
azfs

Jul 17, 2021

Contents

1	Installation	3
2	Limitation	5
2.1	authorization	5
2.2	types of storage	5
3	API Usage	7
3.1	create the client	7
3.2	download data	7
3.3	upload data	8
3.4	enumerating(ls, glob) or checking if file exists	9
3.5	remove, copy files, etc.	9
3.6	manipulate QueueStorage	10
3.7	manipulate TableStorage	10
4	For Users	11
4.1	API Reference	11
4.2	Release History	22
5	For Developers	27
5.1	API Reference (for Developer)	27
5.2	GitHub	28
5.3	References	28
6	Indices and tables	29
	Index	31

code status

package status

AzFS is to provide convenient Python read/write functions for Azure Storage Account.

AzFS can

- list files in blob (also filtered with wildcard *),
- check if file exists,
- read csv as `pd.DataFrame`, and json as dict from blob,
- write `pd.DataFrame` as csv, and dict as json to blob,
- and raise lots of exceptions ! (Thank you for your cooperation)

CHAPTER 1

Installation

AzFS can be installed from pip.

```
pip install azfs
```


2.1 authorization

Supported authentication types are

- Azure Active Directory (AAD) token credential
- connection_string, like *DefaultEndpointsProtocol=https;AccountName=xxxx;AccountKey=xxxx;EndpointSuffix=core.windows.net*

2.2 types of storage

The table below shows the compatibility read/access of AzFS.

account kind	Blob	Data Lake	Queue	File	Table
StorageV2	O	O	O	X	O
StorageV1	O	O	O	X	O
BlobStorage	O				

3.1 create the client

To manipulate files in Azure Blob Storage, firstly you need to create `AzFileClient`.

Credential is not required if `AzFileClient()` is created on AAD (Azure Active Directory).

```
import azfs
from azure.identity import DefaultAzureCredential

# credential is not required if your environment is on AAD
azc = azfs.AzFileClient()
# credential is required if your environment is not on AAD
credential = "[your storage account credential]"
# or
credential = DefaultAzureCredential()
azc = azfs.AzFileClient(credential=credential)

# connection_string is also supported
connection_string = "DefaultEndpointsProtocol=https;AccountName=xxxx;AccountKey=xxxx;
↪EndpointSuffix=core.windows.net"
azc = azfs.AzFileClient(connection_string=connection_string)
```

3.2 download data

AzFS provides function to download csv or json data from Azure Blob Storage. API reference is [get/download](#).

```
import azfs
import pandas as pd

azc = azfs.AzFileClient()
# if your data in BlobStorage
```

(continues on next page)

(continued from previous page)

```

csv_path = "https://{storage_account}.blob.core.windows.net/{container}/***.csv"
json_path = "https://{storage_account}.blob.core.windows.net/{container}/***.json"
data_path = "https://{storage_account}.blob.core.windows.net/{container}/***.another_
↳format"
# if your data in DataLakeStorage
csv_path = "https://{storage_account}.dfs.core.windows.net/{container}/***.csv"
json_path = "https://{storage_account}.dfs.core.windows.net/{container}/***.json"
data_path = "https://{storage_account}.dfs.core.windows.net/{container}/***.another_
↳format"

# read csv as pd.DataFrame
df = azc.read_csv(csv_path, index_col=0)
# or
with azc:
    df = pd.read_csv_az(csv_path, header=None)

# read json
data = azc.read_json(json_path)

# also get data directory
data = azc.get(data_path)
# or, (`download` is an alias for `get`)
data = azc.download(data_path)

```

3.3 upload data

AzFS also provides functions to upload csv or json data to Azure Blob Storage. API reference is [put/upload](#).

```

import azfs
import pandas as pd

azc = azfs.AzFileClient()
# if your data in BlobStorage
csv_path = "https://{storage_account}.blob.core.windows.net/{container}/***.csv"
json_path = "https://{storage_account}.blob.core.windows.net/{container}/***.json"
data_path = "https://{storage_account}.blob.core.windows.net/{container}/***.another_
↳format"

df = pd.DataFrame()
data = {"example": "data"}

# write csv
azc.write_csv(path=csv_path, df=df)
# or
with azc:
    df.to_csv_az(path=csv_path, index=False)

# read json as dict
azc.write_json(path=json_path, data=data, indent=4)

# also put data directory
import json
azc.put(path=json_path, data=json.dumps(data, indent=4))

```

(continues on next page)

(continued from previous page)

```
# or, (`upload` is an alias for `put`)
azc.upload(path=json_path, data=json.dumps(data, indent=4))
```

