ocifs

Release 1.1.4

Allen Hosler

Nov 16, 2022

CONTENTS

CHAPTER

ONE

INSTALLATION

Install using pip:

```
python3 -m pip install ocifs
```

CHAPTER

TWO

OVERVIEW

ocifs is a Pythonic filesystem interface to Oracle Cloud Infrastructure (OCI) Object Storage. It builds on top of oci.

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

The connection gets validated via a configuration file (usually ~/.oci/config) or Resource Principal.

Calling `open()` on a `OCIFileSystem` (typically using a context manager) provides an `OCIFile` for read or write access to a particular key. The object emulates the standard file object (`read`, `write`, `tell`, `seek`), such that functions can interact with OCI Object Storage. Only binary read and write modes are implemented, with blocked caching.

ocifs uses and is based upon fsspec.

CHAPTER

THREE

EXAMPLES

Simple locate and read a file:

```
>>> import ocifs
>>> fs = ocifs.OCIFileSystem()
>>> fs.ls('my-bucket@my-namespace')
['my-bucket@my-namespace/my-file.txt']
>>> with fs.open('my-bucket@my-namespace/my-file.txt', 'rb') as f:
...     print(f.read())
b'Hello, world'
```

(see also `walk` and `glob` in the Unix Operations section)

Reading with delimited blocks:

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

Learn more about read_block.

Writing with blocked caching:

```
>>> fs = ocifs.OCIFileSystem(config=".oci/config")  # uses default credentials
>>> with fs.open('mybucket@mynamespace/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
>>> fs.du('mybucket@mynamespace/new-file')
{'mybucket@mynamespace/new-file': 4194304}
```

Learn more about `fsspec`'s caching system.

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

```
>>> with fs.open('mybucket@mynamespace/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

FOUR

INTEGRATION

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

```
df = pd.read_excel("oci://bucket@namespace/path/file.xls",
                   storage_options={"config": "~/.oci/config"})
```

Use the `storage_options` parameter to pass any additional arguments to `ocifs`, for example security credentials.

CHAPTER

FIVE

AUTHENTICATION AND CREDENTIALS

An OCI config file (API Keys) may be provided as a filepath or a config instance returned from `oci.config.from_file` when creating an `OCIFileSystem` using the `config` argument. For example, two valid inputs to the `config` argument are: `oci.config.from_file("~/.oci/config")` and `~/.oci/config`. Specify the profile using the `profile` argument: `OCIFileSystem(config="~/.oci/config", profile='PROFILE')`.

Alternatively a signer may be used to create an `OCIFileSystem` using the `signer` argument. A Resource Principal Signer can be created using `oci.auth.signers.get_resource_principals_signer()`. An Instance Principal Signer can be created using `oci.auth.signers.InstancePrincipalsSecurityTokenSigner()` If neither config nor signer is provided, `OCIFileSystem` will attempt to create a Resource Principal, then an Instance Principal. However, passing a signer directly is always preferred.

Learn more about using signers with *ocifs* in the Getting Connected tab, or learn more about Resource Principal here.

CHAPTER

SIX

LOGGING

The logger named `ocifs` provides information about the operations of the file system. To see all messages, set the logging level to `INFO`:

```
import logging
logging.getLogger("ocifs").setLevel(logging.INFO)
```

Info mode will print messages to stderr. More advanced logging configuration is possible using Python's standard logging framework.

CHAPTER

SEVEN

LIMITATIONS

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

- file access is always binary (although reading files line by line as strings is possible)
- no permissions/access-control (no `chmod`/`chown` methods)
- `ocifs` only reads the latest version of a file. Versioned support is on the roadmap for a future release.

CHAPTER

EIGHT

A NOTE ON CACHING

To save time and bandwidth, ocifs caches the results from listing buckets and objects. To refresh the cache, set the `refresh` argument to `True` in the `ls()` method. However, `info`, and as a result `exists`, ignores the cache and makes a call to Object Storage directly. If the underlying bucket is modified after a call to list, the change will be reflected in calls to info, but not list (unless `refresh=True`).

CHAPTER

NINE

CONTENTS

9.1 Getting Started

The Oracle Cloud Infrastructure (OCI) Object Storage filesystem (ocifs) is an fsspec implementation for use with Object Storage.

9.1.1 Quickstart with Pandas

Begin by importing `ocifs` and `pandas`. When importing `ocifs`, you are registering the `oci` protocol with `pandas`:

[1]:
```
import ocifs
import pandas as pd
```

Now that the `oci` protocol is registered with `pandas`, you can read and write from and to Object Storage as easily as you can locally. For example, you could read an Excel file, `path/file.xls`, from your bucket in a namespace easily using:

[ ]:
```
df = pd.read_excel("oci://bucket@namespace/path/file.xls",
                   storage_options={"config": "~/.oci/config"})
```

[ ]:
```
df.to_parquet("oci://bucket@namespace/path/file.parquet",
              storage_options={"config": "~/.oci/config"})
```

You could also use Dask:

[ ]:
```
from dask import dataframe as dd

