B2_Python_SDK

Release 1.17.0

Backblaze

Jun 22, 2022

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.v2.InMemoryAccountInfo`:

```
>>> from b2sdk.v2 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.v2 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.v2.B2Api.authorize_account()`

2.2.3 B2Api

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

Typical B2Api operations

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

to find out more, see `b2sdk.v2.B2Api`.

The most practical operation on `B2Api` object is `b2sdk.v2.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.v2.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

To find out more, see `b2sdk.v2.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.v2 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.v2 import ScanPoliciesManager
>>> from b2sdk.v2 import parse_folder
>>> from b2sdk.v2 import Synchronizer
>>> from b2sdk.v2 import SyncReport
>>> import time
>>> import sys

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

>>> source = parse_folder(source, b2_api)
>>> destination = parse_folder(destination, b2_api)
```

(continues on next page)

(continued from previous page)

```
>>> 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)),
            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.v2.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.v2.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)
>>> new_bucket = bucket.update(
        bucket_type=new_bucket_type,
        default_server_side_encryption=EncryptionSetting(mode=EncryptionMode.SSE_B2)
    )
>>> new_bucket.as_dict()
{'accountId': '451862be08d0',
 'bucketId': '5485a1682662eb3e60980d10',
 'bucketInfo': {},
 'bucketName': 'example-mybucket-b2',
 'bucketType': 'allPrivate',
 'corsRules': [],
 'lifecycleRules': [],
 'revision': 3,
 'defaultServerSideEncryption': {'isClientAuthorizedToRead': True,
```

(continues on next page)

(continued from previous page)

```
                                    'value': {'algorithm': 'AES256', 'mode': 'SSE-B2'}}},
}
```

For more information see `b2sdk.v2.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,
    )
<b2sdk.file_version.FileVersion 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.v2.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,
```

(continues on next page)

(continued from previous page)

```
        file_infos=file_info,
        encryption=encryption_setting,
    )
```

Download file

By id

```
>>> from b2sdk.v2 import DoNothingProgressListener

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

>>> downloaded_file = b2_api.download_file_by_id(file_id, progress_listener)  # only the␣
↪headers
    # and the beginning of the file is downloaded at this stage

>>> print('File name:   ', downloaded_file.download_version.file_name)
File name:    som2.pdf
>>> print('File id:     ', downloaded_file.download_version.id_)
File id:      4_z5485a1682662eb3e60980d10_f1195145f42952533_d20190403_m130258_c002_
↪v0001111_t0002
>>> print('File size:   ', downloaded_file.download_version.size)
File size:    1870579
>>> print('Content type:', downloaded_file.download_version.content_type)
Content type: application/pdf
>>> print('Content sha1:', downloaded_file.download_version.content_sha1)
Content sha1: d821849a70922e87c2b0786c0be7266b89d87df0

>>> downloaded_file.save_to(local_file_path)  # this downloads the whole file
```

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'
>>> downloaded_file = bucket.download_file_by_name(b2_file_name)
>>> downloaded_file.save_to(local_file_path)
```

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_version, folder_name in bucket.ls(latest_only=True):
>>>     print(file_version.file_name, file_version.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_version, folder_name in bucket.ls(latest_only=True, recursive=True):
>>>     print(file_version.file_name, file_version.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_version, folder_name in bucket.ls(folder_to_list='test-folder', latest_
↪only=True):
>>>     print(file_version.file_name, file_version.upload_timestamp, folder_name)
test-folder/.bzEmpty 1561005295000 None
test-folder/folder_file.txt 1561005349000 None

# list file versions
>>> for file_version, folder_name in bucket.ls(latest_only=False):
>>>     print(file_version.file_name, file_version.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.v2.Bucket.ls()`.

Get file metadata

```
>>> file_id = '4_z5485a1682662eb3e60980d10_f113f963288e711a6_d20190404_m065910_c002_
↪v0001095_t0044'
>>> file_version = b2_api.get_file_info(file_id)
>>> file_version.as_dict()
{'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"},
 }
```

Update file lock configuration

```
>>> file_id = '4_z5485a1682662eb3e60980d10_f113f963288e711a6_d20190404_m065910_c002_
↪v0001095_t0044'
>>> file_name = 'dummy.pdf'
>>> b2_api.update_file_legal_hold(file_id, file_name, LegalHold.ON)
>>> b2_api.update_file_legal_hold(
        file_id, file_name,
        FileRetentionSetting(RetentionMode.GOVERNANCE, int(time.time() + 100)*1000))
```

This is low-level file API, for high-level operations see *Direct file operations*.

Copy file

```
>>> file_id = '4_z5485a1682662eb3e60980d10_f118df9ba2c5131e8_d20190619_m065809_c002_
↪v0001126_t0040'
>>> new_file_version = bucket.copy(file_id, 'f2_copy.txt')
>>> new_file_version.as_dict()
{'accountId': '451862be08d0',
 'action': 'copy',
 'bucketId': '5485a1682662eb3e60980d10',
 'contentLength': 124,
 'contentSha1': '737637702a0e41dda8b7be79c8db1d369c6eef4a',
 'contentType': 'text/plain',
 'fileId': '4_z5485a1682662eb3e60980d10_f1022e2320daf707f_d20190620_m122848_c002_
↪v0001123_t0020',
 'fileInfo': {'src_last_modified_millis': '1560848707000'},
 'fileName': 'f2_copy.txt',
 'uploadTimestamp': 1561033728000,
```

