globus-sdk-python

Release 1.11.0

Globus Team

Jan 29, 2021

GETTING STARTED

This SDK provides a convenient Pythonic interface to Globus web APIs, including the Transfer API and the Globus Auth API. Documentation for the APIs is available at https://docs.globus.org/api/.

Two interfaces are provided - a low level interface, supporting only `GET`, `PUT`, `POST`, and `DELETE` operations, and a high level interface providing helper methods for common API resources.

Additionally, some tools for interacting with local endpoint definitions are provided.

Source code is available at https://github.com/globus/globus-sdk-python.

CHAPTER

ONE

TABLE OF CONTENTS

1.1 Installation

The Globus SDK requires Python 2.7+ or 3.4+. If a supported version of Python is not already installed on your system, see this Python installation guide .

The simplest way to install the Globus SDK is using the `pip` package manager (https://pypi.python.org/pypi/pip), which is included in most Python installations:

```
pip install globus-sdk
```

This will install the Globus SDK and it's dependencies.

Bleeding edge versions of the Globus SDK can be installed by checking out the git repository and installing it manually:

```
git clone https://github.com/globus/globus-sdk-python.git
cd globus-sdk-python
python setup.py install
```

1.2 Tutorial

1.2.1 First Steps

This is a tutorial in the use of the Globus SDK. It takes you through a simple step-by-step flow for registering your application, getting tokens, and using them with our service.

These are the steps we will take:

1. *Get a Client*
2. *Get and Save Client ID*
3. *Get Some Access Tokens!*
4. *Use Your Tokens, Talk to the Service*

That should be enough to get you up and started. You can also proceed to the *Advanced Tutorial* steps to dig deeper into the SDK.

Step 1: Get a Client

In order to complete an OAuth2 flow to get tokens, you must have a client or "app" definition registered with Globus.

Navigate to the Developer Site and select "Register your app with Globus." You will be prompted to login – do so with the account you wish to use as your app's administrator.

When prompted, create a Project named "SDK Tutorial Project". Projects let you share the administrative burden of a collection of apps, but we won't be sharing the SDK Tutorial Project.

In the "Add. . . " menu for "SDK Tutorial Project", select "Add new app".

Enter the following pieces of information:

- **App Name**: "SDK Tutorial App"
- **Scopes**: "openid", "profile", "email", "urn:globus:auth:scope:transfer.api.globus.org:all"
- **Redirects**: https://auth.globus.org/v2/web/auth-code
- **Required Identity Provider**: <Leave Unchecked>
- **Privacy Policy**: <Leave Blank>
- **Terms & Conditions**: <Leave Blank>
- **Native App**: Check this Box

and click "Create App".

Step 2: Get and Save Client ID

On the "Apps" screen you should now see all of your Projects, probably just "SDK Tutorial Project", and all of the Apps they contain, probably just "SDK Tutorial App". Expand the dropdown for the tutorial App, and you should see an array of attributes of your client, including the ones we specified in Step 1, and a bunch of new things.

We want to get the Client ID from this screen. Feel free to think of this as your App's "username". You can hardcode it into scripts, store it in a config file, or even put it into a database. It's non-secure information and you can treat it as such.

In the rest of the tutorial we will assume in all code samples that it is available in the variable, `CLIENT_ID`.

Step 3: Get Some Access Tokens!

Talking to Globus Services as a user requires that you authenticate to your new App and get it Tokens, credentials proving that you logged into it and gave it permission to access the service.

No need to worry about creating your own login pages and such – for this type of app, Globus provides all of that for you. Run the following code sample to get your Access Tokens:

```
import globus_sdk

CLIENT_ID = '<YOUR_ID_HERE>'

client = globus_sdk.NativeAppAuthClient(CLIENT_ID)
client.oauth2_start_flow()

authorize_url = client.oauth2_get_authorize_url()
print('Please go to this URL and login: {0}'.format(authorize_url))

```

(continues on next page)

(continued from previous page)

```
# this is to work on Python2 and Python3 -- you can just use raw_input() or
# input() for your specific version
get_input = getattr(__builtins__, 'raw_input', input)
auth_code = get_input(
    'Please enter the code you get after login here: ').strip()
token_response = client.oauth2_exchange_code_for_tokens(auth_code)

globus_auth_data = token_response.by_resource_server['auth.globus.org']
globus_transfer_data = token_response.by_resource_server['transfer.api.globus.org']

# most specifically, you want these tokens as strings
AUTH_TOKEN = globus_auth_data['access_token']
TRANSFER_TOKEN = globus_transfer_data['access_token']
```

Managing credentials is one of the more advanced features of the SDK. If you want to read in depth about these steps, please look through our various *Examples*.

Step 4: Use Your Tokens, Talk to the Service

Continuing from the example above, you have two credentials to Globus Services on hand: the `AUTH_TOKEN` and the `TRANSFER_TOKEN`. We'll focus on the `TRANSFER_TOKEN` for now. It's how you authorize access to the Globus Transfer service.

```
# a GlobusAuthorizer is an auxiliary object we use to wrap the token. In
# more advanced scenarios, other types of GlobusAuthorizers give us
# expressive power
authorizer = globus_sdk.AccessTokenAuthorizer(TRANSFER_TOKEN)
tc = globus_sdk.TransferClient(authorizer=authorizer)

# high level interface; provides iterators for list responses
print("My Endpoints:")
for ep in tc.endpoint_search(filter_scope="my-endpoints"):
    print("[{}] {}".format(ep["id"], ep["display_name"]))
```

Note that the `TRANSFER_TOKEN` is only valid for a limited time. You'll have to login again when it expires.

1.2.2 Advanced Tutorial

In the first 4 steps of the Tutorial, we did a lot of hocus-pocus to procure Access Tokens, but we didn't dive into how we are getting them (or why they exist at all). Not only will we talk through more detail on Access Tokens, but we'll also explore more advanced use cases and their near-cousins, Refresh Tokens.

Advanced 1: Exploring the OAuthTokenResponse

We powered through the OAuth2 flow in the basic tutorial. It's worth looking closer at the token response itself, as it is of particular interest. This is the ultimate product of the flow, and it contains all of the credentials that we'll want and need moving forward.

Remember:

```
client = globus_sdk.NativeAppAuthClient(CLIENT_ID)
client.oauth2_start_flow()
```

(continues on next page)

(continued from previous page)

```
print('Please go to this URL and login: {0}'
      .format(client.oauth2_get_authorize_url()))

get_input = getattr(__builtins__, 'raw_input', input)
auth_code = get_input('Please enter the code here: ').strip()
token_response = client.oauth2_exchange_code_for_tokens(auth_code)
```

Though it has a few attributes and methods, by far the most important thing about `token_response` to understand is `token_response.by_resource_server`.

Let's take a look at `str(token_response.by_resource_server)`:

```
>>> str(token_response.by_resource_server)
{
  "auth.globus.org": {
    "access_token": "AQBX8YvVAAAAAAADxhAtF46RxjcFuoxN1oSOmEk-
↪hBqvOejY4imMbZlC0B8THfoFuOK9rshN6TV7I0uwf0hb",
    "scope": "openid email profile",
    "token_type": "Bearer",
    "expires_at_seconds": 1476121216,
    "refresh_token": None
  },
  "transfer.api.globus.org": {
    "access_token": "AQBX8YvVAAAAAAADxg-u9uULMyTkLw4_15ReO_
↪f2E056wLqjAWeLP51pgakLxYmyUDfGTd4SnYCiRjFq3mnj",
    "scope": "urn:globus:auth:scope:transfer.api.globus.org:all",
    "token_type": "Bearer",
    "expires_at_seconds": 1476121286,
    "refresh_token": None
  }
}
```

A token response is structured with the following info:

- Resource Servers: The services (e.x. APIs) which require Tokens. These are the keys, *"auth.globus.org"* and *"transfer.api.globus.org"*
- Access Tokens: Credentials you can use to talk to Resource Servers. We get back separate Access Tokens for each Resource Server. Importantly, this means that if Globus is issuing tokens to *evil.api.example.com*, you don't need to worry that *evil.api.example.com* will ever see tokens valid for Globus Transfer
- Scope: A list of activities that the Access Token is good for against the Resource Server. They are defined and enforced by the Resource Server.
- token_type: With what kind of authorization should the Access Token be used? For the foreseeable future, all Globus tokens are sent as Bearer Auth headers.
- expires_at_seconds: A POSIX timestamp – the time at which the relevant Access Token expires and is no longer accepted by the service.
- Refresh Tokens: Credentials used to replace or "refresh" your access tokens when they expire. If requested, you'll get one for each Resource Server. Details on their usage are in the next Advanced Tutorial

Advanced 2: Refresh Tokens, Never Login Again

Logging in to Globus through the web interface gets pretty old pretty fast. In fact, as soon as you write your first cron job against Globus, you'll need something better. Enter Refresh Tokens: credentials which never expire unless revoked, and which can be used to get new Access Tokens whenever those do expire.

Getting yourself refresh tokens to play with is actually pretty easy. Just tweak your login flow with one argument:

```
client = globus_sdk.NativeAppAuthClient(CLIENT_ID)
client.oauth2_start_flow(refresh_tokens=True)

print('Please go to this URL and login: {0}'
      .format(client.oauth2_get_authorize_url()))

get_input = getattr(__builtins__, 'raw_input', input)
auth_code = get_input('Please enter the code here: ').strip()
token_response = client.oauth2_exchange_code_for_tokens(auth_code)
```

If you peek at the `token_response` now, you'll see that the `"refresh_token"` fields are no longer nulled.

Now we've got a problem though: it's great to say that you can refresh tokens whenever you want, but how do you know when to do that? And what if an Access Token gets revoked before it's ready to expire? It turns out that using these correctly is pretty delicate, but there is a way forward that's pretty much painless.

Let's assume you want to do this with the `globus_sdk.TransferClient`.

```
# let's get stuff for the Globus Transfer service
globus_transfer_data = token_response.by_resource_server['transfer.api.globus.org']
# the refresh token and access token, often abbr. as RT and AT
transfer_rt = globus_transfer_data['refresh_token']
transfer_at = globus_transfer_data['access_token']
expires_at_s = globus_transfer_data['expires_at_seconds']

# Now we've got the data we need, but what do we do?
# That "GlobusAuthorizer" from before is about to come to the rescue

authorizer = globus_sdk.RefreshTokenAuthorizer(
    transfer_rt, client, access_token=transfer_at, expires_at=expires_at_s)

# and try using `tc` to make TransferClient calls. Everything should just
# work -- for days and days, months and months, even years
tc = globus_sdk.TransferClient(authorizer=authorizer)
```

A couple of things to note about this: `access_token` and `expires_at` are optional arguments to `RefreshTokenAuthorizer`. So, if all you've got on hand is a refresh token, it can handle the bootstrapping problem. Also, it's good to know that the `RefreshTokenAuthorizer` will retry the first call that fails with an authorization error. If the second call also fails, it won't try anymore.

Finally, and perhaps most importantly, we must stress that you need to protect your Refresh Tokens. They are an infinite lifetime credential to act as you, so, like passwords, they should only be stored in secure locations.

1.3 Service Clients

The Globus SDK provides a client class for every public Globus API. Each client object takes authentication credentials from config files, environment variables, or programmatically via *GlobusAuthorizers*.

Once instantiated, a Client gives you high-level interface to make API calls, without needing to know Globus API endpoints or their various parameters.

For example, you could use the `TransferClient` to list your task history very simply:

```
from globus_sdk import TransferClient

# you must have transfer_token in your config for this to work
tc = TransferClient()

print("My Last 25 Tasks:")
# `filter` to get Delete Tasks (default is just Transfer Tasks)
for task in tc.task_list(num_results=25, filter="type:TRANSFER,DELETE"):
    print(task["task_id"], task["type"], task["status"])
```

Multi-Thread and Multi-Process Safety

Each Globus SDK client class holds a networking session object to interact with the Globus API. Using a previously created service client object after forking or between multiple threads should be considered unsafe. In multi-processing applications, it is recommended to create service client objects after process forking and to ensure that there is only one service client instance created per process.

Client Types

Common API The Base Client provides methods which are accessible via any client object. These methods correspond directly to HTTP verbs and represent single HTTP requests.

For example, `get()` is a method of `TransferClient` and `AuthClient` objects, and for each of those it sends an HTTP `GET` to the target service.

Globus Auth Using Globus Auth in the SDK is done via several different client types which represent different types of applications.

Additionally, there are objects for managing OAuth2 login flows and customized responses which make it easier to unpack token responses.

The Globus Auth interface also includes the *`IdentityMap`* object, which helps manage bulk identity lookups.

Globus Transfer The Globus Transfer API is usable in the SDK through the *`TransferClient`* object.

Additionally, there are helper objects for assembling Transfer and Delete Task data for submission to the service.

Customized response types simplify handling of paginated response data, producing iterables of the underlying response types.

Globus Search The Globus Search API is usable in the SDK through the *`SearchClient`* object.

Additionally, there is a *`SearchQuery`* object which provides a chainable API for building query documents.