3.4 enumerating(ls, glob) or checking if file exists

ls() lists all files in specified folder, and glob() lists pattern-matched files in all folder. API reference is [enumerating](#).

```
import azfs

azc = azfs.AzFileClient()

# get file_name list of blob
file_name_list = azc.ls("https://{storage_account}.blob.core.windows.net/{container}")
# or if set `attach_prefix` True, get full_path list of blob
file_full_path_list = azc.ls("https://{storage_account}.blob.core.windows.net/{container}", attach_prefix=True)

# find specific file with `*`
file_full_path_list = azc.glob("https://{storage_account}.blob.core.windows.net/{container}/some_folder/*.csv")
# also search deeper directory
file_full_path_list = azc.glob("https://{storage_account}.blob.core.windows.net/{container}/some_folder/**/*.csv")
# or if the directory starts with `a`
file_full_path_list = azc.glob("https://{storage_account}.blob.core.windows.net/{container}/some_folder/a/*.csv")

# check if file exists
is_exists = azc.exists("https://{storage_account}.blob.core.windows.net/{container}/some_folder/test.csv")
```

3.5 remove, copy files, etc...

AzFS also provides remove and copy functions. API reference is [manipulating](#).

```
import azfs

azc = azfs.AzFileClient()

# copy file from `src_path` to `dst_path`
src_path = "https://{storage_account}.blob.core.windows.net/{container}/src_folder/*.csv"
dst_path = "https://{storage_account}.blob.core.windows.net/{container}/dst_folder/*.csv"
is_copied = azc.cp(src_path=src_path, dst_path=dst_path, overwrite=True)

# remove the file
is_removed = azc.rm(path=src_path)

# get file meta info
data = azc.info(path=src_path)
```

3.6 manipulate QueueStorage

AzFS can also manipulate QueueStorage.

```
import azfs
queue_url = "https://{storage_account}.queue.core.windows.net/{queue_name}"

azc = azfs.AzFileClient()
queue_message = azc.get(queue_url)
# message will not be deleted if `delete=False`
# queue_message = azc.get(queue_url, delete=False)

# get message content
queue_content = queue_message.get('content')
```

3.7 manipulate TableStorage

AzFS can now manipulate TableStorage.

API reference is [table storage](#).

```
import azfs
cons = {
    "account_name": "{storage_account_name}",
    "account_key": "{credential}",
    "database_name": "{database_name}"
}

table_client = azfs.TableStorageWrapper(**cons)

# put data, according to the keyword you put
table_client.put(id="1", message="hello_world")

# get data
table_client.get(id="1")
```

4.1 API Reference

4.1.1 AzFileClient

class `azfs.AzFileClient` (*credential: Union[str, azure.identity._credentials.default.DefaultAzureCredential, None] = None, connection_string: Optional[str] = None*)

AzFileClient is

- list files in blob (also with wildcard *),
- check if file exists,
- read csv as `pd.DataFrame`, and json as dict from blob,
- write `pd.DataFrame` as csv, and dict as json to blob,

Examples

```
>>> import azfs
>>> from azure.identity import DefaultAzureCredential
credential is not required if your environment is on AAD
>>> azc = azfs.AzFileClient()
credential is required if your environment is not on AAD
>>> credential = "[your storage account credential]"
>>> azc = azfs.AzFileClient(credential=credential)
# or
>>> credential = DefaultAzureCredential()
>>> azc = azfs.AzFileClient(credential=credential)
connection_string will be also accepted
>>> connection_string = "[your connection_string]"
>>> azc = azfs.AzFileClient(connection_string=connection_string)
```

get/download

`azfs.AzFileClient.get(self, path: str, offset: int = None, length: int = None, **kwargs) → Union[bytes, str, _io.BytesIO, dict]`
get data from Azure Blob Storage.

Parameters

- **path** – Azure Blob path URL format, ex: `https://testazfs.blob.core.windows.net/test_container/test1.csv`
- **offset** –
- **length** –
- ****kwargs** –

Returns some data

Examples

```
>>> import azfs
>>> azc = azfs.AzFileClient()
>>> csv_path = "https://testazfs.blob.core.windows.net/test_container/test1.csv"
you can read csv file in azure blob storage
>>> data = azc.get(path=csv_path)
`download()` is same method as `get()`
>>> data = azc.download(path=csv_path)
```

`azfs.AzFileClient.read_line_iter(self, path: str) → iter`
To read text file in each line with iterator.

