Core API for Python Documentation

The Core API is the underlying interface for all of our official Dropbox mobile apps and our SDKs. This document describes the Python interface to the Core API. For information about the underlying HTTP endpoints, please visit the Core API HTTP documentation.

DropboxClient

This class lets you make Dropbox API calls. You'll need to obtain an OAuth 2 access token first. You can get an access token using either DropboxOAuth2Flow or DropboxOAuth2FlowNoRedirect.

All of the API call methods can raise a dropbox.rest.ErrorResponse exception if the server returns a non-200 or invalid HTTP response. Note that a 401 return status at any point indicates that the access token you're using is no longer valid and the user must be put through the OAuth 2 authorization flow again.

Constructors

DropboxClient(*oauth2_access_token*, *locale*=None, *rest_client*=None)

Construct a `DropboxClient` instance.

Parameters

oauth2_access_token
An OAuth 2 access token (string).

locale
The locale of the user of your application. For example "en" or "en_US". Some API calls return localized data and error messages; this setting tells the server which locale to use. By default, the server uses "en_US".

rest_client
Optional dropbox.rest.RESTClient-like object to use for making requests.

Instance Methods

request(*target*, *params*=None, *method*='POST', *content_server*=False)

An internal method that builds the url, headers, and params for a Dropbox API request. It is exposed if you need to make API calls not implemented in this library or if you need to debug requests.

Parameters

target
The target URL with leading slash (e.g. '/files').

params
A dictionary of parameters to add to the request.

method
An HTTP method (e.g. 'GET' or 'POST').

content_server
A boolean indicating whether the request is to the API content server, for example to fetch the contents of a file rather than its metadata.

Returns

A tuple of `(url, params, headers)` that should be used to make the request. OAuth will be added as needed within these fields.

`account_info()`

Retrieve information about the user's account.

Returns

A dictionary containing account information.
For a detailed description of what this call returns, visit: https://www.dropbox.com/developers/core/docs#account-info

---

`disable_access_token()`

Disable the access token that this `DropboxClient` is using. If this call succeeds, further API calls using this object will fail.

`create_oauth2_access_token()`

If this `DropboxClient` was created with an OAuth 1 access token, this method can be used to create an equivalent OAuth 2 access token. This can be used to upgrade your app's existing access tokens from OAuth 1 to OAuth 2.

---

get_chunked_uploader(*file_obj*, *length*)

Creates a ChunkedUploader to upload the given file-like object.

Parameters

file_obj
The file-like object which is the source of the data being uploaded.

length
The number of bytes to upload.

The expected use of this function is as follows:

```
bigFile = open("data.txt", 'rb')

uploader = myclient.get_chunked_uploader(bigFile, size)
print "uploading: ", size
while uploader.offset < size:
    try:
        upload = uploader.upload_chunked()
    except rest.ErrorResponse, e:
        # perform error handling and retry logic
uploader.finish('/bigFile.txt')
```

The SDK leaves the error handling and retry logic to the developer to implement, as the exact requirements will depend on the application involved.

upload_chunk(*file_obj*, *length*, *offset*=0, *upload_id*=None)

Uploads a single chunk of data from the given file like object. The majority of users should use the ChunkedUploader object, which provides a simpler interface to the chunked_upload API endpoint.

Parameters

file_obj
The source of the data to upload.

length
The number of bytes to upload in one chunk.

Returns

The reply from the server, as a dictionary.

put_file(*full_path*, *file_obj*, *overwrite*=False, *parent_rev*=None)

Upload a file.