ddf = dd.read_csv("oci://bucket@namespace/path/file*.csv",
                  storage_options={"config": "~/.oci/config"})
```

The `storage_options` parameter contains a dictionary of arguments that are passed to the underlying `OCIFileSystem` method. The following `docstring` lists the valid arguments to storage options:

9.1.2 Quickstart to UNIX Operations

You can interact with the filesytem directly using most UNIX commands like `ls`, `cp`, `exists`, `mkdir`, `rm`, `walk`, `find`, and so on.

Instantiate a filesystem from your configuration, see Getting Connected. Every filesystem instance operates within the home region of the configuration. The `cp` command is the only command that has cross-region support. You must create a unique filesystem instance for each region.

[3]:
```
fs = ocifs.OCIFileSystem(config="~/.oci/config", profile="DEFAULT", default_block_
↪size=5*2**20)
```

[ ]:
```
fs.ls("oci://bucket@namespace/path")
# []
```

[ ]:
```
fs.touch("oci://bucket@namespace/path/file.txt")
```

[ ]:
```
fs.exists("oci://bucket@namespace/path/file.txt")
# True
```

[ ]:
```
fs.cat("oci://bucket@namespace/path/file.txt")
# ""
```

[ ]:
```
fs.rm("oci://bucket@namespace", recursive=True)
```

[ ]:
```
fs.exists("oci://bucket@namespace/path/file.txt")
# False
```

Following are examples of how you can use the `OCIFileSystem` and `OCIFile` objects.

9.2 Getting Connected

9.2.1 Configuring Your Connection

There are two IAM policy validation options in `ocifs`. The first option is using an identity policy, which is a configuration file. This is the most commonly used policy, and is the default in this documentation.

[1]:
```
from ocifs import OCIFileSystem

fs = OCIFileSystem(config="~/.oci/config", profile="DEFAULT")
```

Using Pandas and Dask are just as easy:

[ ]:
```
import pandas as pd

pd.read_csv("oci://bucket@namespace/path/file.csv",
            storage_options={"config": "~/.oci/config", "profile": "DEFAULT"})
```

[ ]:
```
from dask import dataframe as dd
```

(continues on next page)

(continued from previous page)

```
dd.read_csv("oci://bucket@namespace/path/file.csv",
            storage_options={"config": "~/.oci/config", "profile": "DEFAULT"})
```

9.2.2 Using Environment Variables

Users can provide default authentication configuration details as enviornment variables. The following environment variables are inspected:

- `OCIFS_IAM_TYPE`: which can be any of: ["api_key", "resource_principal", "instance_principal", "unknown_signer"]
- `OCIFS_CONFIG_LOCATION`: (optional) will be referenced in the case of "api_key", but defaults to the default config location provided by the oci sdk
- `OCIFS_CONFIG_PROFILE`: (optional) will be referenced in the case of "api_key", but defaults to the default config profile provided by the oci sdk

[ ]:
```
import os

os.environ['OCIFS_IAM_TYPE'] = "api_key"
fs = OCIFileSystem()
```

Note, the order of precedence for authentication is: signer arg, config arg, environment variables, then ocifs will attempt to set up Resource Principal, as exemplified below.

9.2.3 Resource Principal

The second policy option is using a resource principal. This policy only works if you're operating within a valid OCI resource, such as an OCI Data Science notebook session. With this option, your resource token path is set by global OCI signing variables.

[ ]:
```
fs = OCIFileSystem()
```

And with pandas or dask:

[ ]:
```
pd.read_csv("oci://bucket@namespace/path/file.csv")
```

[ ]:
```
dd.read_csv("oci://bucket@namespace/path/file.csv")
```

9.2.4 Connecting Using a Signer

Any signer can be passed in using the signer argument.

[ ]:
```
resource_principal_signer = oci.auth.signers.get_resource_principals_signer()
fs_rp = OCIFileSystem(signer=resource_principal_signer)
```

[ ]:
```
instance_principal_signer = oci.auth.signers.InstancePrincipalsSecurityTokenSigner()
fs_ip = OCIFileSystem(signer=instance_principal_signer)
```

And with pandas or dask:

```
[ ]: pd.read_csv("oci://bucket@namespace/path/file.csv",
                 storage_options={"signer": resource_principal_signer})

     dd.read_csv("oci://bucket@namespace/path/file.csv",
                 storage_options={"signer": instance_principal_signer})
```

9.2.5 Connecting to a Different Region

Each filesystem instance has a home region and won't operate outside of that region. The home region defaults to the region of the IAM policy. With a configuration policy, it is `region`. With a resource principal, the region is derived from the `OCI_REGION_METADATA` environment variable.

The `OCIFileSystem` delegates this region set up to the Object Storage Client **init** method in the OCI Python SDK. The `region` argument accepts any valid region identifier and constructs the corresponding service endpoint for the Object Storage Client. The following cell is an example of connecting to the sydney region.