Parameters **path** – Azure Blob path URL format, ex: `https://testazfs.blob.core.windows.net/test_container/test1.csv`

Returns get data of the path as iterator

Examples

```
>>> import azfs
>>> azc = azfs.AzFileClient()
>>> csv_path = "https://testazfs.blob.core.windows.net/test_container/test1.csv"
>>> for l in azc.read_line_iter(path=csv_path):
...     print(l.decode("utf-8"))
```

`azfs.AzFileClient.read_csv(self, path: str, **kwargs) → pandas.core.frame.DataFrame`
get csv data as `pd.DataFrame` from Azure Blob Storage. support `csv` and also `csv.gz`.

Parameters

- **path** – Azure Blob path URL format, ex: `https://testazfs.blob.core.windows.net/test_container/test1.csv`
- ****kwargs** – keywords to put `df.read_csv()`, such as `header`, `encoding`.

Returns `pd.DataFrame`

Examples

```
>>> import azfs
>>> azc = azfs.AzFileClient()
>>> csv_path = "https://testazfs.blob.core.windows.net/test_container/test1.csv"
you can read and write csv file in azure blob storage
>>> df = azc.read_csv(path=csv_path)
Using `with` statement, you can use `pandas`-like methods
>>> with azc:
>>>     df = pd.read_csv_az(path)
```

`azfs.AzFileClient.read_table(self, path: str, **kwargs) → pandas.core.frame.DataFrame`
get tsv data as `pd.DataFrame` from Azure Blob Storage. support tsv.

Parameters

- **path** – Azure Blob path URL format, ex: `https://testazfs.blob.core.windows.net/test_container/test1.tsv`
- ****kwargs** – keywords to put `df.read_csv()`, such as `header`, `encoding`.

Returns `pd.DataFrame`

Examples

```
>>> import azfs
>>> azc = azfs.AzFileClient()
>>> tsv_path = "https://testazfs.blob.core.windows.net/test_container/test1.tsv"
you can read and write csv file in azure blob storage
>>> df = azc.read_table(path=tsv_path)
Using `with` statement, you can use `pandas`-like methods
>>> with azc:
>>>     df = pd.read_table_az(tsv_path)
```

`azfs.AzFileClient.read_pickle(self, path: str, compression='gzip') → pandas.core.frame.DataFrame`
get pickled-pandas data as `pd.DataFrame` from Azure Blob Storage.

Parameters

- **path** – Azure Blob path URL format, ex: `https://testazfs.blob.core.windows.net/test_container/test1.pkl`
- **compression** – acceptable keywords are: `gzip`, `bz2`, `xz`. `gzip` is default value.

Returns `pd.DataFrame`

Examples

```
>>> import azfs
>>> azc = azfs.AzFileClient()
>>> pkl_path = "https://testazfs.blob.core.windows.net/test_container/test1.pkl"
you can read and write csv file in azure blob storage
>>> df = azc.read_pickle(path=pkl_path)
Using `with` statement, you can use `pandas`-like methods
>>> with azc:
>>>     df = pd.read_pickle_az(pkl_path)
```

(continues on next page)

(continued from previous page)

```

you can use difference compression
>>> with azc:
>>>     df = pd.read_pickle_az(pk1_path, compression="bz2")

```

`azfs.AzFileClient.read_json(self, path: str, **kwargs) → dict`
 read json file in Datalake storage.

Parameters

- **path** – Azure Blob path URL format, ex: `https://testazfs.blob.core.windows.net/test_container/test1.json`
- ****kwargs** – keywords to put `json.loads()`, such as `parse_float`.

Returns dict

Examples

```

>>> import azfs
>>> azc = azfs.AzFileClient()
>>> json_path = "https://testazfs.blob.core.windows.net/test_container/test1.json"
you can read and write csv file in azure blob storage
>>> azc.read_json(path=json_path)

```

pyspark-like method

You can read multiple files, using multiprocessing or filters,

`azfs.AzFileClient.read(self, *, path: Union[str, List[str]] = None, use_mp: bool = False, cpu_count: Optional[int] = None, file_format: str = 'csv') → azfs.az_file_client.DataFrameReader`
 read csv, parquet, pickle files in Azure Blob, like PySpark-method.

Parameters

- **path** – Azure Blob path URL format, ex: `https://testazfs.blob.core.windows.net/test_container/test1.csv`
- **use_mp** – Default, False
- **cpu_count** – Default, as same as `mp.cpu_count()`
- **file_format** – determined by which function you call

Returns `pd.DataFrame`

Examples

```

>>> import azfs
>>> azc = azfs.AzFileClient()
>>> blob_path = "https://testazfs.blob.core.windows.net/test_container/test1.csv"
>>> df = azc.read().csv(blob_path)
# result is as same as azc.read_csv(blob_path)
>>> blob_path_list = [
...     "https://testazfs.blob.core.windows.net/test_container/test1.csv",

```

(continues on next page)