(continues on next page)

(continued from previous page)

```
 "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.v2.Bucket.copy()`.

Delete file

```
>>> file_id = '4_z5485a1682662eb3e60980d10_f113f963288e711a6_d20190404_m065910_c002_
↪v0001095_t0044'
>>> file_id_and_name = b2_api.delete_file_version(file_id, 'dummy_new.pdf')
>>> file_id_and_name.as_dict()
{'action': 'delete',
 'fileId': '4_z5485a1682662eb3e60980d10_f113f963288e711a6_d20190404_m065910_c002_
↪v0001095_t0044',
 'fileName': 'dummy_new.pdf'}
```

This is low-level file API, for high-level operations see *Direct file operations*.

Cancel large file uploads

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

2.3.5 Direct file operations

Methods for manipulating object (file) state mentioned in sections above are low level and useful when users have access to basic information, like file id and name. Many API methods, however, return python objects representing files (`b2sdk.v2.FileVersion` and `b2sdk.v2.DownloadVersion`), that provide high-level access to methods manipulating their state. As a rule, these methods don't change properties of python objects they are called on, but return new objects instead.

Obtain file representing objects

b2sdk.v2.FileVersion

By id

```
>>> file_id = '4_z5485a1682662eb3e60980d10_f113f963288e711a6_d20190404_m065910_c002_
↪v0001095_t0044'
>>> file_version = b2_api.get_file_info(file_id)
```

By listing

```
>>> for file_version, folder_name in bucket.ls(latest_only=True, prefix='dir_name'):
>>>     ...
```

b2sdk.v2.DownloadVersion

By id

```
>>> file_id = '4_z5485a1682662eb3e60980d10_f113f963288e711a6_d20190404_m065910_c002_
↪v0001095_t0044'
>>> downloaded_file = b2_api.download_file_by_id(file_id)
>>> download_version = downloaded_file.download_version
```

By name

```
>>> bucket = b2_api.get_bucket_by_name(bucket_name)
>>> b2_file_name = 'dummy_new.pdf'
>>> downloaded_file = bucket.download_file_by_name(b2_file_name)
>>> download_version = downloaded_file.download_version
```

Download (only for b2sdk.v2.FileVersion)

```
>>> file_version.download()
>>> # equivalent to
>>> b2_api.download_file_by_id(file_version.id_)
```

Delete

```
>>> file_version.delete()
>>> download_version.delete()
>>> # equivalent to
>>> b2_api.delete_file_version(file_version.id_, file_version.file_name)
>>> b2_api.delete_file_version(download_version.id_, download_version.file_name)
```

Update file lock

```
>>> file_version.update_legal_hold(LegalHold.ON)
>>> download_version.update_legal_hold(LegalHold.ON)
>>> file_version.update_retention(
        FileRetentionSetting(RetentionMode.GOVERNANCE, int(time.time() + 100)*1000))
>>> download_version.update_retention(
        FileRetentionSetting(RetentionMode.GOVERNANCE, int(time.time() + 100)*1000))
>>> # equivalent to
>>> b2_api.update_file_legal_hold(file_version.id_, file_version.file_name, LegalHold.ON)
>>> b2_api.update_file_legal_hold(download_version.id_, download_version.file_name,␣
↪LegalHold.ON)
>>> b2_api.update_file_legal_hold(
        file_version.id_, file_version.file_name,
        FileRetentionSetting(RetentionMode.GOVERNANCE, int(time.time() + 100)*1000))
>>> b2_api.update_file_legal_hold(
        download_version.id_, download_version.file_name,
        FileRetentionSetting(RetentionMode.GOVERNANCE, int(time.time() + 100)*1000))
```

Usage examples

```
>>> for file_version, folder_name in bucket.ls(latest_only=True, prefix='dir_name'):
>>>     if file_version.mod_time_millis < 1627979193913 and file_version.file_name.
↪endswith('.csv'):
>>>         if file_version.legal_hold.is_on():
>>>             file_version = file_version.update_legal_hold(LegalHold.OFF)
>>>         file_version.delete()
>>>     else:
>>>         file_version.download().save_to(Path('/tmp/dir_name') / file_version.file_
↪name)
```

```
>>> file_id = '4_z5485a1682662eb3e60980d10_f113f963288e711a6_d20190404_m065910_c002_
↪v0001095_t0044'
>>> downloaded_file = b2_api.download_file_by_id(file_id)
>>> download_version = downloaded_file.download_version
>>> if download_version.content_type == 'video/mp4':
>>>     downloaded_file.save_to(Path('/tmp/dir_name') / download_version.file_name)
>>> if download_version.file_retention != NO_RETENTION_FILE_SETTING:
>>>     download_version = download_version.update_retention(
            NO_RETENTION_FILE_SETTING, bypass_governance=True)
>>> download_version.delete()
```

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

- *Encryption Settings*
- *Encryption Types*
- *Sync Encryption Settings Providers*

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.v2.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.v2.Bucket.upload()`	local	no	no	automatic
`b2sdk.v2.Bucket.copy()`	remote	no	no	automatic
`b2sdk.v2.Bucket.concatenate()`	any	no	no	automatic
`b2sdk.v2.Bucket.concatenate_stream()`	any	no	yes	manual
`b2sdk.v2.Bucket.create_file()`	any	yes	no	automatic
`b2sdk.v2.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.v2.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.v2.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.FileVersion 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.v2.Bucket.create_file()` can be used with overalapping ranges in input.

For more information about `concatenate` please see `b2sdk.v2.Bucket.concatenate()` and `b2sdk.v2.CopySource`.

Concatenate files of known size (streamed version)

`b2sdk.v2.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.v2.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.FileVersion 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.v2.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.FileVersion at 0x7fc8cd560552>
```