```
[ ]: fs_sydney = OCIFileSystem(config="~/.oci/config", region="ap-sydney-1")
```

Using Pandas or Dask:

```
[2]: df.to_csv("oci://bucket@namespace/path/file.csv",
               storage_options = {"config": "~/.oci/config", "region": "ap-sydney-1"})
```

```
[ ]: ddf.to_csv("oci://bucket@namespace/path/file.csv",
                storage_options = {"config": "~/.oci/config", "region": "ap-sydney-1"})
```

Note: You must ensure that you have valid cross-region permissions before attempting to instantiate a file system in a non-home region, see the list of valid OCI Region Identifiers.

9.3 Unix Operations

Important: The ocifs SDK isn't a one-to-one adaptor of OCI Object Storage and UNIX filesystem operations. It's a set of convenient wrappings to assist Pandas in natively reading from Object Storage. It supports many of the common UNIX functions, and many of the Object Storage API though not all.

Following are examples of some of the most popular filesystem and file methods. First, you must instantiate your region-specific filesystem instance:

```
[1]: from ocifs import OCIFileSystem

     fs = OCIFileSystem(config="~/.oci/config")
```

9.3.1 Filesystem Operations

9.3.1.1 list

List the files in a bucket or subdirectory using `ls`:

```
[ ]: fs.ls("bucket@namespace/")
     # ['bucket@namespace/file.txt',
     #  'bucket@namespace/data.csv',
     #  'bucket@namespace/folder1/',
     #  'bucket@namespace/folder2/']
```

`list` has the following args: 1) `compartment_id`: a specific compartment from which to list. 2)`detail`: If true, return a list of dictionaries with various details about each object. 3)`refresh`: If true, ignore the cache and pull fresh.

```
[ ]: fs.ls("bucket@namespace/", detail=True)
     # [{'name': 'bucket@namespace/file.txt',
     #   'etag': 'abcdefghijklmnop',
     #   'type': 'file',
     #   'timeCreated': <timestamp when artifact created>,
     #  ... },
     #  ...
     # ]
```

9.3.1.2 touch

The UNIX `touch` command creates empty files in Object Storage. The `data` parameter accepts a bytestream and writes it to the new file.

```
[ ]: fs.touch("bucket@namespace/newfile", data=b"Hello World!")
```

```
[ ]: fs.cat("bucket@namespace/newfile")
     # "Hello World!"
```

9.3.1.3 copy

The `copy` method is a popular UNIX method, and it has a special role in ocifs as the only method capable of cross-tenancy calls. Your IAM Policy must permit you to read and write cross-region to use the `copy` method cross-region. Note: Another benefit of `copy` is that it can move large data between locations in Object Storage without needing to store anything locally.

```
[ ]: fs.copy("bucket@namespace/newfile", "bucket@namespace/newfile-sydney",
             destination_region="ap-sydney-1")
```

9.3.1.4 rm

The `rm` method is another essential UNIX filesystem method. It accepts one additional argument (beyond the path), `recursive`. When `recursive=True`, it is equivalent to an `rm -rf` command. It deletes all files underneath the prefix.

```
[ ]: fs.exists("oci://bucket@namespace/folder/file")
     # True
```

```
[ ]: fs.rm("oci://bucket@namespace/folder", recursive=True)
```

```
[ ]: fs.exists("oci://bucket@namespace/folder/file")
     # False
```

9.3.1.5 glob

Fsspec implementations, including ocifs, support UNIX glob patterns, see Globbing.

```
[ ]: fs.glob("oci://bucket@namespace/folder/*.csv")
     # ["bucket@namespace/folder/part1.csv", "bucket@namespace/folder/part2.csv"]
```

Dask has special support for reading from and writing to a set of files using glob expressions (Pandas doesn't support glob), see Dask's Glob support.

```
[ ]: from dask import dataframe as dd

     ddf = dd.read_csv("oci://bucket@namespace/folder/*.csv")
     ddf.to_csv("oci://bucket@namespace/folder_copy/*.csv")
```

9.3.1.6 walk

Use the UNIX `walk` method for iterating through the subdirectories of a given path. This is a valuable method for determining every file within a bucket or folder.