(continued from previous page)

```

...     "https://testazfs.blob.core.windows.net/test_container/test2.csv"
... ]
>>> df = azc.read().csv(blob_path_list)
# result is as same as pd.concat([each data-frame])
# in addition, you can use `*`
>>> blob_path_pattern = "https://testazfs.blob.core.windows.net/test_container/
↳test*.csv"
>>> df = azc.read().csv(blob_path_pattern)
# you can use multiprocessing with `use_mp` argument
>>> df = azc.read(use_mp=True).csv(blob_path_pattern)
# if you want to filter or apply some method, you can use your defined function,
↳as below
>>> def filter_function(_df: pd.DataFrame, _id: str) -> pd.DataFrame:
...     return _df[_df['_id'] == _id]
>>> df = azc.read(use_mp=True).apply(function=filter_function, _id="aaa").
↳csv(blob_path_pattern)

```

put/upload

azfs.AzFileClient.**put** (*self*, *path: str*, *data*) → bool
upload data to blob or data_lake storage.

Parameters

- **path** – Azure Blob path URL format, ex: https://testazfs.blob.core.windows.net/test_container/test1.csv
- **data** – some data to upload.

Returns True if correctly uploaded

Examples

```

>>> import azfs
>>> azc = azfs.AzFileClient()
>>> csv_path = "https://testazfs.blob.core.windows.net/test_container/test1.csv"
you can write file in azure blob storage
>>> _data = azc.put(path=csv_path)
`download()` is same method as `get()`
>>> _data = azc.upload(path=csv_path)

```

azfs.AzFileClient.**write_csv** (*self*, *path: str*, *df: pandas.core.frame.DataFrame*, ***kwargs*) → bool
output pandas dataframe to csv file in Datalake storage.

Parameters

- **path** – Azure Blob path URL format, ex: https://testazfs.blob.core.windows.net/test_container/test1.csv.
- **df** – pd.DataFrame to upload.
- ****kwargs** – keywords to put df.to_csv(), such as encoding, index.

Returns True if correctly uploaded

Examples

```
>>> import azfs
>>> azc = azfs.AzFileClient()
>>> csv_path = "https://testazfs.blob.core.windows.net/test_container/test1.csv"
you can read and write csv file in azure blob storage
>>> azc.write_csv(path=csv_path, df=df)
Using `with` statement, you can use `pandas`-like methods
>>> with azc:
>>>     df.to_csv_az(csv_path)
```

`azfs.AzFileClient.write_table(self, path: str, df: pandas.core.frame.DataFrame, **kwargs) → bool`
 output pandas dataframe to tsv file in Datalake storage.

Parameters

- **path** – Azure Blob path URL format, ex: `https://testazfs.blob.core.windows.net/test_container/test1.tsv`.
- **df** – `pd.DataFrame` to upload.
- ****kwargs** – keywords to put `df.to_csv()`, such as `encoding`, `index`.

Returns True if correctly uploaded

Examples

```
>>> import azfs
>>> azc = azfs.AzFileClient()
>>> tsv_path = "https://testazfs.blob.core.windows.net/test_container/test1.tsv"
you can read and write csv file in azure blob storage
>>> azc.write_table(path=tsv_path, df=df)
Using `with` statement, you can use `pandas`-like methods
>>> with azc:
>>>     df.to_table_az(tsv_path)
```

`azfs.AzFileClient.write_pickle(self, path: str, df: pandas.core.frame.DataFrame, compression='gzip') → bool`
 output pandas dataframe to tsv file in Datalake storage.

Parameters

- **path** – Azure Blob path URL format, ex: `https://testazfs.blob.core.windows.net/test_container/test1.pkl`
- **df** – `pd.DataFrame` to upload.
- **compression** – acceptable keywords are: `gzip`, `bz2`, `xz`. `gzip` is default value.

Returns `pd.DataFrame`

Examples

```
>>> import azfs
>>> azc = azfs.AzFileClient()
>>> pkl_path = "https://testazfs.blob.core.windows.net/test_container/test1.pkl"
you can read and write csv file in azure blob storage
```

(continues on next page)

(continued from previous page)

```

>>> azc.write_pickle(path=pk1_path, df=df)
Using `with` statement, you can use `pandas`-like methods
>>> with azc:
>>>     df.to_pickle_az(pk1_path)
you can use difference compression
>>> with azc:
>>>     df.to_pickle_az(pk1_path, compression="bz2")

```

`azfs.AzFileClient.write_json(self, path: str, data: dict, **kwargs) → bool`
 output dict to json file in Datalake storage.

