B2_Python_SDK

Release 1.7.0

Backblaze

Apr 22, 2021

CONTENTS

b2sdk is a client library for easy access to all of the capabilities of B2 Cloud Storage.

B2 command-line tool is an example of how it can be used to provide command-line access to the B2 service, but there are many possible applications (including FUSE filesystems, storage backend drivers for backup applications etc).

CHAPTER

ONE

WHY USE B2SDK?

When building an application which uses B2 cloud, it is possible to implement an independent B2 API client, but using **b2sdk** allows for:

- reuse of code that is already written, with hundreds of unit tests
- use of **Syncronizer**, a high-performance, parallel rsync-like utility
- developer-friendly library *api version policy* which guards your program against incompatible changes
- B2 integration checklist is passed automatically
- **raw_simulator** makes it easy to mock the B2 cloud for unit testing purposes
- reporting progress of operations to an object of your choice
- exception hierarchy makes it easy to display informative messages to users
- interrupted transfers are automatically continued
- **b2sdk** has been developed for 3 years before it version 1.0.0 was released. It's stable and mature.

CHAPTER

TWO

DOCUMENTATION INDEX

2.1 Installation Guide

2.1.1 Installing as a dependency

b2sdk can simply be added to `requirements.txt` (or equivalent such as `setup.py`, `.pipfile` etc). In order to properly set a dependency, see *versioning chapter* for details.

Note: The stability of your application depends on correct *pinning of versions*.

2.1.2 Installing a development version

To install **b2sdk**, checkout the repository and run:

```
pip install b2sdk
```

in your python environment.

2.2 Tutorial

2.2.1 AccountInfo

`AccountInfo` object holds information about access keys, tokens, upload urls, as well as a bucket id-name map.

It is the first object that you need to create to use **b2sdk**. Using `AccountInfo`, we'll be able to create a `B2Api` object to manage a B2 account.

In the tutorial we will use *`b2sdk.v1.InMemoryAccountInfo`*:

```
>>> from b2sdk.v1 import InMemoryAccountInfo
>>> info = InMemoryAccountInfo()  # store credentials, tokens and cache in memory
```

With the `info` object in hand, we can now proceed to create a `B2Api` object.

Note: *AccountInfo* section provides guidance for choosing the correct `AccountInfo` class for your application.

2.2.2 Account authorization

```
>>> from b2sdk.v1 import B2Api
>>> b2_api = B2Api(info)
>>> application_key_id = '4a5b6c7d8e9f'
>>> application_key = '001b8e23c26ff6efb941e237deb182b9599a84bef7'
>>> b2_api.authorize_account("production", application_key_id, application_key)
```

Tip: Get credentials from B2 website

To find out more about account authorization, see *b2sdk.v1.B2Api.authorize_account()*

2.2.3 B2Api

B2Api allows for account-level operations on a B2 account.

Typical B2Api operations

authorize_account	Perform account authorization.
create_bucket	Create a bucket.
delete_bucket	Delete a chosen bucket.
list_buckets	Call `b2_list_buckets` and return a list of buckets.
get_bucket_by_name	Return the Bucket matching the given bucket_name.
create_key	Create a new *application key*.
list_keys	List application keys.
delete_key	Delete *application key*.
download_file_by_id	Download a file with the given ID.
list_parts	Generator that yields a *b2sdk.v1.Part* for each of the parts that have been uploaded.
cancel_large_file	Cancel a large file upload.

```
>>> b2_api = B2Api(info)
```

to find out more, see *b2sdk.v1.B2Api*.

The most practical operation on `B2Api` object is *b2sdk.v1.B2Api.get_bucket_by_name()*.

Bucket allows for operations such as listing a remote bucket or transferring files.

2.2.4 Bucket

Initializing a Bucket

Retrieve an existing Bucket

To get a `Bucket` object for an existing B2 Bucket:

```
>>> b2_api.get_bucket_by_name("example-mybucket-b2-1",)
Bucket<346501784642eb3e60980d10,example-mybucket-b2-1,allPublic>
```

Create a new Bucket

To create a bucket:

```
>>> bucket_name = 'example-mybucket-b2-1'
>>> bucket_type = 'allPublic'  # or 'allPrivate'

>>> b2_api.create_bucket(bucket_name, bucket_type)
Bucket<346501784642eb3e60980d10,example-mybucket-b2-1,allPublic>
```

You can optionally store bucket info, CORS rules and lifecycle rules with the bucket. See *b2sdk.v1.B2Api.create_bucket()* for more details.

Note: Bucket name must be unique in B2 (across all accounts!). Your application should be able to cope with a bucket name collision with another B2 user.

Typical Bucket operations

download_file_by_name	Download a file by name.
upload_local_file	Upload a file on local disk to a B2 file.
upload_bytes	Upload bytes in memory to a B2 file.
ls	Pretend that folders exist and yields the information about the files in a folder.
hide_file	Hide a file.
delete_file_version	Delete a file version.
get_download_authorization	Return an authorization token that is valid only for downloading files from the given bucket.
get_download_url	Get file download URL.
update	Update various bucket parameters.
set_type	Update bucket type.
set_info	Update bucket info.

To find out more, see *b2sdk.v1.Bucket*.

2.2.5 Summary

You now know how to use `AccountInfo`, `B2Api` and `Bucket` objects.

To see examples of some of the methods presented above, visit the *quick start guide* section.

2.3 Quick Start Guide

2.3.1 Prepare b2sdk

```
>>> from b2sdk.v1 import *
>>> info = InMemoryAccountInfo()
>>> b2_api = B2Api(info)
>>> application_key_id = '4a5b6c7d8e9f'
>>> application_key = '001b8e23c26ff6efb941e237deb182b9599a84bef7'
>>> b2_api.authorize_account("production", application_key_id, application_key)
```

Tip: Get credentials from B2 website

2.3.2 Synchronization

```
>>> from b2sdk.v1 import ScanPoliciesManager
>>> from b2sdk.v1 import parse_sync_folder
>>> from b2sdk.v1 import Synchronizer
>>> from b2sdk.v1 import SyncReport
>>> import time
>>> import sys

>>> source = '/home/user1/b2_example'
>>> destination = 'b2://example-mybucket-b2'

>>> source = parse_sync_folder(source, b2_api)
>>> destination = parse_sync_folder(destination, b2_api)

>>> policies_manager = ScanPoliciesManager(exclude_all_symlinks=True)

>>> synchronizer = Synchronizer(
        max_workers=10,
        policies_manager=policies_manager,
        dry_run=False,
        allow_empty_source=True,
    )

>>> no_progress = False
>>> encryption_settings_provider = BasicSyncEncryptionSettingsProvider({
        'bucket1': EncryptionSettings(mode=EncryptionMode.SSE_B2),
        'bucket2': EncryptionSettings(
                        mode=EncryptionMode.SSE_C,
                        key=EncryptionKey(secret=b'VkYp3s6v9y$B&E)H@McQfTjWmZq4t7w!',␣
↪id='user-generated-key-id')
                    ),
        'bucket3': None,
    })
>>> with SyncReport(sys.stdout, no_progress) as reporter:
        synchronizer.sync_folders(
            source_folder=source,
            dest_folder=destination,
            now_millis=int(round(time.time() * 1000)),
```

(continues on next page)

(continued from previous page)

```
            reporter=reporter,
            encryption_settings_provider=encryption_settings_provider,
        )
upload some.pdf
upload som2.pdf
```

Tip: Sync is the preferred way of getting data into and out of B2 cloud, because it can achieve *highest performance* due to parallelization of scanning and data transfer operations.

To learn more about sync, see *Synchronizer*.

Sync uses an encryption provider. In principle, it's a mapping between file metadata (bucket_name, file_info, etc) and *EncryptionSetting*. The reason for employing such a mapping, rather than a single *EncryptionSetting*, is the fact that users of Sync do not necessarily know up front what files it's going to upload and download. This approach enables using unique keys, or key identifiers, across files. This is covered in greater detail in *Server-Side Encryption*.

In the example above, Sync will assume *SSE-B2* for all files in *bucket1*, *SSE-C* with the key provided for *bucket2* and rely on bucket default for *bucket3*. Should developers need to provide keys per file (and not per bucket), they need to implement their own `b2sdk.v1.AbstractSyncEncryptionSettingsProvider`.

2.3.3 Bucket actions

List buckets

```
>>> b2_api.list_buckets()
[Bucket<346501784642eb3e60980d10,example-mybucket-b2-1,allPublic>]
>>> for b in b2_api.list_buckets():
        print('%s  %-10s  %s' % (b.id_, b.type_, b.name))
346501784642eb3e60980d10  allPublic   example-mybucket-b2-1
```

Create a bucket

```
>>> bucket_name = 'example-mybucket-b2-1'  # must be unique in B2 (across all␣
↪accounts!)
>>> bucket_type = 'allPublic'  # or 'allPrivate'

>>> b2_api.create_bucket(bucket_name, bucket_type)
Bucket<346501784642eb3e60980d10,example-mybucket-b2-1,allPublic>
```

You can optionally store bucket info, CORS rules and lifecycle rules with the bucket. See *b2sdk.v1.B2Api.create_bucket()*.

Delete a bucket

```
>>> bucket_name = 'example-mybucket-b2-to-delete'
>>> bucket = b2_api.get_bucket_by_name(bucket_name)
>>> b2_api.delete_bucket(bucket)
```

returns *None* if successful, raises an exception in case of error.

Update bucket info

```
>>> new_bucket_type = 'allPrivate'
>>> bucket_name = 'example-mybucket-b2'

>>> bucket = b2_api.get_bucket_by_name(bucket_name)
>>> bucket.update(bucket_type=new_bucket_type,
                  default_server_side_
↪encryption=EncryptionSetting(mode=EncryptionMode.SSE_B2))
{'accountId': '451862be08d0',
 'bucketId': '5485a1682662eb3e60980d10',
 'bucketInfo': {},
 'bucketName': 'example-mybucket-b2',
 'bucketType': 'allPrivate',
 'corsRules': [],
 'lifecycleRules': [],
 'revision': 3,
 'defaultServerSideEncryption': {'isClientAuthorizedToRead': True,
                                 'value': {'algorithm': 'AES256', 'mode': 'SSE-B2'}}},
 }
```

For more information see *b2sdk.v1.Bucket.update()*.

2.3.4 File actions

Tip: Sync is the preferred way of getting files into and out of B2 cloud, because it can achieve *highest performance* due to parallelization of scanning and data transfer operations.

To learn more about sync, see *Synchronizer*.

Use the functions described below only if you *really* need to transfer a single file.

Upload file

```
>>> local_file_path = '/home/user1/b2_example/new.pdf'
>>> b2_file_name = 'dummy_new.pdf'
>>> file_info = {'how': 'good-file'}

>>> bucket = b2_api.get_bucket_by_name(bucket_name)
>>> bucket.upload_local_file(
        local_file=local_file_path,
        file_name=b2_file_name,
        file_infos=file_info,
```

(continues on next page)

(continued from previous page)

```
    )
<b2sdk.file_version.FileVersionInfo at 0x7fc8cd560550>
```

This will work regardless of the size of the file - `upload_local_file` automatically uses large file upload API when necessary.

For more information see *b2sdk.v1.Bucket.upload_local_file()*.

Upload file encrypted with SSE-C

```
>>> local_file_path = '/home/user1/b2_example/new.pdf'
>>> b2_file_name = 'dummy_new.pdf'
>>> file_info = {'how': 'good-file'}
>>> encryption_setting = EncryptionSetting(
        mode=EncryptionMode.SSE_C,
        key=EncryptionKey(secret=b'VkYp3s6v9y$B&E)H@McQfTjWmZq4t7w!', id='user-
↪generated-key-id'),
    )

>>> bucket = b2_api.get_bucket_by_name(bucket_name)
>>> bucket.upload_local_file(
        local_file=local_file_path,
        file_name=b2_file_name,
        file_infos=file_info,
        encryption=encryption_setting,
    )
```

Download file

By id

```
>>> from b2sdk.v1 import DownloadDestLocalFile
>>> from b2sdk.v1 import DoNothingProgressListener

>>> local_file_path = '/home/user1/b2_example/new2.pdf'
>>> file_id = '4_z5485a1682662eb3e60980d10_f1195145f42952533_d20190403_m130258_c002_
↪v0001111_t0002'
>>> download_dest = DownloadDestLocalFile(local_file_path)
>>> progress_listener = DoNothingProgressListener()

>>> b2_api.download_file_by_id(file_id, download_dest, progress_listener)
{'fileId': '4_z5485a1682662eb3e60980d10_f1195145f42952533_d20190403_m130258_c002_
↪v0001111_t0002',
 'fileName': 'som2.pdf',
 'contentType': 'application/pdf',
 'contentLength': 1870579,
 'contentSha1': 'd821849a70922e87c2b0786c0be7266b89d87df0',
 'fileInfo': {'src_last_modified_millis': '1550988084299'}}

>>> print('File name:   ', download_dest.file_name)
File name:    som2.pdf
>>> print('File id:     ', download_dest.file_id)
File id:      4_z5485a1682662eb3e60980d10_f1195145f42952533_d20190403_m130258_c002_
↪v0001111_t0002
```

(continues on next page)

(continued from previous page)

```
>>> print('File size:    ', download_dest.content_length)
File size:    1870579
>>> print('Content type:', download_dest.content_type)
Content type: application/pdf
>>> print('Content sha1:', download_dest.content_sha1)
Content sha1: d821849a70922e87c2b0786c0be7266b89d87df0
```

By name

```
>>> bucket = b2_api.get_bucket_by_name(bucket_name)
>>> b2_file_name = 'dummy_new.pdf'
>>> local_file_name = '/home/user1/b2_example/new3.pdf'
>>> download_dest = DownloadDestLocalFile(local_file_name)
>>> bucket.download_file_by_name(b2_file_name, download_dest)
{'fileId': '4_z5485a1682662eb3e60980d10_f113f963288e711a6_d20190404_m065910_c002_
↪v0001095_t0044',
 'fileName': 'dummy_new.pdf',
 'contentType': 'application/pdf',
 'contentLength': 1870579,
 'contentSha1': 'd821849a70922e87c2b0786c0be7266b89d87df0',
 'fileInfo': {'how': 'good-file'}}
```

Downloading encrypted files

Both methods (*By name* and *By id*) accept an optional *encryption* argument, similarly to *Upload file*. This parameter is necessary for downloading files encrypted with *SSE-C*.

List files

```
>>> bucket_name = 'example-mybucket-b2'
>>> bucket = b2_api.get_bucket_by_name(bucket_name)
>>> for file_info, folder_name in bucket.ls(show_versions=False):
>>>     print(file_info.file_name, file_info.upload_timestamp, folder_name)
f2.txt 1560927489000 None
som2.pdf 1554296578000 None
some.pdf 1554296579000 None
test-folder/.bzEmpty 1561005295000 test-folder/

# Recursive
>>> bucket_name = 'example-mybucket-b2'
>>> bucket = b2_api.get_bucket_by_name(bucket_name)
>>> for file_info, folder_name in bucket.ls(show_versions=False, recursive=True):
>>>     print(file_info.file_name, file_info.upload_timestamp, folder_name)
f2.txt 1560927489000 None
som2.pdf 1554296578000 None
some.pdf 1554296579000 None
test-folder/.bzEmpty 1561005295000 test-folder/
test-folder/folder_file.txt 1561005349000 None
```

Note: The files are returned recursively and in order so all files in a folder are printed one after another. The folder_name is returned only for the first file in the folder.

```
# Within folder
>>> bucket_name = 'example-mybucket-b2'
>>> bucket = b2_api.get_bucket_by_name(bucket_name)
>>> for file_info, folder_name in bucket.ls(folder_to_list='test-folder', show_
↪versions=False):
>>>     print(file_info.file_name, file_info.upload_timestamp, folder_name)
test-folder/.bzEmpty 1561005295000 None
test-folder/folder_file.txt 1561005349000 None

# list file versions
>>> for file_info, folder_name in bucket.ls(show_versions=True):
>>>     print(file_info.file_name, file_info.upload_timestamp, folder_name)
f2.txt 1560927489000 None
f2.txt 1560849524000 None
som2.pdf 1554296578000 None
some.pdf 1554296579000 None
```

For more information see *b2sdk.v1.Bucket.ls()*.

Get file metadata

```
>>> file_id = '4_z5485a1682662eb3e60980d10_f113f963288e711a6_d20190404_m065910_c002_
↪v0001095_t0044'
>>> b2_api.get_file_info(file_id)
{'accountId': '451862be08d0',
 'action': 'upload',
 'bucketId': '5485a1682662eb3e60980d10',
 'contentLength': 1870579,
 'contentSha1': 'd821849a70922e87c2b0786c0be7266b89d87df0',
 'contentType': 'application/pdf',
 'fileId': '4_z5485a1682662eb3e60980d10_f113f963288e711a6_d20190404_m065910_c002_
↪v0001095_t0044',
 'fileInfo': {'how': 'good-file', 'sse_c_key_id': 'user-generated-key-id'},
 'fileName': 'dummy_new.pdf',
 'uploadTimestamp': 1554361150000,
 "serverSideEncryption": {"algorithm": "AES256",
                          "mode": "SSE-C"},
 }
```

Copy file

Please switch to *b2sdk.v1.Bucket.copy()*.

```
>>> file_id = '4_z5485a1682662eb3e60980d10_f118df9ba2c5131e8_d20190619_m065809_c002_
↪v0001126_t0040'
>>> bucket.copy(file_id, 'f2_copy.txt')
{'accountId': '451862be08d0',
 'action': 'copy',
 'bucketId': '5485a1682662eb3e60980d10',
 'contentLength': 124,
 'contentSha1': '737637702a0e41dda8b7be79c8db1d369c6eef4a',
 'contentType': 'text/plain',
 'fileId': '4_z5485a1682662eb3e60980d10_f1022e2320daf707f_d20190620_m122848_c002_
↪v0001123_t0020',
```

(continues on next page)

(continued from previous page)

```
 'fileInfo': {'src_last_modified_millis': '1560848707000'},
 'fileName': 'f2_copy.txt',
 'uploadTimestamp': 1561033728000,
 "serverSideEncryption": {"algorithm": "AES256",
                          "mode": "SSE-B2"}}
```

If the `content length` is not provided and the file is larger than 5GB, `copy` would not succeed and error would be raised. If length is provided, then the file may be copied as a large file. Maximum copy part size can be set by `max_copy_part_size` - if not set, it will default to 5GB. If `max_copy_part_size` is lower than *absoluteMinimumPartSize*, file would be copied in single request - this may be used to force copy in single request large file that fits in server small file limit.

Copying files allows for providing encryption settings for both source and destination files - *SSE-C* encrypted source files cannot be used unless the proper key is provided.

If you want to copy just the part of the file, then you can specify the offset and content length:

```
>>> file_id = '4_z5485a1682662eb3e60980d10_f118df9ba2c5131e8_d20190619_m065809_c002_
↪v0001126_t0040'
>>> bucket.copy(file_id, 'f2_copy.txt', offset=1024, length=2048)
```

Note that content length is required for offset values other than zero.

For more information see *b2sdk.v1.Bucket.copy()*.

Delete file

```
>>> file_id = '4_z5485a1682662eb3e60980d10_f113f963288e711a6_d20190404_m065910_c002_
↪v0001095_t0044'
>>> file_info = b2_api.delete_file_version(file_id, 'dummy_new.pdf')
>>> print(file_info)
{'file_id': '4_z5485a1682662eb3e60980d10_f113f963288e711a6_d20190404_m065910_c002_
↪v0001095_t0044',
 'file_name': 'dummy_new.pdf'}
```

Cancel large file uploads

```
>>> bucket = b2_api.get_bucket_by_name(bucket_name)
>>> for file_version in bucket.list_unfinished_large_files():
        bucket.cancel_large_file(file_version.file_id)
```

2.4 Server-Side Encryption

2.4.1 Cloud

B2 cloud supports Server-Side Encryption. All read and write operations provided by **b2sdk** accept encryption settings as an optional argument. Not supplying this argument means relying on bucket defaults - for **SSE-B2** and for no encryption. In case of **SSE-C**, providing a decryption key is required for successful downloading and copying.

2.4.2 API

Methods and classes that require encryption settings all accept an *EncryptionSetting* object, which holds information about present or desired encryption (mode, algorithm, key, key_id). Some, like copy operations, accept two sets of settings (for source and for destination). Sync, however, accepts an *EncryptionSettingsProvider* object, which is an *EncryptionSetting* factory, yielding them based on file metadata. For details, see

- *b2sdk.encryption.setting*
- *b2sdk.encryption.types*
- *b2sdk.sync.encryption_provider*

2.4.3 High security: use unique keys

B2 cloud does not promote or discourage either reusing encryption keys or using unique keys for *SEE-C*. In applications requiring enhanced security, using unique key per file is a good strategy. **b2sdk** follows a convention, that makes managing such keys easier: *EncryptionSetting* holds a key identifier, aside from the key itself. This key identifier is saved in the metadata of all files uploaded, created or copied via **b2sdk** methods using *SSE-C*, under *sse_c_key_id* in *fileInfo*. This allows developers to create key managers that map those ids to keys, stored securely in a file or a database. Implementing such managers, and linking them to `b2sdk.v1.AbstractSyncEncryptionSettingsProvider` implementations (necessary for using Sync) is outside of the scope of this library.

There is, however, a convention to such managers that authors of this library strongly suggest: if a manager needs to generate a new key-key_id pair for uploading, it's best to commit this data to the underlying storage before commencing the upload. The justification of such approach is: should the key-key_id pair be committed to permanent storage after completing an IO operation, committing could fail after successfully upload the data. This data, however, is now just a random blob, that can never be read, since the key to decrypting it is lost.

This approach comes an overhead: to download a file, its *fileInfo* has to be known. This means that fetching metadata is required before downloading.

2.4.4 Moderate security: a single key with many ids

Should the application's infrastructure require a single key (or a limited set of keys) to be used in a bucket, authors of this library recommend generating a unique key identifier for each file anyway (even though these unique identifiers all point to the same key value). This obfuscates confidential metadata from malicious users, like which files are encrypted with the same key, the total number of different keys, etc.

2.5 Advanced usage patterns

B2 server API allows for creation of an object from existing objects. This allows to avoid transferring data from the source machine if the desired outcome can be (at least partially) constructed from what is already on the server.

The way **b2sdk** exposes this functionality is through a few functions that allow the user to express the desired outcome and then the library takes care of planning and executing the work. Please refer to the table below to compare the support of object creation methods for various usage patterns.

2.5.1 Available methods

Method / supported options	Source	Range overlap	Streaming interface	*Continuation*
b2sdk.v1.Bucket.upload()	local	no	no	automatic
b2sdk.v1.Bucket.copy()	remote	no	no	automatic
b2sdk.v1.Bucket.concatenate()	any	no	no	automatic
b2sdk.v1.Bucket.concatenate_stream()	any	no	yes	manual
b2sdk.v1.Bucket.create_file()	any	yes	no	automatic
b2sdk.v1.Bucket.create_file_stream()	any	yes	yes	manual

Range overlap

Some methods support overlapping ranges between local and remote files. **b2sdk** tries to use the remote ranges as much as possible, but due to limitations of `b2_copy_part` (specifically the minimum size of a part) that may not be always possible. A possible solution for such case is to download a (small) range and then upload it along with another one, to meet the `b2_copy_part` requirements. This can be improved if the same data is already available locally - in such case **b2sdk** will use the local range rather than downloading it.

Streaming interface

Some object creation methods start writing data before reading the whole input (iterator). This can be used to write objects that do not have fully known contents without writing them first locally, so that they could be copied. Such usage pattern can be relevant to small devices which stream data to B2 from an external NAS, where caching large files such as media files or virtual machine images is not an option.

Please see *advanced method support table* to see where streaming interface is supported.

Continuation

Please see *here*

2.5.2 Concatenate files

b2sdk.v1.Bucket.concatenate() accepts an iterable of upload sources (either local or remote). It can be used to glue remote files together, back-to-back, into a new file.

b2sdk.v1.Bucket.concatenate_stream() does not create and validate a plan before starting the transfer, so it can be used to process a large input iterator, at a cost of limited automated continuation.

Concatenate files of known size

```
>>> bucket = b2_api.get_bucket_by_name(bucket_name)
>>> input_sources = [
...     CopySource('4_z5485a1682662eb3e60980d10_f113f963288e711a6_d20190404_m065910_
↪c002_v0001095_t0044', offset=100, length=100),
...     UploadSourceLocalFile('my_local_path/to_file.txt'),
...     CopySource('4_z5485a1682662eb3e60980d10_f1022e2320daf707f_d20190620_m122848_
↪c002_v0001123_t0020', length=2123456789),
... ]
>>> file_info = {'how': 'good-file'}
>>> bucket.concatenate(input_sources, remote_name, file_info)
<b2sdk.file_version.FileVersionInfo at 0x7fc8cd560551>
```

If one of remote source has length smaller than *absoluteMinimumPartSize* then it cannot be copied into large file part. Such remote source would be downloaded and concatenated locally with local source or with other downloaded remote source.

Please note that this method only allows checksum verification for local upload sources. Checksum verification for remote sources is available only when local copy is available. In such case *b2sdk.v1.Bucket.create_file()* can be used with overalapping ranges in input.

For more information about `concatenate` please see *b2sdk.v1.Bucket.concatenate()* and `b2sdk.v1.CopySource`.

Concatenate files of known size (streamed version)

b2sdk.v1.Bucket.concatenate() accepts an iterable of upload sources (either local or remote). The operation would not be planned ahead so it supports very large output objects, but continuation is only possible for local only sources and provided unfinished large file id. See more about continuation in *b2sdk.v1.Bucket.create_file()* paragraph about continuation.

```
>>> bucket = b2_api.get_bucket_by_name(bucket_name)
>>> input_sources = [
...     CopySource('4_z5485a1682662eb3e60980d10_f113f963288e711a6_d20190404_m065910_
↪c002_v0001095_t0044', offset=100, length=100),
...     UploadSourceLocalFile('my_local_path/to_file.txt'),
...     CopySource('4_z5485a1682662eb3e60980d10_f1022e2320daf707f_d20190620_m122848_
↪c002_v0001123_t0020', length=2123456789),
... ]
>>> file_info = {'how': 'good-file'}
>>> bucket.concatenate_stream(input_sources, remote_name, file_info)
<b2sdk.file_version.FileVersionInfo at 0x7fc8cd560551>
```

Concatenate files of unknown size

While it is supported by B2 server, this pattern is currently not supported by **b2sdk**.

2.5.3 Synthethize an object

Using methods described below an object can be created from both local and remote sources while avoiding downloading small ranges when such range is already present on a local drive.

Update a file efficiently

b2sdk.v1.Bucket.create_file() accepts an iterable which *can contain overlapping destination ranges*.