```
[ ]: fs.walk("oci://bucket@namespace/folder")
     # ["bucket@namespace/folder/part1.csv", "bucket@namespace/folder/part2.csv",
     #  "bucket@namespace/folder/subdir/file1.csv", "bucket@namespace/folder/subdir/file2.csv
     ↪"]
```

9.3.1.7 open

This method opens a file and returns an `OCIFile` object. There are examples of what you can do with an `OCIFile` in the next section.

9.3.2 File Operations

After calling open, you get an `OCIFile` object, which is subclassed from fsspec's `AbstractBufferedFile`. This file object can do almost everything a UNIX file can. Following are a few examples, see a full list of methods.

9.3.2.1 read

The `read` method works exactly as you would expect with a UNIX file:

[ ]:
```
import fsspec

with fsspec.open("oci://bucket@namespace/folder/file", 'rb') as f:
    buffer = f.read()
```

[ ]:
```
from ocifs import OCIFileSystem

fs = OCIFileSystem()
with fs.open("oci://bucket@namespace/folder/file", 'rb') as f:
    buffer = f.read()
```

[ ]:
```
file = fs.open("oci://bucket@namespace/folder/file")
buffer = file.read()
file.close()
```

9.3.2.2 seek

The `seek` method is also valuable in navigating files:

[ ]:
```
fs.touch("bucket@namespace/newfile", data=b"Hello World!")
with fs.open("bucket@namespace/newfile") as f:
    f.seek(3)
    print(f.read(1))
    f.seek(0)
    print(f.read(1))

# l
# H
```

9.3.2.3 write

You can use the `write` operation:

[ ]:
```
with fsspec.open("oci://bucket@namespace/newfile", 'wb') as f:
    buffer = f.write(b"new text")

with fsspec.open("oci://bucket@namespace/newfile", 'rb') as f:
    assert f.read() == b"new text"
```

9.3.3 Learn More

There are many more operations that you can use with `ocifs`, see the AbstractBufferedFile spec and the AbstractFileSystem spec.