1.3.1 Common API

All service clients support the low level interface, provided by the `BaseClient`.

class globus_sdk.base.**BaseClient**(*service*, *environment=None*, *base_url=None*, *base_path=None*, *authorizer=None*, *app_name=None*, *http_timeout=None*, **args*, ***kwargs*)

Simple client with error handling for Globus REST APIs. Implemented as a wrapper around a `requests.Session` object, with a simplified interface that does not directly expose anything from requests.

You should *never* try to directly instantiate a `BaseClient`.

Parameters

- **authorizer** (*GlobusAuthorizer*) – A `GlobusAuthorizer` which will generate Authorization headers
- **app_name** (*str*) – Optional "nice name" for the application. Has no bearing on the semantics of client actions. It is just passed as part of the User-Agent string, and may be useful when debugging issues with the Globus Team
- **http_timeout** (*float*) – Number of seconds to wait on HTTP connections. Default is 60. A value of -1 indicates that no timeout should be used (requests can hang indefinitely).

All other parameters are for internal use and should be ignored.

set_app_name(*app_name*)

Set an application name to send to Globus services as part of the User Agent.

Application developers are encouraged to set an app name as a courtesy to the Globus Team, and to potentially speed resolution of issues when interacting with Globus Support.

get(*path*, *params=None*, *headers=None*, *response_class=None*, *retry_401=True*)

Make a GET request to the specified path.

Parameters

- **path** (*str*) – Path for the request, with or without leading slash
- **params** (*dict*) – Parameters to be encoded as a query string
- **headers** (*dict*) – HTTP headers to add to the request
- **response_class** (*class*) – Class for response object, overrides the client's `default_response_class`
- **retry_401** (*bool*) – Retry on 401 responses with fresh Authorization if `self.authorizer` supports it

Returns *GlobusHTTPResponse* object

post(*path*, *json_body=None*, *params=None*, *headers=None*, *text_body=None*, *response_class=None*, *retry_401=True*)

Make a POST request to the specified path.

Parameters

- **path** (*str*) – Path for the request, with or without leading slash
- **params** (*dict*) – Parameters to be encoded as a query string
- **headers** (*dict*) – HTTP headers to add to the request
- **json_body** (*dict*) – Data which will be JSON encoded as the body of the request

- **text_body** (*str or dict*) – Either a raw string that will serve as the request body, or a dict which will be HTTP Form encoded
- **response_class** (*class*) – Class for response object, overrides the client's `default_response_class`
- **retry_401** (*bool*) – Retry on 401 responses with fresh Authorization if `self.authorizer` supports it

Returns *GlobusHTTPResponse* object

delete(*path*, *params=None*, *headers=None*, *response_class=None*, *retry_401=True*)
Make a DELETE request to the specified path.

Parameters

- **path** (*str*) – Path for the request, with or without leading slash
- **params** (*dict*) – Parameters to be encoded as a query string
- **headers** (*dict*) – HTTP headers to add to the request
- **response_class** (*class*) – Class for response object, overrides the client's `default_response_class`
- **retry_401** (*bool*) – Retry on 401 responses with fresh Authorization if `self.authorizer` supports it

Returns *GlobusHTTPResponse* object

put(*path*, *json_body=None*, *params=None*, *headers=None*, *text_body=None*, *response_class=None*, *retry_401=True*)
Make a PUT request to the specified path.

Parameters

- **path** (*str*) – Path for the request, with or without leading slash
- **params** (*dict*) – Parameters to be encoded as a query string
- **headers** (*dict*) – HTTP headers to add to the request
- **json_body** (*dict*) – Data which will be JSON encoded as the body of the request
- **text_body** (*str or dict*) – Either a raw string that will serve as the request body, or a dict which will be HTTP Form encoded
- **response_class** (*class*) – Class for response object, overrides the client's `default_response_class`
- **retry_401** (*bool*) – Retry on 401 responses with fresh Authorization if `self.authorizer` supports it

Returns *GlobusHTTPResponse* object

Responses

Unless noted otherwise, all method return values for Globus SDK Clients are `GlobusResponse` objects.

Some `GlobusResponse` objects are iterables. In those cases, their contents will also be `GlobusResponse` objects.

To customize client methods with additional detail, the SDK uses subclasses of `GlobusResponse`. For example the `GlobusHTTPResponse` attaches HTTP response information.

Generic Response Classes

class globus_sdk.response.**GlobusResponse**(*data*, *client=None*)

Generic response object, with a single `data` member.

The most common response data is a JSON dictionary. To make handling this type of response as seemless as possible, the `GlobusResponse` object also supports direct dictionary item access, as an alias for accessing an item of the underlying `data`. If `data` is not a dictionary, item access will raise `TypeError`.

```
>>> print("Response ID": r["id"]) # alias for r.data["id"]
```

`GlobusResponse` objects *always* wrap some kind of data to return to a caller. Most basic actions on a GlobusResponse are just ways of interacting with this data.

property data

Response data as a Python data structure. Usually a dict or list.

get(**args*, ***kwargs*)

`GlobusResponse.get` is just an alias for `GlobusResponse.data.get`

class globus_sdk.response.**GlobusHTTPResponse**(*http_response*, *client=None*)

Bases: *globus_sdk.response.GlobusResponse*

Response object that wraps an HTTP response from the underlying HTTP library. If the response is JSON, the parsed data will be available in `data`, otherwise `data` will be `None` and `text` should be used instead.

Variables

- **http_status** – HTTP status code returned by the server (int)
- **content_type** – Content-Type header returned by the server (str)

property data

Response data as a Python data structure. Usually a dict or list.

property text

The raw response data as a string.

1.3.2 Globus Transfer

Client

The primary interface for the Globus Transfer API is the `TransferClient` class.

class globus_sdk.**TransferClient**(*authorizer=None*, ***kwargs*)

Bases: *globus_sdk.base.BaseClient*

Client for the Globus Transfer API.

This class provides helper methods for most common resources in the REST API, and basic `get`, `put`, `post`, and `delete` methods from the base rest client that can be used to access any REST resource.

There are two types of helper methods: list methods which return an iterator of *GlobusResponse* objects, and simple methods that return a single *TransferResponse* object.

Some calls are paginated. If a call returns a *PaginatedResource* object, the result is an iterator which can only be walked *once*. If you need to do multiple passes over the result, call `list()` on the `PaginatedResource` or call the original method again to get fresh results.

Detailed documentation is available in the official REST API documentation, which is linked to from the method documentation. Methods that allow arbitrary keyword arguments will pass the extra arguments as query parameters.

Parameters **authorizer** (*GlobusAuthorizer*) – An authorizer instance used for all calls to Globus Transfer

Methods

- *add_endpoint_acl_rule()*
- *add_endpoint_role()*
- *add_endpoint_server()*
- *bookmark_list()*
- *cancel_task()*
- *create_bookmark()*
- *create_endpoint()*
- *create_shared_endpoint()*
- *delete_bookmark()*
- *delete_endpoint()*
- *delete_endpoint_acl_rule()*
- *delete_endpoint_role()*
- *delete_endpoint_server()*
- *endpoint_acl_list()*
- *endpoint_activate()*
- *endpoint_autoactivate()*
- *endpoint_deactivate()*
- *endpoint_get_activation_requirements()*
- *endpoint_manager_acl_list()*
- *endpoint_manager_cancel_status()*
- *endpoint_manager_cancel_tasks()*
- *endpoint_manager_create_pause_rule()*
- *endpoint_manager_delete_pause_rule()*
- *endpoint_manager_get_endpoint()*
- *endpoint_manager_get_pause_rule()*

- *endpoint_manager_get_task()*
- *endpoint_manager_hosted_endpoint_list()*
- *endpoint_manager_monitored_endpoints()*
- *endpoint_manager_pause_rule_list()*
- *endpoint_manager_pause_tasks()*
- *endpoint_manager_resume_tasks()*
- *endpoint_manager_task_event_list()*
- *endpoint_manager_task_list()*
- *endpoint_manager_task_pause_info()*
- *endpoint_manager_task_skipped_errors()*
- *endpoint_manager_task_successful_transfers()*
- *endpoint_manager_update_pause_rule()*
- *endpoint_role_list()*
- *endpoint_search()*
- *endpoint_server_list()*
- *get_bookmark()*
- *get_endpoint()*
- *get_endpoint_acl_rule()*
- *get_endpoint_role()*
- *get_endpoint_server()*
- *get_submission_id()*
- *get_task()*
- *my_effective_pause_rule_list()*
- *my_shared_endpoint_list()*
- *operation_ls()*
- *operation_mkdir()*
- *operation_rename()*
- *operation_symlink()*
- *submit_delete()*
- *submit_transfer()*
- *task_event_list()*
- *task_list()*
- *task_pause_info()*
- *task_skipped_errors()*
- *task_successful_transfers()*
- *task_wait()*

- *update_bookmark()*
- *update_endpoint()*
- *update_endpoint_acl_rule()*
- *update_endpoint_server()*
- *update_task()*

get_endpoint(*endpoint_id*, ***params*)
GET /endpoint/<endpoint_id>

Return type *TransferResponse*

Examples

```
>>> tc = globus_sdk.TransferClient(...)
>>> endpoint = tc.get_endpoint(endpoint_id)
>>> print("Endpoint name:",
>>>       endpoint["display_name"] or endpoint["canonical_name"])
```

External Documentation

See Get Endpoint by ID in the REST documentation for details.

update_endpoint(*endpoint_id*, *data*, ***params*)
PUT /endpoint/<endpoint_id>

Return type *TransferResponse*

Examples

```
>>> tc = globus_sdk.TransferClient(...)
>>> epup = dict(display_name="My New Endpoint Name",
>>>             description="Better Description")
>>> update_result = tc.update_endpoint(endpoint_id, epup)
```

External Documentation

See Update Endpoint by ID in the REST documentation for details.

create_endpoint(*data*)
POST /endpoint/<endpoint_id>

Return type *TransferResponse*

Examples

```
>>> tc = globus_sdk.TransferClient(...)
>>> ep_data = {
>>>   "DATA_TYPE": "endpoint",
>>>   "display_name": display_name,
>>>   "DATA": [
>>>     {
>>>       "DATA_TYPE": "server",
>>>       "hostname": "gridftp.example.edu",
>>>     },
>>>   ],
>>> }
>>> create_result = tc.create_endpoint(ep_data)
>>> endpoint_id = create_result["id"]
```

External Documentation

See Create endpoint in the REST documentation for details.

delete_endpoint(*endpoint_id*)
DELETE /endpoint/<endpoint_id>

Return type *TransferResponse*

Examples

```
>>> tc = globus_sdk.TransferClient(...)
>>> delete_result = tc.delete_endpoint(endpoint_id)
```

External Documentation

See Delete endpoint by id in the REST documentation for details.

endpoint_search(*filter_fulltext=None*, *filter_scope=None*, *num_results=25*, ***params*)
GET /endpoint_search?filter_fulltext=<filter_fulltext>&filter_scope=
↪<filter_scope>

Parameters

- **filter_fulltext** (*str, optional*) – The string to use in a full text search on endpoints. Effectively, the "search query" which is being requested. May be omitted with specific `filter_scope` values.
- **filter_scope** (*str, optional*) – A "scope" within which to search for endpoints. This must be one of the limited and known names known to the service, which can be found documented in the **External Documentation** below. Defaults to searching all endpoints (in which case `filter_fulltext` is required)
- **num_results** (*int or None*) – The number of search results to fetch from the service. May be set to `None` to request the maximum allowable number of results. [Default: `25`]
- **params** (*dict*) – Any additional parameters will be passed through as query params.

Return type *PaginatedResource*, an iterable of *GlobusResponse*

Examples

Search for a given string as a fulltext search:

```
>>> tc = globus_sdk.TransferClient(...)
>>> for ep in tc.endpoint_search('String to search for!'):
>>>     print(ep['display_name'])
```

Search for a given string, but only on endpoints that you own:

```
>>> for ep in tc.endpoint_search('foo', filter_scope='my-endpoints'):
>>>     print('{0} has ID {1}'.format(ep['display_name'], ep['id']))
```

Search results are capped at a number of elements equal to the `num_results` parameter. If you want more than the default, 25, elements, do like so:

```
>>> for ep in tc.endpoint_search('String to search for!',
>>>                              num_results=120):
>>>     print(ep['display_name'])
```

It is important to be aware that the Endpoint Search API limits you to 1000 results for any search query. You can request the maximum number of results either explicitly, with `num_results=1000`, or by stating that you want no limit by setting it to `None`:

```
>>> for ep in tc.endpoint_search('String to search for!',
>>>                              num_results=None):
>>>     print(ep['display_name'])
```

External Documentation

For additional information, see Endpoint Search. in the REST documentation for details.

endpoint_autoactivate(*endpoint_id*, ***params*)

`POST /endpoint/<endpoint_id>/autoactivate`

Return type *TransferResponse*

The following example will try to "auto" activate the endpoint using a credential available from another endpoint or sign in by the user with the same identity provider, but only if the endpoint is not already activated or going to expire within an hour (3600 seconds). If that fails, direct the user to the globus website to perform activation:

Examples

```
>>> tc = globus_sdk.TransferClient(...)
>>> r = tc.endpoint_autoactivate(ep_id, if_expires_in=3600)
>>> while (r["code"] == "AutoActivationFailed"):
>>>     print("Endpoint requires manual activation, please open "
>>>           "the following URL in a browser to activate the "
>>>           "endpoint:")
>>>     print("https://app.globus.org/file-manager?origin_id=%s"
>>>           % ep_id)
>>>     # For python 2.X, use raw_input() instead
>>>     input("Press ENTER after activating the endpoint:")
>>>     r = tc.endpoint_autoactivate(ep_id, if_expires_in=3600)
```

This is the recommended flow for most thick client applications, because many endpoints require activation via OAuth MyProxy, which must be done in a browser anyway. Web based clients can link directly to the URL.