LocalUploadSource has the size determined automatically in this case. This is more efficient than `b2sdk.v2.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.v2.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.FileVersion at 0x7fc8cd560552>
```

LocalUploadSource has the size determined automatically in this case. This is more efficient than `b2sdk.v2.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.v2.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

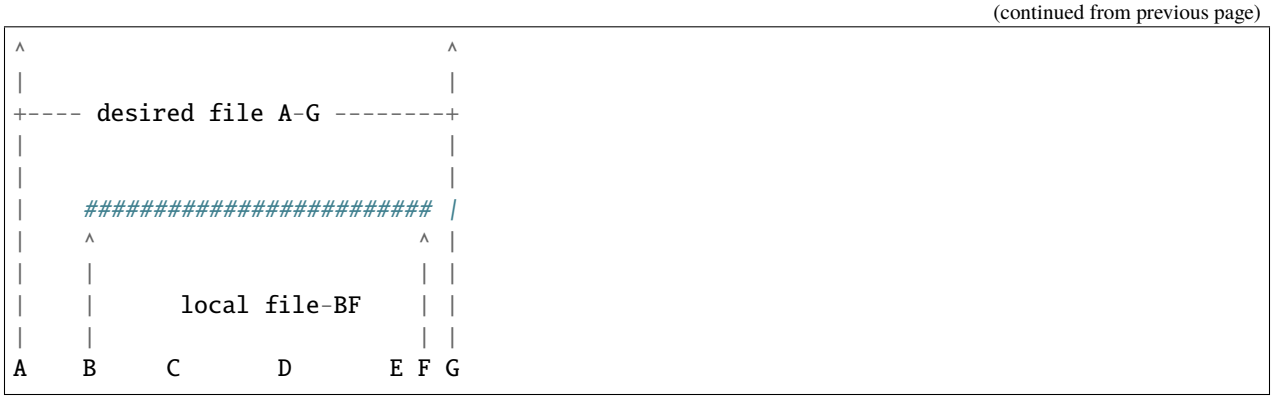
Please note that `b2sdk.v2.Bucket.create_file_stream()` accepts **an ordered iterable** which *can contain overlapping ranges*, so the operation does not need to be planned ahead, but can be streamed, which supports very large output objects.

Scenarios such as below are then possible:

```
A           C       D           G
|           |       |           |
| cloud-AC  |       | cloud-DG  |
|           |       |           |
v           v       v           v
############       #############
```

(continues on next page)

(continued from previous page)




```
>>> 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.FileVersion 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.v2.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.v2.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.v2.Bucket.create_file()` call, because `b2sdk.v2.CopySource` accepts an optional `b2sdk.v2.EncryptionSetting`.

Prioritize remote or local sources

`b2sdk.v2.Bucket.create_file()` and `b2sdk.v2.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*.

```
A              D               G
|              |               |
| cloud-AD     |               |
|              |               |
v              v               |
#################              /
^                              |
|                              |
+---- desired file A-G --------+
|                              |
|                              |
|    #######   #################
|    ^     ^   ^               |
|    |     |   |               |
|    |   local file BC and DE  |
|    |     |   |               |
A    B     C   D               E

A=0, B=50M, C=80M, D=100M, E=200
```

```
>>> 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 matching upload sessions are matching the transfer.

Continuation of create/concantenate

`b2sdk.v2.Bucket.create_file()` supports automatic continuation or manual continuation. `b2sdk.v2.Bucket.create_file_stream()` supports only manual continuation for local-only inputs. The situation looks the same for `b2sdk.v2.Bucket.concatenate()` and `b2sdk.v2.Bucket.concatenate_stream()` (streamed version supports only manual continuation of local sources). Also `b2sdk.v2.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.v2.Bucket.create_file()` - internally `b2sdk.v2.Bucket.create_file()` stores plan information in file info for such inputs, and verifies it before any

copy/upload and `b2sdk.v2.Bucket.create_file_stream()` cannot store this information. Local source only inputs can be safely continued with `b2sdk.v2.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.v2.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.v2.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.v2.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.v2.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>=1.15.0,<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.v2 import ...`. The `.v2` 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 have already 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.v2.ExceptionC` inherits directly from `b2sdk.v2.ExceptionA`, it may one day inherit from `b2sdk.v2.ExceptionB`, which in turn inherits from `b2sdk.v2.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 `as_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.

Note: Replication is currently in a Closed Beta state, where not all B2 accounts have access to the feature. The interface of the beta server API might change and the interface of **b2sdk** around replication may change as well. For the avoidance of doubt, until this message is removed, replication-related functionality of **b2sdk** should be considered as internal interface.

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

B2 Application key

AccountInfo

AccountInfo stores basic information about the account, such as *Application Key ID* and *Application Key*, in order to let `b2sdk.v2.B2Api` perform authenticated requests.

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

- `b2sdk.v2.InMemoryAccountInfo` - a basic implementation with no persistence
- `b2sdk.v2.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.

SqliteAccountInfo

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.v2.UrlPoolAccountInfo`, which has groundwork for url pool functionality). If you cannot use it, inherit directly from `b2sdk.v2.AbstractAccountInfo`.