Note: Following examples *create* new file - data in bucket is immutable, but **b2sdk** can create a new file version with the same name and updated content

---

Append to the end of a file

The assumption here is that the file has been appended to since it was last uploaded to. This assumption is verified by **b2sdk** when possible by recalculating checksums of the overlapping remote and local ranges. If copied remote part sha does not match with locally available source, file creation process would be interrupted and an exception would be raised.

```
>>> bucket = b2_api.get_bucket_by_name(bucket_name)
>>> input_sources = [
...     WriteIntent(
...         data=CopySource(
...             '4_z5485a1682662eb3e60980d10_f113f963288e711a6_d20190404_m065910_c002_
↪v0001095_t0044',
...             offset=0,
...             length=5000000,
...         ),
...         destination_offset=0,
...     ),
...     WriteIntent(
...         data=UploadSourceLocalFile('my_local_path/to_file.txt'),  # of length␣
↪60000000
...         destination_offset=0,
...     ),
... ]
>>> file_info = {'how': 'good-file'}
>>> bucket.create_file(input_sources, remote_name, file_info)
<b2sdk.file_version.FileVersionInfo at 0x7fc8cd560552>
```

LocalUploadSource has the size determined automatically in this case. This is more efficient than *b2sdk.v1.Bucket.concatenate()*, as it can use the overlapping ranges when a remote part is smaller than *absoluteMinimumPartSize* to prevent downloading a range (when concatenating, local source would have destination offset at the end of remote source)

For more information see *b2sdk.v1.Bucket.create_file()*.

Change the middle of the remote file

```
>>> bucket = b2_api.get_bucket_by_name(bucket_name)
>>> input_sources = [
...     WriteIntent(
...         CopySource('4_z5485a1682662eb3e60980d10_f113f963288e711a6_d20190404_
↪m065910_c002_v0001095_t0044', offset=0, length=4000000),
...         destination_offset=0,
...     ),
...     WriteIntent(
...         UploadSourceLocalFile('my_local_path/to_file.txt'),  # length=1024, here␣
↪not passed and just checked from local source using seek
...         destination_offset=4000000,
...     ),
...     WriteIntent(
...         CopySource('4_z5485a1682662eb3e60980d10_f113f963288e711a6_d20190404_
↪m065910_c002_v0001095_t0044', offset=4001024, length=123456789),
...         destination_offset=4001024,
...     ),
... ]
>>> file_info = {'how': 'good-file'}
>>> bucket.create_file(input_sources, remote_name, file_info)
<b2sdk.file_version.FileVersionInfo at 0x7fc8cd560552>
```

LocalUploadSource has the size determined automatically in this case. This is more efficient than *b2sdk.v1.Bucket.concatenate()*, as it can use the overlapping ranges when a remote part is smaller than *absoluteMinimumPartSize* to prevent downloading a range.

For more information see *b2sdk.v1.Bucket.create_file()*.

Synthetize a file from local and remote parts

This is useful for expert usage patterns such as:

- *synthetic backup*
- *reverse synthetic backup*
- mostly-server-side cutting and gluing uncompressed media files such as *wav* and *avi* with rewriting of file headers
- various deduplicated backup scenarios

Please note that *b2sdk.v1.Bucket.create_file_stream()* accepts **an ordered iterable** which *can contain overlapping ranges*, so the operation does not need to be planned ahead, but can be streamed, which supports very large output objects.

Scenarios such as below are then possible:

```
A           C        D            G
|           |        |            |
| cloud-AC  |        | cloud-DG   |
|           |        |            |
v           v        v            v
#############        #############
^                                 ^
|                                 |
+---- desired file A-G --------+
```

(continues on next page)

(continued from previous page)

```
|                                |
|                                |
|    ########################    |
|    ^                      ^    |
|    |                      |    |
|    |      local file-BF   |    |
|    |                      |    |
A    B     C       D       E F G
```

```
>>> bucket = b2_api.get_bucket_by_name(bucket_name)
>>> def generate_input():
...     yield WriteIntent(
...         CopySource('4_z5485a1682662eb3e60980d10_f113f963288e711a6_d20190404_
↪m065910_c002_v0001095_t0044', offset=0, length=lengthC),
...         destination_offset=0,
...     )
...     yield WriteIntent(
...         UploadSourceLocalFile('my_local_path/to_file.txt'), # length = offsetF -␣
↪offsetB
...         destination_offset=offsetB,
...     )
...     yield WriteIntent(
...         CopySource('4_z5485a1682662eb3e60980d10_f113f963288e711a6_d20190404_
↪m065910_c002_v0001095_t0044', offset=0, length=offsetG-offsetD),
...         destination_offset=offsetD,
...     )
...
>>> file_info = {'how': 'good-file'}
>>> bucket.create_file(generate_input(), remote_name, file_info)
<b2sdk.file_version.FileVersionInfo at 0x7fc8cd560552>
```

In such case, if the sizes allow for it (there would be no parts smaller than *absoluteMinimumPartSize*), the only uploaded part will be *C-D*. Otherwise, more data will be uploaded, but the data transfer will be reduced in most cases. *b2sdk.v1.Bucket.create_file()* does not guarantee that outbound transfer usage would be optimal, it uses a simple greedy algorithm with as small look-aheads as possible.

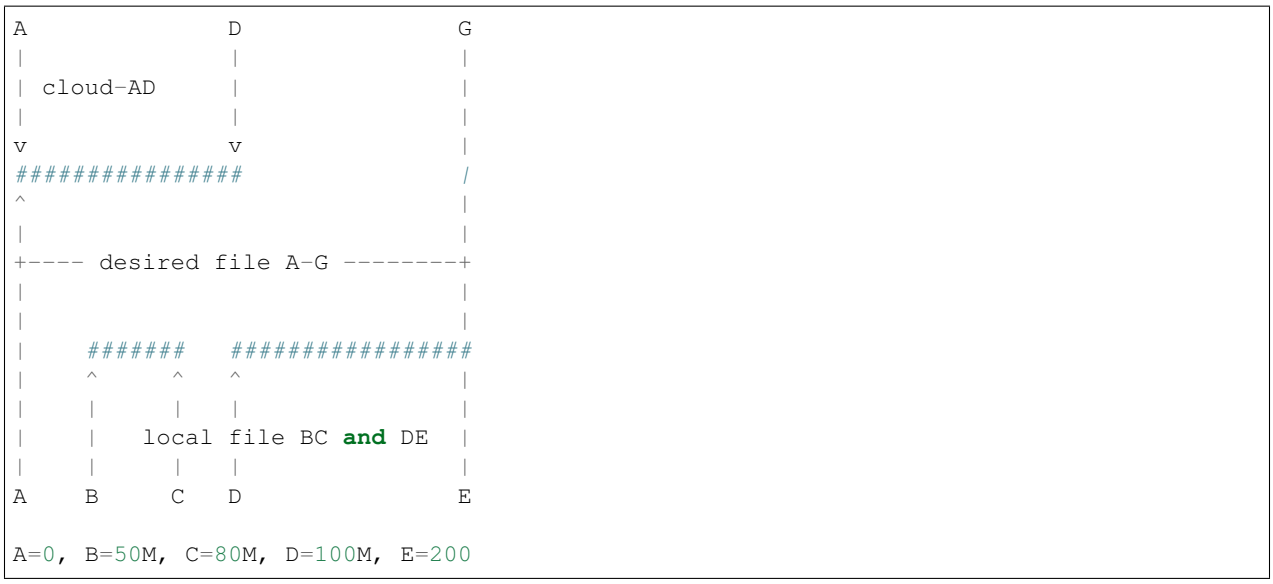
For more information see *b2sdk.v1.Bucket.create_file()*.

Encryption

Even if files *A-C* and *D-G* are encrypted using *SSE-C* with different keys, they can still be used in a single *b2sdk.v1.Bucket.create_file()* call, because `b2sdk.v1.CopySource` accepts an optional `b2sdk.v1.EncryptionSetting`.

Prioritize remote or local sources

b2sdk.v1.Bucket.create_file() and *b2sdk.v1.Bucket.create_file_stream()* support source/origin prioritization, so that planner would know which sources should be used for overlapping ranges. Supported values are: *local*, *remote* and *local_verification*.




```
>>> bucket.create_file(input_sources, remote_name, file_info, prioritize='local')
# planner parts: cloud[A, B], local[B, C], remote[C, D], local[D, E]
```

Here the planner has only used a remote source where remote range was not available, minimizing downloads.

```
>>> planner.create_file(input_sources, remote_name, file_info, prioritize='remote')
# planner parts: cloud[A, D], local[D, E]
```

Here the planner has only used a local source where remote range was not available, minimizing uploads.

```
>>> bucket.create_file(input_sources, remote_name, file_info)
# or
>>> bucket.create_file(input_sources, remote_name, file_info, prioritize='local_
↪verification')
# planner parts: cloud[A, B], cloud[B, C], cloud[C, D], local[D, E]
```

In *local_verification* mode the remote range was artificially split into three parts to allow for checksum verification against matching local ranges.

Note: *prioritize* is just a planner setting - remote parts are always verified if matching local parts exists.

2.5.4 Continuation

Continuation of upload

In order to continue a simple upload session, **b2sdk** checks for any available sessions with of the same `file name`, `file_info` and `media type`, verifying the size of an object as much as possible.

To support automatic continuation, some advanced methods create a plan before starting copy/upload operations, saving the hash of that plan in `file_info` for increased reliability.

If that is not available, `large_file_id` can be extracted via callback during the operation start. It can then be passed into the subsequent call to continue the same task, though the responsibility for passing the exact same input is then on the user of the function. Please see *advanced method support table* to see where automatic continuation is supported. `large_file_id` can also be passed if automatic continuation is available in order to avoid issues where multiple matchin upload sessions are matching the transfer.

Continuation of create/concantenate

b2sdk.v1.Bucket.create_file() supports automatic continuation or manual continuation. *b2sdk.v1.Bucket.create_file_stream()* supports only manual continuation for local-only inputs. The situation looks the same for *b2sdk.v1.Bucket.concatenate()* and *b2sdk.v1.Bucket.concatenate_stream()* (streamed version supports only manual continuation of local sources). Also *b2sdk.v1.Bucket.upload()* and `b2sdk.v2.Bucket.copy()` support both automatic and manual continuation.

Manual continuation

```
>>> def large_file_callback(large_file):
...     # storage is not part of the interface - here only for demonstration purposes
...     storage.store({'name': remote_name, 'large_file_id': large_file.id})
>>> bucket.create_file(input_sources, remote_name, file_info, large_file_
↪callback=large_file_callback)
# ...
>>> large_file_id = storage.query({'name': remote_name})[0]['large_file_id']
>>> bucket.create_file(input_sources, remote_name, file_info, large_file_id=large_
↪file_id)
```

Manual continuation (streamed version)

```
>>> def large_file_callback(large_file):
...     # storage is not part of the interface - here only for demonstration purposes
...     storage.store({'name': remote_name, 'large_file_id': large_file.id})
>>> bucket.create_file_stream(input_sources, remote_name, file_info, large_file_
↪callback=large_file_callback)
# ...
>>> large_file_id = storage.query({'name': remote_name})[0]['large_file_id']
>>> bucket.create_file_stream(input_sources, remote_name, file_info, large_file_
↪id=large_file_id)
```

Streams that contains remote sources cannot be continued with *b2sdk.v1.Bucket.create_file()* - internally *b2sdk.v1.Bucket.create_file()* stores plan information in file info for such inputs, and verifies it before any copy/upload and *b2sdk.v1.Bucket.create_file_stream()* cannot store this information. Local source only inputs can be safely continued with *b2sdk.v1.Bucket.create_file()* in auto continue mode or manual continue mode (because plan information is not stored in file info in such case).

Auto continuation

```
>>> bucket.create_file(input_sources, remote_name, file_info)
```

For local source only input, *b2sdk.v1.Bucket.create_file()* would try to find matching unfinished large file. It will verify uploaded parts checksums with local sources - the most completed, having all uploaded parts matched candidate would be automatically selected as file to continue. If there is no matching candidate (even if there are unfinished files for the same file name) new large file would be started.

In other cases plan information would be generated and *b2sdk.v1.Bucket.create_file()* would try to find unfinished large file with matching plan info in its file info. If there is one or more such unfinished large files, *b2sdk.v1.Bucket.create_file()* would verify checksums for all locally available parts and choose any matching candidate. If all candidates fails on uploaded parts checksums verification, process is interrupted and error raises. In such case corrupted unfinished large files should be cancelled manullay and *b2sdk.v1.Bucket.create_file()* should be retried, or auto continuation should be turned off with *auto_continue=False*

No continuation

```
>>> bucket.create_file(input_sources, remote_name, file_info, auto_continue=False)
```

Note, that this only forces start of a new large file - it is still possible to continue the process with either auto or manual modes.

2.6 Glossary

absoluteMinimumPartSize The smallest large file part size, as indicated during authorization process by the server (in 2019 it used to be `5MB`, but the server can set it dynamincally)

account ID An identifier of the B2 account (not login). Looks like this: `4ba5845d7aaf`.

application key ID Since every *account ID* can have multiple access keys associated with it, the keys need to be distinguished from each other. *application key ID* is an identifier of the access key. There are two types of keys: *master application key* and *non-master application key*.

application key The secret associated with an *application key ID*, used to authenticate with the server. Looks like this: `N2Zug0evLcHDlh_L0Z0AJhiGGdY` or `0a1bce5ea463a7e4b090ef5bd6bd82b851928ab2c6` or `K0014pbwo1zxcIVMnqSNTfWHReU/O3s`

b2sdk version Looks like this: `v1.0.0` or `1.0.0` and makes version numbers meaningful. See *Pinning versions* for more details.

b2sdk interface version Looks like this: `v2` or `b2sdk.v2` and makes maintaining backward compatibility much easier. See *interface versions* for more details.

master application key This is the first key you have access to, it is available on the B2 web application. This key has all capabilities, access to all *buckets*, and has no file prefix restrictions or expiration. The *application key ID* of the master application key is equal to *account ID*.

non-master application key A key which can have restricted capabilities, can only have access to a certain *bucket* or even to just part of it. See https://www.backblaze.com/b2/docs/application_keys.html to learn more. Looks like this: `0014aa9865d6f0000000000b0`

bucket A container that holds files. You can think of buckets as the top-level folders in your B2 Cloud Storage account. There is no limit to the number of files in a bucket, but there is a limit of 100 buckets per account. See https://www.backblaze.com/b2/docs/buckets.html to learn more.

2.7 About API interfaces

2.7.1 Semantic versioning

b2sdk follows Semantic Versioning policy, so in essence the version number is `MAJOR.MINOR.PATCH` (for example `1.2.3`) and:

- we increase *MAJOR* version when we make **incompatible** API changes
- we increase *MINOR* version when we add functionality **in a backwards-compatible manner**, and
- we increase *PATCH* version when we make backwards-compatible **bug fixes** (unless someone relies on the undocumented behavior of a fixed bug)

Therefore when setting up **b2sdk** as a dependency, please make sure to match the version appropriately, for example you could put this in your `requirements.txt` to make sure your code is compatible with the `b2sdk` version your user will get from pypi:

```
b2sdk>=1.0.0,<2.0.0
```

2.7.2 Interface versions

You might notice that the import structure provided in the documentation looks a little odd: `from b2sdk.v1 import ...`. The `.v1` part is used to keep the interface fluid without risk of breaking applications that use the old signatures. With new versions, **b2sdk** will provide functions with signatures matching the old ones, wrapping the new interface in place of the old one. What this means for a developer using **b2sdk**, is that it will just keep working. We havealready deleted some legacy functions when moving from `.v0` to `.v1`, providing equivalent wrappers to reduce the migration effort for applications using pre-1.0 versions of **b2sdk** to fixing imports.

It also means that **b2sdk** developers may change the interface in the future and will not need to maintain many branches and backport fixes to keep compatibility of for users of those old branches.

2.7.3 Interface version compatibility

A *numbered interface* will not be exactly identical throughout its lifespan, which should not be a problem for anyone, however just in case, the acceptable differences that the developer must tolerate, are listed below.

Exceptions

The exception hierarchy may change in a backwards compatible manner and the developer must anticipate it. For example, if `b2sdk.v1.ExceptionC` inherits directly from `b2sdk.v1.ExceptionA`, it may one day inherit from `b2sdk.v1.ExceptionB`, which in turn inherits from `b2sdk.v1.ExceptionA`. Normally this is not a problem if you use `isinstance()` and `super()` properly, but your code should not call the constructor of a parent class by directly naming it or it might skip the middle class of the hierarchy (`ExceptionB` in this example).

Extensions

Even in the same interface version, objects/classes/enums can get additional fields and their representations such as `to_dict()` or `__repr__` (but not `__str__`) may start to contain those fields.

Methods and functions can start accepting new **optional** arguments. New methods can be added to existing classes.

Performance

Some effort will be put into keeping the performance of the old interfaces, but in rare situations old interfaces may end up with a slightly degraded performance after a new version of the library is released. If performance target is absolutely critical to your application, you can pin your dependencies to the middle version (using `b2sdk>=X.Y.0,<X.Y+1.0`) as **b2sdk** *will* increment the middle version when introducing a new interface version if the wrapper for the older interfaces is likely to affect performance.

2.7.4 Public interface

Public interface consists of **public** members of modules listed in *Public API* section. This should be used in 99% of use cases, it's enough to implement anything from a console tool to a FUSE filesystem.

Those modules will generally not change in a backwards-incompatible way between non-major versions. Please see *interface version compatibility* chapter for notes on what changes must be expected.

Hint: If the current version of **b2sdk** is `4.5.6` and you only use the *public* interface, put this in your `requirements.txt` to be safe:

```
b2sdk>=4.5.6,<5.0.0
```

Note: `b2sdk.*._something` and `b2sdk.*.*._something`, while having a name beginning with an underscore, are **NOT** considered public interface.

2.7.5 Internal interface

Some rarely used features of B2 cloud are not implemented in **b2sdk**. Tracking usage of transactions and transferred data is a good example - if it is required, additional work would need to be put into a specialized internal interface layer to enable accounting and reporting.

b2sdk maintainers are *very supportive* in case someone wants to contribute an additional feature. Please consider adding it to the sdk, so that more people can use it. This way it will also receive our updates, unlike a private implementation which would not receive any updates unless you apply them manually (but that's a lot of work and we both know it's not going to happen). In practice, an implementation can be either shared or will quickly become outdated. The license of **b2sdk** is very permissive, but when considering whether to keep your patches private or public, please take into consideration the long-term cost of keeping up with a dynamic open-source project and/or the cost of missing the updates, especially those related to performance and reliability (as those are being actively developed in parallel to documentation).

Internal interface modules are listed in *API Internal* section.

Note: It is OK for you to use our internal interface (better that than copying our source files!), however, if you do, please pin your dependencies to **middle** version, as backwards-incompatible changes may be introduced in a non-major version.

Furthermore, it would be greatly appreciated if an issue was filed for such situations, so that **b2sdk** interface can be improved in a future version in order to avoid strict version pinning.

Hint: If the current version of **b2sdk** is `4.5.6` and you are using the *internal* interface, put this in your requirements.txt:

```
b2sdk>=4.5.6,<4.6.0
```

Hint: Use *Quick Start Guide* to quickly jump to examples

2.8 API Reference

2.8.1 Interface types

b2sdk API is divided into two parts, *public* and *internal*. Please pay attention to which interface type you use.

Tip: *Pinning versions* properly ensures the stability of your application.

2.8.2 Public API

AccountInfo

AccountInfo stores basic information about the account, such as *Application Key ID* and *Application Key*, in order to let `b2sdk.v1.B2Api` perform authenticated requests.

There are two usable implementations provided by **b2sdk**:

- `b2sdk.v1.InMemoryAccountInfo` - a basic implementation with no persistence
- `b2sdk.v1.SqliteAccountInfo` - for console and GUI applications

They both provide the full *AccountInfo interface*.

Note: Backup applications and many server-side applications should *implement their own AccountInfo*, backed by the metadata/configuration database of the application.

AccountInfo implementations

InMemoryAccountInfo

AccountInfo with no persistence.

class b2sdk.v1.**InMemoryAccountInfo**
Implements all methods of *AccountInfo interface*.

Hint: Usage of this class is appropriate for secure Web applications which do not wish to persist any user data.

Using this class for applications such as CLI, GUI or backup is discouraged, as `InMemoryAccountInfo` does not write down the authorization token persistently. That would be slow, as it would force the application to retrieve a new one on every command/click/backup start. Furthermore - an important property of *AccountInfo* is caching the `bucket_name:bucket_id` mapping; in case of `InMemoryAccountInfo` the cache will be flushed between executions of the program.

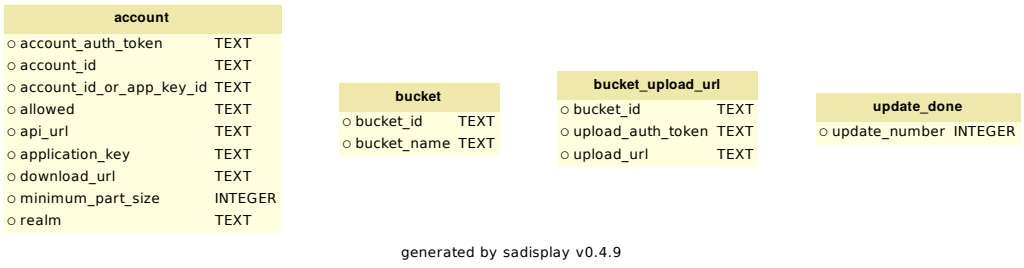
__init__()
The constructor takes no parameters.

SqliteAccountInfo

class b2sdk.v1.**SqliteAccountInfo**
Implements all methods of *AccountInfo interface*.

Uses a SQLite database for persistence and access synchronization between multiple processes. Not suitable for usage over NFS.

Underlying database has the following schema:



Hint: Usage of this class is appropriate for interactive applications installed on a user's machine (i.e.: CLI and GUI applications).

Usage of this class **might** be appropriate for non-interactive applications installed on the user's machine, such as backup applications. An alternative approach that should be considered is to store the *AccountInfo* data alongside the configuration of the rest of the application.

__init__ (*file_name=None*, *last_upgrade_to_run=None*)

If `file_name` argument is empty or `None`, path from `B2_ACCOUNT_INFO` environment variable is used. If that is not available, a default of `~/.b2_account_info` is used.

Parameters

- **file_name** (*str*) – The sqlite file to use; overrides the default.
- **last_upgrade_to_run** (*int*) – For testing only, override the auto-update on the db.

Implementing your own

When building a server-side application or a web service, you might want to implement your own *AccountInfo* class backed by a database. In such case, you should inherit from *b2sdk.v1.UrlPoolAccountInfo*, which has groundwork for url pool functionality). If you cannot use it, inherit directly from *b2sdk.v1.AbstractAccountInfo*.

```
>>> from b2sdk.v1 import UrlPoolAccountInfo
>>> class MyAccountInfo(UrlPoolAccountInfo):
        ...
```

b2sdk.v1.AbstractAccountInfo describes the interface, while *b2sdk.v1.UrlPoolAccountInfo* and `b2sdk.v1.UploadUrlPool` implement a part of the interface for in-memory upload token management.

AccountInfo interface

class b2sdk.v1.**AbstractAccountInfo**

get_s3_api_url()

Return s3_api_url or raises MissingAccountData exception.

Return type str

_abc_impl = <_abc_data object>

AccountInfo helper classes

class b2sdk.v1.**UrlPoolAccountInfo**

> **Caution:** This class is not part of the public interface. To find out how to safely use it, read *this*.

BUCKET_UPLOAD_POOL_CLASS

A url pool class to use for small files.

alias of *b2sdk.account_info.upload_url_pool.UploadUrlPool*

LARGE_FILE_UPLOAD_POOL_CLASS

A url pool class to use for large files.

alias of *b2sdk.account_info.upload_url_pool.UploadUrlPool*

class b2sdk.account_info.upload_url_pool.**UploadUrlPool**
For each key (either a bucket id or large file id), hold a pool of (url, auth_token) pairs.

> **Caution:** This class is not part of the public interface. To find out how to safely use it, read *this*.

put(*key*, *url*, *auth_token*)
Add the url and auth token to the pool for the given key.

Parameters

- **key** (*str*) – bucket ID or large file ID
- **url** (*str*) – bucket or file URL
- **auth_token** (*str*) – authentication token

take(*key*)
Return a (url, auth_token) if one is available, or (None, None) if not.

Parameters **key** (*str*) – bucket ID or large file ID

Return type tuple

clear_for_key(*key*)
Remove an item from the pool by key.

Parameters **key** (*str*) – bucket ID or large file ID

Cache

b2sdk caches the mapping between bucket name and bucket id, so that the user of the library does not need to maintain the mapping to call the api.

class b2sdk.v1.**AbstractCache**

clear()

abstract get_bucket_id_or_none_from_bucket_name(*name*)

abstract get_bucket_name_or_none_from_allowed()

abstract save_bucket(*bucket*)

abstract set_bucket_name_cache(*buckets*)

class b2sdk.v1.**AuthInfoCache**
A cache that stores data persistently in StoredAccountInfo.

__init__(*info*)
Initialize self. See help(type(self)) for accurate signature.

get_bucket_id_or_none_from_bucket_name(*name*)

get_bucket_name_or_none_from_allowed()

save_bucket(*bucket*)

set_bucket_name_cache(*buckets*)

class b2sdk.v1.**DummyCache**
A cache that does nothing.

get_bucket_id_or_none_from_bucket_name(*name*)

get_bucket_name_or_none_from_allowed()

save_bucket(*bucket*)

set_bucket_name_cache(*buckets*)

class b2sdk.v1.**InMemoryCache**
A cache that stores the information in memory.

__init__()
Initialize self. See help(type(self)) for accurate signature.

get_bucket_id_or_none_from_bucket_name(*name*)

get_bucket_name_or_none_from_allowed()

save_bucket(*bucket*)

set_bucket_name_cache(*buckets*)

B2 Api client

class b2sdk.v1.**B2Api**
Provide file-level access to B2 services.

While b2sdk.v1.B2RawApi provides direct access to the B2 web APIs, this class handles several things that simplify the task of uploading and downloading files:

- re-acquires authorization tokens when they expire
- retrying uploads when an upload URL is busy
- breaking large files into parts
- emulating a directory structure (B2 buckets are flat)

Adds an object-oriented layer on top of the raw API, so that buckets and files returned are Python objects with accessor methods.

The class also keeps a cache of information needed to access the service, such as auth tokens and upload URLs.

BUCKET_FACTORY_CLASS
alias of b2sdk.bucket.BucketFactory

BUCKET_CLASS
alias of b2sdk.bucket.Bucket

__init__(*account_info=None*, *cache=None*, *raw_api=None*, *max_upload_workers=10*, *max_copy_workers=10*)
Initialize the API using the given account info.