You also might want messaging or logging depending on why and how the operation succeeded, in which case you'll need to look at the value of the "code" field and either decide on your own messaging or use the response's "message" field.

```
>>> tc = globus_sdk.TransferClient(...)
>>> r = tc.endpoint_autoactivate(ep_id, if_expires_in=3600)
>>> if r['code'] == 'AutoActivationFailed':
>>>     print('Endpoint({}) Not Active! Error! Source message: {}'
>>>           .format(ep_id, r['message']))
>>>     sys.exit(1)
>>> elif r['code'] == 'AutoActivated.CachedCredential':
>>>     print('Endpoint({}) autoactivated using a cached credential.'
>>>           .format(ep_id))
>>> elif r['code'] == 'AutoActivated.GlobusOnlineCredential':
>>>     print(('Endpoint({}) autoactivated using a built-in Globus '
>>>            'credential.').format(ep_id))
>>> elif r['code'] = 'AlreadyActivated':
>>>     print('Endpoint({}) already active until at least {}'
>>>           .format(ep_id, 3600))
```

External Documentation

See Autoactivate endpoint in the REST documentation for details.

endpoint_deactivate(*endpoint_id*, ***params*)
`POST /endpoint/<endpoint_id>/deactivate`

Return type *TransferResponse*

External Documentation

See Deactive endpoint in the REST documentation for details.

endpoint_activate(*endpoint_id*, *requirements_data*, ***params*)
`POST /endpoint/<endpoint_id>/activate`

Return type *TransferResponse*

Consider using autoactivate and web activation instead, described in the example for *endpoint_autoactivate()*.

External Documentation

See Activate endpoint in the REST documentation for details.

endpoint_get_activation_requirements(*endpoint_id*, ***params*)
`GET /endpoint/<endpoint_id>/activation_requirements`

Return type *ActivationRequirementsResponse*

External Documentation

See Get activation requirements in the REST documentation for details.

my_effective_pause_rule_list(*endpoint_id*, ***params*)
`GET /endpoint/<endpoint_id>/my_effective_pause_rule_list`

Return type *IterableTransferResponse*

External Documentation

See Get my effective endpoint pause rules in the REST documentation for details.

my_shared_endpoint_list(*endpoint_id*, ***params*)
`GET /endpoint/<endpoint_id>/my_shared_endpoint_list`

Return type *IterableTransferResponse*

External Documentation

See Get shared endpoint list in the REST documentation for details.

create_shared_endpoint(*data*)
`POST /shared_endpoint`

Parameters **data** (*dict*) – A python dict representation of a `shared_endpoint` document

Return type *TransferResponse*

Examples

```
>>> tc = globus_sdk.TransferClient(...)
>>> shared_ep_data = {
>>>   "DATA_TYPE": "shared_endpoint",
>>>   "host_endpoint": host_endpoint_id,
>>>   "host_path": host_path,
>>>   "display_name": display_name,
```

(continues on next page)

(continued from previous page)

```
>>>   # optionally specify additional endpoint fields
>>>   "description": "my test share"
>>> }
>>> create_result = tc.create_shared_endpoint(shared_ep_data)
>>> endpoint_id = create_result["id"]
```

External Documentation

See Create shared endpoint in the REST documentation for details.

endpoint_server_list(*endpoint_id*, ***params*)
`GET /endpoint/<endpoint_id>/server_list`

Return type *IterableTransferResponse*

External Documentation

See Get endpoint server list in the REST documentation for details.

get_endpoint_server(*endpoint_id*, *server_id*, ***params*)
`GET /endpoint/<endpoint_id>/server/<server_id>`

Return type *TransferResponse*

External Documentation

See Get endpoint server by id in the REST documentation for details.

add_endpoint_server(*endpoint_id*, *server_data*)
`POST /endpoint/<endpoint_id>/server`

Return type *TransferResponse*

External Documentation

See Add endpoint server in the REST documentation for details.

update_endpoint_server(*endpoint_id*, *server_id*, *server_data*)
`PUT /endpoint/<endpoint_id>/server/<server_id>`

Return type *TransferResponse*

External Documentation

See Update endpoint server by id in the REST documentation for details.

delete_endpoint_server(*endpoint_id*, *server_id*)
`DELETE /endpoint/<endpoint_id>/server/<server_id>`

Return type *TransferResponse*

External Documentation

See Delete endpoint server by id in the REST documentation for details.

endpoint_role_list(*endpoint_id*, ***params*)
`GET /endpoint/<endpoint_id>/role_list`

Return type *IterableTransferResponse*

External Documentation

See Get list of endpoint roles in the REST documentation for details.

add_endpoint_role(*endpoint_id*, *role_data*)
`POST /endpoint/<endpoint_id>/role`

Return type *TransferResponse*

External Documentation

See Create endpoint role in the REST documentation for details.

get_endpoint_role(*endpoint_id*, *role_id*, ***params*)
`GET /endpoint/<endpoint_id>/role/<role_id>`

Return type *TransferResponse*

External Documentation

See Get endpoint role by id in the REST documentation for details.

delete_endpoint_role(*endpoint_id*, *role_id*)
`DELETE /endpoint/<endpoint_id>/role/<role_id>`

Return type *TransferResponse*

External Documentation

See Delete endpoint role by id in the REST documentation for details.

endpoint_acl_list(*endpoint_id*, ***params*)
`GET /endpoint/<endpoint_id>/access_list`

Return type *IterableTransferResponse*

External Documentation

See Get list of access rules in the REST documentation for details.

get_endpoint_acl_rule(*endpoint_id*, *rule_id*, ***params*)
`GET /endpoint/<endpoint_id>/access/<rule_id>`

Return type *TransferResponse*

External Documentation

See Get access rule by id in the REST documentation for details.

add_endpoint_acl_rule(*endpoint_id*, *rule_data*)
`POST /endpoint/<endpoint_id>/access`

Parameters

- **endpoint_id** (*str*) – ID of endpoint to which to add the acl
- **rule_data** (*dict*) – A python dict representation of an `access` document

Return type *TransferResponse*

Examples

```
>>> tc = globus_sdk.TransferClient(...)
>>> rule_data = {
>>>   "DATA_TYPE": "access",
>>>   "principal_type": "identity",
>>>   "principal": identity_id,
>>>   "path": "/dataset1/",
>>>   "permissions": "rw",
>>> }
>>> result = tc.add_endpoint_acl_rule(endpoint_id, rule_data)
>>> rule_id = result["access_id"]
```

Note that if this rule is being created on a shared endpoint the "path" field is relative to the "host_path" of the shared endpoint.

External Documentation

See Create access rule in the REST documentation for details.

update_endpoint_acl_rule(*endpoint_id*, *rule_id*, *rule_data*)
`PUT /endpoint/<endpoint_id>/access/<rule_id>`

Return type *TransferResponse*

External Documentation

See Update access rule in the REST documentation for details.

delete_endpoint_acl_rule(*endpoint_id*, *rule_id*)
`DELETE /endpoint/<endpoint_id>/access/<rule_id>`

Return type *TransferResponse*

External Documentation

See Delete access rule in the REST documentation for details.

bookmark_list(***params*)
`GET /bookmark_list`

Return type *IterableTransferResponse*

External Documentation

See Get list of bookmarks in the REST documentation for details.

create_bookmark(*bookmark_data*)
`POST /bookmark`

Return type *TransferResponse*

External Documentation

See Create bookmark in the REST documentation for details.

get_bookmark(*bookmark_id*, ***params*)
`GET /bookmark/<bookmark_id>`

Return type *TransferResponse*

External Documentation

See Get bookmark by id in the REST documentation for details.

update_bookmark(*bookmark_id*, *bookmark_data*)
`PUT /bookmark/<bookmark_id>`

Return type *TransferResponse*

External Documentation

See Update bookmark in the REST documentation for details.

delete_bookmark(*bookmark_id*)
`DELETE /bookmark/<bookmark_id>`

Return type *TransferResponse*

External Documentation

See Delete bookmark by id in the REST documentation for details.

operation_ls(*endpoint_id*, ***params*)
GET /operation/endpoint/<endpoint_id>/ls

Return type *IterableTransferResponse*

Examples

```
>>> tc = globus_sdk.TransferClient(...)
>>> for entry in tc.operation_ls(ep_id, path="/~/project1/"):
>>>     print(entry["name"], entry["type"])
```

External Documentation

See List Directory Contents in the REST documentation for details.

operation_mkdir(*endpoint_id*, *path*, ***params*)
POST /operation/endpoint/<endpoint_id>/mkdir

Return type *TransferResponse*

Examples

```
>>> tc = globus_sdk.TransferClient(...)
>>> tc.operation_mkdir(ep_id, path="/~/newdir/")
```

External Documentation

See Make Directory in the REST documentation for details.

operation_rename(*endpoint_id*, *oldpath*, *newpath*, ***params*)
POST /operation/endpoint/<endpoint_id>/rename

Return type *TransferResponse*

Examples

```
>>> tc = globus_sdk.TransferClient(...)
>>> tc.operation_rename(ep_id, oldpath="/~/file1.txt",
>>>                     newpath="/~/project1data.txt")
```

External Documentation

See Rename in the REST documentation for details.

operation_symlink(*endpoint_id*, *symlink_target*, *path*, ***params*)
POST /operation/endpoint/<endpoint_id>/symlink

Return type *TransferResponse*

The `path` is the name of the symlink, and the `symlink_target` is the path referenced by the symlink.

Examples

```
>>> tc = globus_sdk.TransferClient(...)
>>> tc.operation_symlink(ep_id, symlink_target="/~/file1.txt",
>>>                      path="/~/link-to-file1.txt")
```

External Documentation

See Symlink in the REST documentation for details.

get_submission_id(***params*)
GET /submission_id

Return type *TransferResponse*

Submission IDs are required to submit tasks to the Transfer service via the *submit_transfer* and *submit_delete* methods.

Most users will not need to call this method directly, as the convenience classes *TransferData* and *DeleteData* will call it automatically if they are not passed a `submission_id` explicitly.

External Documentation

See Get a submission id in the REST documentation for more details.

submit_transfer(*data*)

`POST /transfer`

Return type *TransferResponse*

Examples

```
>>> tc = globus_sdk.TransferClient(...)
>>> tdata = globus_sdk.TransferData(tc, source_endpoint_id,
>>>                                 destination_endpoint_id,
>>>                                 label="SDK example",
>>>                                 sync_level="checksum")
>>> tdata.add_item("/source/path/dir/", "/dest/path/dir/",
>>>                recursive=True)
>>> tdata.add_item("/source/path/file.txt",
>>>                "/dest/path/file.txt")
>>> transfer_result = tc.submit_transfer(tdata)
>>> print("task_id =", transfer_result["task_id"])
```

The *data* parameter can be a normal Python dictionary, or a *TransferData* object.

External Documentation

See Submit a transfer task in the REST documentation for more details.

submit_delete(*data*)

`POST /delete`

Return type *TransferResponse*

Examples

```
>>> tc = globus_sdk.TransferClient(...)
>>> ddata = globus_sdk.DeleteData(tc, endpoint_id, recursive=True)
>>> ddata.add_item("/dir/to/delete/")
>>> ddata.add_item("/file/to/delete/file.txt")
>>> delete_result = tc.submit_delete(ddata)
>>> print("task_id =", delete_result["task_id"])
```

The *data* parameter can be a normal Python dictionary, or a *DeleteData* object.

External Documentation

See Submit a delete task in the REST documentation for details.

task_list(*num_results=10*, ***params*)

Get an iterable of task documents owned by the current user.

`GET /task_list`

Parameters

- **num_results** (*int or none*) – The number of tasks to fetch from the service. May be set to `None` to request the maximum allowable number of results. [Default: `10`]

- **params** (*dict, optional*) – Any additional parameters will be passed through as query params.