A typical use case would be as follows:

```
f = open('working-draft.txt', 'rb')
response = client.put_file('/magnum-opus.txt', f)
print "uploaded:", response
```

which would return the metadata of the uploaded file, similar to:

```
{
    'bytes': 77,
    'icon': 'page_white_text',
    'is_dir': False,
    'mime_type': 'text/plain',
    'modified': 'Wed, 20 Jul 2011 22:04:50 +0000',
    'path': '/magnum-opus.txt',
    'rev': '362e2029684fe',
    'revision': 221922,
    'root': 'dropbox',
    'size': '77 bytes',
    'thumb_exists': False
}
```

Parameters

full_path
The full path to upload the file to, *including the file name*. If the destination directory does not yet exist, it will be created.

file_obj
A file-like object to upload. If you would like, you can pass a string as file_obj.

overwrite
Whether to overwrite an existing file at the given path. (Default `False`.) If overwrite is False and a file already exists there, Dropbox will rename the upload to make sure it doesn't overwrite anything. You need to check the metadata returned for the new name. This field should only be True if your intent is to potentially clobber changes to a file that you don't know about.

parent_rev
Optional rev field from the 'parent' of this upload. If your intent is to update the file at the given path, you should pass the parent_rev parameter set to the rev value from the most recent metadata you have of the existing file at that path. If the server has a more recent version of the file at the specified

path, it will automatically rename your uploaded file, spinning off a conflict. Using this parameter effectively causes the overwrite parameter to be ignored. The file will always be overwritten if you send the most-recent parent_rev, and it will never be overwritten if you send a less-recent one.

Returns

A dictionary containing the metadata of the newly uploaded file.
For a detailed description of what this call returns, visit: https://www.dropbox.com/developers/core/docs#files-put

Raises

A dropbox.rest.ErrorResponse with an HTTP status of:

400: Bad request (may be due to many things; check e.error for details).

503: User over quota.

`get_file(from_path, rev=None)`

Download a file.

Example:

```
out = open('magnum-opus.txt', 'wb')
with client.get_file('/magnum-opus.txt') as f:
    out.write(f.read())
```

which would download the file `magnum-opus.txt` and write the contents into the file `magnum-opus.txt` on the local filesystem.

Parameters

from_path
The path to the file to be downloaded.

rev
Optional previous rev value of the file to be downloaded.

Returns

A dropbox.rest.RESTResponse that is the HTTP response for the API request. It is a file-like object that can be read from. You must call `close()` when you're done.

Raises

A dropbox.rest.ErrorResponse with an HTTP status of:

400: Bad request (may be due to many things; check e.error for details).

404: No file was found at the given path, or the file that was there was deleted.

200: Request was okay but response was malformed in some way.

`get_file_and_metadata(`*`from_path, rev`*`=None)`

Download a file alongwith its metadata.

Acts as a thin wrapper around get_file() (see get_file() comments for more details)

A typical usage looks like this:

```
out = open('magnum-opus.txt', 'wb')
f, metadata = client.get_file_and_metadata('/magnum-opus.txt')
with f:
    out.write(f.read())
```

Parameters

from_path
The path to the file to be downloaded.

rev
Optional previous rev value of the file to be downloaded.

Returns

A pair of `(response, metadata)`:

response
A dropbox.rest.RESTResponse that is the HTTP response for the API request. It is a file-like object that can be read from. You must call `close()` when you're done.

metadata
A dictionary containing the metadata of the file (see https://www.dropbox.com/developers/core/docs#metadata for details).

Raises

A dropbox.rest.ErrorResponse with an HTTP status of:

400: Bad request (may be due to many things; check e.error for details).

404: No file was found at the given path, or the file that was there was deleted.

200: Request was okay but response was malformed in some way.

`delta(`*`cursor`*`=None, `*`path_prefix`*`=None)`

A way of letting you keep up with changes to files and folders in a user's Dropbox. You can periodically call delta() to get a list of "delta entries", which are instructions on how to update your local state to match the server's state.

Parameters

cursor
On the first call, omit this argument (or pass in `None`). On subsequent calls, pass in the `cursor` string returned by the previous call.

path_prefix
If provided, results will be limited to files and folders whose paths are equal to or under `path_prefix`. The `path_prefix` is fixed for a given cursor. Whatever `path_prefix` you use on the first `delta()` must also be passed in on subsequent calls that use the returned cursor.

