S3Fs Documentation

Release 0.5.1+42.g1f8e716

Continuum Analytics

Nov 17, 2020

Contents

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 `S3File3System`, 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.

CHAPTER 3

Async

s3fs is implemented using aiobotocore, and offers async functionality. A number of methods of S3FileSystem are async, for 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.

```
loop = ...  # however you create your loop

async def run_program(loop):
    s3 = S3FileSystem(..., asynchronous=True, loop=loop)
    await s3._connect()
    ...  # perform work

asyncio.run(run_program(loop))  # 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.

CHAPTER 6

Credentials

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

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://...'
      }
   )
```

CHAPTER 8

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 maybe 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)
```

CHAPTER 9

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.

CHAPTER 10

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.S3FileSytem(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

CHAPTER 11

Contents

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:dask/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

`S3FileSystem.cat`(path[, recursive, on_error])	Fetch (potentially multiple) paths' contents
`S3FileSystem.du`(path[, total, maxdepth])	Space used by files within a path
`S3FileSystem.exists`(path)	
`S3FileSystem.get`(rpath, lpath[, recursive])	Copy file(s) to local.
`S3FileSystem.glob`(path, **kwargs)	Find files by glob-matching.
`S3FileSystem.info`(path[, version_id, refresh])	Give details of entry at path
`S3FileSystem.ls`(path[, detail, refresh])	List single "directory" with or without details
`S3FileSystem.mkdir`(path[, acl, create_parents])	
`S3FileSystem.mv`(path1, path2[, recursive, . . .])	Move file(s) from one location to another
`S3FileSystem.open`(path[, mode, block_size, . . .])	Return a file-like object from the filesystem
`S3FileSystem.put`(lpath, rpath[, recursive])	Copy file(s) from local.
`S3FileSystem.read_block`(fn, offset, length)	Read a block of bytes from
`S3FileSystem.rm`(path[, recursive])	Delete files.
`S3FileSystem.tail`(path[, size])	Get the last `size` bytes from file
`S3FileSystem.touch`(path[, truncate, data])	Create empty file or truncate

`S3File`(s3, path[, mode, block_size, acl, . . .])	Open S3 key as a file.
`S3File.close`()	Close file
`S3File.flush`([force])	Write buffered data to backend store.
`S3File.info`()	File information about this path
`S3File.read`([length])	Return data from cache, or fetch pieces as necessary
`S3File.seek`(loc[, whence])	Set current file location
`S3File.tell`()	Current file location
`S3File.write`(data)	Write data to buffer.

`S3Map`(root, s3[, check, create])	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, default_cache_type='bytes', version_aware=False, config_kwargs=None, s3_additional_kwargs=None, session=None, username=None, password=None, asynchronous=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.

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 inplace of creating a new session inside S3FileSystem. For example: aiobotocore.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

s3

`transaction` A context within which files are committed together upon exit

Methods

`cat`(path[, recursive, on_error])	Fetch (potentially multiple) paths' contents
`cat_file`(path)	Get the content of a file
`checksum`(path[, refresh])	Unique value for current version of file
`chmod`(path, acl, **kwargs)	Set Access Control on a bucket/key
`clear_instance_cache`()	Clear the cache of filesystem instances.
`connect`([kwargs])	Establish S3 connection object.
`copy`(path1, path2[, recursive])	Copy within two locations in the filesystem
`cp`(path1, path2, **kwargs)	Alias of FilesystemSpec.copy.
`created`(path)	Return the created timestamp of a file as a datetime.datetime
`current`()	Return the most recently created FileSystem
`delete`(path[, recursive, maxdepth])	Alias of FilesystemSpec.rm.
`disk_usage`(path[, total, maxdepth])	Alias of FilesystemSpec.du.
`download`(rpath, lpath[, recursive])	Alias of FilesystemSpec.get.
`du`(path[, total, maxdepth])	Space used by files within a path
`end_transaction`()	Finish write transaction, non-context version
`expand_path`(path[, recursive, maxdepth])	Turn one or more globs or directories into a list of all matching files
`from_json`(blob)	Recreate a filesystem instance from JSON representation
`get`(rpath, lpath[, recursive])	Copy file(s) to local.
`get_delegated_s3pars`([exp])	Get temporary credentials from STS, appropriate for sending across a network.
`get_file`(rpath, lpath, **kwargs)	Copy single remote file to local
`get_mapper`(root[, check, create])	Create key/value store based on this file-system
`get_tags`(path)	Retrieve tag key/values for the given path
`getxattr`(path, attr_name, **kwargs)	Get an attribute from the metadata.
`glob`(path, **kwargs)	Find files by glob-matching.
`head`(path[, size])	Get the first `size` bytes from file
`info`(path[, version_id, refresh])	Give details of entry at path
`invalidate_cache`([path])	Discard any cached directory information
`isdir`(path)	Is this entry directory-like?
`isfile`(path)	Is this entry file-like?
`listdir`(path[, detail])	Alias of FilesystemSpec.ls.
`ls`(path[, detail, refresh])	List single "directory" with or without details
`makedir`(path[, create_parents])	Alias of FilesystemSpec.mkdir.
`makedirs`(path[, exist_ok])	Recursively make directories
`merge`(path, filelist, **kwargs)	Create single S3 file from list of S3 files
`metadata`(path[, refresh])	Return metadata of path.
`mkdirs`(path[, exist_ok])	Alias of FilesystemSpec.makedirs.
`modified`(path[, version_id, refresh])	Return the last modified timestamp of file at *path* as a datetime
`move`(path1, path2, **kwargs)	Alias of FilesystemSpec.mv.
`mv`(path1, path2[, recursive, maxdepth])	Move file(s) from one location to another
`open`(path[, mode, block_size, cache_options])	Return a file-like object from the filesystem
`pipe`(path[, value])	Put value into path
`pipe_file`(path, value, **kwargs)	Set the bytes of given file
`put`(lpath, rpath[, recursive])	Copy file(s) from local.
`put_file`(lpath, rpath, **kwargs)	Copy single file to remote
`put_tags`(path, tags[, mode])	Set tags for given existing key
`read_block`(fn, offset, length[, delimiter])	Read a block of bytes from

Continued on next page

Table 4 – continued from previous page

rename(path1, path2, **kwargs)	Alias of FilesystemSpec.mv.
rm(path[, recursive])	Delete files.
rm_file(path)	Delete a file
setxattr(path[, copy_kwargs])	Set metadata.
sign(path[, expiration])	Create a signed URL representing the given path
size(path)	Size in bytes of file
split_path(path)	Normalise S3 path string into bucket and key.
start_transaction()	Begin write transaction for deferring files, non-context version
stat(path, **kwargs)	Alias of FilesystemSpec.info.
tail(path[, size])	Get the last `size` bytes from file
to_json()	JSON representation of this filesystem instance
touch(path[, truncate, data])	Create empty file or truncate
ukey(path)	Hash of file properties, to tell if it has changed
upload(lpath, rpath[, recursive])	Alias of FilesystemSpec.put.
url(path[, expires])	Generate presigned URL to access path by HTTP
walk(path[, maxdepth])	Return all files belows path

call_s3	
cp_file	
exists	
find	
is_bucket_versioned	
mkdir	
object_version_info	
rmdir	

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(*kwargs={}*)

Establish S3 connection object.

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'
```