Parameters

- **path** – Azure Blob path URL format, ex: `https://testazfs.blob.core.windows.net/test_container/test1.json`
- **data** – dict to upload
- ****kwargs** – keywords to put `json.loads()`, such as `indent`.

Returns True if correctly uploaded

Examples

```

>>> import azfs
>>> azc = azfs.AzFileClient()
>>> json_path = "https://testazfs.blob.core.windows.net/test_container/test1.json"
you can read and write csv file in azure blob storage
>>> azc.write_json(path=json_path, data={"": ""})

```

file enumerating

`azfs.AzFileClient.ls(self, path: str, attach_prefix: bool = False) → list`
 list blob file from blob or dfs.

Parameters

- **path** – Azure Blob path URL format, ex: `https://testazfs.blob.core.windows.net/test_container`
- **attach_prefix** – return `full_path` if True, return only name

Returns list of azure blob files

Examples

```

>>> import azfs
>>> azc = azfs.AzFileClient()
>>> csv_path = "https://testazfs.blob.core.windows.net/test_container"
>>> azc.ls(csv_path)
[
    "test1.csv",
    "test2.csv",
    "test3.csv",
    "directory_1",

```

(continues on next page)

(continued from previous page)

```

    "directory_2"
]
>>> azc.ls(path=path, attach_prefix=True)
[
    "https://testazfs.blob.core.windows.net/test_container/test1.csv",
    "https://testazfs.blob.core.windows.net/test_container/test2.csv",
    "https://testazfs.blob.core.windows.net/test_container/test3.csv",
    "https://testazfs.blob.core.windows.net/test_container/directory_1",
    "https://testazfs.blob.core.windows.net/test_container/directory_2"
]

```

`azfs.AzFileClient.glob(self, pattern_path: str) → List[str]`

Currently only support * (wildcard) . By default, `glob()` lists specified files with formatted-URL.

Parameters `pattern_path` – ex: `https://<storage_account_name>.blob.core.windows.net/<container>/**/*.csv`

Returns lists specified files filtered by wildcard

Examples

```

>>> import azfs
>>> azc = azfs.AzFileClient()
>>> path = "https://testazfs.blob.core.windows.net/test_container/some_folder"
ls() lists all files in some folder like
>>> azc.ls(path)
[
    "test1.csv",
    "test2.csv",
    "test3.csv",
    "test1.json",
    "test2.json",
    "directory_1",
    "directory_2"
]
glob() lists specified files according to the wildcard, and lists with formatted-
↪URL by default
>>> csv_pattern_path = "https://testazfs.blob.core.windows.net/test_container/
↪some_folder/*.csv"
>>> azc.glob(path=csv_pattern_path)
[
    "https://testazfs.blob.core.windows.net/test_container/some_folder/test1.csv",
    "https://testazfs.blob.core.windows.net/test_container/some_folder/test2.csv",
    "https://testazfs.blob.core.windows.net/test_container/some_folder/test3.csv"
]
glob() can use any path
>>> csv_pattern_path = "https://testazfs.blob.core.windows.net/test_container/
↪some_folder/test1.*"
>>> azc.glob(path=csv_pattern_path)
[
    "https://testazfs.blob.core.windows.net/test_container/some_folder/test1.csv",
    "https://testazfs.blob.core.windows.net/test_container/some_folder/test1.json"
]
also deeper folders
>>> csv_pattern_path = "https://testazfs.blob.core.windows.net/test_container/
↪some_folder/**/*.csv"

```

(continues on next page)

(continued from previous page)

```
>>> azc.glob(path=csv_pattern_path)
[
    "https://testazfs.blob.core.windows.net/test_container/some_folder/directory_
    ↳1/deeper_test1.csv",
    "https://testazfs.blob.core.windows.net/test_container/some_folder/directory_
    ↳2/deeper_test2.csv"
]
```

Raises `AzfsInputError` – when `*` is used in `root_folder` under a container.

`azfs.AzFileClient.exists(self, path: str) → bool`
 check if specified file exists or not.

Parameters `path` – Azure Blob path URL format, ex: `https://testazfs.blob.core.windows.net/test_container/test1.csv`

Returns True if files exists, otherwise False

Examples

```
>>> import azfs
>>> azc = azfs.AzFileClient()
>>> csv_path = "https://testazfs.blob.core.windows.net/test_container/test1.csv"
>>> azc.exists(path=csv_path)
True
>>> csv_path = "https://testazfs.blob.core.windows.net/test_container/not_exist_
    ↳test1.csv"
>>> azc.exists(path=csv_path)
False
```

file manipulating

`azfs.AzFileClient.info(self, path: str) → dict`
 get file properties, such as name, creation_time, last_modified_time, size, content_hash(md5).