Parameters

- **account_info** – an instance of *UrlPoolAccountInfo*, or any custom class derived from *AbstractAccountInfo* To learn more about Account Info objects, see here *SqliteAccountInfo*
- **cache** – an instance of the one of the following classes: *DummyCache*, *InMemoryCache*, *AuthInfoCache*, or any custom class derived from *AbstractCache* It is used by B2Api to cache the mapping between bucket name and bucket ids. default is *DummyCache*

- **raw_api** – an instance of one of the following classes: *B2RawApi*, *RawSimulator*, or any custom class derived from *AbstractRawApi* It makes network-less unit testing simple by using *RawSimulator*, in tests and *B2RawApi* in production. default is *B2RawApi*
- **max_upload_workers** (*int*) – a number of upload threads, default is 10
- **max_copy_workers** (*int*) – a number of copy threads, default is 10

property account_info

property cache

property raw_api

> **Warning:** *B2RawApi* attribute is deprecated. *B2Session* expose all *B2RawApi* methods now.

authorize_automatically()
Perform automatic account authorization, retrieving all account data from account info object passed during initialization.

authorize_account(*realm*, *application_key_id*, *application_key*)
Perform account authorization.

Parameters

- **realm** (*str*) – a realm to authorize account in (usually just "production")
- **application_key_id** (*str*) – *application key ID*
- **application_key** (*str*) – user's *application key*

get_account_id()
Return the account ID.

Return type str

create_bucket(*name*, *bucket_type*, *bucket_info=None*, *cors_rules=None*, *lifecycle_rules=None*, *default_server_side_encryption: Optional[*b2sdk.encryption.setting.EncryptionSetting*] = None*)
Create a bucket.

Parameters

- **name** (*str*) – bucket name
- **bucket_type** (*str*) – a bucket type, could be one of the following values: `"allPublic"`, `"allPrivate"`
- **bucket_info** (*dict*) – additional bucket info to store with the bucket
- **cors_rules** (*dict*) – bucket CORS rules to store with the bucket
- **lifecycle_rules** (*dict*) – bucket lifecycle rules to store with the bucket
- **default_server_side_encryption** (*b2sdk.v1.EncryptionSetting*) – default server side encryption settings (`None` if unknown)

Returns a Bucket object

Return type *b2sdk.v1.Bucket*

download_file_by_id(*file_id*, *download_dest*, *progress_listener=None*, *range_=None*, *encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*)

Download a file with the given ID.

Parameters

- **file_id** (*str*) – a file ID
- **download_dest** – an instance of the one of the following classes: *DownloadDestLocalFile*, *DownloadDestBytes*, *DownloadDestProgressWrapper*, *PreSeekedDownloadDest*, or any sub class of *AbstractDownloadDestination*
- **progress_listener** – an instance of the one of the following classes: PartProgressReporter, *TqdmProgressListener*, *SimpleProgressListener*, *DoNothingProgressListener*, *ProgressListenerForTest*, SyncFileReporter, or any sub class of *AbstractProgressListener*
- **range** (*list*) – a list of two integers, the first one is a start position, and the second one is the end position in the file
- **encryption** (*b2sdk.v1.EncryptionSetting*) – encryption settings (None if unknown)

Returns context manager that returns an object that supports iter_content()

get_bucket_by_id(*bucket_id*)

Return a bucket object with a given ID. Unlike get_bucket_by_name, this method does not need to make any API calls.

Parameters **bucket_id** (*str*) – a bucket ID

Returns a Bucket object

Return type *b2sdk.v1.Bucket*

get_bucket_by_name(*bucket_name*)

Return the Bucket matching the given bucket_name.

Parameters **bucket_name** (*str*) – the name of the bucket to return

Returns a Bucket object

Return type *b2sdk.v1.Bucket*

Raises **b2sdk.v1.exception.NonExistentBucket** – if the bucket does not exist in the account

delete_bucket(*bucket*)

Delete a chosen bucket.

Parameters **bucket** (b2sdk.v1.Bucket) – a *bucket* to delete

Return type None

list_buckets(*bucket_name=None*, *bucket_id=None*)

Call b2_list_buckets and return a list of buckets.

When no bucket name nor ID is specified, returns *all* of the buckets in the account. When a bucket name or ID is given, returns just that bucket. When authorized with an *application key* restricted to one *bucket*, you must specify the bucket name, or the request will be unauthorized.

Parameters

- **bucket_name** (*str*) – the name of the one bucket to return

- **bucket_id** (*str*) – the ID of the one bucket to return

Return type list[*b2sdk.v1.Bucket*]

list_parts(*file_id*, *start_part_number=None*, *batch_size=None*)
Generator that yields a *b2sdk.v1.Part* for each of the parts that have been uploaded.

Parameters

- **file_id** (*str*) – the ID of the large file that is not finished
- **start_part_number** (*int*) – the first part number to return; defaults to the first part
- **batch_size** (*int*) – the number of parts to fetch at a time from the server

Return type generator

cancel_large_file(*file_id*)
Cancel a large file upload.

Parameters **file_id** (*str*) – a file ID

Return type None

delete_file_version(*file_id*, *file_name*)
Permanently and irrevocably delete one version of a file.

Parameters

- **file_id** (*str*) – a file ID
- **file_name** (*str*) – a file name

Return type *FileIdAndName*

get_download_url_for_fileid(*file_id*)
Return a URL to download the given file by ID.

Parameters **file_id** (*str*) – a file ID

get_download_url_for_file_name(*bucket_name*, *file_name*)
Return a URL to download the given file by name.

Parameters

- **bucket_name** (*str*) – a bucket name
- **file_name** (*str*) – a file name

create_key(*capabilities*, *key_name*, *valid_duration_seconds=None*, *bucket_id=None*, *name_prefix=None*)
Create a new *application key*.

Parameters

- **capabilities** (*list*) – a list of capabilities
- **key_name** (*str*) – a name of a key
- **valid_duration_seconds** (*int*, *None*) – key auto-expire time after it is created, in seconds, or `None` to not expire
- **bucket_id** (*str*, *None*) – a bucket ID to restrict the key to, or `None` to not restrict
- **name_prefix** (*str*, *None*) – a remote filename prefix to restrict the key to or `None` to not restrict

delete_key(*application_key_id*)
Delete *application key*.

Parameters **application_key_id** (*str*) – an *application key ID*

list_keys(*start_application_key_id=None*)
List application keys.

Parameters **start_application_key_id** (*str*, *None*) – an *application key ID* to start from or None to start from the beginning

get_file_info(*file_id: str*) → Dict[str, Any]
Legacy interface which just returns whatever remote API returns.

Parameters **file_id** (*str*) – the id of the file who's info will be retrieved.

Returns The parsed response

Return type dict

check_bucket_restrictions(*bucket_name*)
Check to see if the allowed field from authorize-account has a bucket restriction.

If it does, checks if the bucket_name for a given api call matches that. If not, it raises a b2sdk.v1.exception.RestrictedBucket error.

Parameters **bucket_name** (*str*) – a bucket name

Raises **b2sdk.v1.exception.RestrictedBucket** – if the account is not allowed to use this bucket

Exceptions

B2 Bucket

class b2sdk.v1.**Bucket**
Provide access to a bucket in B2: listing files, uploading and downloading.

DEFAULT_CONTENT_TYPE = 'b2/x-auto'

__init__(*api*, *id_*, *name=None*, *type_=None*, *bucket_info=None*, *cors_rules=None*, *lifecycle_rules=None*, *revision=None*, *bucket_dict=None*, *options_set=None*, *default_server_side_encryption: b2sdk.encryption.setting.EncryptionSetting = <EncryptionSetting(EncryptionMode.UNKNOWN, None, None)>*)

Parameters

- **api** (b2sdk.v1.B2Api) – an API object
- **id** (*str*) – a bucket id
- **name** (*str*) – a bucket name
- **type** (*str*) – a bucket type
- **bucket_info** (*dict*) – an info to store with a bucket
- **cors_rules** (*dict*) – CORS rules to store with a bucket
- **lifecycle_rules** (*dict*) – lifecycle rules to store with a bucket
- **revision** (*int*) – a bucket revision number
- **bucket_dict** (*dict*) – a dictionary which contains bucket parameters
- **options_set** (*set*) – set of bucket options strings

- **default_server_side_encryption** (*b2sdk.v1.EncryptionSetting*) – default server side encryption settings

get_id()
Return bucket ID.

Return type str

set_info(*new_bucket_info*, *if_revision_is=None*)
Update bucket info.

Parameters

- **new_bucket_info** (*dict*) – new bucket info dictionary
- **if_revision_is** (*int*) – revision number, update the info **only if** *revision* equals to *if_revision_is*

set_type(*bucket_type*)
Update bucket type.

Parameters **bucket_type** (*str*) – a bucket type ("allPublic" or "allPrivate")

update(*bucket_type=None*, *bucket_info=None*, *cors_rules=None*, *lifecycle_rules=None*, *if_revision_is=None*, *default_server_side_encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*)
Update various bucket parameters. For legacy reasons in apiver v1 it returns whatever server returned on b2_update_bucket call, v2 will change that.

Parameters

- **bucket_type** (*str*) – a bucket type
- **bucket_info** (*dict*) – an info to store with a bucket
- **cors_rules** (*dict*) – CORS rules to store with a bucket
- **lifecycle_rules** (*dict*) – lifecycle rules to store with a bucket
- **if_revision_is** (*int*) – revision number, update the info **only if** *revision* equals to *if_revision_is*
- **default_server_side_encryption** (*b2sdk.v1.EncryptionSetting*) – default server side encryption settings (`None` if unknown)

cancel_large_file(*file_id*)
Cancel a large file transfer.

Parameters **file_id** (*str*) – a file ID

download_file_by_id(*file_id*, *download_dest*, *progress_listener=None*, *range_=None*, *encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*)
Download a file by ID.

Note: download_file_by_id actually belongs in *b2sdk.v1.B2Api*, not in *b2sdk.v1.Bucket*; we just provide a convenient redirect here

---

Parameters

- **file_id** (*str*) – a file ID
- **download_dest** – an instance of the one of the following classes: *DownloadDestLocalFile*, *DownloadDestBytes*,

DownloadDestProgressWrapper, *PreSeekedDownloadDest*, or any sub class of *AbstractDownloadDestination*

- **None progress_listener** (b2sdk.v1.AbstractProgressListener,) – a progress listener object to use, or None to not report progress
- **int] range** (*tuple[int,*) – two integer values, start and end offsets
- **encryption** (*b2sdk.v1.EncryptionSetting*) – encryption settings (None if unknown)

download_file_by_name (*file_name*, *download_dest*, *progress_listener=None*, *range_=None*, *encryption: Optional[*b2sdk.encryption.setting.EncryptionSetting*] = None*)

Download a file by name.

See also:

Synchronizer, a *high-performance* utility that synchronizes a local folder with a Bucket.

Parameters

- **file_name** (*str*) – a file name
- **download_dest** – an instance of the one of the following classes: *DownloadDestLocalFile*, *DownloadDestBytes*, *DownloadDestProgressWrapper*, *PreSeekedDownloadDest*, or any sub class of *AbstractDownloadDestination*
- **None progress_listener** (b2sdk.v1.AbstractProgressListener,) – a progress listener object to use, or None to not track progress
- **int] range** (*tuple[int,*) – two integer values, start and end offsets
- **encryption** (*b2sdk.v1.EncryptionSetting*) – encryption settings (None if unknown)

get_file_info_by_id (*file_id: str*) → b2sdk.file_version.FileVersionInfo

Gets a file version's info by ID.

Parameters **file_id** (*str*) – the id of the file who's info will be retrieved.

Return type generator[*b2sdk.v1.FileVersionInfo*]

get_file_info_by_name (*file_name: str*) → b2sdk.file_version.FileVersionInfo

Gets a file version's info by its name.

Parameters **file_name** (*str*) – the name of the file who's info will be retrieved.

Return type generator[*b2sdk.v1.FileVersionInfo*]

get_download_authorization (*file_name_prefix*, *valid_duration_in_seconds*)

Return an authorization token that is valid only for downloading files from the given bucket.

Parameters

- **file_name_prefix** (*str*) – a file name prefix, only files that match it could be downloaded
- **valid_duration_in_seconds** (*int*) – a token is valid only during this amount of seconds

list_parts (*file_id*, *start_part_number=None*, *batch_size=None*)

Get a list of all parts that have been uploaded for a given file.

Parameters

- **file_id** (*str*) – a file ID
- **start_part_number** (*int*) – the first part number to return. defaults to the first part.
- **batch_size** (*int*) – the number of parts to fetch at a time from the server

list_file_versions (*file_name*, *fetch_count=None*)
Lists all of the versions for a single file.

Parameters

- **file_name** (*str*) – the name of the file to list.
- **fetch_count** (*int,None*) – how many entries to list per API call or `None` to use the default. Acceptable values: 1 - 10000

Return type generator[*b2sdk.v1.FileVersionInfo*]

ls (*folder_to_list=''*, *show_versions=False*, *recursive=False*, *fetch_count=10000*)
Pretend that folders exist and yields the information about the files in a folder.

B2 has a flat namespace for the files in a bucket, but there is a convention of using "/" as if there were folders. This method searches through the flat namespace to find the files and "folders" that live within a given folder.

When the *recursive* flag is set, lists all of the files in the given folder, and all of its sub-folders.

Parameters

- **folder_to_list** (*str*) – the name of the folder to list; must not start with "/". Empty string means top-level folder
- **show_versions** (*bool*) – when `True` returns info about all versions of a file, when `False`, just returns info about the most recent versions
- **recursive** (*bool*) – if `True`, list folders recursively
- **fetch_count** (*int,None*) – how many entries to return or `None` to use the default. Acceptable values: 1 - 10000

Return type generator[tuple[*b2sdk.v1.FileVersionInfo*, str]]

Returns generator of (file_version_info, folder_name) tuples

Note: In case of *recursive=True*, folder_name is returned only for first file in the folder.

---

list_unfinished_large_files (*start_file_id=None*, *batch_size=None*, *prefix=None*)
A generator that yields an *b2sdk.v1.UnfinishedLargeFile* for each unfinished large file in the bucket, starting at the given file, filtering by prefix.

Parameters

- **start_file_id** (*str,None*) – a file ID to start from or None to start from the beginning
- **batch_size** (*int,None*) – max file count
- **prefix** (*str,None*) – file name prefix filter

Return type generator[*b2sdk.v1.UnfinishedLargeFile*]

start_large_file (*file_name*, *content_type=None*, *file_info=None*)
Start a large file transfer.

Parameters

- **file_name** (*str*) – a file name
- **content_type** (*str, None*) – the MIME type, or None to accept the default based on file extension of the B2 file name
- **file_info** (*dict, None*) – a file info to store with the file or None to not store anything

upload_bytes(*data_bytes*, *file_name*, *content_type=None*, *file_infos=None*, *progress_listener=None*, *encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*)

Upload bytes in memory to a B2 file.

Parameters

- **data_bytes** (*bytes*) – a byte array to upload
- **file_name** (*str*) – a file name to upload bytes to
- **content_type** (*str, None*) – the MIME type, or None to accept the default based on file extension of the B2 file name
- **file_infos** (*dict, None*) – a file info to store with the file or None to not store anything
- **progress_listener** (b2sdk.v1.AbstractProgressListener, *None*) – a progress listener object to use, or None to not track progress
- **encryption** (*b2sdk.v1.EncryptionSetting*) – encryption settings (None if unknown)

Return type generator[b2sdk.v1.FileVersion]

upload_local_file(*local_file*, *file_name*, *content_type=None*, *file_infos=None*, *sha1_sum=None*, *min_part_size=None*, *progress_listener=None*, *encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*)

Upload a file on local disk to a B2 file.

See also:

Synchronizer, a *high-performance* utility that synchronizes a local folder with a *bucket*.

Parameters

- **local_file** (*str*) – a path to a file on local disk
- **file_name** (*str*) – a file name of the new B2 file
- **content_type** (*str, None*) – the MIME type, or None to accept the default based on file extension of the B2 file name
- **file_infos** (*dict, None*) – a file info to store with the file or None to not store anything
- **sha1_sum** (*str, None*) – file SHA1 hash or None to compute it automatically
- **min_part_size** (*int*) – a minimum size of a part
- **progress_listener** (b2sdk.v1.AbstractProgressListener, *None*) – a progress listener object to use, or None to not report progress
- **encryption** (*b2sdk.v1.EncryptionSetting*) – encryption settings (None if unknown)

Return type *b2sdk.v1.FileVersionInfo*

upload(*upload_source*, *file_name*, *content_type=None*, *file_info=None*, *min_part_size=None*, *progress_listener=None*, *encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*)

Upload a file to B2, retrying as needed.

The source of the upload is an UploadSource object that can be used to open (and re-open) the file. The result of opening should be a binary file whose read() method returns bytes.

The function *opener* should return a file-like object, and it must be possible to call it more than once in case the upload is retried.

Parameters

- **upload_source** (*b2sdk.v1.UploadSource*) – an object that opens the source of the upload
- **file_name** (*str*) – the file name of the new B2 file
- **content_type** (*str*, *None*) – the MIME type, or None to accept the default based on file extension of the B2 file name
- **file_info** (*dict*, *None*) – a file info to store with the file or None to not store anything
- **min_part_size** (*int*, *None*) – the smallest part size to use or None to determine automatically
- **progress_listener** (b2sdk.v1.AbstractProgressListener, *None*) – a progress listener object to use, or None to not report progress
- **encryption** (*b2sdk.v1.EncryptionSetting*) – encryption settings (None if unknown)

Return type *b2sdk.v1.FileVersionInfo*

create_file(*write_intents*, *file_name*, *content_type=None*, *file_info=None*, *progress_listener=None*, *recommended_upload_part_size=None*, *continue_large_file_id=None*, *encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*)

Creates a new file in this bucket using an iterable (list, tuple etc) of remote or local sources.

Source ranges can overlap and remote sources will be prioritized over local sources (when possible). For more information and usage examples please see *Advanced usage patterns*.

Parameters

- **write_intents** (*list[b2sdk.v1.WriteIntent]*) – list of write intents (remote or local sources)
- **new_file_name** (*str*) – file name of the new file
- **content_type** (*str*, *None*) – content_type for the new file, if None content_type would be automatically determined or it may be copied if it resolves as single part remote source copy
- **file_info** (*dict*, *None*) – file_info for the new file, if None it will be set to empty dict or it may be copied if it resolves as single part remote source copy
- **progress_listener** (b2sdk.v1.AbstractProgressListener, *None*) – a progress listener object to use, or None to not report progress
- **recommended_upload_part_size** (*int*, *None*) – the recommended part size to use for uploading local sources or None to determine automatically, but remote sources would be copied with maximum possible part size

- **continue_large_file_id** (*str*, *None*) – large file id that should be selected to resume file creation for multipart upload/copy, None for automatic search for this id
- **encryption** (*b2sdk.v1.EncryptionSetting*) – encryption settings (None if unknown)

create_file_stream(*write_intents_iterator*, *file_name*, *content_type=None*, *file_info=None*, *progress_listener=None*, *recommended_upload_part_size=None*, *continue_large_file_id=None*, *encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*)

Creates a new file in this bucket using a stream of multiple remote or local sources.

Source ranges can overlap and remote sources will be prioritized over local sources (when possible). For more information and usage examples please see *Advanced usage patterns*.

Parameters

- **write_intents_iterator** (*iterator[b2sdk.v1.WriteIntent]*) – iterator of write intents which are sorted ascending by destination_offset
- **new_file_name** (*str*) – file name of the new file
- **content_type** (*str*, *None*) – content_type for the new file, if None content_type would be automatically determined or it may be copied if it resolves as single part remote source copy
- **file_info** (*dict*, *None*) – file_info for the new file, if None it will be set to empty dict or it may be copied if it resolves as single part remote source copy
- **progress_listener** (b2sdk.v1.AbstractProgressListener, *None*) – a progress listener object to use, or None to not report progress
- **recommended_upload_part_size** (*int*, *None*) – the recommended part size to use for uploading local sources or None to determine automatically, but remote sources would be copied with maximum possible part size
- **continue_large_file_id** (*str*, *None*) – large file id that should be selected to resume file creation for multipart upload/copy, if None in multipart case it would always start a new large file
- **encryption** (*b2sdk.v1.EncryptionSetting*) – encryption settings (None if unknown)

concatenate(*outbound_sources*, *file_name*, *content_type=None*, *file_info=None*, *progress_listener=None*, *recommended_upload_part_size=None*, *continue_large_file_id=None*, *encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*)

Creates a new file in this bucket by concatenating multiple remote or local sources.

Parameters

- **outbound_sources** (*list[b2sdk.v1.OutboundTransferSource]*) – list of outbound sources (remote or local)
- **new_file_name** (*str*) – file name of the new file
- **content_type** (*str*, *None*) – content_type for the new file, if None content_type would be automatically determined from file name or it may be copied if it resolves as single part remote source copy
- **file_info** (*dict*, *None*) – file_info for the new file, if None it will be set to empty dict or it may be copied if it resolves as single part remote source copy

- **progress_listener** (b2sdk.v1.AbstractProgressListener, *None*) – a progress listener object to use, or None to not report progress
- **recommended_upload_part_size** (*int*, *None*) – the recommended part size to use for uploading local sources or None to determine automatically, but remote sources would be copied with maximum possible part size
- **continue_large_file_id** (*str*, *None*) – large file id that should be selected to resume file creation for multipart upload/copy, None for automatic search for this id
- **encryption** (*b2sdk.v1.EncryptionSetting*) – encryption settings (None if unknown)

concatenate_stream(*outbound_sources_iterator*, *file_name*, *content_type=None*, *file_info=None*, *progress_listener=None*, *recommended_upload_part_size=None*, *continue_large_file_id=None*, *encryption: Optional[*b2sdk.encryption.setting.EncryptionSetting*] = None*)

Creates a new file in this bucket by concatenating stream of multiple remote or local sources.

Parameters

- **outbound_sources_iterator** (*iterator[*b2sdk.v1.OutboundTransferSource*]*) – iterator of outbound sources
- **new_file_name** (*str*) – file name of the new file
- **content_type** (*str*, *None*) – content_type for the new file, if None content_type would be automatically determined or it may be copied if it resolves as single part remote source copy
- **file_info** (*dict*, *None*) – file_info for the new file, if None it will be set to empty dict or it may be copied if it resolves as single part remote source copy
- **progress_listener** (b2sdk.v1.AbstractProgressListener, *None*) – a progress listener object to use, or None to not report progress
- **recommended_upload_part_size** (*int*, *None*) – the recommended part size to use for uploading local sources or None to determine automatically, but remote sources would be copied with maximum possible part size
- **continue_large_file_id** (*str*, *None*) – large file id that should be selected to resume file creation for multipart upload/copy, if None in multipart case it would always start a new large file
- **encryption** (*b2sdk.v1.EncryptionSetting*) – encryption setting (None if unknown)

get_download_url(*filename*)

Get file download URL.

Parameters **filename** (*str*) – a file name

Return type str

hide_file(*file_name*)

Hide a file.

Parameters **file_name** (*str*) – a file name

Return type *b2sdk.v1.FileVersionInfo*

copy(*file_id*, *new_file_name*, *content_type=None*, *file_info=None*, *offset=0*, *length=None*, *progress_listener=None*, *destination_encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*, *source_encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*, *source_file_info: Optional[dict] = None*, *source_content_type: Optional[str] = None*)

Creates a new file in this bucket by (server-side) copying from an existing file.

Parameters

- **file_id** (*str*) – file ID of existing file to copy from
- **new_file_name** (*str*) – file name of the new file
- **content_type** (*str*, *None*) – content_type for the new file, if `None` and `b2_copy_file` will be used content_type will be copied from source file - otherwise content_type would be automatically determined
- **file_info** (*dict*, *None*) – file_info for the new file, if `None` will and `b2_copy_file` will be used file_info will be copied from source file - otherwise it will be set to empty dict
- **offset** (*int*) – offset of exisiting file that copy should start from
- **length** (*int*, *None*) – number of bytes to copy, if `None` then `offset` have to be `0` and it will use `b2_copy_file` without `range` parameter so it may fail if file is too large. For large files length have to be specified to use `b2_copy_part` instead.
- **progress_listener** (`b2sdk.v1.AbstractProgressListener`, *None*) – a progress listener object to use for multipart copy, or `None` to not report progress
- **destination_encryption** (*b2sdk.v1.EncryptionSetting*) – encryption settings for the destination (`None` if unknown)
- **source_encryption** (*b2sdk.v1.EncryptionSetting*) – encryption settings for the source (`None` if unknown)
- **source_file_info** (*dict*, *None*) – source file's file_info dict, useful when copying files with SSE-C
- **source_content_type** (*str*, *None*) – source file's content type, useful when copying files with SSE-C

copy_file(*file_id*, *new_file_name*, *bytes_range=None*, *metadata_directive=None*, *content_type=None*, *file_info=None*, *destination_encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*, *source_encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*)

Creates a new file in this bucket by (server-side) copying from an existing file.

Parameters

- **file_id** (*str*) – file ID of existing file
- **new_file_name** (*str*) – file name of the new file
- **bytes_range** (*tuple[int, int]*, *None*) – start and end offsets (**inclusive!**), default is the entire file
- **metadata_directive** (`b2sdk.v1.MetadataDirectiveMode`, *None*) – default is *b2sdk.v1.MetadataDirectiveMode.COPY*
- **content_type** (*str*, *None*) – content_type for the new file if metadata_directive is set to *b2sdk.v1.MetadataDirectiveMode.REPLACE*, default will copy the content_type of old file

- **file_info** (*dict, None*) – file_info for the new file if metadata_directive is set to *b2sdk.v1.MetadataDirectiveMode.REPLACE*, default will copy the file_info of old file
- **destination_encryption** (*b2sdk.v1.EncryptionSetting*) – encryption settings for the destination (`None` if unknown)
- **source_encryption** (*b2sdk.v1.EncryptionSetting*) – encryption settings for the source (`None` if unknown)

delete_file_version(*file_id*, *file_name*)
Delete a file version.

Parameters

- **file_id** (*str*) – a file ID
- **file_name** (*str*) – a file name

as_dict()
Return bucket representation as a dictionary.

Return type dict

Data classes

class b2sdk.v1.**FileVersionInfo**(*id_*, *file_name*, *size*, *content_type*, *content_sha1*, *file_info*, *upload_timestamp*, *action*, *content_md5=None*, *server_side_encryption: Optional[*b2sdk.encryption.setting.EncryptionSetting*] = None*)
A structure which represents a version of a file (in B2 cloud).