```
[ ]:
```

9.4 API Reference

9.4.1 ocifs classes

9.4.1.1 OCIFileSystem

class ocifs.core.**OCIFileSystem**(**args*, ***kwargs*)

Bases: `AbstractFileSystem`

Access oci as if it were a file system.

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

Parameters

- **config** (*Union[dict, str, None]*) – Config for the connection to OCI. If a dict, it should be returned from oci.config.from_file If a str, it should be the location of the config file If None, user should have a Resource Principal configured environment If Resource Principal is not available, Instance Principal
- **signer** (*oci.auth.signers*) – A signer from the OCI sdk. More info: oci.auth.signers
- **profile** (*str*) – The profile to use from the config (If the config is passed in)
- **iam_type** (*str (None)*) – The IAM Auth principal type to use. Values can be one of [“api_key”, “resource_principal”, “instance_principal”]
- **region** (*str (None)*) – The Region Identifier that the client should connnect to. Regions can be found here: https://docs.oracle.com/en-us/iaas/Content/General/Concepts/regions.htm
- **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.
- **config_kwargs** (*dict*) – dict of parameters passed to the OCI Client upon connection more info here: oci.object_storage.ObjectStorageClient.__init__
- **oci_additional_kwargs** (*dict*) – dict of parameters that are used when calling oci api methods. Typically used for things like “retry_strategy”.
- **kwargs** (*dict*) – dict of other parameters for oci session This includes default parameters for tenancy, namespace, and region Any other parameters are passed along to AbstractFileSystem’s init method.

async_impl = False

blocksize = 4194304

bulk_delete(*pathlist*, ***kwargs*)

Remove multiple keys with one call :param pathlist: The keys to remove, must all be in the same bucket. :type pathlist: listof strings

cachable = True

cat(*path*, *recursive=False*, *on_error='raise'*, ***kwargs*)

Fetch (potentially multiple) paths' contents

Parameters

- **recursive** (*bool*) – If True, assume the path(s) are directories, and get all the contained files
- **on_error** (*"raise", "omit", "return"*) – If raise, an underlying exception will be raised (converted to KeyError if the type is in self.missing_exceptions); if omit, keys with exception will simply not be included in the output; if "return", all keys are included in the output, but the value will be bytes or an exception instance.
- **kwargs** (*passed to cat_file*) –

Returns

- **dict of {path** (*contents} if there are multiple paths*)
- *or the path has been otherwise expanded*

cat_file(*path*, *start=None*, *end=None*, ***kwargs*)

Get the content of a file

Parameters

- **path** (*URL of file on this filesystems*) –
- **end** (*start,*) – Bytes limits of the read. If negative, backwards from end, like usual python slices. Either can be None for start or end of file, respectively
- **kwargs** (passed to open().) –

cat_ranges(*paths*, *starts*, *ends*, *max_gap=None*, *on_error='return'*, ***kwargs*)

checksum(*path*, ***kwargs*)

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

classmethod clear_instance_cache()

Clear the cache of filesystem instances.

Notes

Unless overridden by setting the cachable class attribute to False, the filesystem class stores a reference to newly created instances. This prevents Python's normal rules around garbage collection from working, since the instances refcount will not drop to zero until clear_instance_cache is called.

connect(*refresh=True*)

Establish oci connection object.

Parameters

refresh (*bool*) – Whether to create new session/client, even if a previous one with the same parameters already exists. If False (default), an existing one will be used if possible.

connect_timeout = 5

copy(*path1*, *path2*, *destination_region=None*, ***kwargs*)

Copy file between locations on OCI

Parameters

- **path1** (*str*) – URI of source data
- **path2** (*str*) – URI of destination data
- **destination_region** (*str*) – the region you want path2 to be written in (defaults region of your config)

copy_basic(*path1*, *path2*, *destination_region=None*, ***kwargs*)

Copy file between locations on OCI

Not allowed where the origin is >50GB

Parameters

- **path1** (*str*) – URI of source data
- **path2** (*str*) – URI of destination data
- **destination_region** (*str*) – the region you want path2 to be written in (defaults region of your config)

cp(*path1*, *path2*, ***kwargs*)

Alias of *AbstractFileSystem.copy*.

cp_file(*path1*, *path2*, ***kwargs*)

created(*path*)

Return the created timestamp of a file as a datetime.datetime

classmethod current()

Return the most recently instantiated FileSystem

If no instance has been created, then create one with defaults

default_block_size = 5242880

delete(*path*, *recursive=False*, *maxdepth=None*)

Alias of *AbstractFileSystem.rm*.

disk_usage(*path*, *total=True*, *maxdepth=None*, ***kwargs*)

Alias of *AbstractFileSystem.du*.

download(*rpath*, *lpath*, *recursive=False*, ***kwargs*)

Alias of *AbstractFileSystem.get*.

du(*path*, *total=True*, *maxdepth=None*, ***kwargs*)

Space used by files within a path

Parameters

- **path** (*str*) –
- **total** (*bool*) – whether to sum all the file sizes

- **maxdepth** (*int or None*) – maximum number of directory levels to descend, None for unlimited.
- **kwargs** (passed to `ls`) –

Returns

- **Dict of {fn** (*size} if total=False, or int otherwise, where numbers*)
- *refer to bytes used.*

end_transaction()

Finish write transaction, non-context version

exists(*path*, ***kwargs*)

Is there a file at the given path

expand_path(*path*, *recursive=False*, *maxdepth=None*)

Turn one or more globs or directories into a list of all matching paths to files or directories.

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

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.
- **ls.** (*kwargs are passed to*) –

static from_json(*blob*)

Recreate a filesystem instance from JSON representation

See `.to_json()` for the expected structure of the input

Parameters
blob (*str*) –

Returns

Return type
file system instance, not necessarily of this particular class.