Parameters `path` – Azure Blob path URL format, ex: `https://testazfs.blob.core.windows.net/test_container/test1.csv`

Returns dict info of some file

Examples

```
>>> import azfs
>>> azc = azfs.AzFileClient()
>>> csv_path = "https://testazfs.blob.core.windows.net/test_container/test1.csv"
>>> azc.info(path=csv_path)
{
    "name": "test1.csv",
    "size": "128KB",
    "creation_time": "",
    "last_modified": "",

```

(continues on next page)

(continued from previous page)

```

    "etag": "etag...",
    "content_type": "",
    "type": "file"
}

```

`azfs.AzFileClient.rm(self, path: str) → bool`

delete the file in blob

Parameters **path** – Azure Blob path URL format, ex: `https://testazfs.blob.core.windows.net/test_container/test1.csv`

Returns True if target file is correctly removed.

`azfs.AzFileClient.cp(self, src_path: str, dst_path: str, overwrite=False) → bool`

copy the data from *src_path* to *dst_path*

Parameters

- **src_path** – Azure Blob path URL format, ex: `https://testazfs.blob.core.windows.net/test_container/test1.csv`
- **dst_path** – Azure Blob path URL format, ex: `https://testazfs.blob.core.windows.net/test_container/test2.csv`
- **overwrite** –

Returns:

4.1.2 TableStorage

class `azfs.TableStorage(account_name: str, account_key: str, database_name: str)`

A class for manipulating TableStorage in Storage Account The class provides simple methods below.

- create
- read
- update
- delete(not yet)

The class is intended to be used as *delegation*, not *extend*.

Parameters

- **account_name** – name of the Storage Account
- **account_key** – key for the Storage Account
- **database_name** – name of the StorageTable database

class `azfs.TableStorageWrapper(account_name, account_key, database_name, partition_key: str, row_key_name: str = 'id_')`

Wrapper for the TableStorage class.

Parameters

- **account_name** – name of the Storage Account
- **account_key** – key for the Storage Account
- **database_name** – name of the StorageTable database
- **partition_key** –

- `row_key_name` –

Examples

```
>>> import json
>>> from datetime import datetime
>>> from pytz import timezone
>>> tokyo = timezone('Asia/Tokyo')
>>> cons = {
...     "account_name": "{storage_account_name}",
...     "account_key": "{credential}",
...     "database_name": "{database_name}"
... }
# you can manipulate data through `simple_table_client`
>>> simple_table_client = TableStorageWrapper(partition_key="simple_table",
↳**cons)
# store data according to the keyword-argument you put
# by default, `id_` is converted to `RowKey`, then `id_` is not stored
>>> simple_table_client.put(id_="1", message="hello_world")
... {'PartitionKey': 'simple_table', 'message': 'hello_world', 'RowKey': '1'}
# can get all data, simply call
>>> simple_table_client.get()
... ..
# or filter with specific value, like
# `id_` is configured as `RowKey` by default
>>> simple_table_client.get(id_="1")
... [
...     {
...         'PartitionKey': 'simple_table',
...         'RowKey': '1',
...         'Timestamp': datetime.datetime(2020, 10, 10, 3, 15, 57, 874427,
↳tzinfo=tzutc()),
...         'message': 'hello_world',
...         'etag': 'W/"datetime'2020-10-10T03%3A15%3A57.8744271Z'"
...     }
... ]
# In addition, you can store data in different way
>>> complex_client = TableStorageWrapper(partition_key="complex_table", **cons)
>>> @complex_client.overwrite_pack_data_to_put()
... def modify_put_data(id_: str, message: str):
...     alt_message = json.dumps({datetime.now(tz=tokyo).isoformat(): message},
↳ensure_ascii=False)
...     return {"id_": id_, "message": alt_message}
# you can store data in a different way
>>> complex_client.put(id_="2", message="hello_world")
... {
...     'PartitionKey': 'complex_table',
...     'message': '{"2020-10-10T12:26:57.442718+09:00": "hello_world"}',
...     'RowKey': '2'
... }
# you can also modify update function, with restriction example
>>> @complex_client.overwrite_pack_data_to_update(allowed={"message": ["ERROR",
↳"RUNNING", "SUCCESS"]})
... def modify_update_data(id_: str, message: str):
...     d = complex_client.get(id_=id_)
...     message_dict = json.loads(d[0]['message'])
...     if type(message_dict) is dict:
```

(continues on next page)

