e2fyi-utils

Release 0.2.2

eterna2 <eterna2@hotmail.com>

Oct 01, 2020

CONTENTS:

CHAPTER

ONE

API REFERENCE

This page contains auto-generated API reference documentation[1].

1.1 e2fyi

e2fyi-utils is an e2fyi namespaced package with the e2fyi.utils subpackage.

1.1.1 Subpackages

1.1.1.1 e2fyi.utils

1.1.1.1.1 Subpackages

1.1.1.1.1.1 e2fyi.utils.aws

Helpers for interacting with S3 buckets.

S3 bucket with preset prefix:

```
from e2fyi.utils.aws import S3Bucket

# upload a dict to s3 bucket
S3Bucket("foo").upload("some_folder/some_file.json", {"foo": "bar"})

# creates a s3 bucket with std prefix rule
foo_bucket = S3Bucket("foo", get_prefix=lambda prefix: "some_folder/%s" % prefix)
foo_bucket.upload("some_file.json", {"foo": "bar"})  # some_folder/some_file.json
```

Uploading to S3 bucket:

```
import logging

import pandas as pd

from e2fyi.utils.aws import S3Bucket
from pydantic import BaseModel

# s3 bucket with prefix rule
s3 = S3Bucket("foo", get_prefix=lambda prefix: "some_folder/%s" % prefix)
```

(continues on next page)

[1] Created with sphinx-autoapi

(continued from previous page)

```
# check if upload is successful
result = s3.upload("some_file.txt", "hello world")
if not result.is_ok:
    logging.exception(result.exception)

# upload string as text/plain file
s3.upload("some_file.txt", "hello world")

# upload dict as application/json file
s3.upload("some_file.json", {"foo": "bar"})

# upload pandas df as text/csv and/or application/json files
df = pd.DataFrame([{"key": "a", "value": 1}, {"key": "b", "value": 2}])
# extra kwargs can be passed to pandas.to_csv method
s3.upload("some_file.csv", df, content_type="text/csv", index=False)
# extra kwargs can be passed to pandas.to_json method
s3.upload("some_file.json", df, content_type="application/json", orient="records")

# upload pydantic models as application/json file
class KeyValue(BaseModel):
    key: str
    value: int
model = KeyValue(key="a", value=1)
s3.upload("some_file.json", model)
```

Listing contents inside S3 buckets:

```
from e2fyi.utils.aws import S3Bucket

# list files
S3Bucket("foo").list("some_folder/")

# list files inside "some_folder/"
foo_bucket.list()
```

1.1.1.1.1.2 Submodules

1.1.1.1.1.3 e2fyi.utils.aws.compat

Module for compatability so that no errors will be thrown even if a optional package is not available.

1.1.1.1.1.4 Module Contents

e2fyi.utils.aws.compat.**LIB_MAGIC_AVAILABLE = True**

1.1.1.1.1.5 e2fyi.utils.aws.s3

utils to interact with s3 buckets.

1.1.1.1.1.6 Module Contents

1.1.1.1.1.7 Classes

S3Bucket	*S3Bucket* is an abstraction of the actual S3 bucket with methods to interact

e2fyi.utils.aws.s3.**T**

e2fyi.utils.aws.s3.**StringOrBytes**

e2fyi.utils.aws.s3.**ALLOWED_DOWNLOAD_ARGS = ['VersionId', 'SSECustomerAlgorithm', 'SSECustom**

e2fyi.utils.aws.s3.**ALLOWED_UPLOAD_ARGS = ['ACL', 'CacheControl', 'ContentDisposition', 'Co**

class e2fyi.utils.aws.s3.**S3Bucket**(*name: str, get_prefix: Callable[[str], str] = _noop, s3client: boto3.client = None*)

S3Bucket is an abstraction of the actual S3 bucket with methods to interact with the actual S3 bucket (e.g. list objects inside the bucket), and some utility methods.

Prefix rules can also be set during the creation of the *S3Bucket* object - i.e. enforce a particular prefix rules for a particular bucket.