Returns

A dict with four keys:

entries
A list of "delta entries" (described below).

reset
If `True`, you should your local state to be an empty folder before processing the list of delta entries. This is only `True` only in rare situations.

cursor
A string that is used to keep track of your current state. On the next call to delta(), pass in this value to return entries that were recorded since the cursor was returned.

has_more
If `True`, then there are more entries available; you can call delta() again immediately to retrieve those entries. If `False`, then wait at least 5 minutes (preferably longer) before checking again.

Delta Entries: Each entry is a 2-item list of one of following forms:

[*path*, *metadata*]: Indicates that there is a file/folder at the given path. You should add the entry to your local path. (The *metadata* value is the same as what would be returned by the `metadata()` call.)

If the new entry includes parent folders that don't yet exist in your local state, create those parent folders in your local state. You will eventually get entries for those parent folders.

If the new entry is a file, replace whatever your local state has at *path* with the new entry.

If the new entry is a folder, check what your local state has at *path*. If it's a file, replace it with the new entry. If it's a folder, apply the new *metadata* to the folder, but do not modify the folder's children.

[*path*, `None`]: Indicates that there is no file/folder at the *path* on Dropbox. To update your local state to match, delete whatever is at *path*, including any children (you will sometimes also get "delete" delta entries for the children, but this is not guaranteed). If your local state doesn't have anything at *path*, ignore this entry.

Remember: Dropbox treats file names in a case-insensitive but case-preserving way. To facilitate this, the *path* strings above are lower-cased versions of the actual path. The *metadata* dicts have the original, case-preserved path.

`create_copy_ref(`*`from_path`*`)`

Creates and returns a copy ref for a specific file. The copy ref can be used to instantly copy that file to the Dropbox of another account.

Paramet
ers

path
The path to the file for a copy ref to be created on.

Returns

A dictionary that looks like the following example:

```
{"expires": "Fri, 31 Jan 2042 21:01:05 +0000", "copy_
ref": "z1X6ATl6aWtzOGq0c3g5Ng"}
```

`add_copy_ref(`*`copy_ref, to_path`*`)`

Adds the file referenced by the copy ref to the specified path

Paramet
ers

**copy_
ref**

A copy ref string that was returned from a create_copy_ref call. The copy_ref can be created from any other Dropbox account, or from the same account.

path
The path to where the file will be created.

Returns

A dictionary containing the metadata of the new copy of the file.

`file_copy(`*`from_path, to_path`*`)`

Copy a file or folder to a new location.

Parameters

from_path
The path to the file or folder to be copied.

to_path
The destination path of the file or folder to be copied. This parameter should include the destination filename (e.g. from_path: '/test.txt', to_path: '/dir/test.txt'). If there's already a file at the to_path it will raise an ErrorResponse.

Returns

A dictionary containing the metadata of the new copy of the file or folder.
For a detailed description of what this call returns, visit: https://www.dropbox.com/developers/core/docs#fileops-copy

Raises

A dropbox.rest.ErrorResponse with an HTTP status of:

400: Bad request (may be due to many things; check e.error for details).

403: An invalid move operation was attempted (e.g. there is already a file at the given destination, or moving a shared folder into a shared folder).

404: No file was found at given from_path.

503: User over storage quota.

`file_create_folder(`*`path`*`)`

Create a folder.

Parameters

path

The path of the new folder.

Returns

A dictionary containing the metadata of the newly created folder.
For a detailed description of what this call returns, visit: https://www.dropbox.com/developers/core/docs#fileops-create-folder

Raises

A dropbox.rest.ErrorResponse with an HTTP status of:

400: Bad request (may be due to many things; check e.error for details).

403: A folder at that path already exists.

`file_delete(`*`path`*`)`

Delete a file or folder.

Parameters

path
The path of the file or folder.