Variables

- **id_** (*str*) – `fileId`
- **file_name** (*str*) – full file name (with path)
- **size** (*int or None*) – size in bytes, can be `None` (unknown)
- **content_type** (*str*) – RFC 822 content type, for example `"application/octet-stream"`
- **content_sha1** (*str or None*) – sha1 checksum of the entire file, can be `None` (unknown) if it is a large file uploaded by a client which did not provide it
- **content_md5** (*str or None*) – md5 checksum of the file, can be `None` (unknown)
- **file_info** (*dict*) – file info dict
- **upload_timestamp** (*int or None*) – in milliseconds since EPOCH (1970-01-01 00:00:00). Can be `None` (unknown).
- **action** (*str*) – `"upload"`, `"hide"` or `"delete"`

LS_ENTRY_TEMPLATE = '%83s %6s %10s %8s %9d %s'

id_

file_name

size

content_type

content_sha1

content_md5

file_info

upload_timestamp

action

server_side_encryption

as_dict()
represents the object as a dict which looks almost exactly like the raw api output for upload/list

format_ls_entry()
legacy method, to be removed in v2: formats a *ls* entry for b2 command line tool

classmethod format_folder_ls_entry(*name*)
legacy method, to be removed in v2: formats a *ls* "folder" consistently with format_ls_entry()

class b2sdk.v1.**FileIdAndName**(*file_id*, *file_name*)
A structure which represents a B2 cloud file with just *file_name* and *fileId* attributes.

Used to return data from calls to *b2sdk.v1.Bucket.delete_file_version()*.

Variables

- **file_id** (*str*) – fileId
- **file_name** (*str*) – full file name (with path)

as_dict()
represents the object as a dict which looks almost exactly like the raw api output for delete_file_version

__dict__ = mappingproxy({'__module__': 'b2sdk.file_version', '__doc__': '\n A struct

class b2sdk.v1.**UnfinishedLargeFile**
A structure which represents a version of a file (in B2 cloud).

Variables

- **file_id** (*str*) – fileId
- **file_name** (*str*) – full file name (with path)
- **account_id** (*str*) – account ID
- **bucket_id** (*str*) – bucket ID
- **content_type** (*str*) – **RFC 822** content type, for example "application/octet-stream"
- **file_info** (*dict*) – file info dict

class b2sdk.v1.**Part**(*file_id*, *part_number*, *content_length*, *content_sha1*)
A structure which represents a *part* of a large file upload.

Variables

- **file_id** (*str*) – fileId
- **part_number** (*int*) – part number, starting with 1
- **content_length** (*str*) – content length, in bytes
- **content_sha1** (*str*) – checksum

Enums

class b2sdk.v1.**MetadataDirectiveMode**(*value*)
Mode of handling metadata when copying a file

COPY = 401
copy metadata from the source file

REPLACE = 402
ignore the source file metadata and set it to provided values

class b2sdk.v1.**NewerFileSyncMode**(*value*)
Mode of handling files newer on destination than on source

SKIP = 101
skip syncing such file

REPLACE = 102
replace the file on the destination with the (older) file on source

RAISE_ERROR = 103
raise a non-transient error, failing the sync operation

class b2sdk.v1.**CompareVersionMode**(*value*)
Mode of comparing versions of files to determine what should be synced and what shouldn't

MODTIME = 201
use file modification time on source filesystem

SIZE = 202
compare using file size

NONE = 203
compare using file name only

class b2sdk.v1.**KeepOrDeleteMode**(*value*)
Mode of dealing with old versions of files on the destination

DELETE = 301
delete the old version as soon as the new one has been uploaded

KEEP_BEFORE_DELETE = 302
keep the old versions of the file for a configurable number of days before deleting them, always keeping the newest version

NO_DELETE = 303
keep old versions of the file, do not delete anything

Progress reporters

Note: Concrete classes described in this chapter implement methods defined in `AbstractProgressListener`

class b2sdk.v1.**AbstractProgressListener**
Interface expected by B2Api upload and download methods to report on progress.

This interface just accepts the number of bytes transferred so far. Subclasses will need to know the total size if they want to report a percent done.

abstract set_total_bytes(*total_byte_count*)
Always called before __enter__ to set the expected total number of bytes.

May be called more than once if an upload is retried.

Parameters **total_byte_count** (*int*) – expected total number of bytes

abstract bytes_completed(*byte_count*)
Report the given number of bytes that have been transferred so far. This is not a delta, it is the total number of bytes transferred so far.

Transfer can fail and restart from beginning so byte count can decrease between calls.

Parameters **byte_count** (*int*) – number of bytes have been transferred

close()
Must be called when you're done with the listener. In well-structured code, should be called only once.

class b2sdk.v1.**TqdmProgressListener**(*description*, **args*, ***kwargs*)
Progress listener based on tqdm library.

class b2sdk.v1.**SimpleProgressListener**(*description*, **args*, ***kwargs*)
Just a simple progress listener which prints info on a console.

class b2sdk.v1.**DoNothingProgressListener**
This listener gives no output whatsoever.

class b2sdk.v1.**ProgressListenerForTest**(**args*, ***kwargs*)
Capture all of the calls so they can be checked.

b2sdk.v1.**make_progress_listener**(*description*, *quiet*)
Return a progress listener object depending on some conditions.

Parameters

- **description** (*str*) – listener description
- **quiet** (*bool*) – if `True`, do not output anything

Returns a listener object

Synchronizer

Synchronizer is a powerful utility with functionality of a basic backup application. It is able to copy entire folders into the cloud and back to a local drive or even between two cloud buckets, providing retention policies and many other options.

The **high performance** of sync is credited to parallelization of:

- listing local directory contents
- listing bucket contents
- uploads
- downloads

Synchronizer spawns threads to perform the operations listed above in parallel to shorten the backup window to a minimum.

Sync Options

Following are the important optional arguments that can be provided while initializing *Synchronizer* class.

- `compare_version_mode`: When comparing the source and destination files for finding whether to replace them or not, *compare_version_mode* can be passed to specify the mode of comparison. For possible values see *b2sdk.v1.CompareVersionMode*. Default value is *b2sdk.v1.CompareVersionMode.MODTIME*
- `compare_threshold`: It's the minimum size (in bytes)/modification time (in seconds) difference between source and destination files before we assume that it is new and replace.
- `newer_file_mode`: To identify whether to skip or replace if source is older. For possible values see *b2sdk.v1.NewerFileSyncMode*. If you don't specify this the sync will raise `b2sdk.v1.exception.DestFileNewer` in case any of the source file is older than destination.
- `keep_days_or_delete`: specify policy to keep or delete older files. For possible values see *b2sdk.v1.KeepOrDeleteMode*. Default is *DO_NOTHING*.
- `keep_days`: if *keep_days_or_delete* is `b2sdk.v1.CompareVersionMode.KEEP_BEFORE_DELETE` then this specify for how many days should we keep.

```
>>> from b2sdk.v1 import ScanPoliciesManager
>>> from b2sdk.v1 import parse_sync_folder
>>> from b2sdk.v1 import Synchronizer
>>> from b2sdk.v1 import KeepOrDeleteMode, CompareVersionMode, NewerFileSyncMode
>>> import time
>>> import sys

>>> source = '/home/user1/b2_example'
>>> destination = 'b2://example-mybucket-b2'

>>> source = parse_sync_folder(source, b2_api)
>>> destination = parse_sync_folder(destination, b2_api)

>>> policies_manager = ScanPoliciesManager(exclude_all_symlinks=True)

>>> synchronizer = Synchronizer(
        max_workers=10,
        policies_manager=policies_manager,
        dry_run=False,
        allow_empty_source=True,
        compare_version_mode=CompareVersionMode.SIZE,
        compare_threshold=10,
        newer_file_mode=NewerFileSyncMode.REPLACE,
        keep_days_or_delete=KeepOrDeleteMode.KEEP_BEFORE_DELETE,
        keep_days=10,
    )
```

We have a file (hello.txt) which is present in destination but not on source (my local), so it will be deleted and since our mode is to keep the delete file, it will be hidden for 10 days in bucket.

```
>>> no_progress = False
>>> with SyncReport(sys.stdout, no_progress) as reporter:
        synchronizer.sync_folders(
            source_folder=source,
            dest_folder=destination,
            now_millis=int(round(time.time() * 1000)),
            reporter=reporter,
```

(continues on next page)

(continued from previous page)

```
        )
upload f1.txt
delete hello.txt (old version)
hide   hello.txt
```

We changed f1.txt and added 1 byte. Since our compare_threshold is 10, it will not do anything.

```
>>> with SyncReport(sys.stdout, no_progress) as reporter:
        synchronizer.sync_folders(
            source_folder=source,
            dest_folder=destination,
            now_millis=int(round(time.time() * 1000)),
            reporter=reporter,
        )
```

We changed f1.txt and added more than 10 bytes. Since our compare_threshold is 10, it will replace the file at destination folder.

```
>>> with SyncReport(sys.stdout, no_progress) as reporter:
        synchronizer.sync_folders(
            source_folder=source,
            dest_folder=destination,
            now_millis=int(round(time.time() * 1000)),
            reporter=reporter,
        )
upload f1.txt
```

Let's just delete the file and not keep - keep_days_or_delete = DELETE You can avoid passing keep_days argument in this case because it will be ignored anyways

```
>>> synchronizer = Synchronizer(
        max_workers=10,
        policies_manager=policies_manager,
        dry_run=False,
        allow_empty_source=True,
        compare_version_mode=CompareVersionMode.SIZE,
        compare_threshold=10,  # in bytes
        newer_file_mode=NewerFileSyncMode.REPLACE,
        keep_days_or_delete=KeepOrDeleteMode.DELETE,
    )

>>> with SyncReport(sys.stdout, no_progress) as reporter:
    synchronizer.sync_folders(
        source_folder=source,
        dest_folder=destination,
        now_millis=int(round(time.time() * 1000)),
        reporter=reporter,
    )
delete f1.txt
delete f1.txt (old version)
delete hello.txt (old version)
upload f2.txt
delete hello.txt (hide marker)
```

As you can see, it deleted f1.txt and it's older versions (no hide this time) and deleted hello.txt also because now we don't want the file anymore. also, we added another file f2.txt which gets uploaded.

Now we changed newer_file_mode to SKIP and compare_version_mode to MODTIME. also uploaded a new version of f2.txt to bucket using B2 web.

```
>>> synchronizer = Synchronizer(
        max_workers=10,
        policies_manager=policies_manager,
        dry_run=False,
        allow_empty_source=True,
        compare_version_mode=CompareVersionMode.MODTIME,
        compare_threshold=10,  # in seconds
        newer_file_mode=NewerFileSyncMode.SKIP,
        keep_days_or_delete=KeepOrDeleteMode.DELETE,
    )
>>> with SyncReport(sys.stdout, no_progress) as reporter:
    synchronizer.sync_folders(
        source_folder=source,
        dest_folder=destination,
        now_millis=int(round(time.time() * 1000)),
        reporter=reporter,
    )
```

As expected, nothing happened, it found a file that was older at source but did not do anything because we skipped.

Now we changed newer_file_mode again to REPLACE and also uploaded a new version of f2.txt to bucket using B2 web.

```
>>> synchronizer = Synchronizer(
        max_workers=10,
        policies_manager=policies_manager,
        dry_run=False,
        allow_empty_source=True,
        compare_version_mode=CompareVersionMode.MODTIME,
        compare_threshold=10,
        newer_file_mode=NewerFileSyncMode.REPLACE,
        keep_days_or_delete=KeepOrDeleteMode.DELETE,
    )
>>> with SyncReport(sys.stdout, no_progress) as reporter:
    synchronizer.sync_folders(
        source_folder=source,
        dest_folder=destination,
        now_millis=int(round(time.time() * 1000)),
        reporter=reporter,
    )
delete f2.txt (old version)
upload f2.txt
```

Handling encryption

The *Synchronizer* object may need *EncryptionSetting* instances to perform downloads and copies. For this reason, the *sync_folder* method accepts an *EncryptionSettingsProvider*, see *Server-Side Encryption* for details.

class b2sdk.v1.**ScanPoliciesManager**

Policy object used when scanning folders for syncing, used to decide which files to include in the list of files to be synced.

Code that scans through files should at least use should_exclude_file() to decide whether each file should be included; it will check include/exclude patterns for file names, as well as patterns for excluding directories.

Code that scans may optionally use should_exclude_directory() to test whether it can skip a directory completely and not bother listing the files and sub-directories in it.

__init__(*exclude_dir_regexes=()*, *exclude_file_regexes=()*, *include_file_regexes=()*, *exclude_all_symlinks=False*, *exclude_modified_before=None*, *exclude_modified_after=None*)

Parameters

- **exclude_dir_regexes** (*tuple*) – a tuple of regexes to exclude directories
- **exclude_file_regexes** (*tuple*) – a tuple of regexes to exclude files
- **include_file_regexes** (*tuple*) – a tuple of regexes to include files
- **exclude_all_symlinks** (*bool*) – if True, exclude all symlinks
- **exclude_modified_before** (*int, optional*) – optionally exclude file versions modified before (in millis)
- **exclude_modified_after** (*int, optional*) – optionally exclude file versions modified after (in millis)

should_exclude_file(*file_path*)

Given the full path of a file, decide if it should be excluded from the scan.

Parameters **file_path** – the path of the file, relative to the root directory being scanned.

Type str

Returns True if excluded.

Return type bool

should_exclude_file_version(*file_version*)

Given the modification time of a file version, decide if it should be excluded from the scan.

Parameters **file_version** – the file version object

Type b2sdk.v1.FileVersion

Returns True if excluded.

Return type bool

should_exclude_directory(*dir_path*)

Given the full path of a directory, decide if all of the files in it should be excluded from the scan.

Parameters **dir_path** (*str*) – the path of the directory, relative to the root directory being scanned. The path will never end in '/'.

Returns True if excluded.

class b2sdk.v1.**Synchronizer**

Copies multiple "files" from source to destination. Optionally deletes or hides destination files that the source does not have.

The synchronizer can copy files:

- From a B2 bucket to a local destination.
- From a local source to a B2 bucket.
- From one B2 bucket to another.
- Between different folders in the same B2 bucket. It will sync only the latest versions of files.

By default, the synchronizer:

- Fails when the specified source directory doesn't exist or is empty. (see `allow_empty_source` argument)
- Fails when the source is newer. (see `newer_file_mode` argument)
- Doesn't delete a file if it's present on the destination but not on the source. (see `keep_days_or_delete` and `keep_days` arguments)
- Compares files based on modification time. (see `compare_version_mode` and `compare_threshold` arguments)

__init__(*max_workers*, *policies_manager=<b2sdk.sync.scan_policies.ScanPoliciesManager object>*, *dry_run=False*, *allow_empty_source=False*, *newer_file_mode=<NewerFileSyncMode.RAISE_ERROR: 103>*, *keep_days_or_delete=<KeepOrDeleteMode.NO_DELETE: 303>*, *compare_version_mode=<CompareVersionMode.MODTIME: 201>*, *compare_threshold=None*, *keep_days=None*)

Initialize synchronizer class and validate arguments

Parameters

- **max_workers** (*int*) – max number of workers
- **policies_manager** – policies manager object
- **dry_run** (*bool*) – test mode, does not actually transfer/delete when enabled
- **allow_empty_source** (*bool*) – if True, do not check whether source folder is empty
- **newer_file_mode** (b2sdk.v1.NewerFileSyncMode) – setting which determines handling for destination files newer than on the source
- **keep_days_or_delete** (b2sdk.v1.KeepOrDeleteMode) – setting which determines if we should delete or not delete or keep for *keep_days*
- **compare_version_mode** (b2sdk.v1.CompareVersionMode) – how to compare the source and destination files to find new ones
- **compare_threshold** (*int*) – should be greater than 0, default is 0
- **keep_days** (*int*) – if keep_days_or_delete is *b2sdk.v1.KeepOrDeleteMode.KEEP_BEFORE_DELETE*, then this should be greater than 0

sync_folders(*source_folder*, *dest_folder*, *now_millis*, *reporter*, *encryption_settings_provider: b2sdk.sync.encryption_provider.AbstractSyncEncryptionSettingsProvider = <b2sdk.sync.encryption_provider.ServerDefaultSyncEncryptionSettingsProvider object>*)

Syncs two folders. Always ensures that every file in the source is also in the destination. Deletes any file versions in the destination older than history_days.

Parameters

- **source_folder** (b2sdk.sync.folder.AbstractFolder) – source folder object
- **dest_folder** (b2sdk.sync.folder.AbstractFolder) – destination folder object
- **now_millis** (*int*) – current time in milliseconds
- **reporter** (*b2sdk.sync.report.SyncReport*, *None*) – progress reporter
- **encryption_settings_provider** (*b2sdk.v1.AbstractSyncEncryptionSettingsProvider*) – encryption setting provider

make_folder_sync_actions(*source_folder*, *dest_folder*, *now_millis*, *reporter*, *policies_manager=<b2sdk.sync.scan_policies.ScanPoliciesManager object>*, *encryption_settings_provider: b2sdk.sync.encryption_provider.AbstractSyncEncryptionSettingsProvider = <b2sdk.sync.encryption_provider.ServerDefaultSyncEncryptionSettingsProvider object>*)

Yield a sequence of actions that will sync the destination folder to the source folder.

Parameters

- **source_folder** (*b2sdk.v1.AbstractFolder*) – source folder object
- **dest_folder** (*b2sdk.v1.AbstractFolder*) – destination folder object
- **now_millis** (*int*) – current time in milliseconds
- **reporter** (b2sdk.v1.SyncReport) – reporter object
- **policies_manager** – policies manager object
- **encryption_settings_provider** (*b2sdk.v1.AbstractSyncEncryptionSettingsProvider*) – encryption setting provider

make_file_sync_actions(*sync_type*, *source_file*, *dest_file*, *source_folder*, *dest_folder*, *now_millis*, *encryption_settings_provider: b2sdk.sync.encryption_provider.AbstractSyncEncryptionSettingsProvider = <b2sdk.sync.encryption_provider.ServerDefaultSyncEncryptionSettingsProvider object>*)

Yields the sequence of actions needed to sync the two files

Parameters

- **sync_type** (*str*) – synchronization type
- **source_file** (*b2sdk.v1.File*) – source file object
- **dest_file** (*b2sdk.v1.File*) – destination file object
- **source_folder** (*b2sdk.v1.AbstractFolder*) – a source folder object
- **dest_folder** (*b2sdk.v1.AbstractFolder*) – a destination folder object
- **now_millis** (*int*) – current time in milliseconds
- **encryption_settings_provider** (*b2sdk.v1.AbstractSyncEncryptionSettingsProvider*) – encryption setting provider

class b2sdk.v1.**SyncReport**

Handle reporting progress for syncing.

Print out each file as it is processed, and puts up a sequence of progress bars.

The progress bars are:

- Step 1/1: count local files
- Step 2/2: compare file lists
- Step 3/3: transfer files

This class is THREAD SAFE, so it can be used from parallel sync threads.

UPDATE_INTERVAL = 0.1

__init__(*stdout*, *no_progress*)

Parameters

- **stdout** – standard output file object
- **no_progress** (*bool*) – if True, do not show progress

close()
Perform a clean-up.

error(*message*)
Print an error, gracefully interleaving it with a progress bar.

Parameters **message** (*str*) – an error message

print_completion(*message*)
Remove the progress bar, prints a message, and puts the progress bar back.

Parameters **message** (*str*) – an error message

update_total(*delta*)
Report that more files have been found for comparison.

Parameters **delta** (*int*) – number of files found since the last check

end_total()
Total files count is done. Can proceed to step 2.

update_compare(*delta*)
Report that more files have been compared.

Parameters **delta** (*int*) – number of files compared

end_compare(*total_transfer_files*, *total_transfer_bytes*)
Report that the comparison has been finished.

Parameters

- **total_transfer_files** (*int*) – total number of transferred files
- **total_transfer_bytes** (*int*) – total number of transferred bytes

update_transfer(*file_delta*, *byte_delta*)
Update transfer info.

Parameters

- **file_delta** (*int*) – number of files transferred
- **byte_delta** (*int*) – number of bytes transferred

local_access_error(*path*)
Add a file access error message to the list of warnings.

Parameters **path** (*str*) – file path

local_permission_error(*path*)
Add a permission error message to the list of warnings.

Parameters **path** (*str*) – file path

symlink_skipped(*path*)

property local_file_count

property local_done

update_local(*delta*)
Report that more files have been found for comparison.

Parameters **delta** (*int*) – number of files found since the last check

end_local()
Total files count is done. Can proceed to step 2.

B2 Utility functions

b2sdk.v1.**b2_url_encode**(*s*)
URL-encode a unicode string to be sent to B2 in an HTTP header.

Parameters **s** (*str*) – a unicode string to encode

Returns URL-encoded string

Return type str

b2sdk.v1.**b2_url_decode**(*s*)
Decode a Unicode string returned from B2 in an HTTP header.

Parameters **s** (*str*) – a unicode string to decode

Returns a Python unicode string.

Return type str

b2sdk.v1.**choose_part_ranges**(*content_length*, *minimum_part_size*)
Return a list of (offset, length) for the parts of a large file.

Parameters

- **content_length** (*int*) – content length value
- **minimum_part_size** (*int*) – a minimum file part size

Return type list

b2sdk.v1.**fix_windows_path_limit**(*path*)
Prefix paths when running on Windows to overcome 260 character path length limit. See https://msdn.microsoft.com/en-us/library/windows/desktop/aa365247(v=vs.85).aspx#maxpath

Parameters **path** (*str*) – a path to prefix

Returns a prefixed path

Return type str

b2sdk.v1.**format_and_scale_fraction**(*numerator*, *denominator*, *unit*)
Pick a good scale for representing a fraction, and format it.

Parameters

- **numerator** (*int*) – a numerator of a fraction
- **denominator** (*int*) – a denominator of a fraction
- **unit** (*str*) – an arbitrary unit name

Returns scaled and formatted fraction

Return type str

b2sdk.v1.**format_and_scale_number**(*x*, *unit*)
Pick a good scale for representing a number and format it.

Parameters

- **x** (*int*) – a number
- **unit** (*str*) – an arbitrary unit name

Returns scaled and formatted number

Return type str

b2sdk.v1.**hex_sha1_of_stream**(*input_stream*, *content_length*)
Return the 40-character hex SHA1 checksum of the first content_length bytes in the input stream.

Parameters

- **input_stream** – stream object, which exposes read() method
- **content_length** (*int*) – expected length of the stream

Return type str

b2sdk.v1.**hex_sha1_of_bytes**(*data: bytes*) → str
Return the 40-character hex SHA1 checksum of the data.

class b2sdk.v1.**TempDir**
Context manager that creates and destroys a temporary directory.

__enter__()
Return the unicode path to the temp dir.

__exit__(*exc_type*, *exc_val*, *exc_tb*)

Write intent

class b2sdk.v1.**WriteIntent**
Wrapper for outbound source that defines destination offset.

__init__(*outbound_source*, *destination_offset=0*)

Parameters

- **outbound_source** (b2sdk.v1.OutboundTransferSource) – data source (remote or local)
- **destination_offset** (*int*) – point of start in destination file

property length
Length of the write intent.

Return type int

property destination_end_offset
Offset of source end in destination file.

Return type int

is_copy()
States if outbound source is remote source and requires copying.

Return type bool

is_upload()
States if outbound source is local source and requires uploading.

Return type bool

classmethod wrap_sources_iterator(*outbound_sources_iterator*)
Helper that wraps outbound sources iterator with write intents.

Can be used in cases similar to concatenate to automatically compute destination offsets

Param iterator[b2sdk.v1.OutboundTransferSource] outbound_sources_iterator: iterator of outbound sources

Return type generator[*b2sdk.v1.WriteIntent*]

Outbound Transfer Source

class b2sdk.v1.**OutboundTransferSource**

Abstract class for defining outbound transfer sources.

Supported outbound transfer sources are:

- `b2sdk.v1.CopySource`
- `b2sdk.v1.UploadSourceBytes`
- `b2sdk.v1.UploadSourceLocalFile`
- `b2sdk.v1.UploadSourceLocalFileRange`
- `b2sdk.v1.UploadSourceStream`
- `b2sdk.v1.UploadSourceStreamRange`

abstract get_content_length()

Return the number of bytes of data in the file.

abstract is_upload()

Return if outbound source is an upload source. :rtype bool:

abstract is_copy()

Return if outbound source is a copy source. :rtype bool:

Download destination

Note: Concrete classes described in this chapter implement methods defined in `AbstractDownloadDestination`

class b2sdk.v1.**AbstractDownloadDestination**

Interface to a destination for a downloaded file.

abstract make_file_context(*file_id*, *file_name*, *content_length*, *content_type*, *content_sha1*, *file_info*, *mod_time_millis*, *range_=None*)

Return a context manager that yields a binary file-like object to use for writing the contents of the file.

Parameters

- **file_id** (*str*) – the B2 file ID from the headers
- **file_name** (*str*) – the B2 file name from the headers
- **content_length** (*str*) – the content length
- **content_type** (*str*) – the content type from the headers
- **content_sha1** (*str*) – the content sha1 from the headers (or `"none"` for large files)
- **file_info** (*dict*) – the user file info from the headers
- **mod_time_millis** (*int*) – the desired file modification date in ms since 1970-01-01

- **range** (*None*, *tuple[int, int]*) – starting and ending offsets of the received file contents. Usually `None`, which means that the whole file is downloaded.

Returns None

class b2sdk.v1.**DownloadDestLocalFile**
Store a downloaded file into a local file and sets its modification time.

__init__(*local_file_path*)
Initialize self. See help(type(self)) for accurate signature.

class b2sdk.v1.**DownloadDestBytes**
Store a downloaded file into bytes in memory.

__init__()
Initialize self. See help(type(self)) for accurate signature.

class b2sdk.v1.**DownloadDestProgressWrapper**
Wrap a DownloadDestination and report progress to a ProgressListener.

__init__(*download_dest*, *progress_listener*)
Initialize self. See help(type(self)) for accurate signature.

class b2sdk.v1.**PreSeekedDownloadDest**
Store a downloaded file into a local file and sets its modification time. Does not truncate the target file, seeks to a given offset just after opening a descriptor.

__init__(*local_file_path*, *seek_target*)
Initialize self. See help(type(self)) for accurate signature.

2.8.3 Internal API

Note: See *Internal interface* chapter to learn when and how to safely use the Internal API

b2sdk.session – B2 Session

class b2sdk.session.**TokenType**(*value*)
Bases: enum.Enum

An enumeration.

API = 'api'

API_TOKEN_ONLY = 'api_token_only'

UPLOAD_PART = 'upload_part'

UPLOAD_SMALL = 'upload_small'

class b2sdk.session.**B2Session**(*account_info=None*, *cache=None*, *raw_api=None*)
Bases: object

A facade that supplies the correct api_url and account_auth_token to methods of underlying raw_api and reauthorizes if necessary.

