a file-like object from the filesystem
<code>S3FileSystem.put(lpath, rpath[, recursive, ...])</code>	Copy file(s) from local.
<code>S3FileSystem.read_block(fn, offset, length)</code>	Read a block of bytes from
<code>S3FileSystem.rm(path[, recursive, maxdepth])</code>	Delete files.
<code>S3FileSystem.tail(path[, size])</code>	Get the last <code>size</code> bytes from file
<code>S3FileSystem.touch(path[, truncate, data])</code>	Create empty file or truncate
<code>S3File(s3, path[, mode, block_size, acl, ...])</code>	Open S3 key as a file.
<code>S3File.close()</code>	Close file
<code>S3File.flush([force])</code>	Write buffered data to backend store.
<code>S3File.info()</code>	File information about this path
<code>S3File.read([length])</code>	Return data from cache, or fetch pieces as necessary
<code>S3File.seek(loc[, whence])</code>	Set current file location
<code>S3File.tell()</code>	Current file location
<code>S3File.write(data)</code>	Write data to buffer.
<code>S3Map(root, s3[, check, create])</code>	Mirror previous class, not implemented in fsspec

```
class s3fs.core.S3FileSystem (anon=False, key=None, secret=None, token=None,
                             use_ssl=True, client_kwargs=None, requester_pays=False,
                             default_block_size=None, default_fill_cache=True, de-
                             fault_cache_type='bytes', version_aware=False, con-
                             fig_kwargs=None, s3_additional_kwargs=None, session=None,
                             username=None, password=None, cache_regions=False, asyn-
                             chronous=False, loop=None, **kwargs)
```

Access S3 as if it were a file system.

This exposes a filesystem-like API (`ls`, `cp`, `open`, etc.) on top of S3 storage.

Provide credentials either explicitly (`key=`, `secret=`) or depend on boto's credential methods. See botocore documentation for more information. If no credentials are available, use `anon=True`.

Parameters

anon [bool (False)] Whether to use anonymous connection (public buckets only). If False, uses the key/secret given, or boto's credential resolver (`client_kwargs`, environment, variables, config files, EC2 IAM server, in that order)

key [string (None)] If not anonymous, use this access key ID, if specified

secret [string (None)] If not anonymous, use this secret access key, if specified

- token** [string (None)] If not anonymous, use this security token, if specified
- use_ssl** [bool (True)] Whether to use SSL in connections to S3; may be faster without, but insecure. If `use_ssl` is also set in `client_kwargs`, the value set in `client_kwargs` will take priority.
- s3_additional_kwargs** [dict of parameters that are used when calling s3 api] methods. Typically used for things like “ServerSideEncryption”.
- client_kwargs** [dict of parameters for the botocore client]
- requester_pays** [bool (False)] If RequesterPays buckets are supported.
- default_block_size: int (None)** If given, the default block size value used for `open()`, if no specific value is given at all time. The built-in default is 5MB.
- default_fill_cache** [Bool (True)] Whether to use cache filling with `open` by default. Refer to `S3File.open`.
- default_cache_type** [string ('bytes')] If given, the default `cache_type` value used for `open()`. Set to “none” if no caching is desired. See `fsspec`’s documentation for other available `cache_type` values. Default `cache_type` is ‘bytes’.
- version_aware** [bool (False)] Whether to support bucket versioning. If enable this will require the user to have the necessary IAM permissions for dealing with versioned objects.
- cache_regions** [bool (False)] Whether to cache bucket regions or not. Whenever a new bucket is used, it will first find out which region it belongs and then use the client for that region.
- asynchronous** [bool (False)] Whether this instance is to be used from inside coroutines.
- config_kwargs** [dict of parameters passed to `botocore.client.Config`]
- kwargs** [other parameters for core session.]
- session** [aiobotocore AioSession object to be used for all connections.] This session will be used in place of creating a new session inside `S3FileSystem`. For example: `aiobotocore.session.AioSession(profile='test_user')`

The following parameters are passed on to `fsspec`:

skip_instance_cache: to control reuse of instances

use_listings_cache, listings_expiry_time, max_paths: to control reuse of directory listings