Returns

A dictionary containing the metadata of the just deleted file.
For a detailed description of what this call returns, visit: https://www.dropbox.com/developers/core/docs#fileops-delete

Raises

A dropbox.rest.ErrorResponse with an HTTP status of:

400: Bad request (may be due to many things; check e.error for details).

404: No file was found at the given path.

`file_move(`*`from_path`*`, `*`to_path`*`)`

Move a file or folder to a new location.

Parameters

from_path
The path to the file or folder to be moved.

to_path
The destination path of the file or folder to be moved. This parameter should include the destination filename (e.g. if `from_path` is `'/test.txt'`,

`to_path` might be `'/dir/test.txt'`). If there's already a file at the `to_path`, this file or folder will be renamed to be unique.

Returns

A dictionary containing the metadata of the new copy of the file or folder. For a detailed description of what this call returns, visit: https://www.dropbox.com/developers/core/docs#fileops-move

Raises

A dropbox.rest.ErrorResponse with an HTTP status of:

400: Bad request (may be due to many things; check e.error for details).

404: No file was found at given from_path.

503: User over storage quota.

---

metadata(*path*, *list*=True, *file_limit*=25000, *hash*=None, *rev*=None, *include_deleted*=False)

Retrieve metadata for a file or folder.

A typical use would be:

```
folder_metadata = client.metadata('/')
print "metadata:", folder_metadata
```

which would return the metadata of the root directory. This will look something like:

```
{
    'bytes': 0,
    'contents': [
        {
           'bytes': 0,
           'icon': 'folder',
           'is_dir': True,
           'modified': 'Thu, 25 Aug 2011 00:03:15 +0000',
           'path': '/Sample Folder',
           'rev': '803beb471',
           'revision': 8,
           'root': 'dropbox',
           'size': '0 bytes',
           'thumb_exists': False
        },
        {
           'bytes': 77,
           'icon': 'page_white_text',
           'is_dir': False,
           'mime_type': 'text/plain',
           'modified': 'Wed, 20 Jul 2011 22:04:50 +0000',
           'path': '/magnum-opus.txt',
                'rev': '362e2029684fe',
                'revision': 221922,
                'root': 'dropbox',
                'size': '77 bytes',
                'thumb_exists': False
            }
        ],
        'hash': 'efdac89c4da886a9cece1927e6c22977',
        'icon': 'folder',
        'is_dir': True,
        'path': '/',
        'root': 'app_folder',
        'size': '0 bytes',
        'thumb_exists': False
    }
```

In this example, the root directory contains two things: `Sample Folder`, which is a folder, and `/magnum-opus.txt`, which is a text file 77 bytes long

Paramet
ers

path
The path to the file or folder.

list
Whether to list all contained files (only applies when path refers to a folder).

file_
limit
The maximum number of file entries to return within a folder. If the number of files in the directory exceeds this limit, an exception is raised. The server will return at max 25,000 files within a folder.

hash
Every directory listing has a hash parameter attached that can then be passed back into this function later to save on bandwidth. Rather than returning an unchanged folder's contents, the server will instead return a 304.

rev
Optional revision of the file to retrieve the metadata for. This parameter only applies for files. If omitted, you'll receive the most recent revision metadata.

includ
e_
delete
d
When listing contained files, include files that have been deleted.

Returns

A dictionary containing the metadata of the file or folder (and contained files if appropriate).
For a detailed description of what this call returns, visit: https://www.dropbox.com/developers/core/docs#metadata

Raises

A dropbox.rest.ErrorResponse with an HTTP status of:

304: Current directory hash matches hash parameters, so contents are unchanged.

400: Bad request (may be due to many things; check e.error for details).

404: No file was found at given path.

406: Too many file entries to return.

`thumbnail(`*`from_path`*`, `*`size`*`='m', `*`format`*`='JPEG')`

Download a thumbnail for an image.