info (*path*, *version_id=None*, *refresh=False*)
Give details of entry at path

Returns a single dictionary, with exactly the same information as `ls` would with `detail=True`.

The default implementation should calls ls and could be overridden by a shortcut. kwargs are passed on to `` `ls(). ``

Some file systems might not be able to measure the file's size, in which case, the returned dict will include `'size':  None`.

Returns

dict with keys: name (full path in the FS), size (in bytes), type (file,

directory, or something else) and other FS-specific keys.

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.

isdir (*path*)
Is this entry directory-like?

ls (*path*, *detail=False*, *refresh=False*, ***kwargs*)
List single "directory" with or without details

Parameters

path [string/bytes] location at which to list files

detail [bool (=True)] if True, each list item is a dict of file properties; otherwise, returns list of filenames

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

kwargs [dict] additional arguments passed on

makedirs (*path*, *exist_ok=False*)
Recursively make directories

Creates directory at path and any intervening required directories. Raises exception if, for instance, the path already exists but is a file.

Parameters

path: str leaf directory name

exist_ok: bool (False) If True, will error if the target already exists

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.

Metadata is cached unless *refresh=True*.

Parameters

path [string/bytes] filename to get metadata for

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

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.

rm (*path*, *recursive=False*, ***kwargs*)
Delete files.

Parameters

path: str or list of str File(s) to delete.

recursive: bool If file(s) are directories, recursively delete contents and then also remove the directory

maxdepth: int or None Depth to pass to walk for finding files to delete, if recursive. If None, there will be no limit and infinite recursion may be possible.

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 fileystem]

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*, ***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.

walk (*path*, *maxdepth=None*, ***kwargs*)
Return all files belows path

List all files, recursing into subdirectories; output is iterator-style, like `os.walk()`. For a simple list of files, `find()` is available.

Note that the "files" outputted will include anything that is not a directory, such as links.

Parameters

path: str Root to recurse into

maxdepth: int Maximum recursion depth. None means limitless, but not recommended on link-based file-systems.

kwargs: passed to ''ls''

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*)
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

Methods

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

readinto1	

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

filter_dict	

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 Version 0.5.0

- Asynchronous filesystem based on `aiobotocore`

11.3.2 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

Index