__init__(*account_info=None*, *cache=None*, *raw_api=None*)
Initialize Session using given account info.

Parameters

- **account_info** – an instance of *UrlPoolAccountInfo*, or any custom class derived from *AbstractAccountInfo* To learn more about Account Info objects, see here *SqliteAccountInfo*
- **cache** – an instance of the one of the following classes: *DummyCache*, *InMemoryCache*, *AuthInfoCache*, or any custom class derived from *AbstractCache* It is used by B2Api to cache the mapping between bucket name and bucket ids. default is *DummyCache*
- **raw_api** – an instance of one of the following classes: *B2RawApi*, *RawSimulator*, or any custom class derived from *AbstractRawApi* It makes network-less unit testing simple by using *RawSimulator*, in tests and *B2RawApi* in production. default is *B2RawApi*

authorize_automatically()
Perform automatic account authorization, retrieving all account data from account info object passed during initialization.

authorize_account(*realm*, *application_key_id*, *application_key*)
Perform account authorization.

Parameters

- **realm** (*str*) – a realm to authorize account in (usually just "production")
- **application_key_id** (*str*) – *application key ID*
- **application_key** (*str*) – user's *application key*

cancel_large_file(*file_id*)

create_bucket(*account_id*, *bucket_name*, *bucket_type*, *bucket_info=None*, *cors_rules=None*, *lifecycle_rules=None*, *default_server_side_encryption=None*)

create_key(*account_id*, *capabilities*, *key_name*, *valid_duration_seconds*, *bucket_id*, *name_prefix*)

delete_key(*application_key_id*)

delete_bucket(*account_id*, *bucket_id*)

delete_file_version(*file_id*, *file_name*)

download_file_from_url(*url*, *range_=None*, *encryption: Optional[*b2sdk.encryption.setting.EncryptionSetting*] = None*)

finish_large_file(*file_id*, *part_sha1_array*)

get_download_authorization(*bucket_id*, *file_name_prefix*, *valid_duration_in_seconds*)

get_file_info_by_id(*file_id: str*) → Dict[str, Any]

get_file_info_by_name(*bucket_name: str*, *file_name: str*) → Dict[str, Any]

get_upload_url(*bucket_id*)

get_upload_part_url(*file_id*)

hide_file(*bucket_id*, *file_name*)

list_buckets(*account_id*, *bucket_id=None*, *bucket_name=None*)

list_file_names(*bucket_id*, *start_file_name=None*, *max_file_count=None*, *prefix=None*)

list_file_versions(*bucket_id*, *start_file_name=None*, *start_file_id=None*, *max_file_count=None*, *prefix=None*)

list_keys(*account_id*, *max_key_count=None*, *start_application_key_id=None*)

list_parts(*file_id*, *start_part_number*, *max_part_count*)

list_unfinished_large_files(*bucket_id*, *start_file_id=None*, *max_file_count=None*, *prefix=None*)

start_large_file(*bucket_id*, *file_name*, *content_type*, *file_info*, *server_side_encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*)

update_bucket(*account_id*, *bucket_id*, *bucket_type=None*, *bucket_info=None*, *cors_rules=None*, *lifecycle_rules=None*, *if_revision_is=None*, *default_server_side_encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*)

upload_file(*bucket_id*, *file_name*, *content_length*, *content_type*, *content_sha1*, *file_infos*, *data_stream*, *server_side_encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*)

upload_part(*file_id*, *part_number*, *content_length*, *sha1_sum*, *input_stream*, *server_side_encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*)

get_download_url_by_id(*file_id*)

get_download_url_by_name(*bucket_name*, *file_name*)

copy_file(*source_file_id*, *new_file_name*, *bytes_range=None*, *metadata_directive=None*, *content_type=None*, *file_info=None*, *destination_bucket_id=None*, *destination_server_side_encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*, *source_server_side_encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*)

copy_part(*source_file_id*, *large_file_id*, *part_number*, *bytes_range=None*, *destination_server_side_encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*, *source_server_side_encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*)

b2sdk.raw_api – B2 raw api wrapper

class b2sdk.raw_api.**MetadataDirectiveMode**(*value*)

Bases: enum.Enum

Mode of handling metadata when copying a file

COPY = 401

copy metadata from the source file

REPLACE = 402

ignore the source file metadata and set it to provided values

class b2sdk.raw_api.**AbstractRawApi**

Bases: object

Direct access to the B2 web apis.

abstract authorize_account(*realm_url*, *application_key_id*, *application_key*)

abstract cancel_large_file(*api_url*, *account_auth_token*, *file_id*)

abstract copy_file(*api_url*, *account_auth_token*, *source_file_id*, *new_file_name*, *bytes_range=None*, *metadata_directive=None*, *content_type=None*, *file_info=None*, *destination_bucket_id=None*, *destination_server_side_encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*, *source_server_side_encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*)

abstract copy_part(*api_url*, *account_auth_token*, *source_file_id*, *large_file_id*, *part_number*, *bytes_range=None*, *destination_server_side_encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*, *source_server_side_encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*)

abstract create_bucket(*api_url*, *account_auth_token*, *account_id*, *bucket_name*, *bucket_type*, *bucket_info=None*, *cors_rules=None*, *lifecycle_rules=None*, *default_server_side_encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*)

abstract create_key(*api_url*, *account_auth_token*, *account_id*, *capabilities*, *key_name*, *valid_duration_seconds*, *bucket_id*, *name_prefix*)

abstract download_file_from_url(*account_auth_token_or_none*, *url*, *range_=None*, *encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*)

abstract delete_key(*api_url*, *account_auth_token*, *application_key_id*)

abstract delete_bucket(*api_url*, *account_auth_token*, *account_id*, *bucket_id*)

abstract delete_file_version(*api_url*, *account_auth_token*, *file_id*, *file_name*)

abstract finish_large_file(*api_url*, *account_auth_token*, *file_id*, *part_sha1_array*)

abstract get_download_authorization(*api_url*, *account_auth_token*, *bucket_id*, *file_name_prefix*, *valid_duration_in_seconds*)

abstract get_file_info_by_id(*api_url: str*, *account_auth_token: str*, *file_id: str*) → Dict[str, Any]

abstract get_file_info_by_name(*download_url: str*, *account_auth_token: str*, *bucket_name: str*, *file_name: str*) → Dict[str, Any]

abstract get_upload_url(*api_url*, *account_auth_token*, *bucket_id*)

abstract get_upload_part_url(*api_url*, *account_auth_token*, *file_id*)

abstract hide_file(*api_url*, *account_auth_token*, *bucket_id*, *file_name*)

abstract list_buckets(*api_url*, *account_auth_token*, *account_id*, *bucket_id=None*, *bucket_name=None*)

abstract list_file_names(*api_url*, *account_auth_token*, *bucket_id*, *start_file_name=None*, *max_file_count=None*, *prefix=None*)

abstract list_file_versions(*api_url*, *account_auth_token*, *bucket_id*, *start_file_name=None*, *start_file_id=None*, *max_file_count=None*, *prefix=None*)

abstract list_keys(*api_url*, *account_auth_token*, *account_id*, *max_key_count=None*, *start_application_key_id=None*)

abstract list_parts(*api_url*, *account_auth_token*, *file_id*, *start_part_number*, *max_part_count*)

abstract list_unfinished_large_files(*api_url*, *account_auth_token*, *bucket_id*, *start_file_id=None*, *max_file_count=None*, *prefix=None*)

abstract start_large_file(*api_url*, *account_auth_token*, *bucket_id*, *file_name*, *content_type*, *file_info*, *server_side_encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*)

abstract update_bucket(*api_url*, *account_auth_token*, *account_id*, *bucket_id*, *bucket_type=None*, *bucket_info=None*, *cors_rules=None*, *lifecycle_rules=None*, *if_revision_is=None*, *default_server_side_encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*)

abstract upload_file(*upload_url*, *upload_auth_token*, *file_name*, *content_length*, *content_type*, *content_sha1*, *file_infos*, *data_stream*, *server_side_encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*)

abstract upload_part(*upload_url*, *upload_auth_token*, *part_number*, *content_length*, *sha1_sum*, *input_stream*, *server_side_encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*)

get_download_url_by_id(*download_url*, *file_id*)

get_download_url_by_name(*download_url*, *bucket_name*, *file_name*)

class b2sdk.raw_api.**B2RawApi**(*b2_http*)

Bases: *b2sdk.raw_api.AbstractRawApi*

Provide access to the B2 web APIs, exactly as they are provided by b2.

Requires that you provide all necessary URLs and auth tokens for each call.

Each API call decodes the returned JSON and returns a dict.

For details on what each method does, see the B2 docs: https://www.backblaze.com/b2/docs/

This class is intended to be a super-simple, very thin layer on top of the HTTP calls. It can be mocked-out for testing higher layers. And this class can be tested by exercising each call just once, which is relatively quick.

__init__(*b2_http*)

Initialize self. See help(type(self)) for accurate signature.

authorize_account(*realm_url*, *application_key_id*, *application_key*)

cancel_large_file(*api_url*, *account_auth_token*, *file_id*)

create_bucket(*api_url*, *account_auth_token*, *account_id*, *bucket_name*, *bucket_type*, *bucket_info=None*, *cors_rules=None*, *lifecycle_rules=None*, *default_server_side_encryption=None*)

create_key(*api_url*, *account_auth_token*, *account_id*, *capabilities*, *key_name*, *valid_duration_seconds*, *bucket_id*, *name_prefix*)

delete_bucket(*api_url*, *account_auth_token*, *account_id*, *bucket_id*)

delete_file_version(*api_url*, *account_auth_token*, *file_id*, *file_name*)

delete_key(*api_url*, *account_auth_token*, *application_key_id*)

download_file_from_url(*account_auth_token_or_none*, *url*, *range_=None*, *encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*)

Issue a streaming request for download of a file, potentially authorized.

Parameters

- **account_auth_token_or_none** (*str*) – an optional account auth token to pass in

- **url** (*str*) – the full URL to download from
- **range** (*tuple*) – two-element tuple for http Range header
- **encryption** (*b2sdk.v1.EncryptionSetting*) – encryption settings for downloading

Returns b2_http response

finish_large_file(*api_url*, *account_auth_token*, *file_id*, *part_sha1_array*)

get_download_authorization(*api_url*, *account_auth_token*, *bucket_id*, *file_name_prefix*, *valid_duration_in_seconds*)

get_file_info_by_id(*api_url: str*, *account_auth_token: str*, *file_id: str*) → Dict[str, Any]

get_file_info_by_name(*download_url: str*, *account_auth_token: str*, *bucket_name: str*, *file_name: str*) → Dict[str, Any]

get_upload_url(*api_url*, *account_auth_token*, *bucket_id*)

get_upload_part_url(*api_url*, *account_auth_token*, *file_id*)

hide_file(*api_url*, *account_auth_token*, *bucket_id*, *file_name*)

list_buckets(*api_url*, *account_auth_token*, *account_id*, *bucket_id=None*, *bucket_name=None*)

list_file_names(*api_url*, *account_auth_token*, *bucket_id*, *start_file_name=None*, *max_file_count=None*, *prefix=None*)

list_file_versions(*api_url*, *account_auth_token*, *bucket_id*, *start_file_name=None*, *start_file_id=None*, *max_file_count=None*, *prefix=None*)

list_keys(*api_url*, *account_auth_token*, *account_id*, *max_key_count=None*, *start_application_key_id=None*)

list_parts(*api_url*, *account_auth_token*, *file_id*, *start_part_number*, *max_part_count*)

list_unfinished_large_files(*api_url*, *account_auth_token*, *bucket_id*, *start_file_id=None*, *max_file_count=None*, *prefix=None*)

start_large_file(*api_url*, *account_auth_token*, *bucket_id*, *file_name*, *content_type*, *file_info*, *server_side_encryption: Optional[*b2sdk.encryption.setting.EncryptionSetting*] = None*)

update_bucket(*api_url*, *account_auth_token*, *account_id*, *bucket_id*, *bucket_type=None*, *bucket_info=None*, *cors_rules=None*, *lifecycle_rules=None*, *if_revision_is=None*, *default_server_side_encryption=None*)

unprintable_to_hex(*string*)

Replace unprintable chars in string with a hex representation.

Parameters **string** – an arbitrary string, possibly with unprintable characters.

Returns the string, with unprintable characters changed to hex (e.g., “”)

check_b2_filename(*filename*)

Raise an appropriate exception with details if the filename is unusable.

See https://www.backblaze.com/b2/docs/files.html for the rules.

Parameters **filename** – a proposed filename in unicode

Returns None if the filename is usable

upload_file(*upload_url*, *upload_auth_token*, *file_name*, *content_length*, *content_type*, *content_sha1*, *file_infos*, *data_stream*, *server_side_encryption: Optional[*b2sdk.encryption.setting.EncryptionSetting*] = None*)

Upload one, small file to b2.

Parameters

- **upload_url** – the upload_url from b2_authorize_account
- **upload_auth_token** – the auth token from b2_authorize_account
- **file_name** – the name of the B2 file
- **content_length** – number of bytes in the file
- **content_type** – MIME type
- **content_sha1** – hex SHA1 of the contents of the file
- **file_infos** – extra file info to upload
- **data_stream** – a file like object from which the contents of the file can be read

Returns

upload_part(*upload_url*, *upload_auth_token*, *part_number*, *content_length*, *content_sha1*, *data_stream*, *server_side_encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*)

copy_file(*api_url*, *account_auth_token*, *source_file_id*, *new_file_name*, *bytes_range=None*, *metadata_directive=None*, *content_type=None*, *file_info=None*, *destination_bucket_id=None*, *destination_server_side_encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*, *source_server_side_encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*)

copy_part(*api_url*, *account_auth_token*, *source_file_id*, *large_file_id*, *part_number*, *bytes_range=None*, *destination_server_side_encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*, *source_server_side_encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*)

b2sdk.raw_api.**test_raw_api**()

Exercise the code in B2RawApi by making each call once, just to make sure the parameters are passed in, and the result is passed back.

The goal is to be a complete test of B2RawApi, so the tests for the rest of the code can use the simulator.

Prints to stdout if things go wrong.

Returns 0 on success, non-zero on failure

b2sdk.raw_api.**test_raw_api_helper**(*raw_api*)

Try each of the calls to the raw api. Raise an exception if anything goes wrong.

This uses a Backblaze account that is just for this test. The account uses the free level of service, which should be enough to run this test a reasonable number of times each day. If somebody abuses the account for other things, this test will break and we'll have to do something about it.

b2sdk.b2http – thin http client wrapper

class b2sdk.b2http.**ResponseContextManager**(*response*)
A context manager that closes a requests.Response when done.

class b2sdk.b2http.**HttpCallback**
A callback object that does nothing. Overrides pre_request and/or post_request as desired.

pre_request(*method*, *url*, *headers*)
Called before processing an HTTP request.

Raises an exception if this request should not be processed. The exception raised must inherit from B2HttpCallbackPreRequestException.

Parameters

- **method** (*str*) – str, one of: 'POST', 'GET', etc.
- **url** (*str*) – the URL that will be used
- **headers** (*dict*) – the header sent with the request

post_request(*method*, *url*, *headers*, *response*)
Called after processing an HTTP request. Should not raise an exception.

Raises an exception if this request should be treated as failing. The exception raised must inherit from B2HttpCallbackPostRequestException.

Parameters

- **method** (*str*) – one of: 'POST', 'GET', etc.
- **url** (*str*) – the URL that will be used
- **headers** (*dict*) – the header sent with the request
- **response** – a response object from the requests library

class b2sdk.b2http.**ClockSkewHook**

post_request(*method*, *url*, *headers*, *response*)
Raise an exception if the clock in the server is too different from the clock on the local host.

The Date header contains a string that looks like: "Fri, 16 Dec 2016 20:52:30 GMT".

Parameters

- **method** (*str*) – one of: 'POST', 'GET', etc.
- **url** (*str*) – the URL that will be used
- **headers** (*dict*) – the header sent with the request
- **response** – a response object from the requests library

class b2sdk.b2http.**B2Http**(*requests_module=None*, *install_clock_skew_hook=True*, *user_agent_append=None*)
A wrapper for the requests module. Provides the operations needed to access B2, and handles retrying when the returned status is 503 Service Unavailable, 429 Too Many Requests, etc.

The operations supported are:

- post_json_return_json
- post_content_return_json
- get_content

The methods that return JSON either return a Python dict or raise a subclass of B2Error. They can be used like this:

```
try:
    response_dict = b2_http.post_json_return_json(url, headers, params)
    ...
except B2Error as e:
    ...
```

TIMEOUT = 900

add_callback(*callback*)
Add a callback that inherits from HttpCallback.

Parameters **callback** (*callable*) – a callback to be added to a chain

post_content_return_json(*url*, *headers*, *data*, *try_count=5*, *post_params=None*)
Use like this:

```
try:
    response_dict = b2_http.post_content_return_json(url, headers, data)
    ...
except B2Error as e:
    ...
```

Parameters

- **url** (*str*) – a URL to call
- **headers** (*dict*) – headers to send.
- **data** – bytes (Python 3) or str (Python 2), or a file-like object, to send

Returns a dict that is the decoded JSON

Return type dict

post_json_return_json(*url*, *headers*, *params*, *try_count=5*)
Use like this:

```
try:
    response_dict = b2_http.post_json_return_json(url, headers, params)
    ...
except B2Error as e:
    ...
```

Parameters

- **url** (*str*) – a URL to call
- **headers** (*dict*) – headers to send.
- **params** (*dict*) – a dict that will be converted to JSON

Returns the decoded JSON document

Return type dict

get_content(*url*, *headers*, *try_count=5*)
Fetches content from a URL.

Use like this:

```
try:
    with b2_http.get_content(url, headers) as response:
        for byte_data in response.iter_content(chunk_size=1024):
            ...
except B2Error as e:
    ...
```

The response object is only guarantee to have:

- headers
- iter_content()

Parameters

- **url** (*str*) – a URL to call
- **headers** (*dict*) – headers to send
- **try_count** (*int*) – a number or retries

Returns Context manager that returns an object that supports iter_content()

head_content(*url: str*, *headers: Dict[str, Any]*, *try_count: int = 5*) → Dict[str, Any]

Does a HEAD instead of a GET for the URL. The response's content is limited to the headers.

Use like this:

```
try:
    response_dict = b2_http.head_content(url, headers)
    ...
except B2Error as e:
    ...
```

The response object is only guaranteed to have:

- headers

Parameters

- **url** (*str*) – a URL to call
- **headers** (*dict*) – headers to send
- **try_count** (*int*) – a number or retries

Returns the decoded response

Return type dict

b2sdk.b2http.**test_http**()

Run a few tests on error diagnosis.

This test takes a while to run and is not used in the automated tests during building. Run the test by hand to exercise the code.

b2sdk.utils

b2sdk.utils.**interruptible_get_result**(*future*)
Wait for the result of a future in a way that can be interrupted by a KeyboardInterrupt.

This is not necessary in Python 3, but is needed for Python 2.

Parameters **future** (*Future*) – a future to get result of

b2sdk.utils.**b2_url_encode**(*s*)
URL-encode a unicode string to be sent to B2 in an HTTP header.

Parameters **s** (*str*) – a unicode string to encode

Returns URL-encoded string

Return type str

b2sdk.utils.**b2_url_decode**(*s*)
Decode a Unicode string returned from B2 in an HTTP header.

Parameters **s** (*str*) – a unicode string to decode

Returns a Python unicode string.

Return type str

b2sdk.utils.**choose_part_ranges**(*content_length*, *minimum_part_size*)
Return a list of (offset, length) for the parts of a large file.

Parameters

- **content_length** (*int*) – content length value
- **minimum_part_size** (*int*) – a minimum file part size

Return type list

b2sdk.utils.**hex_sha1_of_stream**(*input_stream*, *content_length*)
Return the 40-character hex SHA1 checksum of the first content_length bytes in the input stream.

Parameters

- **input_stream** – stream object, which exposes read() method
- **content_length** (*int*) – expected length of the stream

Return type str

b2sdk.utils.**hex_sha1_of_unlimited_stream**(*input_stream*, *limit=None*)

b2sdk.utils.**hex_sha1_of_bytes**(*data: bytes*) → str
Return the 40-character hex SHA1 checksum of the data.

b2sdk.utils.**hex_md5_of_bytes**(*data: bytes*) → str
Return the 32-character hex MD5 checksum of the data.

b2sdk.utils.**md5_of_bytes**(*data: bytes*) → bytes
Return the 16-byte MD5 checksum of the data.

b2sdk.utils.**b64_of_bytes**(*data: bytes*) → str
Return the base64 encoded represtantion of the data.

b2sdk.utils.**validate_b2_file_name**(*name*)
Raise a ValueError if the name is not a valid B2 file name.

Parameters **name** (*str*) – a string to check

b2sdk.utils.**is_file_readable**(*local_path*, *reporter=None*)
Check if the local file has read permissions.

Parameters

- **local_path** (*str*) – a file path
- **reporter** – reporter object to put errors on

Return type bool

b2sdk.utils.**get_file_mtime**(*local_path*)
Get modification time of a file in milliseconds.

Parameters **local_path** (*str*) – a file path

Return type int

b2sdk.utils.**set_file_mtime**(*local_path*, *mod_time_millis*)
Set modification time of a file in milliseconds.

Parameters

- **local_path** (*str*) – a file path
- **mod_time_millis** (*int*) – time to be set

b2sdk.utils.**fix_windows_path_limit**(*path*)
Prefix paths when running on Windows to overcome 260 character path length limit. See https://msdn.microsoft.com/en-us/library/windows/desktop/aa365247(v=vs.85).aspx#maxpath

Parameters **path** (*str*) – a path to prefix

Returns a prefixed path

Return type str

class b2sdk.utils.**TempDir**
Bases: object

Context manager that creates and destroys a temporary directory.

b2sdk.utils.**format_and_scale_number**(*x*, *unit*)
Pick a good scale for representing a number and format it.

Parameters

- **x** (*int*) – a number
- **unit** (*str*) – an arbitrary unit name

Returns scaled and formatted number

Return type str

b2sdk.utils.**format_and_scale_fraction**(*numerator*, *denominator*, *unit*)
Pick a good scale for representing a fraction, and format it.

Parameters

- **numerator** (*int*) – a numerator of a fraction
- **denominator** (*int*) – a denominator of a fraction
- **unit** (*str*) – an arbitrary unit name

Returns scaled and formatted fraction

Return type str

b2sdk.utils.**camelcase_to_underscore**(*input_*)
Convert a camel-cased string to a string with underscores.

Parameters **input** (*str*) – an input string

Returns string with underscores

Return type str

class b2sdk.utils.**B2TraceMeta**(*name*, *bases*, *attrs*, ***kwargs*)
Bases: logfury.v0_1.meta.DefaultTraceMeta

Trace all public method calls, except for ones with names that begin with *get_*.

class b2sdk.utils.**B2TraceMetaAbstract**(*name*, *bases*, *namespace*, ***kwargs*)
Bases: logfury.v0_1.meta.DefaultTraceAbstractMeta

Default class for tracers, to be set as a metaclass for abstract base classes.

class b2sdk.utils.**ConcurrentUsedAuthTokenGuard**(*lock*, *token*)
Bases: object

Context manager preventing two tokens being used simultaneously. Throws UploadTokenUsedConcurrently when unable to acquire a lock Sample usage:

with ConcurrentUsedAuthTokenGuard(lock_for_token, token): # code that uses the token exclusively

__init__(*lock*, *token*)
Initialize self. See help(type(self)) for accurate signature.

b2sdk.cache

class b2sdk.cache.**AbstractCache**
Bases: object

clear()

abstract get_bucket_id_or_none_from_bucket_name(*name*)

abstract get_bucket_name_or_none_from_allowed()

abstract save_bucket(*bucket*)

abstract set_bucket_name_cache(*buckets*)

class b2sdk.cache.**DummyCache**
Bases: *b2sdk.cache.AbstractCache*

A cache that does nothing.

get_bucket_id_or_none_from_bucket_name(*name*)

get_bucket_name_or_none_from_allowed()

save_bucket(*bucket*)

set_bucket_name_cache(*buckets*)

class b2sdk.cache.**InMemoryCache**
Bases: *b2sdk.cache.AbstractCache*

A cache that stores the information in memory.

__init__()
Initialize self. See help(type(self)) for accurate signature.

get_bucket_id_or_none_from_bucket_name(*name*)

get_bucket_name_or_none_from_allowed()

save_bucket(*bucket*)

set_bucket_name_cache(*buckets*)

class b2sdk.cache.**AuthInfoCache**(*info*)
Bases: *b2sdk.cache.AbstractCache*

A cache that stores data persistently in StoredAccountInfo.

__init__(*info*)
Initialize self. See help(type(self)) for accurate signature.

get_bucket_id_or_none_from_bucket_name(*name*)

get_bucket_name_or_none_from_allowed()

save_bucket(*bucket*)

set_bucket_name_cache(*buckets*)

b2sdk.download_dest – Download destination

class b2sdk.download_dest.**AbstractDownloadDestination**
Bases: object

Interface to a destination for a downloaded file.

abstract make_file_context(*file_id*, *file_name*, *content_length*, *content_type*, *content_sha1*, *file_info*, *mod_time_millis*, *range_=None*)
Return a context manager that yields a binary file-like object to use for writing the contents of the file.

Parameters

- **file_id** (*str*) – the B2 file ID from the headers
- **file_name** (*str*) – the B2 file name from the headers
- **content_length** (*str*) – the content length
- **content_type** (*str*) – the content type from the headers
- **content_sha1** (*str*) – the content sha1 from the headers (or "none" for large files)
- **file_info** (*dict*) – the user file info from the headers
- **mod_time_millis** (*int*) – the desired file modification date in ms since 1970-01-01
- **range** (*None*,*tuple[int*,*int]*) – starting and ending offsets of the received file contents. Usually None, which means that the whole file is downloaded.

Returns None

class b2sdk.download_dest.**DownloadDestLocalFile**(*local_file_path*)
Bases: *b2sdk.download_dest.AbstractDownloadDestination*

Store a downloaded file into a local file and sets its modification time.

MODE = 'wb+'

__init__(*local_file_path*)
Initialize self. See help(type(self)) for accurate signature.

make_file_context(*file_id*, *file_name*, *content_length*, *content_type*, *content_sha1*, *file_info*, *mod_time_millis*, *range_=None*)
Return a context manager that yields a binary file-like object to use for writing the contents of the file.

Parameters

- **file_id** (*str*) – the B2 file ID from the headers
- **file_name** (*str*) – the B2 file name from the headers
- **content_length** (*str*) – the content length
- **content_type** (*str*) – the content type from the headers
- **content_sha1** (*str*) – the content sha1 from the headers (or "none" for large files)
- **file_info** (*dict*) – the user file info from the headers
- **mod_time_millis** (*int*) – the desired file modification date in ms since 1970-01-01
- **range** (*None*, *tuple[int, int]*) – starting and ending offsets of the received file contents. Usually None, which means that the whole file is downloaded.