Parameters

from_path
The path to the file to be thumbnailed.

size
A string specifying the desired thumbnail size. Currently supported sizes: `"xs"` (32x32), `"s"` (64x64), `"m"` (128x128), `"l"` (640x480), `"xl"` (1024x768). Check https://www.dropbox.com/developers/core/docs#thumbnails for more details.

format
The image format the server should use for the returned thumbnail data. Either `"JPEG"` or `"PNG"`.

Returns

A dropbox.rest.RESTResponse that is the HTTP response for the API request. It is a file-like object that can be read from. You must call `close()` when you're done.

Raises

A dropbox.rest.ErrorResponse with an HTTP status of:

400: Bad request (may be due to many things; check e.error for details).

404: No file was found at the given from_path, or files of that type cannot be thumbnailed.

415: Image is invalid and cannot be thumbnailed.

`thumbnail_and_metadata(`*`from_path`*`, `*`size`*`='m', `*`format`*`='JPEG')`

Download a thumbnail for an image alongwith its metadata.

Acts as a thin wrapper around thumbnail() (see thumbnail() comments for more details)

Parameters

from_path
The path to the file to be thumbnailed.

size
A string specifying the desired thumbnail size. See thumbnail() for details.

format
The image format the server should use for the returned thumbnail data. Either `"JPEG"` or `"PNG"`.

Returns

A pair of `(response, metadata)`:

response
A dropbox.rest.RESTResponse that is the HTTP response for the API request. It is a file-like object that can be read from. You must call `close()` when you're done.

metadata
A dictionary containing the metadata of the file whose thumbnail was downloaded (see https://www.dropbox.com/developers/core/docs#metadata for details).

Raises

A dropbox.rest.ErrorResponse with an HTTP status of:

400: Bad request (may be due to many things; check e.error for details).

404: No file was found at the given from_path, or files of that type cannot be thumbnailed.

415: Image is invalid and cannot be thumbnailed.

200: Request was okay but response was malformed in some way.

`search(`*`path`*`, `*`query`*`, `*`file_limit`*`=1000, `*`include_deleted`*`=False)`

Search directory for filenames matching query.

Parameters

path
The directory to search within.

query
The query to search on (minimum 3 characters).

file_limit
The maximum number of file entries to return within a folder. The server will return at max 1,000 files.

include_deleted
Whether to include deleted files in search results.

Returns

A list of the metadata of all matching files (up to file_limit entries). For a detailed description of what this call returns, visit: https://www.dropbox.com/developers/core/docs#search

Raises

A dropbox.rest.ErrorResponse with an HTTP status of:

400: Bad request (may be due to many things; check e.error for details).

---

revisions(*path*, *rev_limit*=1000)

Retrieve revisions of a file.

Parameters

path
The file to fetch revisions for. Note that revisions are not available for folders.

rev_limit
The maximum number of file entries to return within a folder. The server will return at max 1,000 revisions.

Returns

A list of the metadata of all matching files (up to rev_limit entries). For a detailed description of what this call returns, visit: https://www.dropbox.com/developers/core/docs#revisions

Raises

A dropbox.rest.ErrorResponse with an HTTP status of:

400: Bad request (may be due to many things; check e.error for details).

404: No revisions were found at the given path.

`restore(path, rev)`

Restore a file to a previous revision.

Parameters

path
The file to restore. Note that folders can't be restored.

rev
A previous rev value of the file to be restored to.

Returns

A dictionary containing the metadata of the newly restored file. For a detailed description of what this call returns, visit: https://www.dropbox.com/developers/core/docs#restore

Raises

A dropbox.rest.ErrorResponse with an HTTP status of:

400: Bad request (may be due to many things; check e.error for details).

404: Unable to find the file at the given revision.

`media(path)`

Get a temporary unauthenticated URL for a media file.

All of Dropbox's API methods require OAuth, which may cause problems in situations where an application expects to be able to hit a URL multiple times (for example, a media player seeking around a video file). This method creates a time-limited URL that can be accessed without any authentication, and returns that to you, along with an expiration time.

Parameters

path
The file to return a URL for. Folders are not supported.