Return type *PaginatedResource*, an iterable of *GlobusResponse*

Examples

Fetch the default number (10) of tasks and print some basic info:

```
>>> tc = TransferClient(...)
>>> for task in tc.task_list():
>>>     print("Task({}): {} -> {}".format(
>>>         task["task_id"], task["source_endpoint"],
>>>         task["destination_endpoint"]))
```

External Documentation

See Task list in the REST documentation for details.

task_event_list(*task_id*, *num_results=10*, ***params*)

List events (for example, faults and errors) for a given Task.

```
GET /task/<task_id>/event_list
```

Parameters

- **task_id** (*str*) – The ID of the task to inspect.
- **num_results** (*int or None*) – The number of events to fetch from the service. May be set to `None` to request the maximum allowable number of results. [Default: `10`]
- **params** (*dict, optional*) – Any additional parameters will be passed through as query params.

Return type *PaginatedResource*, an iterable of *GlobusResponse*

Examples

Fetch the default number (10) of events and print some basic info:

```
>>> tc = TransferClient(...)
>>> task_id = ...
>>> for event in tc.task_event_list(task_id):
>>>     print("Event on Task({}) at {}:\n{}".format(
>>>         task_id, event["time"], event["description"])
```

External Documentation

See Get event list in the REST documentation for details.

get_task(*task_id*, ***params*)

```
GET /task/<task_id>
```

Return type *TransferResponse*

External Documentation

See Get task by id in the REST documentation for details.

update_task(*task_id*, *data*, ***params*)

```
PUT /task/<task_id>
```

Return type *TransferResponse*

External Documentation

See Update task by id in the REST documentation for details.

cancel_task(*task_id*)
`POST /task/<task_id>/cancel`

Return type *TransferResponse*

External Documentation

See Cancel task by id in the REST documentation for details.

task_wait(*task_id*, *timeout=10*, *polling_interval=10*)
Wait until a Task is complete or fails, with a time limit. If the task is "ACTIVE" after time runs out, returns `False`. Otherwise returns `True`.

Parameters

- **task_id** (*str*) – ID of the Task to wait on for completion
- **timeout** (*int, optional*) – Number of seconds to wait in total. Minimum 1. [Default: `10`]
- **polling_interval** (*int, optional*) – Number of seconds between queries to Globus about the Task status. Minimum 1. [Default: `10`]

Examples

If you want to wait for a task to terminate, but want to warn every minute that it doesn't terminate, you could:

```
>>> tc = TransferClient(...)
>>> while not tc.task_wait(task_id, timeout=60):
>>>     print("Another minute went by without {0} terminating"
>>>           .format(task_id))
```

Or perhaps you want to check on a task every minute for 10 minutes, and give up if it doesn't complete in that time:

```
>>> tc = TransferClient(...)
>>> done = tc.task_wait(task_id, timeout=600, polling_interval=60):
>>> if not done:
>>>     print("{0} didn't successfully terminate!"
>>>           .format(task_id))
>>> else:
>>>     print("{0} completed".format(task_id))
```

You could print dots while you wait for a task by only waiting one second at a time:

```
>>> tc = TransferClient(...)
>>> while not tc.task_wait(task_id, timeout=1, polling_interval=1):
>>>     print(".", end="")
>>> print("\n{0} completed!".format(task_id))
```

task_pause_info(*task_id*, ***params*)
`GET /task/<task_id>/pause_info`

Return type *TransferResponse*

External Documentation

See Get task pause info in the REST documentation for details.

task_successful_transfers(*task_id*, *num_results=100*, ***params*)
Get the successful file transfers for a completed Task.

Note: Only files that were actually transferred are included. This does not include directories, files that were checked but skipped as part of a sync transfer, or files which were skipped due to skip_source_errors being set on the task.

`GET /task/<task_id>/successful_transfers`

Parameters

- **task_id** (*str*) – The ID of the task to inspect.
- **num_results** (*int or None, optional*) – The number of file transfer records to fetch from the service. May be set to `None` to request the maximum allowable number of results. [Default: `100`]
- **params** (*dict, optional*) – Any additional parameters will be passed through as query params.

Return type *PaginatedResource*, an iterable of *GlobusResponse*

Examples

Fetch all transferred files for a task and print some basic info:

```
>>> tc = TransferClient(...)
>>> task_id = ...
>>> for info in tc.task_successful_transfers(task_id):
>>>     print("{} -> {}".format(
>>>         info["source_path"], info["destination_path"]))
```

External Documentation

See Get Task Successful Transfers in the REST documentation for details.

task_skipped_errors(*task_id*, *num_results=100*, ***params*)
Get path and error information for all paths that were skipped due to skip_source_errors being set on a completed transfer Task.

`GET /task/<task_id>/skipped_errors`

Parameters

- **task_id** (*str*) – The ID of the task to inspect.
- **num_results** (*int or None, optional*) – The number of file transfer records to fetch from the service. May be set to `None` to request the maximum allowable number of results. [Default: `100`]
- **params** (*dict, optional*) – Any additional parameters will be passed through as query params.

Return type *PaginatedResource*, an iterable of *GlobusResponse*

Examples

Fetch all skipped errors for a task and print some basic info:

```
>>> tc = TransferClient(...)
>>> task_id = ...
>>> for info in tc.task_skipped_errors(task_id):
```

(continues on next page)

(continued from previous page)

```
>>>     print("{} -> {}".format(
>>>         info["error_code"], info["source_path"]))
```

External Documentation

See Get Task Skipped Errors in the REST documentation for details.

endpoint_manager_monitored_endpoints(***params*)
Get endpoints the current user is a monitor or manager on.

`GET endpoint_manager/monitored_endpoints`

Return type iterable of *GlobusResponse*

See Get monitored endpoints in the REST documentation for details.

endpoint_manager_hosted_endpoint_list(*endpoint_id*, ***params*)
Get shared endpoints hosted on the given endpoint.

`GET /endpoint_manager/endpoint/<endpoint_id>/hosted_endpoint_list`

Return type iterable of *GlobusResponse*

See Get hosted endpoint list in the REST documentation for details.

endpoint_manager_get_endpoint(*endpoint_id*, ***params*)
Get endpoint details as an admin.

`GET /endpoint_manager/endpoint/<endpoint_id>`

Return type *TransferResponse*

External Documentation

See Get endpoint as admin in the REST documentation for details.

endpoint_manager_acl_list(*endpoint_id*, ***params*)
Get a list of access control rules on specified endpoint as an admin.

`GET endpoint_manager/endpoint/<endpoint_id>/access_list`

Return type *IterableTransferResponse*

External Documentation

See Get endpoint access list as admin in the REST documentation for details.

endpoint_manager_task_list(*num_results=10*, ***params*)
Get a list of tasks visible via `activity_monitor` role, as opposed to tasks owned by the current user.

`GET endpoint_manager/task_list`

Parameters

- **num_results** (*int or None, optional*) – The number of tasks to fetch from the service. May be set to `None` to request the maximum allowable number of results. [Default: `10`]
- **params** (*dict, optional*) – Any additional parameters will be passed through as query params.

Return type *PaginatedResource*, an iterable of *GlobusResponse*

Examples

Fetch the default number (10) of tasks and print some basic info:

```
>>> tc = TransferClient(...)
>>> for task in tc.endpoint_manager_task_list():
>>>     print("Task({}): {} -> {}\n  was submitted by\n  {}".format(
>>>         task["task_id"], task["source_endpoint"],
>>>         task["destination_endpoint"], task["owner_string"]))
```

Do that same operation on *all* tasks visible via `activity_monitor` status:

```
>>> tc = TransferClient(...)
>>> for task in tc.endpoint_manager_task_list(num_results=None):
>>>     print("Task({}): {} -> {}\n  was submitted by\n  {}".format(
>>>         task["task_id"], task["source_endpoint"],
>>>         task["destination_endpoint"], task["owner_string"]))
```

External Documentation

See Advanced Endpoint Management: Get tasks in the REST documentation for details.

endpoint_manager_get_task(*task_id*, ***params*)

Get task info as an admin. Requires activity monitor effective role on the destination endpoint of the task.

`GET /endpoint_manager/task/<task_id>`

Return type *TransferResponse*

External Documentation

See Get task as admin in the REST documentation for details.

endpoint_manager_task_event_list(*task_id*, *num_results=10*, ***params*)

List events (for example, faults and errors) for a given task as an admin. Requires activity monitor effective role on the destination endpoint of the task.

`GET /task/<task_id>/event_list`

Parameters

- **task_id** (*str*) – The ID of the task to inspect.
- **num_results** (*int or None, optional*) – The number of events to fetch from the service. May be set to `None` to request the maximum allowable number of results. [Default: `10`]
- **params** (*dict, optional*) – Any additional parameters will be passed through as query params.

Return type *PaginatedResource*, an iterable of *GlobusResponse*

External Documentation

See Get task events as admin in the REST documentation for details.

endpoint_manager_task_pause_info(*task_id*, ***params*)

Get details about why a task is paused as an admin. Requires activity monitor effective role on the destination endpoint of the task.

`GET /endpoint_manager/task/<task_id>/pause_info`

Return type *TransferResponse*

External Documentation

See Get task pause info as admin in the REST documentation for details.

endpoint_manager_task_successful_transfers(*task_id*, *num_results=100*, ***params*)
Get the successful file transfers for a completed Task as an admin.

`GET /endpoint_manager/task/<task_id>/successful_transfers`

Parameters

- **task_id** (*str*) – The ID of the task to inspect.
- **num_results** (*int or None, optional*) – The number of file transfer records to fetch from the service. May be set to `None` to request the maximum allowable number of results. [Default: `100`]
- **params** (*dict, optional*) – Any additional parameters will be passed through as query params.

Return type *PaginatedResource*, an iterable of *GlobusResponse*

External Documentation

See Get task successful transfers as admin in the REST documentation for details.

endpoint_manager_task_skipped_errors(*task_id*, *num_results=100*, ***params*)
Get skipped errors for a completed Task as an admin.

`GET /endpoint_manager/task/<task_id>/skipped_errors`

Parameters

- **task_id** (*str*) – The ID of the task to inspect.
- **num_results** (*int or None, optional*) – The number of skipped error records to fetch from the service. May be set to `None` to request the maximum allowable number of results. [Default: `100`]
- **params** (*dict, optional*) – Any additional parameters will be passed through as query params.

Return type *PaginatedResource*, an iterable of *GlobusResponse*

External Documentation

See Get task skipped errors as admin in the REST documentation for details.

endpoint_manager_cancel_tasks(*task_ids*, *message*, ***params*)
Cancel a list of tasks as an admin. Requires activity manager effective role on the task(s) source or destination endpoint(s).

`POST /endpoint_manager/admin_cancel`

Parameters

- **task_ids** (*iterable of str*) – List of task ids to cancel.
- **message** (*str*) – Message given to all users who's tasks have been canceled.
- **params** (*dict, optional*) – Any additional parameters will be passed through as query params.

Return type *TransferResponse*

External Documentation

See Cancel tasks as admin in the REST documentation for details.

endpoint_manager_cancel_status(*admin_cancel_id*, ***params*)
Get the status of an an admin cancel (result of endpoint_manager_cancel_tasks).

```
GET /endpoint_manager/admin_cancel/<admin_cancel_id>
```

Parameters

- **admin_cancel_id** (*str*) – The ID of the the cancel job to inspect.
- **params** (*dict, optional*) – Any additional parameters will be passed through as query params.

Return type *TransferResponse*

External Documentation

See Get cancel status by id in the REST documentation for details.

endpoint_manager_pause_tasks(*task_ids*, *message*, ***params*)
Pause a list of tasks as an admin. Requires activity manager effective role on the task(s) source or destination endpoint(s).

```
POST /endpoint_manager/admin_pause
```

Parameters

- **task_ids** (*iterable of str*) – List of task ids to pause.
- **message** (*str*) – Message given to all users who's tasks have been paused.
- **params** (*dict, optional*) – Any additional parameters will be passed through as query params.

Return type *TransferResponse*

External Documentation

See Pause tasks as admin in the REST documentation for details.

endpoint_manager_resume_tasks(*task_ids*, ***params*)
Resume a list of tasks as an admin. Requires activity manager effective role on the task(s) source or destination endpoint(s).

```
POST /endpoint_manager/admin_resume
```

Parameters

- **task_ids** (*iterable of str*) – List of task ids to resume.
- **params** (*dict, optional*) – Any additional parameters will be passed through as query params.

Return type *TransferResponse*

External Documentation

See Resume tasks as admin in the REST documentation for details.

endpoint_manager_pause_rule_list(*filter_endpoint=None*, ***params*)
Get a list of pause rules on endpoints that the current user has the activity monitor effective role on.

```
GET /endpoint_manager/pause_rule_list
```

Parameters

- **filter_endpoint** (*str*) – An endpoint ID. Limit results to rules on endpoints hosted by this endpoint. Must be activity monitor on this endpoint, not just the hosted endpoints.

- **params** (*dict, optional*) – Any additional parameters will be passed through as query params.