```
>>> from b2sdk.v2 import UrlPoolAccountInfo
>>> class MyAccountInfo(UrlPoolAccountInfo):
        ...
```

`b2sdk.v2.AbstractAccountInfo` describes the interface, while `b2sdk.v2.UrlPoolAccountInfo` and `b2sdk.v2.UploadUrlPool` implement a part of the interface for in-memory upload token management.

AccountInfo interface

AccountInfo helper classes

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.

B2 Api client

Exceptions

B2 Bucket

File locks

Data classes

Downloaded File

Enums

Progress reporters

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

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.v2.CompareVersionMode`. Default value is `b2sdk.v2.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.v2.NewerFileSyncMode`. If you don't specify this the sync will raise `b2sdk.v2.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.v2.KeepOrDeleteMode`. Default is *DO_NOTHING*.
- `keep_days`: if *keep_days_or_delete* is `b2sdk.v2.KeepOrDeleteMode.KEEP_BEFORE_DELETE` then this specifies for how many days should we keep.

```
>>> from b2sdk.v2 import ScanPoliciesManager
>>> from b2sdk.v2 import parse_folder
>>> from b2sdk.v2 import Synchronizer
>>> from b2sdk.v2 import KeepOrDeleteMode, CompareVersionMode, NewerFileSyncMode
>>> import time
>>> import sys

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

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

>>> policies_manager = ScanPoliciesManager(exclude_all_symlinks=True)

>>> synchronizer = Synchronizer(
        max_workers=10,
        policies_manager=policies_manager,
```

(continues on next page)

(continued from previous page)

```
        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,
        )
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,
```

(continues on next page)

(continued from previous page)

```
        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,
```

(continues on next page)

(continued from previous page)

```
        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 further explanation and *Sync Encryption Settings Providers* for public API.

Public API classes

Sync Encryption Settings Providers

B2 Utility functions

Write intent

Outbound Transfer Source

Encryption Settings

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()

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

An enumeration.

API = 'api'

API_TOKEN_ONLY = 'api_token_only'

UPLOAD_PART = 'upload_part'

UPLOAD_SMALL = 'upload_small'

class b2sdk.session.**B2Session**(*account_info: ~typing.Optional[~b2sdk.account_info.abstract.AbstractAccountInfo] = None*, *cache: ~typing.Optional[~b2sdk.cache.AbstractCache] = None*, *api_config: ~b2sdk.api_config.B2HttpApiConfig = <b2sdk.api_config.B2HttpApiConfig object>*)

Bases: object

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

SQLITE_ACCOUNT_INFO_CLASS

alias of SqliteAccountInfo

B2HTTP_CLASS

alias of *B2Http*

__init__(*account_info: ~typing.Optional[~b2sdk.account_info.abstract.AbstractAccountInfo] = None*, *cache: ~typing.Optional[~b2sdk.cache.AbstractCache] = None*, *api_config: ~b2sdk.api_config.B2HttpApiConfig = <b2sdk.api_config.B2HttpApiConfig object>*)

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*

:param api_config

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*, *is_file_lock_enabled: Optional[bool] = None*, *replication: Optional[ReplicationConfiguration] = 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[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[EncryptionSetting] = None*, *file_retention: Optional[FileRetentionSetting] = None*, *legal_hold: Optional[LegalHold] = 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[EncryptionSetting] = None*, *default_retention: Optional[BucketRetentionSetting] = None*, *replication: Optional[ReplicationConfiguration] = None*)

upload_file(*bucket_id*, *file_name*, *content_length*, *content_type*, *content_sha1*, *file_infos*, *data_stream*, *server_side_encryption: Optional[EncryptionSetting] = None*, *file_retention: Optional[FileRetentionSetting] = None*, *legal_hold: Optional[LegalHold] = None*)

upload_part(*file_id*, *part_number*, *content_length*, *sha1_sum*, *input_stream*, *server_side_encryption: Optional[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[EncryptionSetting] = None*, *source_server_side_encryption: Optional[EncryptionSetting] = None*, *file_retention: Optional[FileRetentionSetting] = None*, *legal_hold: Optional[LegalHold] = None*)

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

update_file_retention(*file_id*, *file_name*, *file_retention: FileRetentionSetting*, *bypass_governance: bool = False*)

update_file_legal_hold(*file_id*, *file_name*, *legal_hold: LegalHold*)

b2sdk.raw_api – B2 raw api wrapper

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

Bases: 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[EncryptionSetting] = None*, *source_server_side_encryption: Optional[EncryptionSetting] = None*, *file_retention: Optional[FileRetentionSetting] = None*, *legal_hold: Optional[LegalHold] = 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[EncryptionSetting] = None*, *source_server_side_encryption: Optional[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[EncryptionSetting] = None*, *is_file_lock_enabled: Optional[bool] = None*, *replication: Optional[ReplicationConfiguration] = 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[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[EncryptionSetting] = None*, *file_retention: Optional[FileRetentionSetting] = None*, *legal_hold: Optional[LegalHold] = 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[EncryptionSetting] = None*, *default_retention: Optional[BucketRetentionSetting] = None*, *replication: Optional[ReplicationConfiguration] = None*)

abstract update_file_retention(*api_url*, *account_auth_token*, *file_id*, *file_name*, *file_retention: FileRetentionSetting*, *bypass_governance: bool = False*)

classmethod get_upload_file_headers(*upload_auth_token: str*, *file_name: str*, *content_length: int*, *content_type: str*, *content_sha1: str*, *file_infos: dict*, *server_side_encryption: Optional[EncryptionSetting]*, *file_retention: Optional[FileRetentionSetting]*, *legal_hold: Optional[LegalHold]*) → dict

abstract upload_file(*upload_url*, *upload_auth_token*, *file_name*, *content_length*, *content_type*, *content_sha1*, *file_infos*, *data_stream*, *server_side_encryption: Optional[EncryptionSetting] = None*, *file_retention: Optional[FileRetentionSetting] = None*, *legal_hold: Optional[LegalHold] = None*)

abstract upload_part(*upload_url*, *upload_auth_token*, *part_number*, *content_length*, *sha1_sum*, *input_stream*, *server_side_encryption: Optional[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.**B2RawHTTPApi**(*b2_http*)