get(*rpath*, *lpath*, *recursive=False*, *callback=<fsspec.callbacks.NoOpCallback object>*, ***kwargs*)

Copy file(s) to local.

Copies a specific file or tree of files (if recursive=True). If lpath ends with a "/", it will be assumed to be a directory, and target files will go within. Can submit a list of paths, which may be glob-patterns and will be expanded.

Calls get_file for each source.

get_file(*rpath*, *lpath*, *callback=<fsspec.callbacks.NoOpCallback object>*, *outfile=None*, ***kwargs*)

Copy single remote file to local

get_mapper(*root=''*, *check=False*, *create=False*, *missing_exceptions=None*)

Create key/value store based on this file-system

Makes a MutableMapping interface to the FS at the given root path. See `fsspec.mapping.FSMap` for further details.

glob(*path*, ***kwargs*)

Find files by glob-matching.

If the path ends with '/' and does not contain "*", it is essentially the same as `ls(path)`, returning only files.

We support `"**"`, `"?"` and `"[..]"`. We do not support ^ for pattern negation.

Search path names that contain embedded characters special to this implementation of glob may not produce expected results; e.g., 'foo/bar/*starredfilename*'.

kwargs are passed to `ls`.

head(*path*, *size=1024*)

Get the first `size` bytes from file

info(*path*, ***kwargs*)

Get metadata about a file from a head or list call.

Parameters

- **path** (*str*) – URI of the directory/file
- **kwargs** (*dict*) – additional args for OCI

invalidate_cache(*path=None*)

Deletes the filesystem cache.

Parameters

path (*str (optional)*) – The directory from which to clear. If not specificed, deleted entire cache.

isdir(*path*)

Is this entry directory-like?

isfile(*path*)

Is this entry file-like?

lexists(*path*, ***kwargs*)

If there is a file at the given path (including broken links)

listdir(*path*, *detail=True*, ***kwargs*)

Alias of *AbstractFileSystem.ls*.

ls(*path: str*, *detail: bool = False*, *refresh: bool = False*, ***kwargs*)

List single "directory" with or without details

Parameters

- **path** (*string/bytes*) – location at which to list files
- **detail** (*bool (=False)*) – 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

makedir(*path*, *create_parents=True*, ***kwargs*)

Alias of *AbstractFileSystem.mkdir*.

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 False, will error if the target already exists

metadata(*path*, ***kwargs*)

Get metadata about a file from a head or list call.

Parameters

- **path** (*str*) – URI of the directory/file
- **kwargs** (*dict*) – additional args for OCI

mirror_sync_methods = False

mkdir(*path: str*, *create_parents: bool = True*, *compartment_id: Optional[str] = None*, ***kwargs*)

Make a new bucket or folder

Parameters

- **path** (*str*) – URI of the directory
- **create_parents** (*bool (=True)*) – If Ture, will create all nested dirs
- **compartment_id** (*str*) – If the compartment to create the bucket is different from the compartment of your auth mechanism.
- **kwargs** (*dict*) – additional args for OCI

mkdirs(*path*, *exist_ok=False*)

Alias of *AbstractFileSystem.makedirs*.

modified(*path*)

Return the modified timestamp of a file as a datetime.datetime

move(*path1*, *path2*, ***kwargs*)

Alias of *AbstractFileSystem.mv*.

mv(*path1*, *path2*, *recursive=False*, *maxdepth=None*, ***kwargs*)

Move file(s) from one location to another

open(*path: str*, *mode: str = 'rb'*, *block_size: Optional[int] = None*, *cache_options: Optional[dict] = None*, *compression: Optional[str] = None*, *cache_type: Optional[str] = None*, *autocommit: bool = True*, ***kwargs*)

Open a file for reading or writing

Parameters

- **path** (*string*) – Path of file on oci
- **mode** (*string*) – One of ‘r’, ‘w’, ‘rb’, or ‘wb’. These have the same meaning as they do for the built-in *open* function.

- **block_size** (*int*) – Size of data-node blocks if reading
- **cache_options** (*dict, optional*) – Extra arguments to pass through to the cache.
- **compression** (*string or None*) – If given, open file using compression codec. Can either be a compression name (a key in `fsspec.compression.compr`) or "infer" to guess the compression from the filename suffix.
- **cache_type** (*str*) – Caching policy in read mode Valid types are: {"readahead", "none", "mmap", "bytes"}, default "readahead"
- **autocommit** (*bool*) – If True, the OCIFile will automatically commit the multipart upload when done
- **encoding** (*str*) – The encoding to use if opening the file in text mode. The platform's default text encoding is used if not given.
- **kwargs** (*dict-like*) – Additional parameters used for oci methods. Typically used for ServerSideEncryption.

pipe(*path*, *value=None*, ***kwargs*)

Put value into path

(counterpart to `cat`)

Parameters

- **path** (*string or dict(str, bytes)*) – If a string, a single remote location to put `value` bytes; if a dict, a mapping of {path: bytesvalue}.
- **value** (*bytes, optional*) – If using a single path, these are the bytes to put there. Ignored if `path` is a dict

pipe_file(*path*, *value*, ***kwargs*)

Set the bytes of given file

protocol = ['oci']

put(*lpath*, *rpath*, *recursive=False*, *callback=<fsspec.callbacks.NoOpCallback object>*, ***kwargs*)

Copy file(s) from local.

Copies a specific file or tree of files (if recursive=True). If rpath ends with a "/", it will be assumed to be a directory, and target files will go within.

Calls put_file for each source.

put_file(*lpath*, *rpath*, *callback=<fsspec.callbacks.NoOpCallback object>*, ***kwargs*)

Copy single file to remote

read_block(*fn*, *offset*, *length*, *delimiter=None*)

Read a block of bytes from

Starting at `offset` of the file, read `length` bytes. If `delimiter` is set then we ensure that the read starts and stops at delimiter boundaries that follow the locations `offset` and `offset + length`. If `offset` is zero then we start at zero. The bytestring returned WILL include the end delimiter string.

If offset+length is beyond the eof, reads to eof.

Parameters

- **fn** (*string*) – Path to filename
- **offset** (*int*) – Byte offset to start read
- **length** (*int*) – Number of bytes to read

- **delimiter** (*bytes (optional)*) – Ensure reading starts and stops at delimiter bytestring

Examples