Example:

```
from e2fyi.utils.aws import S3Bucket

# prints key for all resources with prefix "some_folder/"
for resource in S3Bucket("some_bucket").list("some_folder/"):
    print(resource.key)

# prints key for the first 2,000 resources with prefix "some_folder/"
for resource in S3Bucket("some_bucket").list("some_folder/", max_objects=2000):
    print(resource.key)

# creates a s3 bucket with prefix rule
prj_bucket = S3Bucket(
    "some_bucket", get_prefix=lambda prefix: "prj-a/%s" % prefix
)
for resource in prj_bucket.list("some_folder/"):
    print(resource.key)  # prints "prj-a/some_folder/<resource_name>"

# get obj key in the bucket
print(prj_bucket.create_resource_key("foo.json"))  # prints "prj-a/foo.json"

# get obj uri in the bucket
# prints "s3a://some_bucket/prj-a/foo.json"
```

(continues on next page)

(continued from previous page)

```
print(prj_bucket.create_resource_uri("foo.json", "s3a://"))

# create S3Resource in bucket to read in
foo = prj_bucket.create_resource("foo.json", "application/json")
# read "s3a://some_bucket/prj-a/foo.json" and load as a dict (or list)
foo_dict = foo.load()

# create S3Resource in bucket and save to "s3a://some_bucket/prj-a/foo.json"
prj_bucket.create_resource("foo.json", obj={"foo": "bar"}).save()
```

Creates a new instance of s3 bucket.

Args: name (str): name of the bucket get_prefix (Callable[[str], str], optional): function that takes a filename and return the full path to the resource in the bucket.

s3client (boto3.Session.client, optional): use a custom boto3 s3 client.

create_resource_key(*self*, *filename: str*) → str

Creates a resource key based on the s3 bucket name, and configured prefix.

Example:

```
from e2fyi.utils.aws import S3Bucket

s3 = S3Bucket(name="foo", get_prefix=lambda x: "bar/%s" % x)

print(s3.create_resource_key("hello.world"))  # > bar/hello.world
```

Args: filename (str): path for the file.

Returns: str: key for the resource in s3.

create_resource_uri(*self*, *filename: str*, *protocol: str = 's3a://'*) → str

Create a resource uri based on the s3 bucket name, and configured prefix.

Example:

```
from e2fyi.utils.aws import S3Bucket

s3 = S3Bucket(name="foo", get_prefix=lambda x: "bar/%s" % x)

print(s3.create_resource_uri("hello.world"))  # > s3a://foo/bar/hello.world
```

Args: filename (str): path for the file. protocol (str, optional): protocol for the uri. Defaults to "s3a://".

Returns: str: uri string for the resource.

list(*self*, *prefix: str = ''*, *within_project: bool = True*, *max_objects: int = -1*) → Generator[S3Resource[bytes], None, None]

Returns a generator that yield S3Resource objects inside the S3Bucket that matches the provided prefix.

Example:

```
# prints key for all resources with prefix "some_folder/"
for resource in S3Bucket("some_bucket").list("some_folder/"):
    print(resource.key)
```

(continues on next page)

(continued from previous page)

```
# prints key for the first 2,000 resources with prefix "some_folder/"
for resource in S3Bucket(
    "some_bucket").list("some_folder/", max_objects=2000
):
    print(resource.key)

# creates a s3 bucket with prefix rule
prj_bucket = S3Bucket(
    "some_bucket", get_prefix=lambda prefix: "prj-a/%s" % prefix
)
for resource in prj_bucket.list("some_folder/"):
    print(resource.key)  # prints "prj-a/some_folder/<resource_name>"
```

Args: prefix (str, optional): [description]. Defaults to “”. within_project (bool, optional): [description]. Defaults to True. max_objects (int, optional): max number of object to return. Negative

or zero means all objects will be returned. Defaults to -1.

Returns: Generator[S3Resource[bytes], None, None]: [description]

Yields: Generator[S3Resource[bytes], None, None]: [description]

create_resource(*self*, *filename: str*, *content_type: str = ''*, *obj: Any = None*, *protocol: str = 's3a://'*, *metadata: Dict[str, str] = None*, *pandas_kwargs: dict = None*, ***kwargs*) → S3Resource

Creates a new instance of S3Resource binds to the current bucket.

Example:

```
# create S3Resource in bucket to read in
foo = prj_bucket.create_resource("foo.json", "application/json")
# read "s3a://some_bucket/prj-a/foo.json" and load as a dict (or list)
foo_dict = foo.load()

# create S3Resource in bucket and save to "s3a://some_bucket/prj-a/foo.json"
prj_bucket.create_resource("foo.json", obj={"foo": "bar"}).save()
```

Args: filename (str): name of the resource. content_type (str, optional): mime type. Defaults to

“application/octet-stream”.

obj (Any, optional): python object to convert into a resource. Defaults to None.

protocol (str, optional): protocol. Defaults to “s3a://”. stream (Union[io.StringIO, io.BytesIO, IO[StringOrBytes]], optional):

content of the resource. Defaults to None.

metadata (dict, optional): metadata for the object. Defaults to None. pandas_kwargs: Any additional args to pass to *pandas*. **kwargs: Any additional args to pass to *S3Resource*.