Bases: *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*)

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[EncryptionSetting] = None*, *is_file_lock_enabled: Optional[bool] = None*, *replication: Optional[ReplicationConfiguration] = 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[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.v2.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[EncryptionSetting] = None*, *file_retention: Optional[FileRetentionSetting] = None*, *legal_hold: Optional[LegalHold] = 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: Optional[EncryptionSetting] = None*, *default_retention: Optional[BucketRetentionSetting] = None*, *replication: Optional[ReplicationConfiguration] = None*)

update_file_retention(*api_url*, *account_auth_token*, *file_id*, *file_name*, *file_retention: FileRetentionSetting*, *bypass_governance: bool = False*)

update_file_legal_hold(*api_url*, *account_auth_token*, *file_id*, *file_name*, *legal_hold: LegalHold*)

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[EncryptionSetting] = None*, *file_retention: Optional[FileRetentionSetting] = None*, *legal_hold: Optional[LegalHold] = 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[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[EncryptionSetting] = None*, *source_server_side_encryption: Optional[EncryptionSetting] = None*, *file_retention: Optional[FileRetentionSetting] = None*, *legal_hold: Optional[LegalHold] = None*)

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

b2sdk.b2http – thin http client wrapper

b2sdk.b2http.**random**() → x in the interval [0, 1).

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**(*api_config: ~b2sdk.api_config.B2HttpApiConfig = <b2sdk.api_config.B2HttpApiConfig object>*)