Return type *IterableTransferResponse*

External Documentation

See Get pause rules in the REST documentation for details.

endpoint_manager_create_pause_rule(*data*)

Create a new pause rule. Requires the activity manager effective role on the endpoint defined in the rule.

```
POST /endpoint_manager/pause_rule
```

Return type *TransferResponse*

Examples

```
>>> tc = globus_sdk.TransferClient(...)
>>> rule_data = {
>>>   "DATA_TYPE": "pause_rule",
>>>   "message": "Message to users explaining why tasks are paused",
>>>   "endpoint_id": "339abc22-aab3-4b45-bb56-8d40535bfd80",
>>>   "identity_id": None,  # affect all users on endpoint
>>>   "start_time": None  # start now
>>> }
>>> create_result = tc.endpoint_manager_create_pause_rule(ep_data)
>>> rule_id = create_result["id"]
```

External Documentation

See Create pause rule in the REST documentation for details.

endpoint_manager_get_pause_rule(*pause_rule_id*, ***params*)

Get an existing pause rule by ID. Requires the activity manager effective role on the endpoint defined in the rule.

```
GET /endpoint_manager/pause_rule/<pause_rule_id>
```

Parameters

- **pause_rule_id** (*str*) – ID of pause rule to get.
- **params** (*dict, optional*) – Any additional parameters will be passed through as query params.

Return type *TransferResponse*

External Documentation

See Get pause rule in the REST documentation for details.

endpoint_manager_update_pause_rule(*pause_rule_id*, *data*)

Update an existing pause rule by ID. Requires the activity manager effective role on the endpoint defined in the rule. Note that non update-able fields in data will be ignored.

```
PUT /endpoint_manager/pause_rule/<pause_rule_id>
```

Return type *TransferResponse*

Examples

```
>>> tc = globus_sdk.TransferClient(...)
>>> rule_data = {
>>>   "message": "Update to pause, reads are now allowed.",
```

(continues on next page)

(continued from previous page)

```
>>>   "pause_ls": False,
>>>   "pause_task_transfer_read": False
>>> }
>>> update_result = tc.endpoint_manager_update_pause_rule(ep_data)
```

External Documentation

See Update pause rule in the REST documentation for details.

endpoint_manager_delete_pause_rule(*pause_rule_id*, ***params*)

Delete an existing pause rule by ID. Requires the user to see the "editible" field of the rule as True. Any tasks affected by this rule will no longer be once it is deleted.

`DELETE /endpoint_manager/pause_rule/<pause_rule_id>`

Parameters

- **pause_rule_id** (*str*) – The ID of the pause rule to delete.
- **params** (*dict, optional*) – Any additional parameters will be passed through as query params.

Return type *TransferResponse*

External Documentation

See Delete pause rule in the REST documentation for details.

Helper Objects

These helper objects make it easier to correctly create data for consumption by a `TransferClient`.

class globus_sdk.**TransferData**(*transfer_client*, *source_endpoint*, *destination_endpoint*, *label=None*, *submission_id=None*, *sync_level=None*, *verify_checksum=False*, *preserve_timestamp=False*, *encrypt_data=False*, *deadline=None*, *skip_source_errors=False*, *fail_on_quota_errors=False*, *recursive_symlinks='ignore'*, ***kwargs*)

Bases: `dict`

Convenience class for constructing a transfer document, to use as the *data* parameter to *submit_transfer*.

At least one item must be added using *add_item*.

If `submission_id` isn't passed, one will be fetched automatically. The submission ID can be pulled out of here to inspect, but the document can be used as-is multiple times over to retry a potential submission failure (so there shouldn't be any need to inspect it).

Parameters

- **transfer_client** (*TransferClient*) – A `TransferClient` instance which will be used to get a submission ID if one is not supplied. Should be the same instance that is used to submit the transfer.
- **source_endpoint** (*str*) – The endpoint ID of the source endpoint
- **destination_endpoint** (*str*) – The endpoint ID of the destination endpoint
- **label** (*str, optional*) – A string label for the Task

- **submission_id** (*str, optional*) – A submission ID value fetched via *get_submission_id* . Defaults to using `transfer_client.get_submission_id`
- **sync_level** (*int or str, optional*) – The method used to compare items between the source and destination. One of `"exists"`, `"size"`, `"mtime"`, or `"checksum"` See the section below on sync-level for an explanation of values.
- **verify_checksum** (*bool, optional*) – When true, after transfer verify that the source and destination file checksums match. If they don't, re-transfer the entire file and keep trying until it succeeds. This will create CPU load on both the origin and destination of the transfer, and may even be a bottleneck if the network speed is high enough. [default: `False`]
- **preserve_timestamp** (*bool, optional*) – When true, Globus Transfer will attempt to set file timestamps on the destination to match those on the origin. [default: `False`]
- **encrypt_data** (*bool, optional*) – When true, all files will be TLS-protected during transfer. [default: `False`]
- **deadline** (*str or datetime, optional*) – An ISO-8601 timestamp (as a string) or a datetime object which defines a deadline for the transfer. At the deadline, even if the data transfer is not complete, the job will be canceled. We recommend ensuring that the timestamp is in UTC to avoid confusion and ambiguity. Examples of ISO-8601 timestamps include `2017-10-12 09:30Z`, `2017-10-12 12:33:54+00:00`, and `2017-10-12`
- **recursive_symlinks** (*str*) – Specify the behavior of recursive directory transfers when encountering symlinks. One of `"ignore"`, `"keep"`, or `"copy"`. `"ignore"` skips symlinks, `"keep"` creates symlinks at the destination matching the source (without modifying the link path at all), and `"copy"` follows symlinks on the source, failing if the link is invalid. [default: `"ignore"`]
- **skip_source_errors** (*bool, optional*) – When true, source permission denied and file not found errors from the source endpoint will cause the offending path to be skipped. [default: `False`]
- **fail_on_quota_errors** (*bool, optional*) – When true, quota exceeded errors will cause the task to fail. [default: `False`]

Any additional parameters are fed into the dict being created verbatim.

Sync Levels

The values for `sync_level` are used to determine how comparisons are made between files found both on the source and the destination. When files match, no data transfer will occur.

For compatibility, this can be an integer `0`, `1`, `2`, or `3` in addition to the string values.

The meanings are as follows:

value	behavior
`0`, `exists`	Determine whether or not to transfer based on file existence. If the destination file is absent, do the transfer.
`1`, `size`	Determine whether or not to transfer based on the size of the file. If destination file size does not match the source, do the transfer.
`2`, `mtime`	Determine whether or not to transfer based on modification times. If source has a newer modififed time than the destination, do the transfer.
`3`, `checksum`	Determine whether or not to transfer based on checksums of file contents. If source and destination contents differ, as determined by a checksum of their contents, do the transfer.

Examples

See the *submit_transfer* documentation for example usage.

External Documentation

See the Task document definition and Transfer specific fields in the REST documentation for more details on Transfer Task documents.

Methods

- *add_item()*
- *add_symlink_item()*

add_item(*source_path*, *destination_path*, *recursive=False*, *external_checksum=None*, *checksum_algorithm=None*, ***params*)

Add a file or directory to be transfered. If the item is a symlink to a file or directory, the file or directory at the target of the symlink will be transfered.

Appends a transfer_item document to the DATA key of the transfer document.

Note: The full path to the destination file must be provided for file items. Parent directories of files are not allowed. See task submission documentation for more details.

Parameters

- **source_path** (*str*) – Path to the source directory or file to be transfered
- **destination_path** (*str*) – Path to the source directory or file will be transfered to
- **recursive** (*bool*) – Set to True if the target at source path is a directory
- **external_checksum** (*str, optional*) – A checksum to verify source file integrity before the transfer and destination file integrity after the transfer. Cannot be used with directories. Assumed to be an MD5 checksum unless checksum_algorithm is also given.
- **checksum_algorithm** (*str, optional*) – Specifies the checksum algorithm to be used when verify_checksum is True, sync_level is "checksum" or 3, or an external_checksum is given.

add_symlink_item(*source_path*, *destination_path*)

Add a symlink to be transfered as a symlink rather than as the target of the symlink.

Appends a transfer_symlink_item document to the DATA key of the transfer document.

Parameters

- **source_path** (*str*) – Path to the source symlink
- **destination_path** (*str*) – Path to which the source symlink will be transfered

class globus_sdk.**DeleteData**(*transfer_client*, *endpoint*, *label=None*, *submission_id=None*, *recursive=False*, *deadline=None*, ***kwargs*)

Bases: dict

Convenience class for constructing a delete document, to use as the *data* parameter to *submit_delete*.

At least one item must be added using *add_item*.

If `submission_id` isn't passed, one will be fetched automatically. The submission ID can be pulled out of here to inspect, but the document can be used as-is multiple times over to retry a potential submission failure (so there shouldn't be any need to inspect it).

Parameters

- **transfer_client** (*TransferClient*) – A `TransferClient` instance which will be used to get a submission ID if one is not supplied. Should be the same instance that is used to submit the deletion.
- **endpoint** (*str*) – The endpoint ID which is targeted by this deletion Task
- **label** (*str, optional*) – A string label for the Task
- **submission_id** (*str, optional*) – A submission ID value fetched via *get_submission_id*. Defaults to using `transfer_client.get_submission_id`
- **recursive** (*bool*) – Recursively delete subdirectories on the target endpoint [default: `False`]
- **deadline** (*str or datetime, optional*) – An ISO-8601 timestamp (as a string) or a datetime object which defines a deadline for the deletion. At the deadline, even if the data deletion is not complete, the job will be canceled. We recommend ensuring that the timestamp is in UTC to avoid confusion and ambiguity. Examples of ISO-8601 timestamps include `2017-10-12 09:30Z`, `2017-10-12 12:33:54+00:00`, and `2017-10-12`

Examples

See the *submit_delete* documentation for example usage.

External Documentation

See the Task document definition and Delete specific fields in the REST documentation for more details on Delete Task documents.

Methods

- *add_item()*
- *add_symlink_item()*

add_item(*path*, ***params*)

Add a file or directory or symlink to be deleted. If any of the paths are directories, `recursive` must be set True on the top level `DeleteData`. Symlinks will never be followed, only deleted.

Appends a delete_item document to the DATA key of the delete document.

Client Errors

When an error occurs, a `TransferClient` will raise this specialized type of error, rather than a generic `GlobusAPIError`.

class globus_sdk.exc.**TransferAPIError**(*r*)

Bases: `globus_sdk.exc.GlobusAPIError`

Error class for the Transfer API client. In addition to the inherited `code` and `message` instance variables, provides:

Variables **request_id** – Unique identifier for the request, which should be provided when contacting support@globus.org.

Transfer Responses

class globus_sdk.transfer.response.**ActivationRequirementsResponse**(**args*, ***kwargs*)

Bases: globus_sdk.transfer.response.base.TransferResponse

Response class for Activation Requirements responses.

All Activation Requirements documents refer to a specific Endpoint, from whence they were acquired. References to "the Endpoint" implicitly refer to that originating Endpoint, and not to some other Endpoint.

External Documentation

See Activation Requirements Document in the API documentation for details.

active_until(*time_seconds*, *relative_time=True*)

Check if the Endpoint will be active until some time in the future, given as an integer number of seconds. When `relative_time=False`, the `time_seconds` is interpreted as a POSIX timestamp.

This supports queries using both relative and absolute timestamps to better support a wide range of use cases. For example, if I have a task that I know will typically take N seconds, and I want an M second safety margin:

```
>>> num_secs_allowed = N + M
>>> tc = TransferClient(...)
>>> reqs_doc = tc.endpoint_get_activation_requirements(...)
>>> if not reqs_doc.active_until(num_secs_allowed):
>>>     raise Exception("Endpoint won't be active long enough")
>>> ...
```

or, alternatively, if I know that the endpoint must be active until October 18th, 2016 for my tasks to complete:

```
>>> oct18_2016 = 1476803436
>>> tc = TransferClient(...)
>>> reqs_doc = tc.endpoint_get_activation_requirements(...)
>>> if not reqs_doc.active_until(oct18_2016, relative_time=False):
>>>     raise Exception("Endpoint won't be active long enough")
>>> ...
```

Parameters

- **time_seconds** (*int*) – Number of seconds into the future.
- **relative_time** (*bool*) – Defaults to True. When False, `time_seconds` is treated as a POSIX timestamp (i.e. seconds since epoch as an integer) instead of its ordinary behavior.

Returns True if the Endpoint will be active until the deadline, False otherwise

Return type `bool`

property always_activated

Returns True if the endpoint activation never expires (e.g. shared endpoints, globus connect personal endpoints).

Return type `bool`

property supports_auto_activation

Check if the document lists Auto-Activation as an available type of activation. Typically good to use when you need to catch endpoints that require web activation before proceeding.

```
>>> endpoint_id = "..."
>>> tc = TransferClient(...)
>>> reqs_doc = tc.endpoint_get_activation_requirements(endpoint_id)
>>> if not reqs_doc.supports_auto_activation:
>>>     # use `from __future__ import print_function` in py2
>>>     print(("This endpoint requires web activation. "
>>>            "Please login and activate the endpoint here:\n"
>>>            "https://app.globus.org/file-manager?origin_id={}")
>>>           .format(endpoint_id), file=sys.stderr)
>>>     # py3 calls it `input()` in py2, use `raw_input()`
>>>     input("Please Hit Enter When You Are Done")
```

Return type bool

property supports_web_activation

Check if the document lists known types of activation that can be done through the web. If this returns `False`, it means that the endpoint is of a highly unusual type, and you should directly inspect the response's `data` attribute to see what is required. Sending users to the web page for activation is also a fairly safe action to take. Note that `ActivationRequirementsResponse.supports_auto_activation` directly implies `ActivationRequirementsResponse.supports_web_activation`, so these are *not* exclusive.