Returns: S3Resource: a S3Resource related to the active S3Bucket.

upload(*self*, *filepath: str*, *body: Any*, *content_type: str = 'text/plain'*, *protocol: str = 's3a://'*, *metadata: dict = None*, ***kwargs*) → Any

Deprecated since v0.2.0. Use S3Resource.save instead.

1.1.1.1.1.8 e2fyi.utils.aws.s3_resource

Provides *S3Resource* to represent resources in S3 buckets.

1.1.1.1.1.9 Module Contents

1.1.1.1.1.10 Classes

S3Resource	*S3Resource* represents a resource in S3 currently or a local resource that will

e2fyi.utils.aws.s3_resource.**T**

e2fyi.utils.aws.s3_resource.**StringOrBytes**

class e2fyi.utils.aws.s3_resource.**S3Resource**(*filename: str*, *content_type: str = ''*, *bucketname: str = ''*, *prefix: str = ''*, *protocol: str = 's3a://'*, *stream: S3Stream[StringOrBytes] = None*, *s3client: boto3.client = None*, *stats: dict = None*, ***kwargs*)

S3Resource represents a resource in S3 currently or a local resource that will be uploaded to S3. *S3Resource* constructor will automatically attempts to convert any inputs into a *S3Stream*, but for more granular control *S3Stream.from_any* should be used instead to create the *S3Stream*.

S3Resource is a readable stream - i.e. it has *read*, *seek*, and *close*.

Example:

```
import boto3

from e2fyi.utils.aws import S3Resource, S3Stream

# create custom s3 client
s3client = boto3.client(
    's3',
    aws_access_key_id=ACCESS_KEY,
    aws_secret_access_key=SECRET_KEY
)

# creates a local copy of s3 resource with S3Stream from a local file
obj = S3Resource(
    # full path shld be "prefix/some_file.json"
    filename="some_file.json",
    prefix="prefix/",
    # bucket to download from or upload to
    bucketname="some_bucket",
    # or "s3n://" or "s3://"
    protocol="s3a://",
    # uses default client if not provided
    s3client=s3client,
    # attempts to convert to S3Stream if input is not a S3Stream
    stream=S3Stream.from_file("./some_path/some_file.json"),
    # addition kwarg to pass to `s3.upload_fileobj` or `s3.download_fileobj`
    # methods
```

(continues on next page)

(continued from previous page)

```
    Metadata={"label": "foo"}
)
print(obj.key)  # prints "prefix/some_file.json"
print(obj.uri)  # prints "s3a://some_bucket/prefix/some_file.json"

# will attempt to fix prefix and filename if incorrect filename is provided
obj = S3Resource(
    filename="subfolder/some_file.json",
    prefix="prefix/"
)
print(obj.filename)     # prints "some_file.json"
print(obj.prefix)       # prints "prefix/subfolder/"
```

Saving to S3:

```
from e2fyi.utils.aws import S3Resource

# creates a local copy of s3 resource with some python object
obj = S3Resource(
    filename="some_file.txt",
    prefix="prefix/",
    bucketname="some_bucket",
    stream={"some": "dict"},
)

# upload obj to s3 bucket "some_bucket" with the key "prefix/some_file.json"
# with the json string content.
obj.save()

# upload to s3 bucket "another_bucket" instead with a metadata tag.
obj.save("another_bucket", MetaData={"label": "foo"})
```

Reading from S3:

```
from e2fyi.utils.aws import S3Resource
from pydantic import BaseModel

# do not provide a stream input to the S3Resource constructor
obj = S3Resource(
    filename="some_file.json",
    prefix="prefix/",
    bucketname="some_bucket",
    content_type="application/json"
)

# read the resource like a normal file object from S3
data = obj.read()
print(type(data))       # prints <class 'str'>

# read and load json string into a dict or list
# for content_type == "application/json" only
data_obj = obj.load()
print(type(data_obj))   # prints <class 'dict'> or <class 'list'>


# read and convert into a pydantic model
class Person(BaseModel):
```

(continues on next page)

(continued from previous page)

```
    name: str
    age: int

# automatically unpack the dict
data_obj = obj.load(lambda name, age: Person(name=name, age=age))
# alternatively, do not unpack
data_obj = obj.load(lambda data: Person(**data), unpack=False)
print(type(data_obj))    # prints <class 'Person'>
```

Creates a new instance of S3Resource, which will use *boto3.s3.transfer.S3Transfer* under the hood to download/upload the s3 resource.