Examples

```
>>> s3 = S3FileSystem(anon=False) # doctest: +SKIP
>>> s3.ls('my-bucket/') # doctest: +SKIP
['my-file.txt']
```

```
>>> with s3.open('my-bucket/my-file.txt', mode='rb') as f: # doctest: +SKIP
...     print(f.read()) # doctest: +SKIP
b'Hello, world!'
```

Attributes

loop

s3

transaction A context within which files are committed together upon exit

Methods

<code>cat(path[, recursive, on_error])</code>	Fetch (potentially multiple) paths' contents
<code>cat_file(path[, start, end])</code>	Get the content of a file
<code>checksum(path[, refresh])</code>	Unique value for current version of file
<code>chmod(path, acl, **kwargs)</code>	Set Access Control on a bucket/key
<code>clear_instance_cache()</code>	Clear the cache of filesystem instances.
<code>connect([refresh, kwargs])</code>	Establish S3 connection object.
<code>copy(path1, path2[, recursive, on_error])</code>	Copy within two locations in the filesystem
<code>cp(path1, path2, **kwargs)</code>	Alias of <i>AbstractFileSystem.copy</i> .
<code>created(path)</code>	Return the created timestamp of a file as a date-time.datetime
<code>current()</code>	Return the most recently created FileSystem
<code>delete(path[, recursive, maxdepth])</code>	Alias of <i>AbstractFileSystem.rm</i> .
<code>disk_usage(path[, total, maxdepth])</code>	Alias of <i>AbstractFileSystem.du</i> .
<code>download(rpath, lpath[, recursive])</code>	Alias of <i>AbstractFileSystem.get</i> .
<code>du(path[, total, maxdepth])</code>	Space used by files within a path
<code>end_transaction()</code>	Finish write transaction, non-context version
<code>expand_path(path[, recursive, maxdepth])</code>	Turn one or more globs or directories into a list of all matching paths to files or directories.
<code>find(path[, maxdepth, withdirs, detail, prefix])</code>	List all files below path.
<code>from_json(blob)</code>	Recreate a filesystem instance from JSON representation
<code>get(rpath, lpath[, recursive, callback])</code>	Copy file(s) to local.
<code>get_delegated_s3pars([exp])</code>	Get temporary credentials from STS, appropriate for sending across a network.
<code>get_file(rpath, lpath[, callback])</code>	Copy single remote file to local
<code>get_mapper(root[, check, create])</code>	Create key/value store based on this file-system
<code>get_tags(path)</code>	Retrieve tag key/values for the given path
<code>getxattr(path, attr_name, **kwargs)</code>	Get an attribute from the metadata.
<code>glob(path, **kwargs)</code>	Find files by glob-matching.
<code>head(path[, size])</code>	Get the first <code>size</code> bytes from file
<code>info(path, **kwargs)</code>	Give details of entry at path
<code>invalidate_cache([path])</code>	Discard any cached directory information
<code>invalidate_region_cache()</code>	Invalidate the region cache (associated with buckets) if <code>cache_regions</code> is turned on.
<code>isfile(path)</code>	Is this entry file-like?
<code>lexists(path, **kwargs)</code>	If there is a file at the given path (including broken links)
<code>listdir(path[, detail])</code>	Alias of <i>AbstractFileSystem.ls</i> .
<code>ls(path[, detail])</code>	List objects at path.
<code>makedir(path[, create_parents])</code>	Alias of <i>AbstractFileSystem.mkdir</i> .
<code>merge(path, filelist, **kwargs)</code>	Create single S3 file from list of S3 files
<code>metadata(path[, refresh])</code>	Return metadata of path.
<code>makedirs(path[, exist_ok])</code>	Alias of <i>AbstractFileSystem.makedirs</i> .
<code>modified(path[, version_id, refresh])</code>	Return the last modified timestamp of file at <code>path</code> as a datetime
<code>move(path1, path2, **kwargs)</code>	Alias of <i>AbstractFileSystem.mv</i> .