```
>>> fs.read_block('data/file.csv', 0, 13)
b'Alice, 100\nBo'
>>> fs.read_block('data/file.csv', 0, 13, delimiter=b'\n')
b'Alice, 100\nBob, 200\n'
```

Use `length=None` to read to the end of the file. >>> fs.read_block('data/file.csv', 0, None, delimiter=b'n') # doctest: +SKIP b'Alice, 100nBob, 200nCharlie, 300'

See also:

`fsspec.utils.read_block()`

read_bytes(*path*, *start=None*, *end=None*, ***kwargs*)

Alias of *AbstractFileSystem.cat_file*.

read_text(*path*, *encoding=None*, *errors=None*, *newline=None*, ***kwargs*)

Get the contents of the file as a string.

Parameters

- **path** (*str*) – URL of file on this filesystems
- **newline** (*encoding, errors,*) –

read_timeout = 15

rename(*path1*, *path2*, ***kwargs*)

Alias of *AbstractFileSystem.mv*.

rm(*path*, *recursive=False*, ***kwargs*)

Remove keys and/or bucket.

Parameters

- **path** (*str*) – The location to remove.
- **recursive** (*bool (True)*) – Whether to remove also all entries below, i.e., which are returned by *walk()*.

rm_file(*path*)

Delete a file

rmdir(*path*, ***kwargs*)

Remove a directory, if empty

root_marker = ''

sep = '/'

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 – The signed URL

Return type
str

:raises NotImplementedError : if method is not implemented for a filesystem:

size(*path*)

Size in bytes of file

sizes(*paths*)

Size in bytes of each file in a list of paths

split_path(*path*, ***kwargs*)

Normalise OCI path string into bucket and key. :param path: Input path, like *oci://mybucket@mynamespace/path/to/file* :type path: string

Examples