See https://boto3.amazonaws.com/v1/documentation/api/latest/reference/customizations/s3.html#boto3.s3.transfer.S3Transfer

Args: filename (str): filename of the object. content_type (str, optional): mime type of the object. Defaults to “”. bucketname (str, optional): name of the bucket the obj is or should be.

Defaults to “”.

prefix (str, optional): prefix to be added to the filename to get the s3 object key. Defaults to “application/octet-stream”.

protocol (str, optional): s3 client protocol. Defaults to “s3a://”. stream (S3Stream[StringOrBytes], optional): data stream. Defaults to None. s3_client (boto3.client, optional): s3 client to use to retrieve

resource. Defaults to None.

Metadata (dict, optional): metadata for the object. Defaults to None. **kwargs: Any additional args to pass to *boto3.s3.transfer.S3Transfer*

function.

property content_type (*self*) → str
mime type of the resource

property key (*self*) → str
Key for the resource.

property uri (*self*) → str
URI to the resource.

property stream (*self*) → S3Stream[StringOrBytes]
data stream for the resource.

read (*self*, *size=-1*) → StringOrBytes
duck-typing for a readable stream.

seek (*self*, *offset: int*, *whence: int = 0*) → int
duck-typing for readable stream. See https://docs.python.org/3/library/io.html

Change the stream position to the given byte offset. offset is interpreted relative to the position indicated by whence. The default value for whence is SEEK_SET. Values for whence are:

SEEK_SET or 0 – start of the stream (the default); offset should be zero or positive

SEEK_CUR or 1 – current stream position; offset may be negative

SEEK_END or 2 – end of the stream; offset is usually negative

Return the new absolute position.

close(*self*) → 'S3Resource'
Close the resource stream.

get_value(*self*) → StringOrBytes
Retrieve the entire contents of the S3Resource.

load(*self*, *constructor: Callable[..., T] = None*, *unpack: bool = True*) → Union[dict, list, T]
load the content of the stream into memory using *json.loads*. If a *constructor* is provided, it will be used to create a new object. Setting *unpack* to be true will unpack the content when creating the object with the *constructor* (i.e. * for list, ** for dict)

Args:

constructor (Callable[..., T], optional): A constructor function. Defaults to None.

unpack (bool, optional): whether to unpack the content when passing it to the constructor. Defaults to True.

Raises: TypeError: [description]

Returns: Union[dict, list, T]: [description]

save(*self*, *bucketname: str = None*, *s3client: boto3.client = None*, ***kwargs*) → 'S3Resource'
Saves the current S3Resource to the provided s3 bucket (in constructor or in arg). Extra args can be pass to *boto3.s3.transfer.S3Transfer* via keyword arguments of the same name.

See https://boto3.amazonaws.com/v1/documentation/api/latest/reference/customizations/s3.html#boto3.s3.transfer.S3Transfer.ALLOWED_UPLOAD_ARGS

Args:

bucketname (str, optional): bucket to save the resource to. Overwrites the bucket name provided in the constructor. Defaults to None.

s3client (boto3.client, optional): custom s3 client to use. Defaults to None.

**kwargs: additional args to pass to *boto3.s3.transfer.S3Transfer*.

Raises: ValueError: "S3 bucket name must be provided."

Returns: S3Resource: S3Resource object.

1.1.1.1.1.11 e2fyi.utils.aws.s3_stream

Provides *S3Stream* which represents the data stream to and from S3 buckets.

1.1.1.1.1.12 Module Contents

1.1.1.1.1.13 Classes

S3Stream	*S3Stream* represents the data stream of a S3 resource, and provides static

e2fyi.utils.aws.s3_stream.**StringOrBytes**

class e2fyi.utils.aws.s3_stream.**S3Stream**(*stream: Union[IO, BinaryIO, io.BytesIO, io.StringIO]*, *content_type: str = 'application/octet-stream'*)
S3Stream represents the data stream of a S3 resource, and provides static methods to convert any python objects

into a stream. This is generally used with *S3Resource* to upload or download resource from S3 buckets.