For example,

```
>>> tc = TransferClient(...)
>>> reqs_doc = tc.endpoint_get_activation_requirements(...)
>>> if not reqs_doc.supports_web_activation:
>>>     # use `from __future__ import print_function` in py2
>>>     print("Highly unusual endpoint. " +
>>>           "Cannot webactivate. Raw doc: " +
>>>           str(reqs_doc), file=sys.stderr)
>>>     print("Sending user to web anyway, just in case.",
>>>           file=sys.stderr)
>>> ...
```

Return type bool

class globus_sdk.transfer.response.**IterableTransferResponse**(*http_response*, *client=None*)

Bases: `globus_sdk.transfer.response.base.TransferResponse`

Response class for non-paged list oriented resources. Allows top level fields to be accessed normally via standard item access, and also provides a convenient way to iterate over the sub-item list in the `DATA` key:

```
>>> print("Path:", r["path"])
>>> # Equivalent to: for item in r["DATA"]
>>> for item in r:
>>>     print(item["name"], item["type"])
```

class globus_sdk.transfer.response.**TransferResponse**(*http_response*, *client=None*)

Bases: *globus_sdk.response.GlobusHTTPResponse*

Base class for *TransferClient* responses.

PaginatedResource Responses

The `PaginatedResource` class should not typically be instantiated directly, but is returned from several `TransferClient` methods. It is an iterable of `GlobusRepsonse` objects.

class globus_sdk.transfer.paging.**PaginatedResource**(*client_method*, *path*, *client_kwargs*, *num_results=None*, *max_results_per_call=1000*, *max_total_results=None*, *offset=0*, *paging_style=0*)

Bases: *globus_sdk.response.GlobusResponse*, `object`

A `PaginatedResource` is an iterable response which implements the Python iterator interface. As such, **you can only iterate over PaginatedResources once**. Future iterations will be empty.

If you need fresh results, make a call for a new `PaginatedResource`, and if you want to cache and reuse results, convert to a list or other structure. You may also want to read the docs on the *data* property.

Because paginated data can be large, you will tend to get the best performance by being sure to only iterate over the results once.

property data

To get the "data" on a PaginatedResource, fetch all pages and convert them into the only python data structure that makes sense: a list.

Note that this forces iteration/evaluation of all pages from the API. It therefore may cause singificant IO spikes when used. You should avoid using the `PaginatedResource.data` property whenever possible.

1.3.3 Globus Auth

There are several types of client object for communicating with the Globus Auth service. A client object may represent your application (as the driver of authentication and authorization flows), in which case the `NativeAppAuthClient` or `ConfidentialAppAuthClient` classes should generally be used.

class globus_sdk.**AuthClient**(*client_id=None*, *authorizer=None*, ***kwargs*)

Bases: *globus_sdk.base.BaseClient*

Client for the Globus Auth API

This class provides helper methods for most common resources in the Auth API, and the common low-level interface from *BaseClient* of `get`, `put`, `post`, and `delete` methods, which can be used to access any API resource.

There are generally two types of resources, distinguished by the type of authentication which they use. Resources available to end users of Globus are authenticated with a Globus Auth Token ("Authentication: Bearer ..."), while resources available to OAuth Clients are authenticated using Basic Auth with the Client's ID and Secret. Some resources may be available with either authentication type.

Examples

Initializing an `AuthClient` to authenticate a user making calls to the Globus Auth service with an access token takes the form

```
>>> from globus_sdk import AuthClient, AccessTokenAuthorizer
>>> ac = AuthClient(authorizer=AccessTokenAuthorizer('<token_string>'))
```

You can, of course, use other kinds of Authorizers (notably the `RefreshTokenAuthorizer`).

Methods

- *get_identities()*
- *oauth2_exchange_code_for_tokens()*
- *oauth2_get_authorize_url()*
- *oauth2_refresh_token()*
- *oauth2_revoke_token()*
- *oauth2_token()*
- *oauth2_userinfo()*
- *oauth2_validate_token()*

get_identities(*usernames=None*, *ids=None*, *provision=False*, ***params*)
GET /v2/api/identities

Given `usernames=<U>` or (exclusive) `ids=<I>` as keyword arguments, looks up identity information for the set of identities provided. `<U>` and `<I>` in this case are comma-delimited strings listing multiple Identity Usernames or Identity IDs, or iterables of strings, each of which is an Identity Username or Identity ID.

If Globus Auth's identity auto-provisioning behavior is desired, `provision=True` may be specified.

Available with any authentication/client type.

Examples

```
>>> ac = globus_sdk.AuthClient(...)
>>> # by IDs
>>> r = ac.get_identities(ids="46bd0f56-e24f-11e5-a510-131bef46955c")
>>> r.data
{u'identities': [{u'email': None,
   u'id': u'46bd0f56-e24f-11e5-a510-131bef46955c',
   u'identity_provider': u'7daddf46-70c5-45ee-9f0f-7244fe7c8707',
   u'name': None,
   u'organization': None,
   u'status': u'unused',
   u'username': u'globus@globus.org'}]}
>>> ac.get_identities(
>>>     ids=",".join(
>>>         ("46bd0f56-e24f-11e5-a510-131bef46955c",
>>>          "168edc3d-c6ba-478c-9cf8-541ff5ebdc1c"))
...
>>> # or by usernames
>>> ac.get_identities(usernames='globus@globus.org')
...
>>> ac.get_identities(
>>>     usernames='globus@globus.org,auth@globus.org')
...
```

You could also use iterables:

```
>>> ac.get_identities(
>>>     usernames=['globus@globus.org', 'auth@globus.org'])
...
>>> ac.get_identities(
```

(continues on next page)

(continued from previous page)

```
>>>     ids=["46bd0f56-e24f-11e5-a510-131bef46955c",
>>>          "168edc3d-c6ba-478c-9cf8-541ff5ebdc1c"])
...
```

External Documentation

See Identities Resources in the API documentation for details.

oauth2_get_authorize_url(*additional_params=None*)

Get the authorization URL to which users should be sent. This method may only be called after `oauth2_start_flow` has been called on this `AuthClient`.

Parameters **additional_params** (*dict, optional*) – Additional query parameters to include in the authorize URL. Primarily for internal use

Return type `string`

oauth2_exchange_code_for_tokens(*auth_code*)

Exchange an authorization code for a token or tokens.

Return type *OAuthTokenResponse*

Parameters **auth_code** (*str*) – An auth code typically obtained by sending the user to the authorize URL. The code is a very short-lived credential which this method is exchanging for tokens. Tokens are the credentials used to authenticate against Globus APIs.

oauth2_refresh_token(*refresh_token*, *additional_params=None*)

Exchange a refresh token for a *OAuthTokenResponse*, containing an access token.

Does a token call of the form

```
refresh_token=<refresh_token>
grant_type=refresh_token
```

plus any additional parameters you may specify.

Parameters

- **refresh_token** (*str*) – A Globus Refresh Token as a string
- **additional_params** (*dict, optional*) – A dict of extra params to encode in the refresh call.

oauth2_validate_token(*token*, *additional_params=None*)

Validate a token. It can be an Access Token or a Refresh token.

This call can be used to check tokens issued to your client, confirming that they are or are not still valid. The resulting response has the form `{"active": True}` when the token is valid, and `{"active": False}` when it is not.

It is not necessary to validate tokens immediately after receiving them from the service – any tokens which you are issued will be valid at that time. This is more for the purpose of doing checks like

- confirm that `oauth2_revoke_token` succeeded
- at application boot, confirm no need to do fresh login

Parameters

- **token** (*str*) – The token which should be validated. Can be a refresh token or an access token

- **additional_params** (*dict, optional*) – Additional parameters to include in the validation body. Primarily for internal use

Examples

Revoke a token and confirm that it is no longer active:

```
>>> from globus_sdk import ConfidentialAppAuthClient
>>> ac = ConfidentialAppAuthClient(CLIENT_ID, CLIENT_SECRET)
>>> ac.oauth2_revoke_token('<token_string>')
>>> data = ac.oauth2_validate_token('<token_string>')
>>> assert not data['active']
```

During application boot, check if the user needs to do a login, even if a token is present:

```
>>> from globus_sdk import ConfidentialAppAuthClient
>>> ac = ConfidentialAppAuthClient(CLIENT_ID, CLIENT_SECRET)
>>> # this is not an SDK function, but a hypothetical function which
>>> # you use to load a token out of configuration data
>>> tok = load_token_from_config(...)
>>>
>>> if not tok or not ac.oauth2_validate_token(tok)['active']:
>>>     # do_new_login() is another hypothetical helper
>>>     tok = do_new_login()
>>> # at this point, tok is expected to be a valid token
```

oauth2_revoke_token(*token*, *additional_params=None*)

Revoke a token. It can be an Access Token or a Refresh token.

This call should be used to revoke tokens issued to your client, rendering them inert and not further usable. Typically, this is incorporated into "logout" functionality, but it should also be used if the client detects that its tokens are in an unsafe location (e.x. found in a world-readable logfile).

You can check the "active" status of the token after revocation if you want to confirm that it was revoked.

Parameters

- **token** (*str*) – The token which should be revoked
- **additional_params** – Additional parameters to include in the revocation body, which can help speed the revocation process. Primarily for internal use

Examples

```
>>> from globus_sdk import ConfidentialAppAuthClient
>>> ac = ConfidentialAppAuthClient(CLIENT_ID, CLIENT_SECRET)
>>> ac.oauth2_revoke_token('<token_string>')
```

oauth2_token(*form_data*, *response_class=<class 'globus_sdk.auth.token_response.OAuthTokenResponse'>*)

This is the generic form of calling the OAuth2 Token endpoint. It takes `form_data`, a dict which will be encoded in a form POST body on the request.

Generally, users of the SDK should not call this method unless they are implementing OAuth2 flows.

Parameters **response_class** (*class, optional*) – This is used by calls to the oauth2_token endpoint which need to specialize their responses. For example, *oauth2_get_dependent_tokens* requires a specialize response class to handle the dramatically different format of the Dependent Token Grant response

Return type response_class

oauth2_userinfo()

Call the Userinfo endpoint of Globus Auth. Userinfo is specified as part of the OpenID Connect (OIDC) standard, and Globus Auth's Userinfo is OIDC-compliant.

The exact data returned will depend upon the set of OIDC-related scopes which were used to acquire the token being used for this call. For details, see the **External Documentation** below.

Examples

```
>>> ac = AuthClient(...)
>>> info = ac.oauth2_userinfo()
>>> print('Effective Identity "{}" has Full Name "{}" and Email "{}"'
>>>       .format(info["sub"], info["name"], info["email"]))
```

External Documentation

See Userinfo in the API documentation for details.

class globus_sdk.**NativeAppAuthClient**(*client_id*, ***kwargs*)

Bases: globus_sdk.auth.client_types.base.AuthClient

This type of AuthClient is used to represent a Native App's communications with Globus Auth. It requires a Client ID, and cannot take an authorizer.

Native Apps are applications, like the Globus CLI, which are run client-side and therefore cannot keep secrets. Unable to possess client credentials, several Globus Auth interactions have to be specialized to accommodate the absence of a secret.

Any keyword arguments given are passed through to the AuthClient constructor.

Methods

- *oauth2_refresh_token()*
- *oauth2_start_flow()*

oauth2_start_flow(*requested_scopes=None*, *redirect_uri=None*, *state='_default'*, *verifier=None*, *refresh_tokens=False*, *prefill_named_grant=None*)

Starts a Native App OAuth2 flow.

This is done internally by instantiating a *GlobusNativeAppFlowManager*

While the flow is in progress, the NativeAppAuthClient becomes non thread-safe as temporary state is stored during the flow.