Continued on next page

Table 4 – continued from previous page

<code>mv(path1, path2[, recursive, maxdepth])</code>	Move file(s) from one location to another
<code>open(path[, mode, block_size, ...])</code>	Return a file-like object from the filesystem
<code>pipe(path[, value])</code>	Put value into path
<code>pipe_file(path, value, **kwargs)</code>	Set the bytes of given file
<code>put(lpath, rpath[, recursive, callback])</code>	Copy file(s) from local.
<code>put_file(lpath, rpath[, callback])</code>	Copy single file to remote
<code>put_tags(path, tags[, mode])</code>	Set tags for given existing key
<code>read_block(fn, offset, length[, delimiter])</code>	Read a block of bytes from
<code>rename(path1, path2, **kwargs)</code>	Alias of <i>AbstractFileSystem.mv</i> .
<code>rm(path[, recursive, maxdepth])</code>	Delete files.
<code>rm_file(path)</code>	Delete a file
<code>set_session([refresh, kwargs])</code>	Establish S3 connection object.
<code>setxattr(path[, copy_kwargs])</code>	Set metadata.
<code>sign(path[, expiration])</code>	Create a signed URL representing the given path
<code>size(path)</code>	Size in bytes of file
<code>sizes(paths)</code>	Size in bytes of each file in a list of paths
<code>split_path(path)</code>	Normalise S3 path string into bucket and key.
<code>start_transaction()</code>	Begin write transaction for deferring files, non-context version
<code>stat(path, **kwargs)</code>	Alias of <i>AbstractFileSystem.info</i> .
<code>tail(path[, size])</code>	Get the last <code>size</code> bytes from file
<code>to_json()</code>	JSON representation of this filesystem instance
<code>touch(path[, truncate, data])</code>	Create empty file or truncate
<code>ukey(path)</code>	Hash of file properties, to tell if it has changed
<code>upload(lpath, rpath[, recursive])</code>	Alias of <i>AbstractFileSystem.put</i> .
<code>url(path[, expires, client_method])</code>	Generate presigned URL to access path by HTTP
<code>walk(path[, maxdepth])</code>	Return all files belows path

<code>call_s3</code>	
<code>cat_ranges</code>	
<code>close_session</code>	
<code>cp_file</code>	
<code>exists</code>	
<code>get_s3</code>	
<code>is_bucket_versioned</code>	
<code>isdir</code>	
<code>makedirs</code>	
<code>mkdir</code>	
<code>object_version_info</code>	
<code>rmdir</code>	

checksum (*path*, *refresh=False*)

Unique value for current version of file

If the checksum is the same from one moment to another, the contents are guaranteed to be the same. If the checksum changes, the contents *might* have changed.

Parameters

path [string/bytes] path of file to get checksum for

refresh [bool (=False)] if False, look in local cache for file details first

chmod (*path, acl, **kwargs*)

Set Access Control on a bucket/key

See <http://docs.aws.amazon.com/AmazonS3/latest/dev/acl-overview.html#canned-acl>

Parameters

path [string] the object to set

acl [string] the value of ACL to apply

connect (*refresh=False, kwargs={}*)

Establish S3 connection object. Returns `Session` to be closed later with `await .close()`

find (*path, maxdepth=None, withdirs=None, detail=False, prefix=""*)

List all files below path. Like posix `find` command without conditions Parameters `path` : str
`maxdepth`: int or None

If not None, the maximum number of levels to descend

withdirs: bool Whether to include directory paths in the output. This is True when used by `glob`, but users usually only want files.

prefix: str Only return files that match `^{path}/{prefix}` (if there is an exact match `filename == {path}/{prefix}`, it also will be included)

get_delegated_s3pars (*exp=3600*)

Get temporary credentials from STS, appropriate for sending across a network. Only relevant where the key/secret were explicitly provided.

Parameters

exp [int] Time in seconds that credentials are good for

Returns

dict of parameters

get_tags (*path*)

Retrieve tag key/values for the given path

Returns

{str: str}

getxattr (*path, attr_name, **kwargs*)

Get an attribute from the metadata.

Examples

```
>>> mys3fs.getxattr('mykey', 'attribute_1') # doctest: +SKIP
'value_1'
```

invalidate_cache (*path=None*)

Discard any cached directory information

Parameters

path: string or None If None, clear all listings cached else listings at or under given path.

invalidate_region_cache ()

Invalidate the region cache (associated with buckets) if `cache_regions` is turned on.

merge (*path*, *filelist*, ***kwargs*)

Create single S3 file from list of S3 files

Uses multi-part, no data is downloaded. The original files are not deleted.