Examples:

```
import io

import pandas as pd

from e2fyi.utils.aws import S3Stream
from pydantic import BaseModel

# create a s3 stream
stream = S3Stream(io.StringIO("random text"), "text/plain")
print(stream.read())         # prints "random text"
print(stream.content_type)   # prints "text/plain"

# string
stream = S3Stream.from_any("hello world")
print(stream.read())         # prints "hello world"
print(stream.content_type)   # prints "text/plain"

# dict
stream = S3Stream.from_any({"foo": "bar"})
print(stream.read())         # prints "{"foo": "bar"}"
print(stream.content_type)   # prints "application/json"

# pandas dataframe as csv
df = pd.DataFrame([{"key": "a", "value": 1}, {"key": "b", "value": 2}])
stream = S3Stream.from_any(df, index=False)  # do not include index column
print(stream.read())         # prints string as csv format for the dataframe
print(stream.content_type)   # prints "application/csv"

# pandas dataframe as json
stream = S3Stream.from_any(df, orient="records")  # orient dataframe as records
print(stream.read())         # prints string as json list for the dataframe
print(stream.content_type)   # prints "application/json"


# pydantic model
class Person(BaseModel):
    name: str
    age: int
stream = S3Stream.from_any(Person(name="william", age=21))
print(stream.read())         # prints "{"name": "william", "age": 21}"
print(stream.content_type)   # prints "application/json"


# any other python objects
class Pet:
    name: str
    age: int
stream = S3Stream.from_any(Pet(name="kopi", age=1))
print(stream.read())         # prints some binary bytes
print(stream.content_type)   # prints "application/octet-stream"
```

Creates a new S3Stream.

Args: stream (Union[IO, BinaryIO, io.BytesIO, io.StringIO]): any stream object. content_type (str, optional): mime type for the data in the stream.

Defaults to "application/octet-stream".

read (*self*, *size=-1*) → StringOrBytes
duck-typing for a readable stream.

seek (*self*, *offset: int*, *whence: int = 0*) → int
duck-typing for readable stream. See https://docs.python.org/3/library/io.html

Change the stream position to the given byte offset. offset is interpreted relative to the position indicated by whence. The default value for whence is SEEK_SET. Values for whence are:

SEEK_SET or 0 – start of the stream (the default); offset should be zero or positive

SEEK_CUR or 1 – current stream position; offset may be negative

SEEK_END or 2 – end of the stream; offset is usually negative

Return the new absolute position.

seekable (*self*) → bool
Whether if a stream is seekable

tell (*self*) → int
Return the current stream position.

close (*self*) → 'S3Stream'
Close the resource stream.

get_value (*self*) → StringOrBytes
Retrieve the entire contents of the S3Resource.

classmethod from_any (*cls*, *obj: Any*, *content_type: str = ''*, *output_as='csv'*, ***kwargs*) → 'S3Stream'
Returns a S3Stream from any python object.

Args: obj (Any): any python object. content_type (str, optional): mime type. Defaults to "". output_as (str, optional): format to output as if obj is a pandas object.

Defaults to "csv".

****kwargs: additional keyword arguments to pass to *pandas.to_csv***, *pandas.to_json*, *json.dumps* or *joblib.dumps* methods depending on the input object.

Returns: S3Stream: S3Stream object.

static from_file (*filepath: str*, *content_type: str = ''*) → 'S3Stream'
Returns a S3Stream from a file. If content_type is not provided, *python-magic* will be used to infer the mime type from the file data.

Args: filepath (str): path to the file. content_type (str, optional): mime type of the file. Defaults to "".

Returns: [type]: [description]

classmethod from_object (*cls, obj: Union[str, bytes, dict, BaseModel, object], content_type: str = 'application/octet-stream', **kwargs*) → 'S3Stream'
Returns a S3Stream from any string, bytes, dict, pydantic models, or any python object.

Dicts and pydantic models will be converted into a JSON string stream with *json.dumps* and the content type "application/json".

Anything that is not a string, bytes, dict, or pydantic model will be converted into a pickle binary stream with *joblib*.

Any extra keyword arguments will be passed to *json.dumps* or *joblib*.