Returns None

write_to_local_file_context(*mod_time_millis*)

class b2sdk.download_dest.**PreSeekedDownloadDest**(*local_file_path*, *seek_target*)
Bases: *b2sdk.download_dest.DownloadDestLocalFile*

Store a downloaded file into a local file and sets its modification time. Does not truncate the target file, seeks to a given offset just after opening a descriptor.

MODE = 'rb+'

__init__(*local_file_path*, *seek_target*)
Initialize self. See help(type(self)) for accurate signature.

write_to_local_file_context(**args*, ***kwargs*)

class b2sdk.download_dest.**DownloadDestBytes**
Bases: *b2sdk.download_dest.AbstractDownloadDestination*

Store a downloaded file into bytes in memory.

__init__()
Initialize self. See help(type(self)) for accurate signature.

make_file_context(*file_id*, *file_name*, *content_length*, *content_type*, *content_sha1*, *file_info*, *mod_time_millis*, *range_=None*)
Return a context manager that yields a binary file-like object to use for writing the contents of the file.

Parameters

- **file_id** (*str*) – the B2 file ID from the headers
- **file_name** (*str*) – the B2 file name from the headers
- **content_length** (*str*) – the content length
- **content_type** (*str*) – the content type from the headers
- **content_sha1** (*str*) – the content sha1 from the headers (or "none" for large files)
- **file_info** (*dict*) – the user file info from the headers
- **mod_time_millis** (*int*) – the desired file modification date in ms since 1970-01-01

- **range** (*None*, *tuple[int, int]*) – starting and ending offsets of the received file contents. Usually `None`, which means that the whole file is downloaded.

Returns None

capture_bytes_context()
Remember the bytes written in self.bytes_written.

get_bytes_written()

class b2sdk.download_dest.**DownloadDestProgressWrapper**(*download_dest*, *progress_listener*)
Bases: *b2sdk.download_dest.AbstractDownloadDestination*

Wrap a DownloadDestination and report progress to a ProgressListener.

__init__(*download_dest*, *progress_listener*)
Initialize self. See help(type(self)) for accurate signature.

make_file_context(*file_id*, *file_name*, *content_length*, *content_type*, *content_sha1*, *file_info*, *mod_time_millis*, *range_=None*)
Return a context manager that yields a binary file-like object to use for writing the contents of the file.

Parameters

- **file_id** (*str*) – the B2 file ID from the headers
- **file_name** (*str*) – the B2 file name from the headers
- **content_length** (*str*) – the content length
- **content_type** (*str*) – the content type from the headers
- **content_sha1** (*str*) – the content sha1 from the headers (or `"none"` for large files)
- **file_info** (*dict*) – the user file info from the headers
- **mod_time_millis** (*int*) – the desired file modification date in ms since 1970-01-01
- **range** (*None*, *tuple[int, int]*) – starting and ending offsets of the received file contents. Usually `None`, which means that the whole file is downloaded.

Returns None

write_file_and_report_progress_context(*file_id*, *file_name*, *content_length*, *content_type*, *content_sha1*, *file_info*, *mod_time_millis*, *range_*)

b2sdk.stream.chained ChainedStream

class b2sdk.stream.chained.**ChainedStream**(*stream_openers*)
Bases: `b2sdk.stream.base.ReadOnlyStreamMixin`, `io.IOBase`

Chains multiple streams in single stream, sort of what `itertools.chain` does for iterators.

Cleans up buffers of underlying streams when closed.

Can be seeked to beginning (when retrying upload, for example). Closes underlying streams as soon as they reaches EOF, but clears their buffers when the chained stream is closed for underlying streams that follow `b2sdk.v1.StreamOpener` cleanup interface, for example `b2sdk.v1.CachedBytesStreamOpener`

__init__(*stream_openers*)

Parameters **stream_openeres** (*list*) – list of callables that return opened streams

property stream
Return currently processed stream.

seekable()
Return whether object supports random access.

If False, seek(), tell() and truncate() will raise OSError. This method may need to do a test seek().

tell()
Return current stream position.

seek(*pos*, *whence=0*)
Resets stream to the beginning.

Parameters

- **pos** (*int*) – only allowed value is `0`
- **whence** (*int*) – only allowed value is `0`

readable()
Return whether object was opened for reading.

If False, read() will raise OSError.

read(*size=None*)
Read at most *size* bytes from underlying streams, or all available data, if *size* is None or negative. Open the streams only when their data is needed, and possibly leave them open and part-way read for further reading - by subsequent calls to this method.

Parameters **size** (*int*, *None*) – number of bytes to read. If omitted, `None`, or negative data is read and returned until EOF from final stream is reached

Returns data read from the stream

close()
Flush and close the IO object.

This method has no effect if the file is already closed.

class b2sdk.stream.chained.**StreamOpener**
Bases: `object`

Abstract class to define stream opener with cleanup.

cleanup()
Clean up stream opener after chained stream closes.

Can be used for cleaning cached data that are stored in memory to allow resetting chained stream without getting this data more than once, eg. data downloaded from external source.

b2sdk.stream.hashing StreamWithHash

class b2sdk.stream.hashing.**StreamWithHash**(*stream*, *stream_length=None*)
Bases: b2sdk.stream.base.ReadOnlyStreamMixin, *b2sdk.stream.wrapper.StreamWithLengthWrapper*

Wrap a file-like object, calculates SHA1 while reading and appends hash at the end.

__init__(*stream*, *stream_length=None*)

Parameters **stream** – the stream to read from

seek(*pos*, *whence=0*)
Seek to a given position in the stream.

Parameters **pos** (*int*) – position in the stream

read(*size=None*)
Read data from the stream.

Parameters **size** (*int*) – number of bytes to read

Returns read data

Return type bytes|None

classmethod get_digest()

b2sdk.stream.progress Streams with progress reporting

class b2sdk.stream.progress.**AbstractStreamWithProgress**(*stream*, *progress_listener*, *offset=0*)
Bases: *b2sdk.stream.wrapper.StreamWrapper*

Wrap a file-like object and updates a ProgressListener as data is read / written. In the abstract class, read and write methods do not update the progress - child classes shall do it.

__init__(*stream*, *progress_listener*, *offset=0*)

Parameters

- **stream** – the stream to read from or write to
- **progress_listener** (b2sdk.v1.AbstractProgressListener) – the listener that we tell about progress
- **offset** (*int*) – the starting byte offset in the file

class b2sdk.stream.progress.**ReadingStreamWithProgress**(**args*, ***kwargs*)
Bases: *b2sdk.stream.progress.AbstractStreamWithProgress*

Wrap a file-like object, updates progress while reading.

__init__(**args*, ***kwargs*)

Parameters

- **stream** – the stream to read from or write to
- **progress_listener** (b2sdk.v1.AbstractProgressListener) – the listener that we tell about progress
- **offset** (*int*) – the starting byte offset in the file

read(*size=None*)
Read data from the stream.

Parameters **size** (*int*) – number of bytes to read

Returns data read from the stream

seek(*pos*, *whence=0*)
Seek to a given position in the stream.

Parameters **pos** (*int*) – position in the stream

Returns new absolute position

Return type int

class b2sdk.stream.progress.**WritingStreamWithProgress**(*stream*, *progress_listener*, *offset=0*)

Bases: *b2sdk.stream.progress.AbstractStreamWithProgress*

Wrap a file-like object; updates progress while writing.

write(*data*)

Write data to the stream.

Parameters **data** (*bytes*) – data to write to the stream

b2sdk.stream.range RangeOfInputStream

class b2sdk.stream.range.**RangeOfInputStream**(*stream*, *offset*, *length*)

Bases: b2sdk.stream.base.ReadOnlyStreamMixin, *b2sdk.stream.wrapper.StreamWithLengthWrapper*

Wrap a file-like object (read only) and read the selected range of the file.

__init__(*stream*, *offset*, *length*)

Parameters

- **stream** – a seekable stream
- **offset** (*int*) – offset in the stream
- **length** (*int*) – max number of bytes to read

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

Seek to a given position in the stream.

Parameters **pos** (*int*) – position in the stream relative to steam offset

Returns new position relative to stream offset

Return type int

tell()

Return current stream position relative to offset.

Return type int

read(*size=None*)

Read data from the stream.

Parameters **size** (*int*) – number of bytes to read

Returns data read from the stream

Return type bytes

b2sdk.stream.range.**wrap_with_range**(*stream*, *stream_length*, *range_offset*, *range_length*)

b2sdk.stream.wrapper StreamWrapper

class b2sdk.stream.wrapper.**StreamWrapper**(*stream*)
Bases: io.IOBase

Wrapper for a file-like object.

__init__(*stream*)

Parameters **stream** – the stream to read from or write to

seekable()
Return whether object supports random access.

If False, seek(), tell() and truncate() will raise OSError. This method may need to do a test seek().

seek(*pos*, *whence=0*)
Seek to a given position in the stream.

Parameters **pos** (*int*) – position in the stream

Returns new absolute position

Return type int

tell()
Return current stream position.

Return type int

truncate(*size=None*)
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.

flush()
Flush the stream.

readable()
Return whether object was opened for reading.

If False, read() will raise OSError.

read(*size=None*)
Read data from the stream.

Parameters **size** (*int*) – number of bytes to read

Returns data read from the stream

writable()
Return whether object was opened for writing.

If False, write() will raise OSError.

write(*data*)
Write data to the stream.

Parameters **data** – a data to write to the stream

close()
Flush and close the IO object.

This method has no effect if the file is already closed.

class b2sdk.stream.wrapper.**StreamWithLengthWrapper**(*stream*, *length=None*)
Bases: *b2sdk.stream.wrapper.StreamWrapper*

Wrapper for a file-like object that supports *__len__* interface

__init__(*stream*, *length=None*)

Parameters

- **stream** – the stream to read from or write to
- **length** (*int*) – length of the stream

b2sdk.sync.action

class b2sdk.sync.action.**AbstractAction**
Bases: object

An action to take, such as uploading, downloading, or deleting a file. Multi-threaded tasks create a sequence of Actions which are then run by a pool of threads.

An action can depend on other actions completing. An example of this is making sure a CreateBucketAction happens before an UploadFileAction.

run(*bucket*, *reporter*, *dry_run=False*)
Main action routine.

Parameters

- **bucket** (*b2sdk.bucket.Bucket*) – a Bucket object
- **reporter** – a place to report errors
- **dry_run** (*bool*) – if True, perform a dry run

abstract get_bytes()
Return the number of bytes to transfer for this action.

Return type int

abstract do_action(*bucket*, *reporter*)
Perform the action, returning only after the action is completed.

Parameters

- **bucket** (*b2sdk.bucket.Bucket*) – a Bucket object
- **reporter** – a place to report errors

abstract do_report(*bucket*, *reporter*)
Report the action performed.

Parameters

- **bucket** (*b2sdk.bucket.Bucket*) – a Bucket object
- **reporter** – a place to report errors

class b2sdk.sync.action.**B2UploadAction**(*local_full_path*, *relative_name*, *b2_file_name*, *mod_time_millis*, *size*, *encryption_settings_provider: b2sdk.sync.encryption_provider.AbstractSyncEncryptionSettingsProvider*)
Bases: *b2sdk.sync.action.AbstractAction*

File uploading action.

__init__(*local_full_path*, *relative_name*, *b2_file_name*, *mod_time_millis*, *size*, *encryption_settings_provider: b2sdk.sync.encryption_provider.AbstractSyncEncryptionSettingsProvider*)

Parameters

- **local_full_path** (*str*) – a local file path
- **relative_name** (*str*) – a relative file name
- **b2_file_name** (*str*) – a name of a new remote file
- **mod_time_millis** (*int*) – file modification time in milliseconds
- **size** (*int*) – a file size
- **encryption_settings_provider** (*b2sdk.v1.AbstractSyncEncryptionSettingsProvider*) – encryption setting provider

get_bytes()

Return file size.

Return type int

do_action(*bucket*, *reporter*)

Perform the uploading action, returning only after the action is completed.

Parameters

- **bucket** (*b2sdk.v1.Bucket*) – a Bucket object
- **reporter** – a place to report errors

do_report(*bucket*, *reporter*)

Report the uploading action performed.

Parameters

- **bucket** (*b2sdk.bucket.Bucket*) – a Bucket object
- **reporter** – a place to report errors

class b2sdk.sync.action.**B2HideAction**(*relative_name*, *b2_file_name*)

Bases: *b2sdk.sync.action.AbstractAction*

__init__(*relative_name*, *b2_file_name*)

Parameters

- **relative_name** (*str*) – a relative file name
- **b2_file_name** (*str*) – a name of a remote file

get_bytes()

Return file size.

Returns always zero

Return type int

do_action(*bucket*, *reporter*)

Perform the hiding action, returning only after the action is completed.

Parameters

- **bucket** (*b2sdk.bucket.Bucket*) – a Bucket object
- **reporter** – a place to report errors

do_report(*bucket*, *reporter*)
Report the hiding action performed.

Parameters

- **bucket** (*b2sdk.bucket.Bucket*) – a Bucket object
- **reporter** – a place to report errors

class b2sdk.sync.action.**B2DownloadAction**(*source_file:* b2sdk.sync.file.B2File, *b2_file_name:* *str*, *local_full_path:* *str*, *encryption_settings_provider:* b2sdk.sync.encryption_provider.AbstractSyncEncryptionSettingsProvider)
Bases: *b2sdk.sync.action.AbstractAction*

__init__(*source_file:* b2sdk.sync.file.B2File, *b2_file_name:* *str*, *local_full_path:* *str*, *encryption_settings_provider:* b2sdk.sync.encryption_provider.AbstractSyncEncryptionSettingsProvider)

Parameters

- **source_file** (*b2sdk.v1.B2File*) – the file to be downloaded
- **b2_file_name** (*str*) – b2_file_name
- **local_full_path** (*str*) – a local file path
- **encryption_settings_provider** (*b2sdk.v1.AbstractSyncEncryptionSettingsProvider*) – encryption setting provider

get_bytes()
Return file size.

Return type int

do_action(*bucket*, *reporter*)
Perform the downloading action, returning only after the action is completed.

Parameters

- **bucket** (b2sdk.v1.Bucket) – a Bucket object
- **reporter** – a place to report errors

do_report(*bucket*, *reporter*)
Report the downloading action performed.

Parameters

- **bucket** (*b2sdk.bucket.Bucket*) – a Bucket object
- **reporter** – a place to report errors

class b2sdk.sync.action.**B2CopyAction**(*b2_file_name:* *str*, *source_file:* b2sdk.sync.file.B2File, *dest_b2_file_name*, *source_bucket:* *b2sdk.bucket.Bucket*, *destination_bucket:* *b2sdk.bucket.Bucket*, *encryption_settings_provider:* b2sdk.sync.encryption_provider.AbstractSyncEncryptionSettingsProvider)
Bases: *b2sdk.sync.action.AbstractAction*

File copying action.

__init__(*b2_file_name:* *str*, *source_file:* b2sdk.sync.file.B2File, *dest_b2_file_name*, *source_bucket:* *b2sdk.bucket.Bucket*, *destination_bucket:* *b2sdk.bucket.Bucket*, *encryption_settings_provider:* b2sdk.sync.encryption_provider.AbstractSyncEncryptionSettingsProvider)

Parameters

- **b2_file_name** (*str*) – a b2_file_name
- **source_file** (*b2sdk.v1.B2File*) – the file to be copied
- **dest_b2_file_name** (*str*) – a name of a destination remote file
- **source_bucket** (Bucket) – bucket to copy from
- **destination_bucket** (Bucket) – bucket to copy to
- **encryption_settings_provider** (*b2sdk.v1.AbstractSyncEncryptionSettingsProvider*) – encryption setting provider

get_bytes()
Return file size.

Return type int

do_action(*bucket*, *reporter*)
Perform the copying action, returning only after the action is completed.

Parameters

- **bucket** (*b2sdk.bucket.Bucket*) – a Bucket object
- **reporter** – a place to report errors

do_report(*bucket*, *reporter*)
Report the copying action performed.

Parameters

- **bucket** (*b2sdk.bucket.Bucket*) – a Bucket object
- **reporter** – a place to report errors

class b2sdk.sync.action.**B2DeleteAction**(*relative_name*, *b2_file_name*, *file_id*, *note*)
Bases: *b2sdk.sync.action.AbstractAction*

__init__(*relative_name*, *b2_file_name*, *file_id*, *note*)

Parameters

- **relative_name** (*str*) – a relative file name
- **b2_file_name** (*str*) – a name of a remote file
- **file_id** (*str*) – a file ID
- **note** (*str*) – a deletion note

get_bytes()
Return file size.

Returns always zero

Return type int

do_action(*bucket*, *reporter*)
Perform the deleting action, returning only after the action is completed.

Parameters

- **bucket** (*b2sdk.bucket.Bucket*) – a Bucket object
- **reporter** – a place to report errors

do_report(*bucket*, *reporter*)
Report the deleting action performed.

Parameters

- **bucket** (*b2sdk.bucket.Bucket*) – a Bucket object
- **reporter** – a place to report errors

class b2sdk.sync.action.**LocalDeleteAction**(*relative_name*, *full_path*)
Bases: *b2sdk.sync.action.AbstractAction*

__init__(*relative_name*, *full_path*)

Parameters

- **relative_name** (*str*) – a relative file name
- **full_path** – a full local path

Type str

get_bytes()
Return file size.

Returns always zero

Return type int

do_action(*bucket*, *reporter*)
Perform the deleting of a local file action, returning only after the action is completed.

Parameters

- **bucket** (*b2sdk.bucket.Bucket*) – a Bucket object
- **reporter** – a place to report errors

do_report(*bucket*, *reporter*)
Report the deleting of a local file action performed.

Parameters

- **bucket** (*b2sdk.bucket.Bucket*) – a Bucket object
- **reporter** – a place to report errors

b2sdk.sync.exception

exception b2sdk.sync.exception.**EnvironmentEncodingError**(*filename*, *encoding*)
Bases: b2sdk.exception.B2Error

Raised when a file name can not be decoded with system encoding.

__init__(*filename*, *encoding*)

Parameters

- **filename** (*str, bytes*) – an encoded file name
- **encoding** (*str*) – file name encoding

exception b2sdk.sync.exception.**InvalidArgument**(*parameter_name*, *message*)
Bases: b2sdk.exception.B2Error

Raised when one or more arguments are invalid

__init__(*parameter_name*, *message*)

Parameters

- **parameter_name** – name of the function argument
- **message** – brief explanation of misconfiguration

exception b2sdk.sync.exception.**IncompleteSync**(**args*, ***kwargs*)
Bases: b2sdk.exception.B2SimpleError

exception b2sdk.sync.exception.**UnSyncableFilename**(*message*, *filename*)
Bases: b2sdk.exception.B2Error

Raised when a filename is not supported by the sync operation

__init__(*message*, *filename*)

Parameters

- **message** – brief explanation of why the filename was not supported
- **filename** – name of the file which is not supported

b2sdk.sync.file

class b2sdk.sync.file.**File**(*name*, *versions: List[*b2sdk.sync.file.FileVersion*]*)
Bases: object

Hold information about one file in a folder.

The name is relative to the folder in all cases.

Files that have multiple versions (which only happens in B2, not in local folders) include information about all of the versions, most recent first.

__init__(*name*, *versions: List[*b2sdk.sync.file.FileVersion*]*)

Parameters

- **name** (*str*) – a relative file name
- **versions** (*List[*FileVersion*]*) – a list of file versions

name

versions

latest_version() → *b2sdk.sync.file.FileVersion*
Return the latest file version.

class b2sdk.sync.file.**B2File**(*name*, *versions: List[*b2sdk.sync.file.B2FileVersion*]*)
Bases: *b2sdk.sync.file.File*

Hold information about one file in a folder in B2 cloud.

__init__(*name*, *versions: List[*b2sdk.sync.file.B2FileVersion*]*)

Parameters

- **name** (*str*) – a relative file name
- **versions** (*List[*B2FileVersion*]*) – a list of file versions

latest_version() → *b2sdk.sync.file.B2FileVersion*
Return the latest file version.

name

versions

class b2sdk.sync.file.**FileVersion**(*id_*, *file_name*, *mod_time*, *action*, *size*)
Bases: object

Hold information about one version of a file.

__init__(*id_*, *file_name*, *mod_time*, *action*, *size*)

Parameters

- **id** (*str*) – the B2 file id, or the local full path name
- **file_name** (*str*) – a relative file name
- **mod_time** (*int*) – modification time, in milliseconds, to avoid rounding issues with millisecond times from B2
- **action** (*str*) – "hide" or "upload" (never "start")
- **size** (*int*) – a file size

id_

name

mod_time

action

size

class b2sdk.sync.file.**B2FileVersion**(*file_version_info: b2sdk.file_version.FileVersionInfo*)
Bases: *b2sdk.sync.file.FileVersion*

__init__(*file_version_info: b2sdk.file_version.FileVersionInfo*)

Parameters

- **id** (*str*) – the B2 file id, or the local full path name
- **file_name** (*str*) – a relative file name
- **mod_time** (*int*) – modification time, in milliseconds, to avoid rounding issues with millisecond times from B2
- **action** (*str*) – "hide" or "upload" (never "start")
- **size** (*int*) – a file size

file_version_info

property id_

property name

property mod_time

property action

property size

b2sdk.sync.folder

class b2sdk.sync.folder.**AbstractFolder**
Bases: object

Interface to a folder full of files, which might be a B2 bucket, a virtual folder in a B2 bucket, or a directory on a local file system.

Files in B2 may have multiple versions, while files in local folders have just one.

abstract all_files(*reporter*, *policies_manager=<b2sdk.sync.scan_policies.ScanPoliciesManager object>*)
Return an iterator over all of the files in the folder, in the order that B2 uses.

It also performs filtering using policies manager.

No matter what the folder separator on the local file system is, "/" is used in the returned file names.

If a file is found, but does not exist (for example due to a broken symlink or a race), reporter will be informed about each such problem.

Parameters

- **reporter** – a place to report errors
- **policies_manager** – a policies manager object

abstract folder_type()
Return one of: 'b2', 'local'.

Return type str

abstract make_full_path(*file_name*)
Return the full path to the file.

Parameters **file_name** (*str*) – a file name

Return type str

b2sdk.sync.folder.**join_b2_path**(*b2_dir*, *b2_name*)
Like os.path.join, but for B2 file names where the root directory is called ''.

Parameters

- **b2_dir** (*str*) – a directory path
- **b2_name** (*str*) – a file name

class b2sdk.sync.folder.**LocalFolder**(*root*)
Bases: *b2sdk.sync.folder.AbstractFolder*

Folder interface to a directory on the local machine.

__init__(*root*)
Initialize a new folder.

Parameters **root** (*str*) – path to the root of the local folder. Must be unicode.

folder_type()
Return folder type.

Return type str

all_files(*reporter*, *policies_manager=<b2sdk.sync.scan_policies.ScanPoliciesManager object>*)
Yield all files.

Parameters

- **reporter** – a place to report errors
- **policies_manager** – a policy manager object, default is DEFAULT_SCAN_MANAGER

make_full_path(*file_name*)
Convert a file name into an absolute path, ensure it is not outside self.root

Parameters **file_name** (*str*) – a file name

ensure_present()
Make sure that the directory exists.

ensure_non_empty()
Make sure that the directory exists and is non-empty.

class b2sdk.sync.folder.**B2Folder**(*bucket_name*, *folder_name*, *api*)
Bases: *b2sdk.sync.folder.AbstractFolder*

Folder interface to b2.

__init__(*bucket_name*, *folder_name*, *api*)

Parameters

- **bucket_name** (*str*) – a name of the bucket
- **folder_name** (*str*) – a folder name
- **api** (*b2sdk.api.B2Api*) – an API object

all_files(*reporter*, *policies_manager=<b2sdk.sync.scan_policies.ScanPoliciesManager object>*)
Yield all files.

Parameters

- **reporter** – a place to report errors
- **policies_manager** – a policies manager object, default is DEFAULT_SCAN_MANAGER

folder_type()
Return folder type.

Return type str

make_full_path(*file_name*)
Make an absolute path from a file name.

Parameters **file_name** (*str*) – a file name

b2sdk.sync.folder_parser

b2sdk.sync.folder_parser.**parse_sync_folder**(*folder_name*, *api*)
Take either a local path, or a B2 path, and returns a Folder object for it.

B2 paths look like: b2://bucketName/path/name. The '//' is optional, because the previous sync command didn't use it.

Anything else is treated like a local folder.

Parameters

- **folder_name** (*str*) – a name of the folder, either local or remote
- **api** (*b2sdk.api.B2Api*) – an API object

b2sdk.sync.policy

class b2sdk.sync.policy.**NewerFileSyncMode**(*value*)
Bases: enum.Enum

Mode of handling files newer on destination than on source

SKIP = 101
skip syncing such file

REPLACE = 102
replace the file on the destination with the (older) file on source

RAISE_ERROR = 103
raise a non-transient error, failing the sync operation

class b2sdk.sync.policy.**CompareVersionMode**(*value*)
Bases: enum.Enum

Mode of comparing versions of files to determine what should be synced and what shouldn't

MODTIME = 201
use file modification time on source filesystem

SIZE = 202
compare using file size

NONE = 203
compare using file name only

class b2sdk.sync.policy.**AbstractFileSyncPolicy**(*source_file*, *source_folder*, *dest_file*, *dest_folder*, *now_millis*, *keep_days*, *newer_file_mode*, *compare_threshold*, *compare_version_mode=<CompareVersionMode.MODTIME: 201>*, *encryption_settings_provider: b2sdk.sync.encryption_provider.AbstractSyncEncryptionSettingsP = <b2sdk.sync.encryption_provider.ServerDefaultSyncEncryptio object>*)
Bases: object

Abstract policy class.

DESTINATION_PREFIX = NotImplemented

SOURCE_PREFIX = NotImplemented

__init__(*source_file*, *source_folder*, *dest_file*, *dest_folder*, *now_millis*, *keep_days*, *newer_file_mode*, *compare_threshold*, *compare_version_mode=<CompareVersionMode.MODTIME: 201>*, *encryption_settings_provider: b2sdk.sync.encryption_provider.AbstractSyncEncryptionSettingsProvider = <b2sdk.sync.encryption_provider.ServerDefaultSyncEncryptionSettingsProvider object>*)

Parameters

- **source_file** (*b2sdk.v1.File*) – source file object
- **source_folder** (*b2sdk.v1.AbstractFolder*) – source folder object
- **dest_file** (*b2sdk.v1.File*) – destination file object
- **dest_folder** (*b2sdk.v1.AbstractFolder*) – destination folder object

- **now_millis** (*int*) – current time in milliseconds
- **keep_days** (*int*) – days to keep before delete
- **newer_file_mode** (*b2sdk.v1.NEWER_FILE_MODES*) – setting which determines handling for destination files newer than on the source
- **compare_threshold** (*int*) – when comparing with size or time for sync
- **compare_version_mode** (*b2sdk.v1.COMPARE_VERSION_MODES*) – how to compare source and destination files
- **encryption_settings_provider** (*b2sdk.v1.AbstractSyncEncryptionSettingsProvider*) – encryption setting provider

classmethod files_are_different(*source_file*, *dest_file*, *compare_threshold=None*, *compare_version_mode=<CompareVersionMode.MODTIME: 201>*, *newer_file_mode=<NewerFileSyncMode.RAISE_ERROR: 103>*)

Compare two files and determine if the the destination file should be replaced by the source file.