Parameters

path [str] The final file to produce

filelist [list of str] The paths, in order, to assemble into the final file.

metadata (*path*, *refresh=False*, ***kwargs*)

Return metadata of path.

Parameters

path [string/bytes] filename to get metadata for

refresh [bool (=False)] (ignored)

modified (*path*, *version_id=None*, *refresh=False*)

Return the last modified timestamp of file at *path* as a datetime

put_tags (*path*, *tags*, *mode='o'*)

Set tags for given existing key

Tags are a str:str mapping that can be attached to any key, see <https://docs.aws.amazon.com/awsaccountbilling/latest/aboutv2/allocation-tag-restrictions.html>

This is similar to, but distinct from, key metadata, which is usually set at key creation time.

Parameters

path: str Existing key to attach tags to

tags: dict str, str Tags to apply.

mode: One of 'o' or 'm' 'o': Will over-write any existing tags. 'm': Will merge in new tags with existing tags. Incurs two remote calls.

set_session (*refresh=False*, *kwargs={}*)

Establish S3 connection object. Returns —— Session to be closed later with `await .close()`

setxattr (*path*, *copy_kwargs=None*, ***kw_args*)

Set metadata.

Attributes have to be of the form documented in the [‘Metadata Reference’](#).

Parameters

kw_args [key-value pairs like `field="value"`, where the values must be] strings. Does not alter existing fields, unless the field appears here - if the value is None, delete the field.

copy_kwargs [dict, optional] dictionary of additional params to use for the underlying `s3.copy_object`.

Examples

```
>>> mys3file.setxattr(attribute_1='value1', attribute_2='value2') # doctest:␣
↪+SKIP
# Example for use with copy_args
>>> mys3file.setxattr(copy_kwargs={'ContentType': 'application/pdf'},
...     attribute_1='value1') # doctest: +SKIP
```

<http://docs.aws.amazon.com/AmazonS3/latest/dev/UsingMetadata.html#object-metadata>

sign (*path*, *expiration=100*, ***kwargs*)

Create a signed URL representing the given path

Some implementations allow temporary URLs to be generated, as a way of delegating credentials.

Parameters

path [str] The path on the filesystem

expiration [int] Number of seconds to enable the URL for (if supported)

Returns

URL [str] The signed URL

Raises

NotImplementedError [if method is not implemented for a filesystem]

split_path (*path*) → Tuple[str, str, Optional[str]]

Normalise S3 path string into bucket and key.

Parameters

path [string] Input path, like `s3://mybucket/path/to/file`

Examples

```
>>> split_path("s3://mybucket/path/to/file")
['mybucket', 'path/to/file', None]
```

```
>>> split_path("s3://mybucket/path/to/versioned_file?versionId=some_version_id
↪")
['mybucket', 'path/to/versioned_file', 'some_version_id']
```

touch (*path*, *truncate=True*, *data=None*, ***kwargs*)

Create empty file or truncate

url (*path*, *expires=3600*, *client_method='get_object'*, ***kwargs*)

Generate presigned URL to access path by HTTP

Parameters

path [string] the key path we are interested in

expires [int] the number of seconds this signature will be good for.

class `s3fs.core.S3File` (*s3*, *path*, *mode='rb'*, *block_size=5242880*, *acl=""*, *version_id=None*,
fill_cache=True, *s3_additional_kwargs=None*, *autocommit=True*,
cache_type='bytes', *requester_pays=False*, *cache_options=None*)

Open S3 key as a file. Data is only loaded and cached on demand.

Parameters

s3 [S3FileSystem] botocore connection

path [string] S3 bucket/key to access

mode [str] One of 'rb', 'wb', 'ab'. These have the same meaning as they do for the built-in `open` function.

block_size [int] read-ahead size for finding delimiters

fill_cache [bool] If seeking to new a part of the file beyond the current buffer, with this True, the buffer will be filled between the sections to best support random access. When reading only a few specific chunks out of a file, performance may be better if False.

acl: str Canned ACL to apply

version_id [str] Optional version to read the file at. If not specified this will default to the current version of the object. This is only used for reading.

requester_pays [bool (False)] If RequesterPays buckets are supported.