See: https://joblib.readthedocs.io/en/latest/generated/joblib.dump.html#joblib.dump

Args: obj (Union[str, bytes, dict, pydantic.BaseModel, object]): [description] content_type (str, optional): [description]. Defaults to

"application/octet-stream".

****kwargs: Additional keyword arguments to pass to *joblib.dump* or** *json.dumps*.

Returns: S3Stream: S3Stream object.

static from_pandas(*df: Union[pd.DataFrame, pd.Series], output_as: str = 'csv', **kwargs: dict*) → 'S3Stream'

Returns a S3Stream object from a pandas dataframe or series. When output as a "csv", content type will be "application/csv", otherwise it will be "application/json".

Example:

```
import pandas

from e2fyi.utils.aws.s3_stream import S3Stream

# create some pandas dataframe
df = pd.DataFrame([...])

# create a csv stream, and don't output an index column.
csv_stream = S3Stream.from_pandas(df, index=False)

# create a json stream - output as records
json_stream = S3Stream.from_pandas(df, orient="records")
```

Args: df (Union[pd.DataFrame, pd.Series]): pandas dataframe or series. output_as (str, optional): either "csv" or "json". Defaults to "csv". **kwargs: additional keyword arguments to pass to either *pandas.to_csv*

or *pandas.to_json* methods.

Returns: S3Stream: S3Stream object.

static from_io(*stream: Union[IO, BinaryIO, io.StringIO, io.BytesIO], content_type: str = 'application/octet-stream'*) → 'S3Stream'

Returns a S3Stream object from an io stream.

Args:

stream (Union[IO, BinaryIO, io.StringIO, io.BytesIO]): any stream object.

content_type (str, optional): mime type of the stream. Defaults to "application/octet-stream".

Returns: S3Stream: S3Stream object.

1.1.1.1.1.14 e2fyi.utils.core

Core helpers.

Wrapping function with Result:

```
import logging

from e2fyi.utils.core import Result


def load_from_file(filepath: str) -> Result[string]:
    try:
        with open(filepath, "r") as fp:
            return Result(fp.read())
    except IOError as err:
        return Result(exception=err)

data = load_from_file("some_file.json")

# print with a default value fallback
print(data.with_default("default value"))

# print data if ok, else log exception
if data.is_ok:
    print(data)
else:
    logging.exception(data.exception)
```

1.1.1.1.1.15 Submodules

1.1.1.1.1.16 e2fyi.utils.core.maybe

This module provides a Maybe generic class to describe an unreliable output.

1.1.1.1.1.17 Module Contents

1.1.1.1.1.18 Classes

Maybe	Maybe is a generic class used to describe an output that can potentially

e2fyi.utils.core.maybe.**T**

class e2fyi.utils.core.maybe.**Maybe**(*value: Optional[T] = None*, *exception: BaseException = None*)

Maybe is a generic class used to describe an output that can potentially fails (e.g. raise exception).

Example:

```
import logging

from e2fyi.utils.core import Maybe
```

(continues on next page)

(continued from previous page)

```
def load_from_file(filepath: str) -> Maybe[string]:
    try:
        with open(filepath, "r") as fp:
            return Maybe(fp.read())
    except IOError as err:
        return Maybe(exception=err)

data = load_from_file("some_file.json")

# print with a default value fallback
print(data.with_default("default value"))

# print data if ok, else log exception
if data.is_ok:
    print(data)
else:
    logging.exception(data.exception)
```

Creates a new instance of Maybe. If an exception is provided, the Maybe value is considered to be not ok.

property is_ok (*self*) → bool
whether the output is generated successfully.

with_default (*self*, *default_value: T*) → T
returns a default value if the outputs is not generated successfully.

1.1.1.1.1.19 e2fyi.utils.core.results

This module is deprecated since v0.2.0.

1.1.1.1.1.20 Module Contents

1.1.1.1.1.21 Classes

Result	*Result* is deprecated since v0.2.0. Please use *e2fyi.utils.core.Maybe* instead.

e2fyi.utils.core.results.**T**

class e2fyi.utils.core.results.**Result** (*value: Optional[T] = None*, *exception: BaseException = None*)
Result is deprecated since v0.2.0. Please use *e2fyi.utils.core.Maybe* instead.

Deprecated class.

CHAPTER

TWO

E2FYI-UTILS

`e2fyi-utils` is an `e2fyi` namespaced python package with `utils` subpackage (i.e. `e2fyi.utils`) which holds a collections of useful helper classes and functions to interact with various cloud resources.