(continued from previous page)

```

...     message_dict[datetime.now(tz=tokyo).isoformat()] = message
...     else:
...         message_dict = {datetime.now(tz=tokyo).isoformat(): message}
...
...     data = {
...         "id_": id_,
...         "message": json.dumps(message_dict, ensure_ascii=False)
...     }
...     return data
>>> complex_client.update(id_="2", message="RUNNING")
... {
...     'PartitionKey': 'complex_table',
...     'RowKey': '2',
...     'message': '{"2020-10-10T12:26:57.442718+09:00": "hello_world", "2020-10-
↪10T13:00:23.602943+09:00": "RUNNING"}'
... }

```

4.1.3 BlobPathDecoder

class `azfs.BlobPathDecoder` (*path: Union[None, str] = None*)
 Decode Azure Blob Storage URL format class

Examples

```

>>> import azfs
>>> path = "https://testazfs.blob.core.windows.net/test_container/test1.csv"
>>> blob_path_decoder = azfs.BlobPathDecoder()
>>> blob_path_decoder.decode(path=path).get()
(testazfs, blob, test_container, test1.csv)
>>> blob_path_decoder.decode(path=path).get_with_url()
(https://testazfs.blob.core.windows.net", blob, test_container, test1.csv)

```

4.2 Release History

4.2.1 0.2.14 (2021-01-11)

- add default logger
 - `logger = getLogger("azfs")` (#144)
- update `azfs.ExportDecorator()`
 - ignore exception if `ignore_error` is `True` (#142)

4.2.2 0.2.13 (2020-12-22)

- update `azfs.ExportDecorator()`
 - accept `default_parameter` each argument (#128)
- BUG FIX: cli command `$ azfs decorator -n {file_name}`

- NameError occurred (#132)

4.2.3 0.2.12 (2020-12-15)

- update `azfs.ExportDecorator()`
 - attach `under_bar` each argument to avoid conflict (#130)

4.2.4 0.2.11 (2020-12-13)

- update `azfs.ExportDecorator()`
 - accept basic argument (#123)
 - add appropriate error (#124)
- add cli command `$ azfs decorator -n {file_name}`
 - to avoid PEP8 violation on PyCharm(#121)

4.2.5 0.2.10 (2020-12-10)

- update `azfs.ExportDecorator()` (#120)
 - accept multiple `file_name` and multiple return values from user-defined function
 - accept `str` or `dict`, when `import_decorator()`

4.2.6 0.2.9 (2020-12-08)

- add Experimental feature: `azfs.ExportDecorator()`

4.2.7 0.2.8 (2020-11-24)

- BUG FIX: `azc.read(use_mp=True)` is not working correctly, if credential is `AzureDefaultCredentials`.
- BUG FIX: `glob(path)` is not working correctly if path contains special characters like `(or)`, etc.

4.2.8 0.2.7 (2020-11-13)

- BUG FIX: `write_json(ensure_ascii=False)` is not working correctly if data-json has non-ascii character.

4.2.9 0.2.6 (2020-11-11)

- add multiprocessing-read, as `azc.read(use_mp=True).csv()`.
- add `apply()` function, as `azc.read(use_mp=True).apply(function=some_function).csv()`
- modify `glob()`: not working under a certain directory.

4.2.10 0.2.5 (2020-11-01)

- modify `exists()`: use `info()` instead of `_get()`.
- add pyspark-like read method, such as `azc.read().csv()`, `azc.read().parquet()`.
- add `__all__`, and organize the directory.

4.2.11 0.2.4 (2020-10-10)

- add `TableStorage`, and `TableStorageWrapper` class to manipulate `TableStorage`.

4.2.12 0.2.3 (2020-09-09)

- Modify `AzfsInputError` message from Japanese to English.

4.2.13 0.2.2 (2020-08-26)

- partially implement `AzFileSystem`: at least it works.
- add type hinting
- add `read_parquet()`: to read parquet file in Azure.

4.2.14 0.2.1 (2020-07-19)

- partially adopt chunk-reading for csv file.
- implement `AzfsFileClient::read_line_iter()`: to read text file in each line with iterator.
- implement `BlobPathDecoder.add_pattern()`: to decode user-defined path

4.2.15 0.2.0 (2020-07-14)

- can upload over 100MB data to `DataLakeStorage`.
- support `connection_string`: in `azfs.AzFileClient()`

4.2.16 0.1.10 (2020-07-05)

- implement new functions: read/write `pickle` and `tsv` format.
- implement `AzContextManager()`: set and get attributes to pandas easily
- code optimization `glob()`: make execution faster, and add limitation (not allowed to `glob()` in root folder under a container)

4.2.17 0.1.9 (2020-06-30)

- implement `_get_service_client()`: to prepare adopting new authorization
- code optimization: `BlobPathDecoder` class

4.2.18 0.1.8 (2020-06-22)