See also:

S3FileSystem.open used to create `S3File` objects

Examples

```
>>> s3 = S3FileSystem() # doctest: +SKIP
>>> with s3.open('my-bucket/my-file.txt', mode='rb') as f: # doctest: +SKIP
...     ... # doctest: +SKIP
```

Attributes

closed

details

full_name

Methods

<code>close()</code>	Close file
<code>commit()</code>	Move from temp to final destination
<code>discard()</code>	Throw away temporary file
<code>fileno()</code>	Returns underlying file descriptor if one exists.
<code>flush([force])</code>	Write buffered data to backend store.
<code>getxattr(xattr_name, **kwargs)</code>	Get an attribute from the metadata.
<code>info()</code>	File information about this path
<code>isatty()</code>	Return whether this is an 'interactive' stream.
<code>metadata([refresh])</code>	Return metadata of file.
<code>read([length])</code>	Return data from cache, or fetch pieces as necessary
<code>readable()</code>	Whether opened for reading
<code>readinto(b)</code>	mirrors builtin file's readinto method
<code>readline()</code>	Read until first occurrence of newline character
<code>readlines()</code>	Return all data, split by the newline character
<code>readuntil([char, blocks])</code>	Return data between current position and first occurrence of char
<code>seek(loc[, whence])</code>	Set current file location
<code>seekable()</code>	Whether is seekable (only in read mode)
<code>setxattr([copy_kwargs])</code>	Set metadata.
<code>tell()</code>	Current file location
<code>truncate</code>	Truncate file to size bytes.
<code>url(**kwargs)</code>	HTTP URL to read this file (if it already exists)

Continued on next page

Table 5 – continued from previous page

<code>writable()</code>	Whether opened for writing
<code>write(data)</code>	Write data to buffer.
<code>writelines(lines, /)</code>	Write a list of lines to stream.

<code>readinto1</code>	
------------------------	--

commit()
Move from temp to final destination

discard()
Throw away temporary file

getxattr (*xattr_name*, ***kwargs*)
Get an attribute from the metadata. See `getxattr()`.

Examples

```
>>> mys3file.getxattr('attribute_1') # doctest: +SKIP
'value_1'
```

metadata (*refresh=False*, ***kwargs*)
Return metadata of file. See `metadata()`.
Metadata is cached unless *refresh=True*.

setxattr (*copy_kwargs=None*, ***kwargs*)
Set metadata. See `setxattr()`.

Examples

```
>>> mys3file.setxattr(attribute_1='value1', attribute_2='value2') # doctest: ↵
↵+SKIP
```

url (***kwargs*)
HTTP URL to read this file (if it already exists)

`s3fs.mapping.S3Map` (*root*, *s3*, *check=False*, *create=False*)
Mirror previous class, not implemented in `fsspec`

class `s3fs.utils.ParamKwargsHelper` (*s3*)
Utility class to help extract the subset of keys that an `s3` method is actually using

Parameters

s3 [boto S3FileSystem]

Methods

<code>filter_dict</code>	
--------------------------	--

class `s3fs.utils.SSEParams` (*server_side_encryption=None*, *sse_customer_algorithm=None*,
sse_customer_key=None, *sse_kms_key_id=None*)

Methods

to_kwargs	
-----------	--

11.3 Changelog

11.3.1 2021.11.1

- deal with missing ETag (#557)
- ClientPayloadError to retryable (#556)
- pin aiobotocore (#555)

11.3.2 2021.11.0

- move to fsspec org
- doc tweaks (#546, 540)
- redondant argument in `_rm_versioned_bucket_contents` (#439)
- allow `client_method` in `url/sign` (POST, etc) (#536)
- revert `list_v2->head` for info (#545)

11.3.3 2021.10.1

- allow other methods than GET to `url/sign` (#536)

11.3.4 2021.10.0

No changes (just released to keep pin with fsspec)

11.3.5 2021.09.0

- check for bucket also with `get_bucket_location` (#533)
- update versioneer (#531)