API documentation can be found at https://e2fyi-utils.readthedocs.io/en/latest/.

Change logs are available in CHANGELOG.md.

- Python 3.6 and above
- Licensed under Apache-2.0.

2.1 Quickstart

```
pip install e2fyi-utils>=0.2
```

2.1.1 S3Stream

`S3Stream` represents the data stream of a S3 resource, and provides static methods to convert any python objects into a stream. This is generally used with `S3Resource` to upload or download resource from S3 buckets.

NOTE:

- `str`, `float`, `int`, and `bool` will be saved as txt files with mime type "text/plain".
- `pydantic` models, `dict` or `list` will be saved as json files with mime type "application/json" (fallback to pickle if unable to serialize into json string).
- `pandas` dataframe or series can be saved as either a csv ("application/csv") or json format ("application/json").
- path to files will be read with `open` and mime type will be inferred (fallback to "application/octet-stream").
- all other python objects will be pickled with `joblib`.

```python
import io

import pandas as pd

from e2fyi.utils.aws import S3Stream
from pydantic import BaseModel

# create a s3 stream
stream = S3Stream(io.StringIO("random text"), "text/plain")
print(stream.read())         # prints "random text"
print(stream.content_type)   # prints "text/plain"

# string
stream = S3Stream.from_any("hello world")
print(stream.read())         # prints "hello world"
print(stream.content_type)   # prints "text/plain"

# dict
stream = S3Stream.from_any({"foo": "bar"})
print(stream.read())         # prints "{"foo": "bar"}"
print(stream.content_type)   # prints "application/json"

# pandas dataframe as csv
df = pd.DataFrame([{"key": "a", "value": 1}, {"key": "b", "value": 2}])
stream = S3Stream.from_any(df, index=False)  # do not include index column
print(stream.read())         # prints string as csv format for the dataframe
print(stream.content_type)   # prints "application/csv"

# pandas dataframe as json
stream = S3Stream.from_any(df, orient="records")  # orient dataframe as records
print(stream.read())         # prints string as json list for the dataframe
print(stream.content_type)   # prints "application/json"


# pydantic model
class Person(BaseModel):
    name: str
    age: int
stream = S3Stream.from_any(Person(name="william", age=21))
print(stream.read())         # prints "{"name": "william", "age": 21}"
print(stream.content_type)   # prints "application/json"


# any other python objects
class Pet:
    name: str
    age: int
stream = S3Stream.from_any(Pet(name="kopi", age=1))
print(stream.read())         # prints some binary bytes
print(stream.content_type)   # prints "application/octet-stream"
```

2.1.2 S3Resource

`S3Resource` represents a resource in S3 currently or a local resource that will be uploaded to S3. `S3Resource` constructor will automatically attempts to convert any inputs into a `S3Stream`, but for more granular control `S3Stream.from_any` should be used instead to create the `S3Stream`.

`S3Resource` is a readable stream - i.e. it has `read`, `seek`, and `close`.

> NOTE:
>
> See https://boto3.amazonaws.com/v1/documentation/api/latest/reference/customizations/s3.html for additional keyword arguments that can be passed to S3Resource.

2.1.2.1 Example: Creating S3Resource from local python object or file.

```
import boto3

from e2fyi.utils.aws import S3Resource, S3Stream

# create custom s3 client
s3client = boto3.client(
    's3',
    aws_access_key_id=ACCESS_KEY,
    aws_secret_access_key=SECRET_KEY
)

# creates a local copy of s3 resource with S3Stream from a local file
obj = S3Resource(
    # full path shld be "prefix/some_file.json"
    filename="some_file.json",
    prefix="prefix/",
    # bucket to download from or upload to
    bucketname="some_bucket",
    # or "s3n://" or "s3://"
    protocol="s3a://",
    # uses default client if not provided
    s3client=s3client,
    # attempts to convert to S3Stream if input is not a S3Stream
    stream=S3Stream.from_file("./some_path/some_file.json"),
    # addition kwarg to pass to `s3.upload_fileobj` or `s3.download_fileobj` methods
    Metadata={"label": "foo"}
)
print(obj.key)  # prints "prefix/some_file.json"
print(obj.uri)  # prints "s3a://some_bucket/prefix/some_file.json"

# will attempt to fix prefix and filename if incorrect filename is provided
obj = S3Resource(
    filename="subfolder/some_file.json",
    prefix="prefix/"
)
print(obj.filename)     # prints "some_file.json"
print(obj.prefix)       # prints "prefix/subfolder/"
```