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:
    ...
```

Please note that the timeout/retry system, including class-level variables, is not a part of the interface and is subject to change.

TIMEOUT = 128

TIMEOUT_FOR_COPY = 1200

TIMEOUT_FOR_UPLOAD = 128

TRY_COUNT_DATA = 20

TRY_COUNT_DOWNLOAD = 20

TRY_COUNT_HEAD = 5

TRY_COUNT_OTHER = 5

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: int = 20*, *post_params=None*, *_timeout: Optional[int] = 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: int = 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: int = 20*)

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

class b2sdk.b2http.**NotDecompressingHTTPAdapter**(*pool_connections=10*, *pool_maxsize=10*, *max_retries=0*, *pool_block=False*)

HTTP adapter that uses *b2sdk.requests.NotDecompressingResponse* instead of the default `requests.Response` class.

build_response(*req*, *resp*)

Builds a `Response` object from a urllib3 response. This should not be called from user code, and is only exposed for use when subclassing the `HTTPAdapter`

Parameters

- **req** – The `PreparedRequest` used to generate the response.
- **resp** – The urllib3 response object.

Return type
requests.Response

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.requests – modified requests.models.Response class

This file contains modified parts of the requests module (https://github.com/psf/requests, models.py), original Copyright 2019 Kenneth Reitz

Changes made to the original source: see NOTICE

class b2sdk.requests.**NotDecompressingResponse**

iter_content(*chunk_size=1*, *decode_unicode=False*)

Iterates over the response data. When stream=True is set on the request, this avoids reading the content at once into memory for large responses. The chunk size is the number of bytes it should read into memory. This is not necessarily the length of each item returned as decoding can take place.

chunk_size must be of type int or None. A value of None will function differently depending on the value of *stream*. stream=True will read data as it arrives in whatever size the chunks are received. If stream=False, data is returned as a single chunk.

If decode_unicode is True, content will be decoded using the best available encoding based on the response.

classmethod from_builtin_response(*response: Response*)

Create a *b2sdk.requests.NotDecompressingResponse* object from a `requests.Response` object. Don't use `Response.__getstate__` and `Response.__setstate__` because these assume that the content has been consumed, which will never be true in our case.

b2sdk.utils

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_file**(*path_*)

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: DefaultTraceMeta

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

class b2sdk.utils.**B2TraceMetaAbstract**(*name*, *bases*, *namespace*, ***kwargs*)

Bases: 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*)

b2sdk.utils.**current_time_millis**()

File times are in integer milliseconds, to avoid roundoff errors.

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 get_bucket_name_or_none_from_bucket_id(*bucket_id: str*) → Optional[str]

abstract save_bucket(*bucket*)

abstract set_bucket_name_cache(*buckets*)

class b2sdk.cache.**DummyCache**

Bases: *AbstractCache*

A cache that does nothing.

get_bucket_id_or_none_from_bucket_name(*name*)

get_bucket_name_or_none_from_bucket_id(*bucket_id: str*) → Optional[str]

get_bucket_name_or_none_from_allowed()

save_bucket(*bucket*)

set_bucket_name_cache(*buckets*)

class b2sdk.cache.**InMemoryCache**

Bases: *AbstractCache*

A cache that stores the information in memory.

__init__()

get_bucket_id_or_none_from_bucket_name(*name*)

get_bucket_name_or_none_from_bucket_id(*bucket_id: str*) → Optional[str]

get_bucket_name_or_none_from_allowed()

save_bucket(*bucket*)

set_bucket_name_cache(*buckets*)

class b2sdk.cache.**AuthInfoCache**(*info*)

Bases: *AbstractCache*

A cache that stores data persistently in StoredAccountInfo.

__init__(*info*)

get_bucket_id_or_none_from_bucket_name(*name*)

get_bucket_name_or_none_from_bucket_id(*bucket_id*) → Optional[str]

get_bucket_name_or_none_from_allowed()

save_bucket(*bucket*)

set_bucket_name_cache(*buckets*)

b2sdk.stream.chained ChainedStream

class b2sdk.stream.chained.**ChainedStream**(*stream_openers*)

Bases: ReadOnlyStreamMixin, 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.v2.StreamOpener` cleanup interface, for example `b2sdk.v2.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: ReadOnlyStreamMixin, *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: *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.v2.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: *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.v2.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: *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: ReadOnlyStreamMixin, *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

`close`()

Flush and close the IO object.

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

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

b2sdk.stream.wrapper StreamWrapper

`class b2sdk.stream.wrapper.StreamWrapper`(*stream*)

Bases: 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

`class b2sdk.stream.wrapper.StreamWithLengthWrapper`(*stream*, *length=None*)

Bases: *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.scan.folder_parser`

`b2sdk.scan.folder`

`b2sdk.scan.path`

`b2sdk.scan.policies`

`b2sdk.scan.scan`

b2sdk.sync.action

b2sdk.sync.exception

b2sdk.sync.policy

b2sdk.sync.policy_manager

b2sdk.sync.sync

b2sdk.transfer.inbound.downloader.abstract – Downloader base class

class b2sdk.transfer.inbound.downloader.abstract.**EmptyHasher**(**args*, ***kwargs*)

Bases: object

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

update(*data*)

digest()

hexdigest()

copy()

class b2sdk.transfer.inbound.downloader.abstract.**AbstractDownloader**(*thread_pool: Optional[ThreadPoolExecutor] = None*, *force_chunk_size: Optional[int] = None*, *min_chunk_size: Optional[int] = None*, *max_chunk_size: Optional[int] = None*, *align_factor: Optional[int] = None*, *check_hash: bool = True*, ***kwargs*)

Bases: object

REQUIRES_SEEKING = True

DEFAULT_THREAD_POOL_CLASS

alias of ThreadPoolExecutor

DEFAULT_ALIGN_FACTOR = 4096

__init__(*thread_pool: Optional[ThreadPoolExecutor] = None*, *force_chunk_size: Optional[int] = None*, *min_chunk_size: Optional[int] = None*, *max_chunk_size: Optional[int] = None*, *align_factor: Optional[int] = None*, *check_hash: bool = True*, ***kwargs*)

is_suitable(*download_version: DownloadVersion*, *allow_seeking: bool*)