Returns

A dictionary that looks like the following example:

```
{'url': 'https://dl.dropboxusercontent.com/1/view/
abcdefghijk/example',
 'expires': 'Thu, 16 Sep 2011 01:01:25 +0000'}
```

For a detailed description of what this call returns, visit: https://www.dropbox.com/developers/core/docs#media

Raises

A dropbox.rest.ErrorResponse with an HTTP status of:

400: Bad request (may be due to many things; check e.error for details).

404: Unable to find the file at the given path.

---

share(*path*, *short_url*=True)

Create a shareable link to a file or folder.

Shareable links created on Dropbox are time-limited, but don't require any authentication, so they can be given out freely. The time limit should allow at least a day of shareability, though users have the ability to disable a link from their account if they like.

Parameters

path
The file or folder to share.

Returns

A dictionary that looks like the following example:

```
{'url': u'https://db.tt/c0mFuu1Y', 'expires': 'Tue, 01
Jan 2030 00:00:00 +0000'}
```

For a detailed description of what this call returns, visit: https://www.dropbox.com/developers/core/docs#shares

Raises

A dropbox.rest.ErrorResponse with an HTTP status of:

400: Bad request (may be due to many things; check e.error for details).

404: Unable to find the file at the given path.

ChunkedUploader

Contains the logic around a chunked upload, which uploads a large file to Dropbox via the /chunked_upload endpoint.

Instance Methods

`upload_chunked(`*`chunk_size`*`=4194304)`

Uploads data from this ChunkedUploader's file_obj in chunks, until an error occurs. Throws an exception when an error occurs, and can be called again to resume the upload.

Paramet
ers

**chunk
_size**
The number of bytes to put in each chunk. (Default 4 MB.)

`finish(`*`path`*`, `*`overwrite`*`=False, `*`parent_rev`*`=None)`

Commits the bytes uploaded by this ChunkedUploader to a file in the users dropbox.

Paramet
ers

path
The full path of the file in the Dropbox.

**overw
rite**
Whether to overwrite an existing file at the given path. (Default `False`.) If overwrite is False and a file already exists there, Dropbox will rename the upload to make sure it doesn't overwrite anything. You need to check the metadata returned for the new name. This field should only be True if your intent is to potentially clobber changes to a file that you don't know about.

**paren
t_rev**
Optional rev field from the 'parent' of this upload. If your intent is to update the file at the given path, you should pass the parent_rev parameter set to the rev value from the most recent metadata you have of the existing file at that path. If the server has a more recent version of the file at the specified path, it will automatically rename your uploaded file, spinning off a conflict. Using this parameter effectively causes the overwrite parameter to be ignored. The file will always be overwritten if you send the most-recent parent_rev, and it will never be overwritten if you send a less-recent one.

DropboxOAuth2Flow

OAuth 2 authorization helper. Use this for web apps.

OAuth 2 has a two-step authorization process. The first step is having the user authorize your app. The second involves getting an OAuth 2 access token from Dropbox.

Example:

```
from dropbox.client import DropboxOAuth2Flow, DropboxClient

def get_dropbox_auth_flow(web_app_session):
    redirect_uri = "https://my-web-server.org/dropbox-auth-finish")
    return DropboxOAuth2Flow(APP_KEY, APP_SECRET, redirect_uri,
                             web_app_session, "dropbox-auth-csrf-token")

# URL handler for /dropbox-auth-start
def dropbox_auth_start(web_app_session, request):
    authorize_url = get_dropbox_auth_flow(web_app_session).start()
    redirect_to(authorize_url)

# URL handler for /dropbox-auth-finish
def dropbox_auth_finish(web_app_session, request):
    try:
        access_token, user_id, url_state = \
                get_dropbox_auth_flow(web_app_session).finish(request.query_
params)
    except DropboxOAuth2Flow.BadRequestException, e:
        http_status(400)
    except DropboxOAuth2Flow.BadStateException, e:
        # Start the auth flow again.
        redirect_to("/dropbox-auth-start")
    except DropboxOAuth2Flow.CsrfException, e:
        http_status(403)
    except DropboxOAuth2Flow.NotApprovedException, e:
        flash('Not approved?  Why not?')
        return redirect_to("/home")
    except DropboxOAuth2Flow.ProviderException, e:
        logger.log("Auth error: %s" % (e,))
        http_status(403)
```