Parameters

- **source_file** (*b2sdk.v1.File*) – source file object
- **dest_file** (*b2sdk.v1.File*) – destination file object
- **compare_threshold** (*int*) – compare threshold when comparing by time or size
- **compare_version_mode** (b2sdk.v1.CompareVersionMode) – source file version comparator method
- **newer_file_mode** (b2sdk.v1.NewerFileSyncMode) – newer destination handling method

get_all_actions()

Yield file actions.

class b2sdk.sync.policy.**DownPolicy**(*source_file*, *source_folder*, *dest_file*, *dest_folder*, *now_millis*, *keep_days*, *newer_file_mode*, *compare_threshold*, *compare_version_mode=<CompareVersionMode.MODTIME: 201>*, *encryption_settings_provider: b2sdk.sync.encryption_provider.AbstractSyncEncryptionSettingsProvider = <b2sdk.sync.encryption_provider.ServerDefaultSyncEncryptionSettingsProvider object>*)

Bases: *b2sdk.sync.policy.AbstractFileSyncPolicy*

File is synced down (from the cloud to disk).

DESTINATION_PREFIX = 'local://'

SOURCE_PREFIX = 'b2://'

class b2sdk.sync.policy.**UpPolicy**(*source_file*, *source_folder*, *dest_file*, *dest_folder*, *now_millis*, *keep_days*, *newer_file_mode*, *compare_threshold*, *compare_version_mode=<CompareVersionMode.MODTIME: 201>*, *encryption_settings_provider: b2sdk.sync.encryption_provider.AbstractSyncEncryptionSettingsProvider = <b2sdk.sync.encryption_provider.ServerDefaultSyncEncryptionSettingsProvider object>*)

Bases: *b2sdk.sync.policy.AbstractFileSyncPolicy*

File is synced up (from disk the cloud).

DESTINATION_PREFIX = 'b2://'

SOURCE_PREFIX = 'local://'

class b2sdk.sync.policy.**UpAndDeletePolicy**(*source_file*, *source_folder*, *dest_file*, *dest_folder*, *now_millis*, *keep_days*, *newer_file_mode*, *compare_threshold*, *compare_version_mode=<CompareVersionMode.MODTIME: 201>*, *encryption_settings_provider: b2sdk.sync.encryption_provider.AbstractSyncEncryptionSettingsProvide = <b2sdk.sync.encryption_provider.ServerDefaultSyncEncryptionSetting object>*)

Bases: *b2sdk.sync.policy.UpPolicy*

File is synced up (from disk to the cloud) and the delete flag is SET.

class b2sdk.sync.policy.**UpAndKeepDaysPolicy**(*source_file*, *source_folder*, *dest_file*, *dest_folder*, *now_millis*, *keep_days*, *newer_file_mode*, *compare_threshold*, *compare_version_mode=<CompareVersionMode.MODTIME: 201>*, *encryption_settings_provider: b2sdk.sync.encryption_provider.AbstractSyncEncryptionSettingsProv = <b2sdk.sync.encryption_provider.ServerDefaultSyncEncryptionSett object>*)

Bases: *b2sdk.sync.policy.UpPolicy*

File is synced up (from disk to the cloud) and the keepDays flag is SET.

class b2sdk.sync.policy.**DownAndDeletePolicy**(*source_file*, *source_folder*, *dest_file*, *dest_folder*, *now_millis*, *keep_days*, *newer_file_mode*, *compare_threshold*, *compare_version_mode=<CompareVersionMode.MODTIME: 201>*, *encryption_settings_provider: b2sdk.sync.encryption_provider.AbstractSyncEncryptionSettingsProv = <b2sdk.sync.encryption_provider.ServerDefaultSyncEncryptionSett object>*)

Bases: *b2sdk.sync.policy.DownPolicy*

File is synced down (from the cloud to disk) and the delete flag is SET.

class b2sdk.sync.policy.**DownAndKeepDaysPolicy**(*source_file*, *source_folder*, *dest_file*, *dest_folder*, *now_millis*, *keep_days*, *newer_file_mode*, *compare_threshold*, *compare_version_mode=<CompareVersionMode.MODTIME: 201>*, *encryption_settings_provider: b2sdk.sync.encryption_provider.AbstractSyncEncryptionSettingsPr = <b2sdk.sync.encryption_provider.ServerDefaultSyncEncryptionS object>*)

Bases: *b2sdk.sync.policy.DownPolicy*

File is synced down (from the cloud to disk) and the keepDays flag is SET.

class b2sdk.sync.policy.**CopyPolicy**(*source_file*, *source_folder*, *dest_file*, *dest_folder*, *now_millis*, *keep_days*, *newer_file_mode*, *compare_threshold*, *compare_version_mode=<CompareVersionMode.MODTIME: 201>*, *encryption_settings_provider: b2sdk.sync.encryption_provider.AbstractSyncEncryptionSettingsProvider = <b2sdk.sync.encryption_provider.ServerDefaultSyncEncryptionSettingsProvider object>*)

Bases: *b2sdk.sync.policy.AbstractFileSyncPolicy*

File is copied (server-side).

DESTINATION_PREFIX = 'b2://'

SOURCE_PREFIX = 'b2://'

class b2sdk.sync.policy.**CopyAndDeletePolicy**(*source_file*, *source_folder*, *dest_file*, *dest_folder*, *now_millis*, *keep_days*, *newer_file_mode*, *compare_threshold*, *compare_version_mode=<CompareVersionMode.MODTIME: 201>*, *encryption_settings_provider: b2sdk.sync.encryption_provider.AbstractSyncEncryptionSettingsProvi = <b2sdk.sync.encryption_provider.ServerDefaultSyncEncryptionSett object>*)

Bases: *b2sdk.sync.policy.CopyPolicy*

File is copied (server-side) and the delete flag is SET.

class b2sdk.sync.policy.**CopyAndKeepDaysPolicy**(*source_file*, *source_folder*, *dest_file*, *dest_folder*, *now_millis*, *keep_days*, *newer_file_mode*, *compare_threshold*, *compare_version_mode=<CompareVersionMode.MODTIME: 201>*, *encryption_settings_provider: b2sdk.sync.encryption_provider.AbstractSyncEncryptionSettingsPr = <b2sdk.sync.encryption_provider.ServerDefaultSyncEncryptionS object>*)

Bases: *b2sdk.sync.policy.CopyPolicy*

File is copied (server-side) and the keepDays flag is SET.

b2sdk.sync.policy.**make_b2_delete_note**(*version*, *index*, *transferred*)

Create a note message for delete action.

Parameters

- **version** (*b2sdk.v1.FileVersionInfo*) – an object which contains file version info
- **index** (*int*) – file version index
- **transferred** (*bool*) – if True, file has been transferred, False otherwise

b2sdk.sync.policy.**make_b2_delete_actions**(*source_file*, *dest_file*, *dest_folder*, *transferred*)

Create the actions to delete files stored on B2, which are not present locally.

Parameters

- **source_file** (*b2sdk.v1.File*) – source file object
- **dest_file** (*b2sdk.v1.File*) – destination file object
- **dest_folder** (*b2sdk.v1.AbstractFolder*) – destination folder

- **transferred** (*bool*) – if True, file has been transferred, False otherwise

b2sdk.sync.policy.**make_b2_keep_days_actions**(*source_file*, *dest_file*, *dest_folder*, *transferred*, *keep_days*, *now_millis*)

Create the actions to hide or delete existing versions of a file stored in b2.

When keepDays is set, all files that were visible any time from keepDays ago until now must be kept. If versions were uploaded 5 days ago, 15 days ago, and 25 days ago, and the keepDays is 10, only the 25-day old version can be deleted. The 15 day-old version was visible 10 days ago.

Parameters

- **source_file** (*b2sdk.v1.File*) – source file object
- **dest_file** (*b2sdk.v1.File*) – destination file object
- **dest_folder** (*b2sdk.v1.AbstractFolder*) – destination folder object
- **transferred** (*bool*) – if True, file has been transferred, False otherwise
- **keep_days** (*int*) – how many days to keep a file
- **now_millis** (*int*) – current time in milliseconds

b2sdk.sync.policy_manager

class b2sdk.sync.policy_manager.**SyncPolicyManager**

Bases: object

Policy manager; implement a logic to get a correct policy class and create a policy object based on various parameters.

__init__()

Initialize self. See help(type(self)) for accurate signature.

get_policy(*sync_type*, *source_file:* b2sdk.sync.file.File, *source_folder*, *dest_file:* b2sdk.sync.file.File, *dest_folder*, *now_millis*, *delete*, *keep_days*, *newer_file_mode*, *compare_threshold*, *compare_version_mode*, *encryption_settings_provider*)

Return a policy object.

Parameters

- **sync_type** (*str*) – synchronization type
- **source_file** (*b2sdk.v1.File*) – source file name
- **source_folder** (*str*) – a source folder path
- **dest_file** (*b2sdk.v1.File*) – destination file name
- **dest_folder** (*str*) – a destination folder path
- **now_millis** (*int*) – current time in milliseconds
- **delete** (*bool*) – delete policy
- **keep_days** (*int*) – keep for days policy
- **newer_file_mode** (b2sdk.v1.NewerFileSyncMode) – setting which determines handling for destination files newer than on the source
- **compare_threshold** (*int*) – difference between file modification time or file size
- **compare_version_mode** (b2sdk.v1.CompareVersionMode) – setting which determines how to compare source and destination files

- **encryption_settings_provider** (*b2sdk.v1.AbstractSyncEncryptionSettingsProvider*) – an object which decides which encryption to use (if any)

Returns a policy object

get_policy_class(*sync_type*, *delete*, *keep_days*)
Get policy class by a given sync type.

Parameters

- **sync_type** (*str*) – synchronization type
- **delete** (*bool*) – if True, delete files and update from source
- **keep_days** (*int*) – keep for *keep_days* before delete

Returns a policy class

b2sdk.sync.scan_policies

class b2sdk.sync.scan_policies.**RegexSet**(*regex_iterable*)
Bases: object

Hold a (possibly empty) set of regular expressions and know how to check whether a string matches any of them.

__init__(*regex_iterable*)

Parameters **regex_iterable** – an interable which yields regexes

matches(*s*)
Check whether a string matches any of regular expressions.

Parameters **s** (*str*) – a string to check

Return type bool

b2sdk.sync.scan_policies.**convert_dir_regex_to_dir_prefix_regex**(*dir_regex*)
The patterns used to match directory names (and file names) are allowed to match a prefix of the name. This 'feature' was unintentional, but is being retained for compatibility.

This means that a regex that matches a directory name can't be used directly to match against a file name and test whether the file should be excluded because it matches the directory.

The pattern 'photos' will match directory names 'photos' and 'photos2', and should exclude files 'photos/kitten.jpg', and 'photos2/puppy.jpg'. It should not exclude 'photos.txt', because there is no directory name that matches.

On the other hand, the pattern 'photos$' should match 'photos/kitten.jpg', but not 'photos2/puppy.jpg', nor 'photos.txt'

If the original regex is valid, there are only two cases to consider: either the regex ends in '$' or does not.

Parameters **dir_regex** (*str*) – a regular expression string or literal

class b2sdk.sync.scan_policies.**IntegerRange**(*begin*, *end*)
Bases: object

Hold a range of two integers. If the range value is None, it indicates that the value should be treated as -Inf (for begin) or +Inf (for end).

__init__(*begin*, *end*)

Parameters

- **begin** (*int*) – begin position of the range (included)
- **end** (*int*) – end position of the range (included)

class b2sdk.sync.scan_policies.**ScanPoliciesManager**(*exclude_dir_regexes=()*, *exclude_file_regexes=()*, *include_file_regexes=()*, *exclude_all_symlinks=False*, *exclude_modified_before=None*, *exclude_modified_after=None*)

Bases: object

Policy object used when scanning folders for syncing, used to decide which files to include in the list of files to be synced.

Code that scans through files should at least use should_exclude_file() to decide whether each file should be included; it will check include/exclude patterns for file names, as well as patterns for excluding directories.

Code that scans may optionally use should_exclude_directory() to test whether it can skip a directory completely and not bother listing the files and sub-directories in it.

__init__(*exclude_dir_regexes=()*, *exclude_file_regexes=()*, *include_file_regexes=()*, *exclude_all_symlinks=False*, *exclude_modified_before=None*, *exclude_modified_after=None*)

Parameters

- **exclude_dir_regexes** (*tuple*) – a tuple of regexes to exclude directories
- **exclude_file_regexes** (*tuple*) – a tuple of regexes to exclude files
- **include_file_regexes** (*tuple*) – a tuple of regexes to include files
- **exclude_all_symlinks** (*bool*) – if True, exclude all symlinks
- **exclude_modified_before** (*int, optional*) – optionally exclude file versions modified before (in millis)
- **exclude_modified_after** (*int, optional*) – optionally exclude file versions modified after (in millis)

should_exclude_file(*file_path*)

Given the full path of a file, decide if it should be excluded from the scan.

Parameters **file_path** – the path of the file, relative to the root directory being scanned.

Type str

Returns True if excluded.

Return type bool

should_exclude_file_version(*file_version*)

Given the modification time of a file version, decide if it should be excluded from the scan.

Parameters **file_version** – the file version object

Type b2sdk.v1.FileVersion

Returns True if excluded.

Return type bool

should_exclude_directory(*dir_path*)

Given the full path of a directory, decide if all of the files in it should be excluded from the scan.

Parameters **dir_path** (*str*) – the path of the directory, relative to the root directory being scanned. The path will never end in '/'.

Returns True if excluded.

b2sdk.sync.sync

b2sdk.sync.sync.**next_or_none**(*iterator*)
Return the next item from the iterator, or None if there are no more.

b2sdk.sync.sync.**zip_folders**(*folder_a*, *folder_b*, *reporter*, *policies_manager=<b2sdk.sync.scan_policies.ScanPoliciesManager object>*)
Iterate over all of the files in the union of two folders, matching file names.

Each item is a pair (file_a, file_b) with the corresponding file in both folders. Either file (but not both) will be None if the file is in only one folder.

Parameters

- **folder_a** (b2sdk.sync.folder.AbstractFolder) – first folder object.
- **folder_b** (b2sdk.sync.folder.AbstractFolder) – second folder object.
- **reporter** – reporter object
- **policies_manager** – policies manager object

Returns yields two element tuples

b2sdk.sync.sync.**count_files**(*local_folder*, *reporter*, *policies_manager*)
Count all of the files in a local folder.

Parameters

- **local_folder** (b2sdk.sync.folder.AbstractFolder) – a folder object.
- **reporter** – reporter object

class b2sdk.sync.sync.**KeepOrDeleteMode**(*value*)
Bases: enum.Enum

Mode of dealing with old versions of files on the destination

DELETE = 301
delete the old version as soon as the new one has been uploaded

KEEP_BEFORE_DELETE = 302
keep the old versions of the file for a configurable number of days before deleting them, always keeping the newest version

NO_DELETE = 303
keep old versions of the file, do not delete anything

class b2sdk.sync.sync.**Synchronizer**(*max_workers*, *policies_manager=<b2sdk.sync.scan_policies.ScanPoliciesManager object>*, *dry_run=False*, *allow_empty_source=False*, *newer_file_mode=<NewerFileSyncMode.RAISE_ERROR: 103>*, *keep_days_or_delete=<KeepOrDeleteMode.NO_DELETE: 303>*, *compare_version_mode=<CompareVersionMode.MODTIME: 201>*, *compare_threshold=None*, *keep_days=None*)
Bases: object

Copies multiple "files" from source to destination. Optionally deletes or hides destination files that the source does not have.

The synchronizer can copy files:

- From a B2 bucket to a local destination.
- From a local source to a B2 bucket.
- From one B2 bucket to another.
- Between different folders in the same B2 bucket. It will sync only the latest versions of files.

By default, the synchronizer:

- Fails when the specified source directory doesn't exist or is empty. (see `allow_empty_source` argument)
- Fails when the source is newer. (see `newer_file_mode` argument)
- Doesn't delete a file if it's present on the destination but not on the source. (see `keep_days_or_delete` and `keep_days` arguments)
- Compares files based on modification time. (see `compare_version_mode` and `compare_threshold` arguments)

__init__(*max_workers*, *policies_manager=<b2sdk.sync.scan_policies.ScanPoliciesManager object>*, *dry_run=False*, *allow_empty_source=False*, *newer_file_mode=<NewerFileSyncMode.RAISE_ERROR: 103>*, *keep_days_or_delete=<KeepOrDeleteMode.NO_DELETE: 303>*, *compare_version_mode=<CompareVersionMode.MODTIME: 201>*, *compare_threshold=None*, *keep_days=None*)

Initialize synchronizer class and validate arguments

Parameters

- **max_workers** (*int*) – max number of workers
- **policies_manager** – policies manager object
- **dry_run** (*bool*) – test mode, does not actually transfer/delete when enabled
- **allow_empty_source** (*bool*) – if True, do not check whether source folder is empty
- **newer_file_mode** (b2sdk.v1.NewerFileSyncMode) – setting which determines handling for destination files newer than on the source
- **keep_days_or_delete** (b2sdk.v1.KeepOrDeleteMode) – setting which determines if we should delete or not delete or keep for *keep_days*
- **compare_version_mode** (b2sdk.v1.CompareVersionMode) – how to compare the source and destination files to find new ones
- **compare_threshold** (*int*) – should be greater than 0, default is 0
- **keep_days** (*int*) – if keep_days_or_delete is *b2sdk.v1.KeepOrDeleteMode.KEEP_BEFORE_DELETE*, then this should be greater than 0

sync_folders(*source_folder*, *dest_folder*, *now_millis*, *reporter*, *encryption_settings_provider: b2sdk.sync.encryption_provider.AbstractSyncEncryptionSettingsProvider = <b2sdk.sync.encryption_provider.ServerDefaultSyncEncryptionSettingsProvider object>*)

Syncs two folders. Always ensures that every file in the source is also in the destination. Deletes any file versions in the destination older than history_days.

Parameters

- **source_folder** (b2sdk.sync.folder.AbstractFolder) – source folder object

- **dest_folder** (b2sdk.sync.folder.AbstractFolder) – destination folder object
- **now_millis** (*int*) – current time in milliseconds
- **reporter** (*b2sdk.sync.report.SyncReport*, *None*) – progress reporter
- **encryption_settings_provider** (*b2sdk.v1.AbstractSyncEncryptionSettingsProvider*) – encryption setting provider

make_folder_sync_actions(*source_folder*, *dest_folder*, *now_millis*, *reporter*, *policies_manager=<b2sdk.sync.scan_policies.ScanPoliciesManager object>*, *encryption_settings_provider: b2sdk.sync.encryption_provider.AbstractSyncEncryptionSettingsProvider = <b2sdk.sync.encryption_provider.ServerDefaultSyncEncryptionSettingsProvider object>*)

Yield a sequence of actions that will sync the destination folder to the source folder.

Parameters

- **source_folder** (*b2sdk.v1.AbstractFolder*) – source folder object
- **dest_folder** (*b2sdk.v1.AbstractFolder*) – destination folder object
- **now_millis** (*int*) – current time in milliseconds
- **reporter** (b2sdk.v1.SyncReport) – reporter object
- **policies_manager** – policies manager object
- **encryption_settings_provider** (*b2sdk.v1.AbstractSyncEncryptionSettingsProvider*) – encryption setting provider

make_file_sync_actions(*sync_type*, *source_file*, *dest_file*, *source_folder*, *dest_folder*, *now_millis*, *encryption_settings_provider: b2sdk.sync.encryption_provider.AbstractSyncEncryptionSettingsProvider = <b2sdk.sync.encryption_provider.ServerDefaultSyncEncryptionSettingsProvider object>*)

Yields the sequence of actions needed to sync the two files

Parameters

- **sync_type** (*str*) – synchronization type
- **source_file** (*b2sdk.v1.File*) – source file object
- **dest_file** (*b2sdk.v1.File*) – destination file object
- **source_folder** (*b2sdk.v1.AbstractFolder*) – a source folder object
- **dest_folder** (*b2sdk.v1.AbstractFolder*) – a destination folder object
- **now_millis** (*int*) – current time in milliseconds
- **encryption_settings_provider** (*b2sdk.v1.AbstractSyncEncryptionSettingsProvider*) – encryption setting provider

b2sdk.sync.encryption_provider

class b2sdk.sync.encryption_provider.**AbstractSyncEncryptionSettingsProvider**
Object which provides an appropriate EncryptionSetting object for sync, i.e. complex operations with multiple sources and destinations

abstract get_setting_for_upload(*bucket: b2sdk.bucket.Bucket*, *b2_file_name: str*, *file_info: Optional[dict]*, *length: int*) → Optional[*b2sdk.encryption.setting.EncryptionSetting*]
Return an EncryptionSetting for uploading an object or None if server should decide.

WARNING: the signature of this method is not final yet and not part of the public interface

abstract get_source_setting_for_copy(*bucket: b2sdk.bucket.Bucket*, *source_file_version_info: b2sdk.file_version.FileVersionInfo*) → Optional[*b2sdk.encryption.setting.EncryptionSetting*]
Return an EncryptionSetting for a source of copying an object or None if not required

WARNING: the signature of this method is not final yet and not part of the public interface

abstract get_destination_setting_for_copy(*bucket: b2sdk.bucket.Bucket*, *dest_b2_file_name: str*, *source_file_version_info: b2sdk.file_version.FileVersionInfo*, *target_file_info: Optional[dict] = None*) → Optional[*b2sdk.encryption.setting.EncryptionSetting*]
Return an EncryptionSetting for a destination for copying an object or None if server should decide

WARNING: the signature of this method is not final yet and not part of the public interface

abstract get_setting_for_download(*bucket: b2sdk.bucket.Bucket*, *file_version_info: b2sdk.file_version.FileVersionInfo*) → Optional[*b2sdk.encryption.setting.EncryptionSetting*]
Return an EncryptionSetting for downloading an object from, or None if not required

WARNING: the signature of this method is not final yet and not part of the public interface

class b2sdk.sync.encryption_provider.**ServerDefaultSyncEncryptionSettingsProvider**
Encryption settings provider which assumes setting-less reads and a bucket default for writes.

get_setting_for_upload(**args*, ***kwargs*) → None
Return an EncryptionSetting for uploading an object or None if server should decide.

WARNING: the signature of this method is not final yet and not part of the public interface

get_source_setting_for_copy(**args*, ***kwargs*) → None
Return an EncryptionSetting for a source of copying an object or None if not required

WARNING: the signature of this method is not final yet and not part of the public interface

get_destination_setting_for_copy(**args*, ***kwargs*) → None
Return an EncryptionSetting for a destination for copying an object or None if server should decide

WARNING: the signature of this method is not final yet and not part of the public interface

get_setting_for_download(**args*, ***kwargs*) → None
Return an EncryptionSetting for downloading an object from, or None if not required

WARNING: the signature of this method is not final yet and not part of the public interface

class b2sdk.sync.encryption_provider.**BasicSyncEncryptionSettingsProvider**(*read_bucket_settings: Dict[str, Optional[b2sdk.encryption.s write_bucket_settings: Dict[str, Optional[b2sdk.encryption.s*

Basic encryption setting provider that supports exactly one encryption setting per bucket for reading and one encryption setting per bucket for writing

WARNING: This class can be used by B2CLI for SSE-B2, but it's still in development

get_setting_for_upload(*bucket*, **args*, ***kwargs*) → Optional[*b2sdk.encryption.setting.EncryptionSetting*]

Return an EncryptionSetting for uploading an object or None if server should decide.

WARNING: the signature of this method is not final yet and not part of the public interface

get_source_setting_for_copy(*bucket*, **args*, ***kwargs*) → None

Return an EncryptionSetting for a source of copying an object or None if not required

WARNING: the signature of this method is not final yet and not part of the public interface

get_destination_setting_for_copy(*bucket*, **args*, ***kwargs*) → Optional[*b2sdk.encryption.setting.EncryptionSetting*]

Return an EncryptionSetting for a destination for copying an object or None if server should decide

WARNING: the signature of this method is not final yet and not part of the public interface

get_setting_for_download(*bucket*, **args*, ***kwargs*) → None

Return an EncryptionSetting for downloading an object from, or None if not required

WARNING: the signature of this method is not final yet and not part of the public interface

b2sdk.encryption.setting

class b2sdk.encryption.setting.**EncryptionKey**(*secret: Optional[bytes]*, *key_id: Optional[str]*)

Hold information about encryption key: the key itself, and its id. The id may be None, if it's not set in encrypted file's fileInfo. The secret may be None, if encryption metadata is read from the server.

SECRET_REPR = '****'**

as_dict()

Dump EncryptionKey as dict for serializing a to json for requests.

key_b64()

key_md5()

class b2sdk.encryption.setting.**EncryptionSetting**(*mode: b2sdk.encryption.types.EncryptionMode*, *algorithm: Optional[b2sdk.encryption.types.EncryptionAlgorithm] = None*, *key: Optional[b2sdk.encryption.setting.EncryptionKey] = None*)

Hold information about encryption mode, algorithm and key (for bucket default, file version info or even upload)

serialize_to_json_for_request()

as_dict()

Represent the setting as a dict, for example:

```
{
    'mode': 'SSE-C',
    'algorithm': 'AES256',
    'customerKey': 'U3hWbVlxM3Q2djl5JEImRSlIQE1jUWZUalduWnI0dTc=',
    'customerKeyMd5': 'SWx9GFv5BTT1jdwf48Bx+Q=='
}
```

```
{
    'mode': 'SSE-B2',
    'algorithm': 'AES256'
}
```

or

```
{
    'mode': 'none'
}
```

add_to_upload_headers(*headers*)

add_to_download_headers(*headers*)

add_key_id_to_file_info(*file_info: Optional[dict]*)

class b2sdk.encryption.setting.**EncryptionSettingFactory**

classmethod from_file_version_dict(*file_version_dict: dict*) → *b2sdk.encryption.setting.EncryptionSetting*

Returns EncryptionSetting for the given file_version_dict retrieved from the api

```
...
"serverSideEncryption": {"algorithm": "AES256", "mode": "SSE-B2"},
"fileInfo": {"sse_c_key_id": "key-identifier"}
...
```

classmethod from_bucket_dict(*bucket_dict: dict*) → Optional[*b2sdk.encryption.setting.EncryptionSetting*]

Returns EncryptionSetting for the given bucket dict retrieved from the api, or None if unauthorized

Example inputs:

```
...
"defaultServerSideEncryption": {
    "isClientAuthorizedToRead" : true,
    "value": {
      "algorithm" : "AES256",
      "mode" : "SSE-B2"
    }
}
...
```

unset:

```
 ...
"defaultServerSideEncryption": {
```

(continues on next page)

(continued from previous page)

```
    "isClientAuthorizedToRead" : true,
    "value": {
      "mode" : "none"
    }
}
...
```

unknown:

```
...
"defaultServerSideEncryption": {
    "isClientAuthorizedToRead" : false
}
...
```

classmethod from_response_headers(*headers*)

b2sdk.encryption.setting.**SSE_NONE = <EncryptionSetting(EncryptionMode.NONE, None, None)>**
Commonly used "no encryption" setting

b2sdk.encryption.setting.**SSE_B2_AES = <EncryptionSetting(EncryptionMode.SSE_B2, EncryptionA**
Commonly used SSE-B2 setting

b2sdk.encryption.types

class b2sdk.encryption.types.**EncryptionAlgorithm**(*value*)
Encryption algorithm.