11.3.6 2021.08.1

- retry on `IncompleteRead` (#525)
- fix `isdir` for missing bucket (#522)
- raise for `glob("**")` (#5167)

11.3.7 2021.08.0

- fix for aiobotocore update (#510)

11.3.8 2021.07.0

- make bucket in put(recursive) (#496)
- non-truthy prefixes (#497)
- implement rm_file (#499)

11.3.9 2021.06.1

- bucket region caching (#495)

11.3.10 2021.06.0

- support “prefix” in directory listings (#486)
- support negative index in cat_file (#487, 488)
- don't require ETag in file details (#480)

11.3.11 2021.05.0

- optimize `info`, “exists” (and related) calls for non-version aware mode
- copy with entries without ETag (#480)
- find not to corrupts parent listing (#476)
- short listing to determine directory (#472, 471)

11.3.12 Version 2021.04.0

- switch to calver and fsspec pin
- py36 (#462)
- async fixes (#456, 452)

11.3.13 Version 0.6.0

- update for fsspec 0.9.0 (#448)
- better errors (#443)
- cp to preserve ETAG (#441)
- CI (#435, #427, #395)
- 5GB PUT (#425)
- partial cat (#389)

- [direct find \(#360\)](#)

11.3.14 Version 0.5.0

- Asynchronous filesystem based on `aiobotocore`

11.3.15 Version 0.4.0

- New instances no longer need reconnect ([PR #244](#)) by [Martin Durant](#)
- Always use multipart uploads when not autocommitting ([PR #243](#)) by [Marius van Niekerk](#)
- Create `CONTRIBUTING.md` ([PR #248](#)) by [Jacob Tomlinson](#)
- Use autofunction for `S3Map sphinx autosummary` ([PR #251](#)) by [James Bourbeau](#)
- Miscellaneous doc updates ([PR #252](#)) by [James Bourbeau](#)
- Support for Python 3.8 ([PR #264](#)) by [Tom Augspurger](#)
- Improved performance for `isdir` ([PR #259](#)) by [Nate Yoder](#)
- Increased the minimum required version of `fsspec` to 0.6.0

CHAPTER 12

Indices and tables

- `genindex`
- `modindex`
- `search`

C

checksum() (*s3fs.core.S3FileSystem method*), 27
chmod() (*s3fs.core.S3FileSystem method*), 27
commit() (*s3fs.core.S3File method*), 32
connect() (*s3fs.core.S3FileSystem method*), 28

D

discard() (*s3fs.core.S3File method*), 32

F

find() (*s3fs.core.S3FileSystem method*), 28

G

get_delegated_s3pars() (*s3fs.core.S3FileSystem method*), 28
get_tags() (*s3fs.core.S3FileSystem method*), 28
getxattr() (*s3fs.core.S3File method*), 32
getxattr() (*s3fs.core.S3FileSystem method*), 28

I

invalidate_cache() (*s3fs.core.S3FileSystem method*), 28
invalidate_region_cache() (*s3fs.core.S3FileSystem method*), 28

M

merge() (*s3fs.core.S3FileSystem method*), 28
metadata() (*s3fs.core.S3File method*), 32
metadata() (*s3fs.core.S3FileSystem method*), 29
modified() (*s3fs.core.S3FileSystem method*), 29

P

ParamKwargsHelper (*class in s3fs.utils*), 32
put_tags() (*s3fs.core.S3FileSystem method*), 29

S

S3File (*class in s3fs.core*), 30
S3FileSystem (*class in s3fs.core*), 24

S3Map() (*in module s3fs.mapping*), 32
set_session() (*s3fs.core.S3FileSystem method*), 29
setxattr() (*s3fs.core.S3File method*), 32
setxattr() (*s3fs.core.S3FileSystem method*), 29
sign() (*s3fs.core.S3FileSystem method*), 30
split_path() (*s3fs.core.S3FileSystem method*), 30
SSEParams (*class in s3fs.utils*), 32

T

touch() (*s3fs.core.S3FileSystem method*), 30

U

url() (*s3fs.core.S3File method*), 32
url() (*s3fs.core.S3FileSystem method*), 30