```
>>> split_path("oci://mybucket@mynamespace/path/to/file")
['mybucket', 'mynamespace', 'path/to/file']
```

start_transaction()

Begin write transaction for deferring files, non-context version

stat(*path*, ***kwargs*)

Alias of *AbstractFileSystem.info*.

tail(*path*, *size=1024*)

Get the last `size` bytes from file

to_json()

JSON representation of this filesystem instance

Returns
str – protocol (text name of this class's protocol, first one in case of multiple), args (positional args, usually empty), and all other kwargs as their own keys.

Return type
JSON structure with keys cls (the python location of this class),

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

Create empty file or truncate

Parameters

- **path** (*string/bytes*) – location at which to list files
- **truncate** (*bool (=True)*) – if True, delete the existing file, replace with empty file
- **data** (*bool*) – if provided, writes this content to the file
- **kwargs** (*dict*) – additional arguments passed on

property transaction

A context within which files are committed together upon exit

Requires the file class to implement *.commit()* and *.discard()* for the normal and exception cases.

ukey(*path*)

Hash of file properties, to tell if it has changed

unstrip_protocol(*name*)

Format FS-specific path to generic, including protocol

upload(*lpath*, *rpath*, *recursive=False*, ***kwargs*)

Alias of *AbstractFileSystem.put*.

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.
- **topdown** (*bool (True)*) – Whether to walk the directory tree from the top downwards or from the bottom upwards.
- **kwargs** (passed to `ls`) –

write_bytes(*path*, *value*, ***kwargs*)

Alias of *AbstractFileSystem.pipe_file*.

write_text(*path*, *value*, *encoding=None*, *errors=None*, *newline=None*, ***kwargs*)

Write the text to the given file.

An existing file will be overwritten.

Parameters

- **path** (*str*) – URL of file on this filesystems
- **value** (*str*) – Text to write.
- **newline** (*encoding, errors,*) –

9.4.1.2 OCIFile

class ocifs.core.**OCIFile**(*fs: OCIFileSystem*, *path: str*, *mode: str = 'rb'*, *block_size: int = 5242880*, *autocommit: bool = True*, *cache_type: str = 'bytes'*, *cache_options: Optional[dict] = None*, *additional_kwargs: Optional[dict] = None*, *size: Optional[int] = None*, ***kwargs*)

Bases: `AbstractBufferedFile`

Open OCI URI as a file.

This imitates the native python file object. Data is only loaded and cached on demand.

Parameters

- **fs** (OCIFileSystem) – instance of FileSystem
- **path** (*str*) – location in file-system
- **mode** (*str*) – Normal file modes. Currently only 'w', 'wb', 'r' or 'rb'.
- **block_size** (*int*) – Buffer size for reading or writing, 'default' for class default
- **autocommit** (*bool*) – Whether to write to final destination; may only impact what happens when file is being closed.
- **cache_type** (*{"readahead", "none", "mmap", "bytes"}, default "readahead"*) – Caching policy in read mode. See the definitions in core.
- **cache_options** (*dict*) – Additional options passed to the constructor for the cache specified by *cache_type*.
- **size** (*int*) – If given and in read mode, suppressed having to look up the file size
- **kwargs** – Gets stored as self.kwargs

DEFAULT_BLOCK_SIZE = 5242880

MAXIMUM_BLOCK_SIZE = 5368709120

MINIMUM_BLOCK_SIZE = 5242880

close()

Close file

Finalizes writes, discards cache

property closed

commit(***kwargs*)

Move from temp to final destination

property details

discard()

Throw away temporary file

fileno()

Returns underlying file descriptor if one exists.

OSError is raised if the IO object does not use a file descriptor.

flush(*force=False*)

Write buffered data to backend store.

Writes the current buffer, if it is larger than the block-size, or if the file is being closed.

Parameters

force (*bool*) – When closing, write the last block even if it is smaller than blocks are allowed to be. Disallows further writing to this file.

property full_name

info()

File information about this path

isatty()

Return whether this is an 'interactive' stream.

Return False if it can't be determined.

read(*length=-1*)

Return data from cache, or fetch pieces as necessary

Parameters

length (*int (-1)*) – Number of bytes to read; if <0, all remaining bytes.

readable()

Whether opened for reading

readinto(*b*)

mirrors builtin file's readinto method

https://docs.python.org/3/library/io.html#io.RawIOBase.readinto

readinto1(*b*)

readline()

Read until first occurrence of newline character

Note that, because of character encoding, this is not necessarily a true line ending.

readlines()

Return all data, split by the newline character

readuntil(*char=b'\n'*, *blocks=None*)

Return data between current position and first occurrence of char

char is included in the output, except if the end of the tile is encountered first.

Parameters

- **char** (*bytes*) – Thing to find
- **blocks** (*None or int*) – How much to read in each go. Defaults to file blocksize - which may mean a new read on every call.

retries = 5

seek(*loc*, *whence=0*)

Set current file location

Parameters

- **loc** (*int*) – byte location
- **whence** (*{0, 1, 2}*) – from start of file, current location or end of file, resp.

seekable()

Whether is seekable (only in read mode)

tell()

Current file location

truncate()

Truncate file to size bytes.

File pointer is left unchanged. Size defaults to the current IO position as reported by tell(). Returns the new size.

`writable`()

Whether opened for writing

`write`(*data*)

Write data to buffer.

Buffer only sent on flush() or if buffer is greater than or equal to blocksize.

Parameters
`data` (*bytes*) – Set of bytes to be written.

`writelines`(*lines*, */*)

Write a list of lines to stream.

Line separators are not added, so it is usual for each of the lines provided to have a line separator at the end.

9.5 FAQS

9.5.1 Frequently Asked Questions

Is ocifs asynchronous?

No. Ocifs currently inherits the AbstractFile and AbstractFileSystem classes, rather than the Async versions of these. This development may happen in the future if there's enough interest.

Does ocifs use multipart uploads?

Yes. Ocifs uses multipart uploads to upload files to Object Storage when the file is larger than 5 GB.

Can I bring my own signer?

Yes. Whether you want to use Resource Principal, Instance Principal, or some other type of OCI signer, simply pass that signer to the OCIFileSystem init method.

Can I use ocifs with a different region?

Yes, pass this region into the OCIFileSystem init method, and ensure your auth credentials work cross-region.

Can I use ocifs with a different tenancy?

Yes, pass this tenancy into the OCIFileSystem init method, and ensure your auth credentials work cross-region.

Can I set auth once and forget?

Yes, you can use environment variables: *OCIFS_IAM_TYPE*, *OCIFS_CONFIG_LOCATION*, *OCIFS_CONFIG_PROFILE*. Read more in the Getting Connected section.

Can I refresh the cache from pandas?

Yes, you can. Pass *storage_options = {"refresh": "True"}* into any pandas read method.

9.6 Telemetry

ocifs utilizes user-agent to record telemetry on usage: ocifs version used, operating system, python version, etc. This allows the ocifs team to prioritize future development.

CHAPTER

TEN

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

O

P

R

S

T

U

W