AES256 = 'AES256'

class b2sdk.encryption.types.**EncryptionMode**(*value*)
Encryption mode.

UNKNOWN = None
unknown encryption mode (sdk doesn't know or used key has no rights to know)

NONE = 'none'
no encryption (plaintext)

SSE_B2 = 'SSE-B2'
server-side encryption with key maintained by B2

SSE_C = 'SSE-C'
server-side encryption with key provided by the client

can_be_set_as_bucket_default()

b2sdk.transfer.inbound.downloader.abstract – Downloader base class

class b2sdk.transfer.inbound.downloader.abstract.**AbstractDownloader**(*force_chunk_size=None*, *min_chunk_size=None*, *max_chunk_size=None*)

Bases: object

__init__(*force_chunk_size=None*, *min_chunk_size=None*, *max_chunk_size=None*)
Initialize self. See help(type(self)) for accurate signature.

abstract is_suitable(*metadata*, *progress_listener*)
Analyze metadata (possibly against options passed earlier to constructor to find out whether the given download request should be handled by this downloader).

abstract download(*file*, *response*, *metadata*, *session*, *encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*)
@returns (bytes_read, actual_sha1)

b2sdk.transfer.inbound.downloader.parallel – ParallelTransferer

class b2sdk.transfer.inbound.downloader.parallel.**ParallelDownloader**(*max_streams*, *min_part_size*, **args*, ***kwargs*)
Bases: *b2sdk.transfer.inbound.downloader.abstract.AbstractDownloader*

FINISH_HASHING_BUFFER_SIZE = 1048576

__init__(*max_streams*, *min_part_size*, **args*, ***kwargs*)

Parameters

- **max_streams** – maximum number of simultaneous streams
- **min_part_size** – minimum amount of data a single stream will retrieve, in bytes

is_suitable(*metadata*, *progress_listener*)
Analyze metadata (possibly against options passed earlier to constructor to find out whether the given download request should be handled by this downloader).

download(*file*, *response*, *metadata*, *session*, *encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*)
Download a file from given url using parallel download sessions and stores it in the given download_destination.

Parameters

- **file** – an opened file-like object to write to
- **response** – the response of the first request made to the cloud service with download intent

Returns

class b2sdk.transfer.inbound.downloader.parallel.**WriterThread**(*file*, *max_queue_depth*)
Bases: threading.Thread

A thread responsible for keeping a queue of data chunks to write to a file-like object and for actually writing them down. Since a single thread is responsible for synchronization of the writes, we avoid a lot of issues between userspace and kernelspace that would normally require flushing buffers between the switches of the writer. That would kill performance and not synchronizing would cause data corruption (probably we'd end up with a file with unexpected blocks of zeros preceding the range of the writer that comes second and writes further into the file).

The object of this class is also responsible for backpressure: if items are added to the queue faster than they can be written (see GCP VMs with standard PD storage with faster CPU and network than local storage, https://github.com/Backblaze/B2_Command_Line_Tool/issues/595), then `obj.queue.put(item)` will block, slowing down the producer.

The recommended minimum value of `max_queue_depth` is equal to the amount of producer threads, so that if all producers submit a part at the exact same time (right after network issue, for example, or just after starting

the read), they can continue their work without blocking. The writer should be able to store at least one data chunk before a new one is retrieved, but it is not guaranteed.

Therefore, the recommended value of `max_queue_depth` is higher - a double of the amount of producers, so that spikes on either end (many producers submit at the same time / consumer has a latency spike) can be accommodated without sacrificing performance.

Please note that a size of the chunk and the queue depth impact the memory footprint. In a default setting as of writing this, that might be 10 downloads, 8 producers, 1MB buffers, 2 buffers each = 8*2*10 = 160 MB (+ python buffers, operating system etc).

__init__(*file*, *max_queue_depth*)

This constructor should always be called with keyword arguments. Arguments are:

group should be None; reserved for future extension when a ThreadGroup class is implemented.

target is the callable object to be invoked by the run() method. Defaults to None, meaning nothing is called.

name is the thread name. By default, a unique name is constructed of the form "Thread-N" where N is a small decimal number.

args is the argument tuple for the target invocation. Defaults to ().

kwargs is a dictionary of keyword arguments for the target invocation. Defaults to {}.

If a subclass overrides the constructor, it must make sure to invoke the base class constructor (Thread.__init__()) before doing anything else to the thread.

run()

Method representing the thread's activity.

You may override this method in a subclass. The standard run() method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the args and kwargs arguments, respectively.

class b2sdk.transfer.inbound.downloader.parallel.**AbstractDownloaderThread**(*session*, *writer*, *part_to_download*, *chunk_size*, *encryption: Optional[*b2sdk.encryption. = *None*)

Bases: `threading.Thread`

__init__(*session*, *writer*, *part_to_download*, *chunk_size*, *encryption: Optional[*b2sdk.encryption.setting.EncryptionSetting*] = None*)

Parameters

- **session** – raw_api wrapper
- **writer** – where to write data
- **part_to_download** – PartToDownload object
- **chunk_size** – internal buffer size to use for writing and hashing

abstract run()

Method representing the thread's activity.

You may override this method in a subclass. The standard run() method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the args and kwargs arguments, respectively.

class b2sdk.transfer.inbound.downloader.parallel.**FirstPartDownloaderThread**(*response*, *hasher*, **args*, ***kwargs*)

Bases: *b2sdk.transfer.inbound.downloader.parallel.AbstractDownloaderThread*

__init__(*response*, *hasher*, **args*, ***kwargs*)

Parameters

- **response** – response of the original GET call
- **hasher** – hasher object to feed to as the stream is written

run()

Method representing the thread's activity.

You may override this method in a subclass. The standard run() method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the args and kwargs arguments, respectively.

class b2sdk.transfer.inbound.downloader.parallel.**NonHashingDownloaderThread**(*url*, **args*, ***kwargs*)

Bases: *b2sdk.transfer.inbound.downloader.parallel.AbstractDownloaderThread*

__init__(*url*, **args*, ***kwargs*)

Parameters **url** – url of the target file

run()

Method representing the thread's activity.

You may override this method in a subclass. The standard run() method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the args and kwargs arguments, respectively.

class b2sdk.transfer.inbound.downloader.parallel.**PartToDownload**(*cloud_range*, *local_range*)

Bases: object

Hold the range of a file to download, and the range of the local file where it should be stored.

__init__(*cloud_range*, *local_range*)

Initialize self. See help(type(self)) for accurate signature.

b2sdk.transfer.inbound.downloader.parallel.**gen_parts**(*cloud_range*, *local_range*, *part_count*)

Generate a sequence of PartToDownload to download a large file as a collection of parts.

b2sdk.transfer.inbound.downloader.range – transfer range toolkit

class b2sdk.transfer.inbound.downloader.range.**Range**(*start*, *end*)
Bases: object

HTTP ranges use an *inclusive* index at the end.

__init__(*start*, *end*)
Initialize self. See help(type(self)) for accurate signature.

classmethod from_header(*raw_range_header*)
Factory method which returns an object constructed from Range http header.

raw_range_header example: 'bytes=0-11'

size()

subrange(*sub_start*, *sub_end*)
Return a range that is part of this range.

Parameters

- **sub_start** – index relative to the start of this range.
- **sub_end** – (Inclusive!) index relative to the start of this range.

Returns a new Range

as_tuple()

b2sdk.transfer.inbound.downloader.simple – SimpleDownloader

class b2sdk.transfer.inbound.downloader.simple.**SimpleDownloader**(**args*, ***kwargs*)
Bases: *b2sdk.transfer.inbound.downloader.abstract.AbstractDownloader*

__init__(**args*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

is_suitable(*metadata*, *progress_listener*)
Analyze metadata (possibly against options passed earlier to constructor to find out whether the given download request should be handled by this downloader).

download(*file*, *response*, *metadata*, *session*, *encryption: Optional[*b2sdk.encryption.setting.EncryptionSetting*] = None*)
@returns (bytes_read, actual_sha1)

b2sdk.transfer.inbound.download_manager – Manager of downloaders

class b2sdk.transfer.inbound.download_manager.**DownloadManager**(*services*)
Bases: object

Handle complex actions around downloads to free raw_api from that responsibility.

DEFAULT_MAX_STREAMS = 8

DEFAULT_MIN_PART_SIZE = 104857600

MIN_CHUNK_SIZE = 8192

MAX_CHUNK_SIZE = 1048576

__init__(*services*)
Initialize the DownloadManager using the given services object.

Parameters **services** (*b2sdk.v1.Services*) –

download_file_from_url(*url*, *download_dest*, *progress_listener=None*, *range_=None*, *encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*)

Parameters

- **url** – url from which the file should be downloaded
- **download_dest** – where to put the file when it is downloaded
- **progress_listener** – where to notify about progress downloading
- **range** – 2-element tuple containing data of http Range header
- **encryption** (*b2sdk.v1.EncryptionSetting*) – encryption setting (`None` if unknown)

b2sdk.transfer.inbound.file_metadata

class b2sdk.transfer.inbound.file_metadata.**FileMetadata**(*file_id*, *file_name*, *content_type*, *content_length*, *content_sha1*, *file_info*)

Bases: `object`

Hold information about a file which is being downloaded.

UNVERIFIED_CHECKSUM_PREFIX = 'unverified:'

__init__(*file_id*, *file_name*, *content_type*, *content_length*, *content_sha1*, *file_info*)
Initialize self. See help(type(self)) for accurate signature.

file_id

file_name

content_type

content_length

content_sha1

content_sha1_verified

file_info

classmethod from_response(*response*)

as_info_dict()

b2sdk.transfer.outbound.upload_source

class b2sdk.transfer.outbound.upload_source.**AbstractUploadSource**

Bases: `b2sdk.transfer.outbound.outbound_source.OutboundTransferSource`

The source of data for uploading to b2.

abstract get_content_sha1()
Return a 40-character string containing the hex SHA1 checksum of the data in the file.

abstract open()
Return a binary file-like object from which the data can be read. :return:

is_upload()
Return if outbound source is an upload source. :rtype bool:

is_copy()
Return if outbound source is a copy source. :rtype bool:

is_sha1_known()

class b2sdk.transfer.outbound.upload_source.**UploadSourceBytes**(*data_bytes*, *content_sha1=None*)
Bases: *b2sdk.transfer.outbound.upload_source.AbstractUploadSource*

__init__(*data_bytes*, *content_sha1=None*)
Initialize self. See help(type(self)) for accurate signature.

get_content_length()
Return the number of bytes of data in the file.

get_content_sha1()
Return a 40-character string containing the hex SHA1 checksum of the data in the file.

open()
Return a binary file-like object from which the data can be read. :return:

is_sha1_known()

class b2sdk.transfer.outbound.upload_source.**UploadSourceLocalFile**(*local_path*, *content_sha1=None*)
Bases: *b2sdk.transfer.outbound.upload_source.AbstractUploadSource*

__init__(*local_path*, *content_sha1=None*)
Initialize self. See help(type(self)) for accurate signature.

check_path_and_get_size()

get_content_length()
Return the number of bytes of data in the file.

get_content_sha1()
Return a 40-character string containing the hex SHA1 checksum of the data in the file.

open()
Return a binary file-like object from which the data can be read. :return:

is_sha1_known()

class b2sdk.transfer.outbound.upload_source.**UploadSourceLocalFileRange**(*local_path*, *content_sha1=None*, *offset=0*, *length=None*)
Bases: *b2sdk.transfer.outbound.upload_source.UploadSourceLocalFile*

__init__(*local_path*, *content_sha1=None*, *offset=0*, *length=None*)
Initialize self. See help(type(self)) for accurate signature.

open()
Return a binary file-like object from which the data can be read. :return:

class b2sdk.transfer.outbound.upload_source.**UploadSourceStream**(*stream_opener*, *stream_length=None*, *stream_sha1=None*)

Bases: *b2sdk.transfer.outbound.upload_source.AbstractUploadSource*

__init__(*stream_opener*, *stream_length=None*, *stream_sha1=None*)
Initialize self. See help(type(self)) for accurate signature.

get_content_length()
Return the number of bytes of data in the file.

get_content_sha1()
Return a 40-character string containing the hex SHA1 checksum of the data in the file.

open()
Return a binary file-like object from which the data can be read. :return:

is_sha1_known()

class b2sdk.transfer.outbound.upload_source.**UploadSourceStreamRange**(*stream_opener*, *offset*, *stream_length*, *stream_sha1=None*)

Bases: *b2sdk.transfer.outbound.upload_source.UploadSourceStream*

__init__(*stream_opener*, *offset*, *stream_length*, *stream_sha1=None*)
Initialize self. See help(type(self)) for accurate signature.

open()
Return a binary file-like object from which the data can be read. :return:

b2sdk.raw_simulator – B2 raw api simulator

b2sdk.raw_simulator.**get_bytes_range**(*data_bytes*, *bytes_range*)
Slice bytes array using bytes range

class b2sdk.raw_simulator.**KeySimulator**(*account_id*, *name*, *application_key_id*, *key*, *capabilities*, *expiration_timestamp_or_none*, *bucket_id_or_none*, *bucket_name_or_none*, *name_prefix_or_none*)

Bases: object

Hold information about one application key, which can be either a master application key, or one created with create_key().

__init__(*account_id*, *name*, *application_key_id*, *key*, *capabilities*, *expiration_timestamp_or_none*, *bucket_id_or_none*, *bucket_name_or_none*, *name_prefix_or_none*)
Initialize self. See help(type(self)) for accurate signature.

as_key()

as_created_key()
Return the dict returned by b2_create_key.

This is just like the one for b2_list_keys, but also includes the secret key.

get_allowed()
Return the 'allowed' structure to include in the response from b2_authorize_account.

class b2sdk.raw_simulator.**PartSimulator**(*file_id*, *part_number*, *content_length*, *content_sha1*, *part_data*)

Bases: object

__init__(*file_id*, *part_number*, *content_length*, *content_sha1*, *part_data*)
Initialize self. See help(type(self)) for accurate signature.

as_list_parts_dict()

class b2sdk.raw_simulator.**FileSimulator**(*account_id*, *bucket_id*, *file_id*, *action*, *name*, *content_type*, *content_sha1*, *file_info*, *data_bytes*, *upload_timestamp*, *range_=None*, *server_side_encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*)
Bases: object

One of three: an unfinished large file, a finished file, or a deletion marker.

CHECK_ENCRYPTION = True

__init__(*account_id*, *bucket_id*, *file_id*, *action*, *name*, *content_type*, *content_sha1*, *file_info*, *data_bytes*, *upload_timestamp*, *range_=None*, *server_side_encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*)
Initialize self. See help(type(self)) for accurate signature.

classmethod dont_check_encryption()

sort_key()
Return a key that can be used to sort the files in a bucket in the order that b2_list_file_versions returns them.

as_download_headers(*range_=None*)

as_upload_result()

as_list_files_dict()

as_start_large_file_result()

add_part(*part_number*, *part*)

finish(*part_sha1_array*)

is_visible()
Does this file show up in b2_list_file_names?

list_parts(*start_part_number*, *max_part_count*)

check_encryption(*request_encryption: Optional[b2sdk.encryption.setting.EncryptionSetting]*)

class b2sdk.raw_simulator.**FakeRequest**(*url*, *headers*)
Bases: tuple

headers
Alias for field number 1

url
Alias for field number 0

class b2sdk.raw_simulator.**FakeResponse**(*file_sim*, *url*, *range_=None*)
Bases: object

__init__(*file_sim*, *url*, *range_=None*)
Initialize self. See help(type(self)) for accurate signature.

iter_content(*chunk_size=1*)

property request

close()

class b2sdk.raw_simulator.**BucketSimulator**(*api*, *account_id*, *bucket_id*, *bucket_name*, *bucket_type*, *bucket_info=None*, *cors_rules=None*, *lifecycle_rules=None*, *options_set=None*, *default_server_side_encryption=None*)

Bases: object

FIRST_FILE_NUMBER = 9999

FIRST_FILE_ID = '9999'

FILE_SIMULATOR_CLASS
alias of *b2sdk.raw_simulator.FileSimulator*

RESPONSE_CLASS
alias of *b2sdk.raw_simulator.FakeResponse*

__init__(*api*, *account_id*, *bucket_id*, *bucket_name*, *bucket_type*, *bucket_info=None*, *cors_rules=None*, *lifecycle_rules=None*, *options_set=None*, *default_server_side_encryption=None*)
Initialize self. See help(type(self)) for accurate signature.

bucket_dict(*account_auth_token*)

cancel_large_file(*file_id*)

delete_file_version(*file_id*, *file_name*)

download_file_by_id(*file_id*, *url*, *range_=None*, *encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*)

download_file_by_name(*file_name*, *url*, *range_=None*, *encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*)

finish_large_file(*file_id*, *part_sha1_array*)

get_file_info_by_id(*file_id*)

get_file_info_by_name(*file_name*)

get_upload_url()

get_upload_part_url(*file_id*)

hide_file(*file_name*)

copy_file(*file_id*, *new_file_name*, *bytes_range=None*, *metadata_directive=None*, *content_type=None*, *file_info=None*, *destination_bucket_id=None*, *destination_server_side_encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*, *source_server_side_encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*)

list_file_names(*start_file_name=None*, *max_file_count=None*, *prefix=None*)

list_file_versions(*start_file_name=None*, *start_file_id=None*, *max_file_count=None*, *prefix=None*)

list_parts(*file_id*, *start_part_number*, *max_part_count*)

list_unfinished_large_files(*start_file_id=None*, *max_file_count=None*, *prefix=None*)

start_large_file(*file_name*, *content_type*, *file_info*, *server_side_encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*)

upload_file(*upload_id*, *upload_auth_token*, *file_name*, *content_length*, *content_type*, *content_sha1*, *file_infos*, *data_stream*, *server_side_encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*)

upload_part(*file_id*, *part_number*, *content_length*, *sha1_sum*, *input_stream*, *server_side_encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*)

class b2sdk.raw_simulator.**RawSimulator**

Bases: *b2sdk.raw_api.AbstractRawApi*

Implement the same interface as B2RawApi by simulating all of the calls and keeping state in memory.

The intended use for this class is for unit tests that test things built on top of B2RawApi.

BUCKET_SIMULATOR_CLASS

alias of *b2sdk.raw_simulator.BucketSimulator*

API_URL = 'http://api.example.com'

S3_API_URL = 'http://s3.api.example.com'

DOWNLOAD_URL = 'http://download.example.com'

MIN_PART_SIZE = 200

MAX_DURATION_IN_SECONDS = 86400000

UPLOAD_PART_MATCHER = re.compile('https://upload.example.com/part/([^/]*)')

UPLOAD_URL_MATCHER = re.compile('https://upload.example.com/([^/]*)/([^/]*)')

DOWNLOAD_URL_MATCHER = re.compile('http://download.example.com(?:/b2api/v[0-9]+/b2_dow

__init__()

Initialize self. See help(type(self)) for accurate signature.

expire_auth_token(*auth_token*)

Simulate the auth token expiring.

The next call that tries to use this auth token will get an auth_token_expired error.

create_account()

Return (accountId, masterApplicationKey) for a newly created account.

set_upload_errors(*errors*)

Store a sequence of exceptions to raise on upload. Each one will be raised in turn, until they are all gone. Then the next upload will succeed.

authorize_account(*realm_url*, *application_key_id*, *application_key*)

cancel_large_file(*api_url*, *account_auth_token*, *file_id*)

create_bucket(*api_url*, *account_auth_token*, *account_id*, *bucket_name*, *bucket_type*, *bucket_info=None*, *cors_rules=None*, *lifecycle_rules=None*, *default_server_side_encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*)

create_key(*api_url*, *account_auth_token*, *account_id*, *capabilities*, *key_name*, *valid_duration_seconds*, *bucket_id*, *name_prefix*)

delete_file_version(*api_url*, *account_auth_token*, *file_id*, *file_name*)

delete_bucket(*api_url*, *account_auth_token*, *account_id*, *bucket_id*)

download_file_from_url(*account_auth_token_or_none*, *url*, *range_=None*, *encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*)

delete_key(*api_url*, *account_auth_token*, *application_key_id*)

finish_large_file(*api_url*, *account_auth_token*, *file_id*, *part_sha1_array*)

get_download_authorization(*api_url*, *account_auth_token*, *bucket_id*, *file_name_prefix*, *valid_duration_in_seconds*)

get_file_info_by_id(*api_url*, *account_auth_token*, *file_id*)

get_file_info_by_name(*api_url*, *account_auth_token*, *bucket_name*, *file_name*)

get_upload_url(*api_url*, *account_auth_token*, *bucket_id*)

get_upload_part_url(*api_url*, *account_auth_token*, *file_id*)

hide_file(*api_url*, *account_auth_token*, *bucket_id*, *file_name*)

copy_file(*api_url*, *account_auth_token*, *source_file_id*, *new_file_name*, *bytes_range=None*, *metadata_directive=None*, *content_type=None*, *file_info=None*, *destination_bucket_id=None*, *destination_server_side_encryption=None*, *source_server_side_encryption=None*)

copy_part(*api_url*, *account_auth_token*, *source_file_id*, *large_file_id*, *part_number*, *bytes_range=None*, *destination_server_side_encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*, *source_server_side_encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*)

list_buckets(*api_url*, *account_auth_token*, *account_id*, *bucket_id=None*, *bucket_name=None*)

list_file_names(*api_url*, *account_auth_token*, *bucket_id*, *start_file_name=None*, *max_file_count=None*, *prefix=None*)

list_file_versions(*api_url*, *account_auth_token*, *bucket_id*, *start_file_name=None*, *start_file_id=None*, *max_file_count=None*, *prefix=None*)

list_keys(*api_url*, *account_auth_token*, *account_id*, *max_key_count=1000*, *start_application_key_id=None*)

list_parts(*api_url*, *account_auth_token*, *file_id*, *start_part_number*, *max_part_count*)

list_unfinished_large_files(*api_url*, *account_auth_token*, *bucket_id*, *start_file_id=None*, *max_file_count=None*, *prefix=None*)

start_large_file(*api_url*, *account_auth_token*, *bucket_id*, *file_name*, *content_type*, *file_info*, *server_side_encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*)

update_bucket(*api_url*, *account_auth_token*, *account_id*, *bucket_id*, *bucket_type=None*, *bucket_info=None*, *cors_rules=None*, *lifecycle_rules=None*, *if_revision_is=None*, *default_server_side_encryption=None*)

upload_file(*upload_url*, *upload_auth_token*, *file_name*, *content_length*, *content_type*, *content_sha1*, *file_infos*, *data_stream*, *server_side_encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*)

upload_part(*upload_url*, *upload_auth_token*, *part_number*, *content_length*, *sha1_sum*, *input_stream*, *server_side_encryption: Optional[b2sdk.encryption.setting.EncryptionSetting] = None*)

2.9 Contributors Guide

We encourage outside contributors to perform changes on our codebase. Many such changes have been merged already. In order to make it easier to contribute, core developers of this project:

- provide guidance (through the issue reporting system)
- provide tool assisted code review (through the Pull Request system)
- maintain a set of unit tests
- maintain a set of integration tests (run with a production cloud)
- maintain development automation tools using nox that can easily:
  - format the code using yapf
  - runs linters to find subtle/potential issues with maintainability
  - run the test suite on multiple Python versions using pytest
- maintain Continuous Integration (by using GitHub Actions) that:
  - runs all sorts of linters
  - checks if the Python distribution can be built
  - runs all tests on a matrix of 6 versions of Python (including pypy) and 3 operating systems (Linux, Mac OS X and Windows)
  - checks if the documentation can be built properly
- maintain other Continuous Integration tools (coverage tracker)

You'll need to have nox installed:

- `pip install nox`

With `nox`, you can run different sessions (default are `lint` and `test`):

- `format` -> Format the code.
- `lint` -> Run linters.
- `test` (`test-3.5`, `test-3.6`, `test-3.7`, `test-3.8`, `test-3.9`) -> Run test suite.
- `cover` -> Perform coverage analysis.
- `build` -> Build the distribution.
- `deploy` -> Deploy the distribution to the PyPi.
- `doc` -> Build the documentation.
- `doc_cover` -> Perform coverage analysis for the documentation.

For example:

```
$ nox -s format
nox > Running session format
nox > Creating virtual environment (virtualenv) using python3.9 in .nox/format
...

$ nox -s format
nox > Running session format
nox > Re-using existing virtual environment at .nox/format.
```

(continues on next page)

(continued from previous page)

```
...

$ nox --no-venv -s format
nox > Running session format
...
```

Sessions `test`, `unit`, and `integration` can run on many Python versions, 3.5-3.9 by default.

Sessions other than `test` use the last given Python version, 3.9 by default.

You can change it:

```
export NOX_PYTHONS=3.6,3.7
```

With the above setting, session `test` will run on Python 3.6 and 3.7, and all other sessions on Python 3.7.

Given Python interpreters should be installed in the operating system or via pyenv.

2.9.1 Linting

To run all available linters:

```
$ nox -s lint
```

2.9.2 Testing

To run all tests on every available Python version:

```
$ nox -s test
```

To run all tests on a specific version:

```
$ nox -s test-3.9
```

To run just unit tests:

```
$ nox -s unit-3.9
```

To run just integration tests:

```
$ export B2_TEST_APPLICATION_KEY=your_app_key
$ export B2_TEST_APPLICATION_KEY_ID=your_app_key_id
$ nox -s integration-3.9
```

2.9.3 Documentation

To build the documentation and watch for changes (including the source code):

```
$ nox -s doc
```

To just build the documentation:

```
$ nox --non-interactive -s doc
```

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

b

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

Z