Analyze download_version (possibly against options passed earlier to constructor to find out whether the given download request should be handled by this downloader).

abstract download(*file: IOBase*, *response: Response*, *download_version: DownloadVersion*, *session: B2Session*, *encryption: Optional[EncryptionSetting] = None*)

@returns (bytes_read, actual_sha1)

b2sdk.transfer.inbound.downloader.parallel – ParallelTransferer

class b2sdk.transfer.inbound.downloader.parallel.**ParallelDownloader**(*min_part_size: int*, *max_streams: Optional[int] = None*, ***kwargs*)

Bases: *AbstractDownloader*

FINISH_HASHING_BUFFER_SIZE = 1048576

__init__(*min_part_size: int*, *max_streams: Optional[int] = None*, ***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(*download_version: DownloadVersion*, *allow_seeking: bool*)

Analyze download_version (possibly against options passed earlier to constructor to find out whether the given download request should be handled by this downloader).

download(*file: IOBase*, *response: Response*, *download_version: DownloadVersion*, *session:* B2Session, *encryption: Optional[EncryptionSetting] = None*)

Download a file from given url using parallel download sessions and stores it in the given download_destination.

class b2sdk.transfer.inbound.downloader.parallel.**WriterThread**(*file*, *max_queue_depth*)

Bases: 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.

b2sdk.transfer.inbound.downloader.parallel.**download_first_part**(*response: Response*, *hasher*, *session:* B2Session, *writer:* WriterThread, *first_part:* PartToDownload, *chunk_size: int*, *encryption:* *Optional[EncryptionSetting] = None*) → None

Parameters

- **response** – response of the original GET call
- **hasher** – hasher object to feed to as the stream is written
- **session** – B2 API session
- **writer** – thread responsible for writing downloaded data
- **first_part** – definition of the part to be downloaded
- **chunk_size** – size (in bytes) of read data chunks
- **encryption** – encryption mode, algorithm and key

b2sdk.transfer.inbound.downloader.parallel.**download_non_first_part**(*url: str*, *session:* B2Session, *writer:* WriterThread, *part_to_download:* PartToDownload, *chunk_size: int*, *encryption:* *Optional[EncryptionSetting] = None*) → None

Parameters

- **url** – download URL
- **session** – B2 API session
- **writer** – thread responsible for writing downloaded data
- **part_to_download** – definition of the part to be downloaded
- **chunk_size** – size (in bytes) of read data chunks
- **encryption** – encryption mode, algorithm and key

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*)

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.simple – SimpleDownloader

class b2sdk.transfer.inbound.downloader.simple.**SimpleDownloader**(*thread_pool: Optional[ThreadPoolExecutor] = None*, *force_chunk_size: Optional[int] = None*, *min_chunk_size: Optional[int] = None*, *max_chunk_size: Optional[int] = None*, *align_factor: Optional[int] = None*, *check_hash: bool = True*, ***kwargs*)

Bases: *AbstractDownloader*

REQUIRES_SEEKING = False

download(*file: IOBase*, *response: Response*, *download_version: DownloadVersion*, *session:* B2Session, *encryption: Optional[EncryptionSetting] = None*)

@returns (bytes_read, actual_sha1)

__init__(*thread_pool: Optional[ThreadPoolExecutor] = None*, *force_chunk_size: Optional[int] = None*, *min_chunk_size: Optional[int] = None*, *max_chunk_size: Optional[int] = None*, *align_factor: Optional[int] = None*, *check_hash: bool = True*, ***kwargs*)

b2sdk.transfer.inbound.download_manager – Manager of downloaders

class b2sdk.transfer.inbound.download_manager.**DownloadManager**(*write_buffer_size: Optional[int] = None*, *check_hash: bool = True*, ***kwargs*)

Bases: TransferManager, ThreadPoolMixin

Handle complex actions around downloads to free raw_api from that responsibility.

DEFAULT_MIN_PART_SIZE = 104857600

MIN_CHUNK_SIZE = 8192

MAX_CHUNK_SIZE = 1048576

PARALLEL_DOWNLOADER_CLASS

alias of *ParallelDownloader*

SIMPLE_DOWNLOADER_CLASS

alias of *SimpleDownloader*

__init__(*write_buffer_size: Optional[int] = None*, *check_hash: bool = True*, ***kwargs*)

Initialize the DownloadManager using the given services object.

download_file_from_url(*url*, *progress_listener=None*, *range_=None*, *encryption: Optional[EncryptionSetting] = None*) → DownloadedFile

Parameters

- **url** – url from which the file should be downloaded
- **progress_listener** – where to notify about downloading progress
- **range** – 2-element tuple containing data of http Range header
- **encryption** (*b2sdk.v2.EncryptionSetting*) – encryption setting (None if unknown)

b2sdk.transfer.outbound.upload_source

class b2sdk.transfer.outbound.upload_source.**AbstractUploadSource**

Bases: 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: *AbstractUploadSource*

__init__(*data_bytes*, *content_sha1=None*)

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: *AbstractUploadSource*

__init__(*local_path*, *content_sha1=None*)

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: *UploadSourceLocalFile*

__init__(*local_path*, *content_sha1=None*, *offset=0*, *length=None*)

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: *AbstractUploadSource*

__init__(*stream_opener*, *stream_length=None*, *stream_sha1=None*)

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: *UploadSourceStream*

__init__(*stream_opener*, *offset*, *stream_length*, *stream_sha1=None*)

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*)

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*)

as_list_parts_dict()

class b2sdk.raw_simulator.**FileSimulator**(*account_id*, *bucket*, *file_id*, *action*, *name*, *content_type*, *content_sha1*, *file_info*, *data_bytes*, *upload_timestamp*, *range_=None*, *server_side_encryption: Optional[EncryptionSetting] = None*, *file_retention: Optional[FileRetentionSetting] = None*, *legal_hold: LegalHold = LegalHold.UNSET*, *replication_status: Optional[ReplicationStatus] = None*)

Bases: object

One of three: an unfinished large file, a finished file, or a deletion marker.

CHECK_ENCRYPTION = True

SPECIAL_FILE_INFOS = {'b2-cache-control': 'Cache-Control', 'b2-content-disposition': 'Content-Disposition', 'b2-content-encoding': 'Content-Encoding', 'b2-content-language': 'Content-Language', 'b2-expires': 'Expires'}

__init__(*account_id*, *bucket*, *file_id*, *action*, *name*, *content_type*, *content_sha1*, *file_info*, *data_bytes*, *upload_timestamp*, *range_=None*, *server_side_encryption: Optional[EncryptionSetting] = None*, *file_retention: Optional[FileRetentionSetting] = None*, *legal_hold: LegalHold = LegalHold.UNSET*, *replication_status: Optional[ReplicationStatus] = None*)

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(*account_auth_token_or_none*, *range_=None*)