Parameters

- **requested_scopes** (*str or iterable of str, optional*) – The scopes on the token(s) being requested, as a space-separated string or iterable of strings. Defaults to openid profile email urn:globus:auth:scope:transfer.api.globus.org:all
- **redirect_uri** – The page that users should be directed to after authenticating at the authorize URL. Defaults to 'https://auth.globus.org/v2/web/auth-code', which displays the resulting auth_code for users to copy-paste back into your application (and thereby be passed back to the GlobusNativeAppFlowManager)
- **state** (*str, optional*) – The redirect_uri page will have this included in a query parameter, so you can use it to pass information to that page if you use a custom page. It defaults to the string '_default'
- **verifier** (*str, optional*) – A secret used for the Native App flow. It will by default be a freshly generated random string, known only to this GlobusNativeAppFlowManager instance

- **refresh_tokens** (*bool, optional*) – When True, request refresh tokens in addition to access tokens. [Default: `False`]
- **prefill_named_grant** (*str, optional*) – Prefill the named grant label on the consent page

Examples

You can see an example of this flow *in the usage examples*

External Documentation

The Globus Auth specification for Native App grants details the modifications to the Authorization Code grant flow as The PKCE Security Protocol

oauth2_refresh_token(*refresh_token*)

`NativeAppAuthClient` specializes the refresh token grant to include its client ID as a parameter in the POST body. It needs this specialization because it cannot authenticate the refresh grant call with client credentials, as is normal.

class globus_sdk.**ConfidentialAppAuthClient**(*client_id*, *client_secret*, ***kwargs*)

Bases: `globus_sdk.auth.client_types.base.AuthClient`

This is a specialized type of `AuthClient` used to represent an App with a Client ID and Client Secret wishing to communicate with Globus Auth. It must be given a Client ID and a Client Secret, and furthermore, these will be used to establish a `BasicAuthorizer <globus_sdk.authorizers.BasicAuthorizer` for authorization purposes. Additionally, the Client ID is stored for use in various calls.

Confidential Applications (i.e. Applications with are not Native Apps) are those like the Sample Data Portal, which have their own credentials for authenticating against Globus Auth.

Any keyword arguments given are passed through to the `AuthClient` constructor.

Methods

- *oauth2_client_credentials_tokens()*
- *oauth2_get_dependent_tokens()*
- *oauth2_start_flow()*
- *oauth2_token_introspect()*

oauth2_client_credentials_tokens(*requested_scopes=None*)

Perform an OAuth2 Client Credentials Grant to get access tokens which directly represent your client and allow it to act on its own (independent of any user authorization). This method does not use a `GlobusOAuthFlowManager` because it is not at all necessary to do so.

Parameters **requested_scopes** (*str, optional*) – Space-separated scope names being requested for the access token(s). Defaults to a set of commonly desired scopes for Globus.

Return type *OAuthTokenResponse*

For example, with a Client ID of "CID1001" and a Client Secret of "RAND2002", you could use this grant type like so:

```
>>> client = ConfidentialAppAuthClient("CID1001", "RAND2002")
>>> tokens = client.oauth2_client_credentials_tokens()
>>> transfer_token_info = (
...     tokens.by_resource_server["transfer.api.globus.org"])
>>> transfer_token = transfer_token_info["access_token"]
```

oauth2_start_flow(*redirect_uri*, *requested_scopes=None*, *state='_default'*, *refresh_tokens=False*)
Starts or resumes an Authorization Code OAuth2 flow.

Under the hood, this is done by instantiating a *GlobusAuthorizationCodeFlowManager*

Parameters

- **redirect_uri** (str `redirect_uri` (*string*)) – The page that users should be directed to after authenticating at the authorize URL.
- **requested_scopes** (*str or iterable of str, optional*) – The scopes on the token(s) being requested, as a space-separated string or an iterable of strings. Defaults to `openid profile email urn:globus:auth:scope:transfer.api.globus.org:all`
- **state** (*str, optional*) – This string allows an application to pass information back to itself in the course of the OAuth flow. Because the user will navigate away from the application to complete the flow, this parameter lets the app pass an arbitrary string from the starting page to the `redirect_uri`
- **refresh_tokens** (*bool, optional*) – When True, request refresh tokens in addition to access tokens. [Default: `False`]

Examples

You can see an example of this flow *in the usage examples*

External Documentation

The Authorization Code Grant flow is described in the Globus Auth Specification

oauth2_get_dependent_tokens(*token*, *additional_params=None*)
Does a Dependent Token Grant against Globus Auth. This exchanges a token given to this client for a new set of tokens which give it access to resource servers on which it depends. This grant type is intended for use by Resource Servers playing out the following scenario:

1. User has tokens for Service A, but Service A requires access to Service B on behalf of the user
2. Service B should not see tokens scoped for Service A
3. Service A therefore requests tokens scoped only for Service B, based on tokens which were originally scoped for Service A...

In order to do this exchange, the tokens for Service A must have scopes which depend on scopes for Service B (the services' scopes must encode their relationship). As long as that is the case, Service A can use this Grant to get those "Dependent" or "Downstream" tokens for Service B.

Parameters

- **token** (*str*) – A Globus Access Token as a string
- **additional_params** (*dict, optional*) – Additional parameters to include in the request body

Return type *OAuthTokenResponse*

oauth2_token_introspect(*token*, *include=None*)
POST /v2/oauth2/token/introspect

Get information about a Globus Auth token.

```
>>> ac = globus_sdk.ConfidentialAppAuthClient(
...     CLIENT_ID, CLIENT_SECRET)
>>> ac.oauth2_token_introspect('<token_string>')
```

Get information about a Globus Auth token including the full identity set of the user to whom it belongs

```
>>> ac = globus_sdk.ConfidentialAppAuthClient(
...     CLIENT_ID, CLIENT_SECRET)
>>> data = ac.oauth2_token_introspect(
...     '<token_string>', include='identity_set')
>>> for identity in data['identity_set']:
>>>     print('token authenticates for "{}"'.format(identity))
```

Parameters

- **token** (*str*) – An Access Token as a raw string, being evaluated
- **include** (*str, optional*) – A value for the `include` parameter in the request body. Default is to omit the parameter.

External Documentation

See Token Introspection in the API documentation for details.

Helper Objects

The `IdentityMap` is a specialized object which aids in the particular use-case in which the Globus Auth `get_identities` API is being used to resolve large numbers of usernames or IDs. It combines caching, request batching, and other functionality.

class globus_sdk.**IdentityMap**(*auth_client*, *identity_ids=None*, *id_batch_size=None*)

Bases: `object`

There's a common pattern of having a large batch of Globus Auth Identities which you want to inspect. For example, you may have a list of identity IDs fetched from Access Control Lists on Globus Endpoints. In order to display these identities to an end user, you may want to resolve them to usernames.

However, naively looking up the identities one-by-one is very inefficient. It's best to do batched lookups with multiple identities at once. In these cases, an `IdentityMap` can be used to do those batched lookups for you.

An `IdentityMap` is a mapping-like type which converts Identity IDs and Identity Names to Identity records (dictionaries) using the Globus Auth API.

Note: `IdentityMap` objects are not full Mappings in the same sense as python dicts and similar objects. By design, they only implement a small part of the Mapping protocol.

The basic usage pattern is

- create an `IdentityMap` with an AuthClient which will be used to call out to Globus Auth
- seed the `IdentityMap` with IDs and Usernames via *add()* (you can also do this during initialization)
- retrieve identity IDs or Usernames from the map

Because the map can be populated with a collection of identity IDs and Usernames prior to lookups being performed, it can improve the efficiency of these operations up to 100x over individual lookups.

If you attempt to retrieve an identity which has not been previously added to the map, it will be immediately added. But adding many identities beforehand will improve performance.

The `IdentityMap` will cache its results so that repeated lookups of the same Identity will not repeat work. It will also map identities both by ID and by Username, regardless of how they're initially looked up.

Warning: If an Identity is not found in Globus Auth, it will trigger a KeyError when looked up. Your code must be ready to handle KeyErrors when doing a lookup.

Correct usage looks something like so:

```
ac = globus_sdk.AuthClient(...)
idmap = globus_sdk.IdentityMap(
    ac, ["foo@globusid.org", "bar@uchicago.edu"]
)
idmap.add("baz@xsede.org")
# adding by ID is also valid
idmap.add("c699d42e-d274-11e5-bf75-1fc5bf53bb24")
# map ID to username
assert (
    idmap["c699d42e-d274-11e5-bf75-1fc5bf53bb24"]["username"]
    == "go@globusid.org"
)
# map username to ID
assert (
    idmap["go@globusid.org"]["id"]
    == "c699d42e-d274-11e5-bf75-1fc5bf53bb24"
)
```

And simple handling of errors:

```
try:
    record = idmap["no-such-valid-id@example.org"]
except KeyError:
    username = "NO_SUCH_IDENTITY"
else:
    username = record["username"]
```

or you may achieve this by using the *get()* method:

```
# internally handles the KeyError and returns the default value
record = idmap.get("no-such-valid-id@example.org", None)
username = record["username"] if record is not None else "NO_SUCH_IDENTITY"
```

Parameters

- **auth_client** (*AuthClient*) – The client object which will be used for lookups against Globus Auth
- **identity_ids** (*iterable of str*) – A list or other iterable of usernames or identity IDs (potentially mixed together) which will be used to seed the `IdentityMap` 's tracking of unresolved Identities.
- **id_batch_size** (*int, optional*) – A non-default batch size to use when communicating with Globus Auth. Leaving this set to the default is strongly recommended.

Methods

- *__delitem__()*
- *__getitem__()*
- *add()*

- *get()*

__delitem__(*key*)
IdentityMap supports `del map[key]`. Note that this only removes lookup values from the cache and will not impact the set of unresolved/pending IDs.

__getitem__(*key*)
IdentityMap supports dict-like lookups with `map[key]`

__init__(*auth_client*, *identity_ids=None*, *id_batch_size=None*)
Initialize self. See help(type(self)) for accurate signature.

add(*identity_id*)
Add a username or ID to the IdentityMap for batch lookups later.

Returns True if the ID was added for lookup. Returns False if it was rejected as a duplicate of an already known name.

Parameters **identity_id** (*str*) – A string Identity ID or Identity Name (a.k.a. "username") to add

get(*key*, *default=None*)
A dict-like get() method which accepts a default value.

Auth Responses

class globus_sdk.auth.token_response.**OAuthTokenResponse**(**args*, ***kwargs*)
Bases: *globus_sdk.response.GlobusHTTPResponse*

Class for responses from the OAuth2 code for tokens exchange used in 3-legged OAuth flows.

property by_resource_server
Representation of the token response in a dict indexed by resource server.

Although OAuthTokenResponse.data is still available and valid, this representation is typically more desirable for applications doing inspection of access tokens and refresh tokens.

property by_scopes
Representation of the token response in a dict-like object indexed by scope name (or even space delimited scope names, so long as they match the same token).

If you request scopes *scope1 scope2 scope3*, where *scope1* and *scope2* are for the same service (and therefore map to the same token), but *scope3* is for a different service, the following forms of access are valid:

```
>>> tokens = ...
>>> # single scope
>>> token_data = tokens.by_scopes['scope1']
>>> token_data = tokens.by_scopes['scope2']
>>> token_data = tokens.by_scopes['scope3']
>>> # matching scopes
>>> token_data = tokens.by_scopes['scope1 scope2']
>>> token_data = tokens.by_scopes['scope2 scope1']
```

decode_id_token(*auth_client=None*)
A parsed ID Token (OIDC) as a dict.

Parameters **auth_client** (*AuthClient*) – Deprecated parameter for providing the AuthClient used to request this token back to the OAuthTokenResponse. The SDK now tracks this internally, so it is no longer necessary.

class globus_sdk.auth.token_response.**OAuthDependentTokenResponse**(**args*, ***kwargs*)

Bases: *globus_sdk.auth.token_response.OAuthTokenResponse*

Class for responses from the OAuth2 code for tokens retrieved by the OAuth2 Dependent Token Extension Grant. For more complete docs, see *oauth2_get_dependent_tokens*

decode_id_token(*auth_client*)

A parsed ID Token (OIDC) as a dict.

Parameters **auth_client** (*AuthClient*) – Deprecated parameter for providing the AuthClient used to request this token back to the OAuthTokenResponse. The SDK now tracks this internally, so it is no longer necessary.

OAuth2 Flows & Explanation

OAuth2

Globus offers Authentication and Authorization services through an OAuth2 service, Globus Auth.

Globus Auth acts as an Authorization Server, and allows users to authenticate with, and link together, identities from a wide range of Identity Providers.

Although the *AuthClient* class documentation covers normal interactions with Globus Auth, the OAuth2 flows are significantly more complex.

This section documents the supported types of authentication and how to carry them out, as well as providing some necessary background on various OAuth2 elements.

Credentials are for Users and also for Applications

It is very important that our goal in OAuth2 is not to get credentials for an application on its own, but rather for the application as a *client* to Globus which is acting *on behalf of a user*.

Therefore, if you are writing an application called **foo**, and a user **bar@example.com** is using **foo**, the credentials produced belong to the combination of **foo** and **bar@example.com**. The resulting credentials represent the rights and permission for **foo** to perform actions for **bar@example.com** on systems authenticated via Globus.

OAuth2 Flows

If you want to get started doing OAuth2 flows, you should read the *tutorial* and look at the *examples*.

Flow Managers

These objects represent in-progress OAuth2 authentication flows. Most typically, you should not use these objects, but rather rely on the *globus_sdk.AuthClient* object to manage one of these for you through its `oauth2_*` methods.