2.1.2.2 Example: Upload S3Resource to S3

```
# creates a local copy of s3 resource with some python object
obj = S3Resource(
    filename="some_file.txt",
    prefix="prefix/",
    bucketname="some_bucket",
    stream={"some": "dict"},
)

# upload obj to s3 bucket "some_bucket" with the key "prefix/some_file.json"
# with the json string content.
obj.save()

# upload to s3 bucket "another_bucket" instead with a metadata tag.
obj.save("another_bucket", MetaData={"label": "foo"})
```

2.1.2.3 Example: Read S3Resource from S3

```
from pydantic import BaseModel

# do not provide a stream input to the S3Resource constructor
obj = S3Resource(
    filename="some_file.json",
    prefix="prefix/",
    bucketname="some_bucket",
    content_type="application/json"
)

# read the resource like a normal file object from S3
data = obj.read()
print(type(data))       # prints <class 'str'>

# read and load json string into a dict or list
# for content_type == "application/json" only
data_obj = obj.load()
print(type(data_obj))   # prints <class 'dict'> or <class 'list'>


# read and convert into a pydantic model
class Person(BaseModel):
    name: str
    age: int

# automatically unpack the dict
data_obj = obj.load(lambda name, age: Person(name=name, age=age))
# alternatively, do not unpack
data_obj = obj.load(lambda data: Person(**data), unpack=False)
print(type(data_obj))   # prints <class 'Person'>
```

2.1.3 S3Bucket

`S3Bucket` is an abstraction of the actual S3 bucket with methods to interact with the actual S3 bucket (e.g. list objects inside the bucket), and some utility methods.

Prefix rules can also be set during the creation of the `S3Bucket` object - i.e. enforce a particular prefix rules for a particular bucket.

2.1.3.1 Quickstart

```
from e2fyi.utils.aws import S3Bucket

# prints key for all resources with prefix "some_folder/"
for resource in S3Bucket("some_bucket").list("some_folder/"):
    print(resource.key)

# prints key for the first 2,000 resources with prefix "some_folder/"
for resource in S3Bucket("some_bucket").list("some_folder/", max_objects=2000):
    print(resource.key)

# creates a s3 bucket with prefix rule
prj_bucket = S3Bucket("some_bucket", get_prefix=lambda prefix: "prj-a/%s" % prefix)
for resource in prj_bucket.list("some_folder/"):
    print(resource.key)  # prints "prj-a/some_folder/<resource_name>"
    print(resource.stats)  # prints metadata for the resource (e.g. size, etag)

# get obj key in the bucket
print(prj_bucket.create_resource_key("foo.json"))  # prints "prj-a/foo.json"

# get obj uri in the bucket
# prints "s3a://some_bucket/prj-a/foo.json"
print(prj_bucket.create_resource_uri("foo.json", "s3a://"))

# create S3Resource in bucket to read in
foo = prj_bucket.create_resource("foo.json", "application/json")
# read "s3a://some_bucket/prj-a/foo.json" and load as a dict (or list)
foo_dict = foo.load()

# create S3Resource in bucket and save to "s3a://some_bucket/prj-a/foo.json"
prj_bucket.create_resource("foo.json", obj={"foo": "bar"}).save()
```

2.1.4 e2fyi.utils.core.Maybe

`Maybe` represents an uncertain object (exception might be raised so no value will be returned). This is generally used inside a function where all exceptions will be caught.

NOTE:

- `Maybe.value` is the actual returned value.
- `Maybe.exception` is the exception caught (if any).
- `Maybe.with_default` method can be used to provide a default value if no value is returned.
- `Maybe.is_ok` method can be used to check if any value is returned.

```
import logging

from e2fyi.utils.core import Maybe


def load_from_file(filepath: str) -> Maybe[string]:
    """loads the content of a file."""
    try:
        with open(filepath, "r") as fp:
            return Maybe(fp.read())
    except IOError as err:
        return Maybe(exception=err)

data = load_from_file("some_file.json")

# print with a default value fallback
print(data.with_default("default value"))

# print data if ok, else log exception
if data.is_ok:
    print(data)
else:
    logging.exception(data.exception)
```

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

e

INDEX

T

U

W