Constructors

DropboxOAuth2Flow(*consumer_key*, *consumer_secret*, *redirect_uri*, *session*, *csrf_token_session_key*, *locale*=None, *rest_client*=None)

Construct an instance.

Paramet
ers

**consu
mer_
key**
Your API app's "app key".

**consu
mer_
secret**
Your API app's "app secret".

**redire
ct_uri**

The URI that the Dropbox server will redirect the user to after the user finishes authorizing your app. This URI must be HTTPS-based and pre-registered with the Dropbox servers, though localhost URIs are allowed without pre-registration and can be either HTTP or HTTPS.

session
A dict-like object that represents the current user's web session (will be used to save the CSRF token).

csrf_token_session_key
The key to use when storing the CSRF token in the session (for example: "dropbox-auth-csrf-token").

locale
The locale of the user of your application. For example "en" or "en_US". Some API calls return localized data and error messages; this setting tells the server which locale to use. By default, the server uses "en_US".

rest_client
Optional dropbox.rest.RESTClient-like object to use for making requests.

Instance Methods

`start(url_state=None)`

Starts the OAuth 2 authorization process.

This function builds an "authorization URL". You should redirect your user's browser to this URL, which will give them an opportunity to grant your app access to their Dropbox account. When the user completes this process, they will be automatically redirected to the `redirect_uri` you passed in to the constructor.

This function will also save a CSRF token to `session[csrf_token_session_key]` (as provided to the constructor). This CSRF token will be checked on finish() to prevent request forgery.

Parameters

url_state
Any data that you would like to keep in the URL through the authorization process. This exact value will be returned to you by finish().

Returns

The URL for a page on Dropbox's website. This page will let the user "approve" your app, which gives your app permission to access the user's Dropbox account. Tell the user to visit this URL and approve your app.

`finish(`*`query_params`*`)`

Call this after the user has visited the authorize URL (see start()), approved your app and was redirected to your redirect URI.

Parameters

query_params
The query parameters on the GET request to your redirect URI.

Returns

A tuple of `(access_token, user_id, url_state)`. `access_token` can be used to construct a DropboxClient. `user_id` is the Dropbox user ID (string) of the user that just approved your app. `url_state` is the value you originally passed in to start().

Raises

BadRequestException
If the redirect URL was missing parameters or if the given parameters were not valid.

BadStateException
If there's no CSRF token in the session.

CsrfException
If the `'state'` query parameter doesn't contain the CSRF token from the user's session.

NotApprovedException
If the user chose not to approve your app.

ProviderException

If Dropbox redirected to your redirect URI with some unexpected error identifier and error message.

DropboxOAuth2FlowNoRedirect

OAuth 2 authorization helper for apps that can't provide a redirect URI (such as the command-line example apps).

Example:

```
from dropbox.client import DropboxOAuth2FlowNoRedirect, DropboxClient
from dropbox import rest as dbrest

auth_flow = DropboxOAuth2FlowNoRedirect(APP_KEY, APP_SECRET)

authorize_url = auth_flow.start()
print "1. Go to: " + authorize_url
print "2. Click \"Allow\" (you might have to log in first)."
print "3. Copy the authorization code."
auth_code = raw_input("Enter the authorization code here: ").strip()

try:
    access_token, user_id = auth_flow.finish(auth_code)
except dbrest.ErrorResponse, e:
    print('Error: %s' % (e,))
    return

c = DropboxClient(access_token)
```

Constructors

DropboxOAuth2FlowNoRedirect(*consumer_key*, *consumer_secret*, *locale*=None, *rest_client*=None)

Construct an instance.