as_upload_result(*account_auth_token*)

as_list_files_dict(*account_auth_token*)

is_allowed_to_read_file_retention(*account_auth_token*)

is_allowed_to_read_file_legal_hold(*account_auth_token*)

as_start_large_file_result(*account_auth_token*)

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[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**(*account_auth_token_or_none*, *file_sim*, *url*, *range_=None*)

Bases: object

__init__(*account_auth_token_or_none*, *file_sim*, *url*, *range_=None*)

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*, *is_file_lock_enabled: Optional[bool] = None*, *replication: Optional[ReplicationConfiguration] = None*)

Bases: object

FIRST_FILE_NUMBER = 9999

FIRST_FILE_ID = '9999'

FILE_SIMULATOR_CLASS

alias of *FileSimulator*

RESPONSE_CLASS

alias of *FakeResponse*

MAX_SIMPLE_COPY_SIZE = 200

__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*, *is_file_lock_enabled: Optional[bool] = None*, *replication: Optional[ReplicationConfiguration] = None*)

is_allowed_to_read_bucket_encryption_setting(*account_auth_token*)

is_allowed_to_read_bucket_retention(*account_auth_token*)

bucket_dict(*account_auth_token*)

cancel_large_file(*file_id*)

delete_file_version(*file_id*, *file_name*)

download_file_by_id(*account_auth_token_or_none*, *file_id*, *url*, *range_=None*, *encryption: Optional[EncryptionSetting] = None*)

download_file_by_name(*account_auth_token_or_none*, *file_name*, *url*, *range_=None*, *encryption: Optional[EncryptionSetting] = None*)

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

get_file_info_by_id(*account_auth_token*, *file_id*)

get_file_info_by_name(*account_auth_token*, *file_name*)

get_upload_url(*account_auth_token*)

get_upload_part_url(*account_auth_token*, *file_id*)

hide_file(*account_auth_token*, *file_name*)

update_file_retention(*account_auth_token*, *file_id*, *file_name*, *file_retention: FileRetentionSetting*, *bypass_governance: bool = False*)

update_file_legal_hold(*account_auth_token*, *file_id*, *file_name*, *legal_hold: LegalHold*)

copy_file(*account_auth_token*, *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[EncryptionSetting] = None*, *source_server_side_encryption: Optional[EncryptionSetting] = None*, *file_retention: Optional[FileRetentionSetting] = None*, *legal_hold: Optional[LegalHold] = None*)

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

list_file_versions(*account_auth_token*, *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(*account_auth_token*, *start_file_id=None*, *max_file_count=None*, *prefix=None*)

start_large_file(*account_auth_token*, *file_name*, *content_type*, *file_info*, *server_side_encryption: Optional[EncryptionSetting] = None*, *file_retention: Optional[FileRetentionSetting] = None*, *legal_hold: Optional[LegalHold] = None*)

upload_file(*upload_id: str*, *upload_auth_token: str*, *file_name: str*, *content_length: int*, *content_type: str*, *content_sha1: str*, *file_infos: dict*, *data_stream*, *server_side_encryption: Optional[EncryptionSetting] = None*, *file_retention: Optional[FileRetentionSetting] = None*, *legal_hold: Optional[LegalHold] = None*)

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

class b2sdk.raw_simulator.**RawSimulator**(*b2_http=None*)

Bases: *AbstractRawApi*

Implement the same interface as B2RawHTTPApi 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 B2RawHTTPApi.

BUCKET_SIMULATOR_CLASS

alias of *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_PART_ID = 10000

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_download_file_by_id\\?fileId=(?P<file_id>[^/]+)|/file/(?P<bucket_name>[^/]+)/(?P<file_name>.+))$')

__init__(*b2_http=None*)

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[EncryptionSetting] = None*, *is_file_lock_enabled: Optional[bool] = None*, *replication: Optional[ReplicationConfiguration] = 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*)

update_file_retention(*api_url*, *account_auth_token*, *file_id*, *file_name*, *file_retention: FileRetentionSetting*, *bypass_governance: bool = False*)

update_file_legal_hold(*api_url*, *account_auth_token*, *file_id*, *file_name*, *legal_hold: bool*)

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[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*, *file_retention: Optional[FileRetentionSetting] = None*, *legal_hold: Optional[LegalHold] = None*)

copy_part(*api_url*, *account_auth_token*, *source_file_id*, *large_file_id*, *part_number*, *bytes_range=None*, *destination_server_side_encryption: Optional[EncryptionSetting] = None*, *source_server_side_encryption: Optional[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[EncryptionSetting] = None*, *file_retention: Optional[FileRetentionSetting] = None*, *legal_hold: Optional[LegalHold] = 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: Optional[EncryptionSetting] = None*, *default_retention: Optional[BucketRetentionSetting] = None*, *replication: Optional[ReplicationConfiguration] = None*)

classmethod get_upload_file_headers(*upload_auth_token: str*, *file_name: str*, *content_length: int*, *content_type: str*, *content_sha1: str*, *file_infos: dict*, *server_side_encryption: Optional[EncryptionSetting]*, *file_retention: Optional[FileRetentionSetting]*, *legal_hold: Optional[LegalHold]*) → dict

upload_file(*upload_url: str*, *upload_auth_token: str*, *file_name: str*, *content_length: int*, *content_type: str*, *content_sha1: str*, *file_infos: dict*, *data_stream*, *server_side_encryption: Optional[EncryptionSetting] = None*, *file_retention: Optional[FileRetentionSetting] = None*, *legal_hold: Optional[LegalHold] = None*)

upload_part(*upload_url*, *upload_auth_token*, *part_number*, *content_length*, *sha1_sum*, *input_stream*, *server_side_encryption: Optional[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-3.7`, `test-3.8`, `test-3.9`, `test-3.10`) -> 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 python3.10 in .nox/format
...

$ nox -s format
nox > Running session format
nox > Re-using existing virtual environment at .nox/format.
...

$ nox --no-venv -s format
nox > Running session format
...
```

Sessions `test`, `unit`, and `integration` can run on many Python versions, 3.7-3.10 by default.

Sessions other than `test` use the last given Python version, 3.10 by default.

You can change it:

```
export NOX_PYTHONS=3.7,3.8
```

With the above setting, session `test` will run on Python 3.7 and 3.8, and all other sessions on Python 3.8.

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-3.10
```

To run just unit tests:

```
$ nox -s unit-3.10
```

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-3.10
```

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

K

L

M

N

O

P

R

S

T

U

V

W