- modify `write_csv()`: apply default encoding is UTF-8
- modify `exists()`

4.2.19 0.1.7 (2020-06-07)

- modify `ls()`:
- modify `put()` and `get()` in `Queue`: apply Base64 encode/decode

4.2.20 0.1.6 (2020-06-02)

- check compatibility on Python 3.6 and Python 3.8
- modify `glob()`: compile regex

4.2.21 0.1.5 (2020-05-30)

- implementing `glob()`
- add `prefix`-parameter to `ls()`
- add `Queue` operation class

4.2.22 0.1.4 (2020-05-14)

- add `**kwargs` to read/write functions

4.2.23 0.1.3 (2020-05-12)

- add implementing candidate methods
- modify `ls()`: add filter

4.2.24 0.1.2 (2020-05-10)

- remove if-statement using metaclass
- add test on PyTest

4.2.25 0.1.1 (2020-05-03)

- add `DataLakeClient`

4.2.26 0.1.0 (2020-04-29)

- initial release

5.1 API Reference (for Developer)

5.1.1 AzfsClient

class `azfs.clients.AzfsClient` (*credential*, *connection_string*)
 Abstract Client for AzBlobClient, AzDataLakeClient and AzQueueClient.

Examples

```
>>> blob_client = AzfsClient(credential="...").get_client("blob")
# or
>>> datalake_client = AzfsClient(credential="...").get_client("dfs")
# AzfsClient provide easy way to access functions implemented in AzBlobClient and
↳AzDataLakeClient, as below
>>> data_path = "https://testazfs.blob.core.windows.net/test_container/test1.json"
>>> data = AzfsClient(credential="...").get_client("blob").get(path=data_path)
```

5.1.2 ClientInterface

class `azfs.clients.client_interface.ClientInterface` (*credential*: *Union[str, azure.identity_credentials.default.DefaultAzureCredential, None]*, *connection_string*: *Optional[str] = None*)

The class provides Azure Blob, DataLake and Queue Client interface. Abstract methods below are implemented in each inherited classes.

- `_get_file_client`
- `_get_service_client`
- `_get_container_client`

- `_ls`
- `_get`
- `_put`

Clients

```
azfs.clients.client_interface.ClientInterface.get_file_client_from_path(self,
                                                                    path:
                                                                    str)
                                                                    →
                                                                    Union[azure.storage.blob._l
                                                                    azure.storage.filedatalake._c
                                                                    azure.storage.queue._queue
```

get file_client from given path

Parameters `path` – Azure path that `BlobPathDecode()` can decode

Returns `Union[BlobClient, DataLakeFileClient, QueueClient]`

```
azfs.clients.client_interface.ClientInterface.get_container_client_from_path(self,
                                                                    path:
                                                                    str)
                                                                    →
                                                                    Union[azure.storage
                                                                    azure.storage.filedat
```

get container_client from given path

Parameters `path` – Azure path that `BlobPathDecode()` can decode

Returns `Union[ContainerClient, FileSystemClient]`

5.2 GitHub

AzFS repository is [here](#).

5.3 References

- [azure-sdk-for-python/storage](#)
- [filesystem_spec](#)

CHAPTER 6

Indices and tables

- `genindex`
- `search`

A

AzFileClient (class in azfs), 11
AzfsClient (class in azfs.clients), 27

B

BlobPathDecoder (class in azfs), 22

C

ClientInterface (class
azfs.clients.client_interface), 27
cp() (in module azfs.AzFileClient), 20

E

exists() (in module azfs.AzFileClient), 19

G

get() (in module azfs.AzFileClient), 12
get_container_client_from_path() (in module
azfs.clients.client_interface.ClientInterface),
28
get_file_client_from_path() (in module
azfs.clients.client_interface.ClientInterface),
28
glob() (in module azfs.AzFileClient), 18

I

info() (in module azfs.AzFileClient), 19

L

ls() (in module azfs.AzFileClient), 17

P

put() (in module azfs.AzFileClient), 15

R

read() (in module azfs.AzFileClient), 14
read_csv() (in module azfs.AzFileClient), 12
read_json() (in module azfs.AzFileClient), 14

read_line_iter() (in module azfs.AzFileClient), 12
read_pickle() (in module azfs.AzFileClient), 13
read_table() (in module azfs.AzFileClient), 13
rm() (in module azfs.AzFileClient), 20

T

TableStorage (class in azfs), 20
TableStorageWrapper (class in azfs), 20

in W

write_csv() (in module azfs.AzFileClient), 15
write_json() (in module azfs.AzFileClient), 17
write_pickle() (in module azfs.AzFileClient), 16
write_table() (in module azfs.AzFileClient), 16