Parameters

consumer_key
Your API app's "app key"

consumer_secret
Your API app's "app secret"

locale

The locale of the user of your application. For example "en" or "en_US". Some API calls return localized data and error messages; this setting tells the server which locale to use. By default, the server uses "en_US".

rest_ client
Optional dropbox.rest.RESTClient-like object to use for making requests.

Instance Methods

`start()`

Starts the OAuth 2 authorization process.

Returns

The URL for a page on Dropbox's website. This page will let the user "approve" your app, which gives your app permission to access the user's Dropbox account. Tell the user to visit this URL and approve your app.

`finish(`*`code`*`)`

If the user approves your app, they will be presented with an "authorization code". Have the user copy/paste that authorization code into your app and then call this method to get an access token.

Paramet ers

code
The authorization code shown to the user when they approved your app.

Returns

A pair of `(access_token, user_id)`. `access_token` is a string that can be passed to DropboxClient. `user_id` is the Dropbox user ID (string) of the user that just approved your app.

Raises

The same exceptions as DropboxOAuth2Flow.finish().

RESTClient

A class with all static methods to perform JSON REST requests that is used internally by the Dropbox Client API. It provides just enough gear to make requests and get responses as JSON data (when applicable). All requests happen over SSL.

Class Methods

`request(*n, **kw)`

Perform a REST request and parse the response.

Parameters

method
An HTTP method (e.g. `'GET'` or `'POST'`).

url
The URL to make a request to.

post_params
A dictionary of parameters to put in the body of the request. This option may not be used if the body parameter is given.

body
The body of the request. Typically, this value will be a string. It may also be a file-like object. The body parameter may not be used with the post_params parameter.

headers
A dictionary of headers to send with the request.

raw_response
Whether to return a RESTResponse object. Default `False`. It's best enabled for requests that return large amounts of data that you would want to `.read()` incrementally rather than loading into memory. Also use this for calls where you need to read metadata like status or headers, or if the body is not JSON.

Returns

The JSON-decoded data from the server, unless `raw_response` is set, in which case a RESTResponse object is returned instead.

Raises

ErrorResponse
The returned HTTP status is not 200, or the body was not parsed from JSON successfully.

RESTSocketError
A `socket.error` was raised while contacting Dropbox.

`GET(*n, **kw)`

Perform a GET request using RESTClient.request().

`POST(*n, **kw)`

Perform a POST request using RESTClient.request().

`PUT(*n, **kw)`

Perform a PUT request using RESTClient.request().

RESTResponse

Responses to requests can come in the form of `RESTResponse`. These are thin wrappers around the socket file descriptor. read() and close() are implemented. It is important to call close() to return the connection back to the connection pool to be reused. If a connection is not closed by the caller it may leak memory. The object makes a best-effort attempt upon destruction to call close(), but it's still best to explicitly call close().

Instance Methods

`read(`*`amt`*`=None)`

Read data off the underlying socket.

Parameters

`amt`
Amount of data to read. Defaults to `None`, indicating to read everything.

Returns

Data off the socket. If `amt` is not `None`, at most `amt` bytes are returned. An empty string when the socket has no data.

Raises

`ValueError`
If the `RESTResponse` has already been closed.

`close()`

Closes the underlying socket.

`getheaders()`

Returns a dictionary of the response headers.

`getheader(`*`name`*`, `*`default`*`=None)`

Returns a given response header.

ErrorResponse

Exception raised when DropboxClient exeriences a problem.

For example, this is raised when the server returns an unexpected non-200 HTTP response.

Properties

`body`

HTTP response body (string or JSON dict).

`error_msg`

Error message for developer (optional).

`headers`

HTTP response headers (a list of (header, value) tuples).

`reason`

HTTP response reason (a string).

`status`

HTTP response status (an int).

`user_error_msg`

Error message for end user (optional).

RESTSocketError

A light wrapper for `socket.error` that adds some more information.