All Flow Managers inherit from the `GlobusOAuthFlowManager` abstract class. They are a combination of a store for OAuth2 parameters specific to the authentication method you are using and methods which act upon those parameters.

class globus_sdk.auth.**GlobusNativeAppFlowManager**(*auth_client*, *requested_scopes=None*, *redirect_uri=None*, *state='_default'*, *verifier=None*, *refresh_tokens=False*, *prefill_named_grant=None*)

Bases: *globus_sdk.auth.oauth2_flow_manager.GlobusOAuthFlowManager*

This is the OAuth flow designated for use by clients wishing to authenticate users in the absence of a Client Secret. Because these applications run "natively" in the user's environment, they cannot protect a secret. Instead, a temporary secret is generated solely for this authentication attempt.

Parameters

- **auth_client** (*NativeAppAuthClient*) – The `NativeAppAuthClient` object on which this flow is based. It is used to extract default values for the flow, and also to make calls to the Auth service.
- **requested_scopes** (*str or iterable of str, optional*) – The scopes on the token(s) being requested, as a space-separated string or iterable of strings. Defaults to `openid profile email urn:globus:auth:scope:transfer.api.globus.org:all`
- **redirect_uri** (*str, optional*) – The page that users should be directed to after authenticating at the authorize URL. Defaults to 'https://auth.globus.org/v2/web/auth-code', which displays the resulting `auth_code` for users to copy-paste back into your application (and thereby be passed back to the `GlobusNativeAppFlowManager`)
- **state** (*str, optional*) – The `redirect_uri` page will have this included in a query parameter, so you can use it to pass information to that page if you use a custom page. It defaults to the string '_default'
- **verifier** (*str, optional*) – A secret used for the Native App flow. It will by default be a freshly generated random string, known only to this `GlobusNativeAppFlowManager` instance
- **refresh_tokens** (*bool, optional*) – When True, request refresh tokens in addition to access tokens. [Default: `False`]
- **prefill_named_grant** (*str, optional*) – Prefill the named grant label on the consent page

exchange_code_for_tokens(*auth_code*)

The second step of the Native App flow, exchange an authorization code for access tokens (and refresh tokens if specified).

Return type *OAuthTokenResponse*

get_authorize_url(*additional_params=None*)

Start a Native App flow by getting the authorization URL to which users should be sent.

Parameters **additional_params** (*dict, optional*) – Additional query parameters to include in the authorize URL. Primarily for internal use

Return type `string`

The returned URL string is encoded to be suitable to display to users in a link or to copy into their browser. Users will be redirected either to your provided `redirect_uri` or to the default location, with the `auth_code` embedded in a query parameter.

class globus_sdk.auth.**GlobusAuthorizationCodeFlowManager**(*auth_client*, *redirect_uri*, *requested_scopes=None*, *state='_default'*, *refresh_tokens=False*)

Bases: *globus_sdk.auth.oauth2_flow_manager.GlobusOAuthFlowManager*

This is the OAuth flow designated for use by Clients wishing to authenticate users in a web application backed by a server-side component (e.g. an API). The key constraint is that there is a server-side system that can keep a Client Secret without exposing it to the web client. For example, a Django application can rely on the webserver to own the secret, so long as it doesn't embed it in any of the pages it generates.

The application sends the user to get a temporary credential (an `auth_code`) associated with its Client ID. It then exchanges that temporary credential for a token, protecting the exchange with its Client Secret (to prove that it really is the application that the user just authorized).

Parameters

- **auth_client** (*ConfidentialAppAuthClient*) – The `AuthClient` used to extract default values for the flow, and also to make calls to the Auth service.
- **redirect_uri** (*str*) – The page that users should be directed to after authenticating at the authorize URL.
- **requested_scopes** (*str or iterable of str, optional*) – The scopes on the token(s) being requested, as a space-separated string or iterable of strings. Defaults to `openid profile email urn:globus:auth:scope:transfer.api.globus.org:all`
- **state** (*str, optional*) – This string allows an application to pass information back to itself in the course of the OAuth flow. Because the user will navigate away from the application to complete the flow, this parameter lets the app pass an arbitrary string from the starting page to the `redirect_uri`
- **refresh_tokens** (*bool, optional*) – When True, request refresh tokens in addition to access tokens. [Default: `False`]

exchange_code_for_tokens(*auth_code*)

The second step of the Authorization Code flow, exchange an authorization code for access tokens (and refresh tokens if specified)

Return type *OAuthTokenResponse*

get_authorize_url(*additional_params=None*)

Start a Authorization Code flow by getting the authorization URL to which users should be sent.

Parameters **additional_params** (*dict, optional*) – Additional parameters to include in the authorize URL. Primarily for internal use

Return type `string`

The returned URL string is encoded to be suitable to display to users in a link or to copy into their browser. Users will be redirected either to your provided `redirect_uri` or to the default location, with the `auth_code` embedded in a query parameter.

Abstract Flow Manager

class globus_sdk.auth.oauth2_flow_manager.**GlobusOAuthFlowManager**

Bases: `object`

An abstract class definition that defines the interface for the Flow Managers for Globus Auth. Flow Managers are really just bundles of parameters to Globus Auth's OAuth2 mechanisms, along with some useful utility methods. Primarily they can be used as a simple way of tracking small amounts of state in your application as it leverages Globus Auth for authentication.

For sophisticated use cases, the provided Flow Managers will *NOT* be sufficient, but you should consider the provided objects a model.

This way of managing OAuth2 flows is inspired by oauth2client. However, because `oauth2client` has an uncertain future (as of 2016-08-31), and we would have to wrap it in order to provide a clean API surface anyway, we implement our own set of Flow objects.

exchange_code_for_tokens(*auth_code*)

This method takes an auth_code and produces a response object containing one or more tokens. Most typically, this is the second step of the flow, and consumes the auth_code that was sent to a redirect URI used in the authorize step.

The exchange process may be parameterized over attributes of the specific flow manager instance which is generating it.

Parameters **auth_code** (*str*) – The authorization code which was produced from the authorization flow

Return type `OAuthTokenResponse`

get_authorize_url()

This method consumes no arguments or keyword arguments, and produces a string URL for the Authorize Step of a 3-legged OAuth2 flow. Most typically, this is the first step of the flow, and the user may be redirected to the URL or provided with a link.

The authorize_url may be (usually is) parameterized over attributes of the specific flow manager instance which is generating it.

Return type `string`

Resource Servers and Scopes

What are Resource Servers, and how do they interact with scopes?

If you look at a *OAuthTokenResponse*, you will notice that it organizes information under Resource Servers, including one access token (and optionally one refresh token) per Resource Server. This can appear confusing, especially as the Resource Servers in this response do not map one-to-one onto the scopes that your application requested.

This is a brief description Resource Servers to make sense of this response.

Short Version

Resource Servers are just the OAuth2 name for services which use scopes on tokens to control access to their resources.

Resource Servers may have multiple scopes

When you request tokens, you do so with a set of scopes. Our default set consists of `openid profile email urn:globus:auth:scope:transfer.api.globus.org:all`. That means you can get OpenID Connect data in general, profile data, email address, and access to Globus Transfer resources (in that order).

However, for those four scopes, there aren't four distinct services – there are only two. `openid`, `profile`, and `email` all correspond to the service at `auth.globus.org` (Globus Auth) while `urn:globus:auth:scope:transfer.api.globus.org:all` corresponds to `transfer.api.globus.org` (Globus Transfer).

As a result, we don't get four tokens for our four scopes – we get two tokens, one for the first three scopes, and one for the last scope. Those tokens can be organized better by their relevant Resource Server than by their scope names, which is why we use the `token_response.by_resource_server` description.

Why Not Just One Token?

The reason for separate tokens at all (as opposed to one token with all four scopes) is to limit the exposure of tokens for different services.

As a motivating example, consider a new service registered as Resource Server in Globus belonging to another organization – `serv.example.com`. `serv.example.com` should not see tokens scoped for Globus Transfer, and Globus Transfer shouldn't see tokens scoped for `serv.example.com`.

Using a single token for all Resource Servers would make isolating services in this way impossible.

1.3.4 Globus Search

class globus_sdk.**SearchClient**(*authorizer=None*, ***kwargs*)

Bases: *globus_sdk.base.BaseClient*

Client for the Globus Search API

This class provides helper methods for most common resources in the API, and basic `get`, `put`, `post`, and `delete` methods from the base client that can be used to access any API resource.

Parameters **authorizer** (*GlobusAuthorizer*) – An authorizer instance used for all calls to Globus Search

Methods

Methods

- *create_entry()*
- *delete_by_query()*
- *delete_entry()*
- *delete_subject()*
- *get_entry()*
- *get_index()*
- *get_query_template()*
- *get_query_template_list()*
- *get_subject()*
- *get_task()*
- *get_task_list()*
- *ingest()*
- *post_search()*
- *search()*
- *update_entry()*

get_index(*index_id*, ***params*)
GET /v1/index/<index_id>

Examples

```
>>> sc = globus_sdk.SearchClient(...)
>>> index = sc.get_index(index_id)
>>> assert index['index_id'] == index_id
>>> print(index["display_name"],
>>>       "(" + index_id + "):",
>>>       index["description"])
```

External Documentation

See Get Index Metadata in the API documentation for details.

search(*index_id*, *q*, *offset=0*, *limit=10*, *query_template=None*, *advanced=False*, ***params*)
GET /v1/index/<index_id>/search

Examples

```
>>> sc = globus_sdk.SearchClient(...)
>>> result = sc.search(index_id, 'query string')
>>> advanced_result = sc.search(index_id, 'author: "Ada Lovelace"',
>>>                             advanced=True)
```

External Documentation

See GET Search Query in the API documentation for details.

post_search(*index_id*, *data*)
POST /v1/index/<index_id>/search

Examples

```
>>> sc = globus_sdk.SearchClient(...)
>>> query_data = {
>>>   "@datatype": "GSearchRequest",
>>>   "q": "user query",
>>>   "filters": [
>>>     {
>>>       "type": "range",
>>>       "field_name": "path.to.date",
>>>       "values": [
>>>         {"from": "*",
>>>          "to": "2014-11-07"}
>>>       ]
>>>     }
>>>   ],
>>>   "facets": [
>>>     {"name": "Publication Date",
>>>      "field_name": "path.to.date",
>>>      "type": "date_histogram",
>>>      "date_interval": "year"}
>>>   ],
>>>   "sort": [
>>>     {"field_name": "path.to.date",
>>>      "order": "asc"}
>>>   ]
>>> }
>>> search_result = sc.post_search(index_id, query_data)
```

External Documentation

See POST Search Query in the API documentation for details.

ingest(*index_id*, *data*)

`POST /v1/index/<index_id>/ingest`

Examples

```
>>> sc = globus_sdk.SearchClient(...)
>>> ingest_data = {
>>>   "ingest_type": "GMetaEntry",
>>>   "ingest_data": {
>>>     "subject": "https://example.com/foo/bar",
>>>     "visible_to": ["public"],
>>>     "content": {
>>>       "foo/bar": "some val"
>>>     }
>>>   }
>>> }
>>> sc.ingest(index_id, ingest_data)
```

or with multiple entries at once via a GMetaList:

```
>>> sc = globus_sdk.SearchClient(...)
>>> ingest_data = {
>>>   "ingest_type": "GMetaList",
>>>   "ingest_data": {
>>>     "gmeta": [
>>>       {
>>>         "subject": "https://example.com/foo/bar",
>>>         "visible_to": ["public"],
>>>         "content": {
>>>           "foo/bar": "some val"
>>>         }
>>>       },
>>>       {
>>>         "subject": "https://example.com/foo/bar",
>>>         "id": "otherentry",
>>>         "visible_to": ["public"],
>>>         "content": {
>>>           "foo/bar": "some otherval"
>>>         }
>>>       }
>>>     ]
>>>   }
>>> }
>>> sc.ingest(index_id, ingest_data)
```

External Documentation

See Ingest in the API documentation for details.

delete_by_query(*index_id*, *data*)

`POST /v1/index/<index_id>/delete_by_query`

Examples

```
>>> sc = globus_sdk.SearchClient(...)
>>> query_data = {
```

(continues on next page)

(continued from previous page)

```
>>>   "q": "user query",
>>>   "filters": [
>>>     {
>>>       "type": "range",
>>>       "field_name": "path.to.date",
>>>       "values": [
>>>         {"from": "*",
>>>          "to": "2014-11-07"}
>>>       ]
>>>     }
>>>   ]
>>> }
>>> sc.delete_by_query(index_id, query_data)
```

External Documentation

See Delete By Query in the API documentation for details.

get_subject (*index_id*, *subject*, ***params*)
`GET /v1/index/<index_id>/subject`

Examples

Fetch the data for subject `http://example.com/abc` from index `index_id`:

```
>>> sc = globus_sdk.SearchClient(...)
>>> subject_data = sc.get_subject(index_id, 'http://example.com/abc')
```

External Documentation

See Get Subject in the API documentation for details.

delete_subject (*index_id*, *subject*, ***params*)
`DELETE /v1/index/<index_id>/subject`

Examples

Delete all data for subject `http://example.com/abc` from index `index_id`, even data which is not visible to the current user:

```
>>> sc = globus_sdk.SearchClient(...)
>>> subject_data = sc.get_subject(index_id, 'http://example.com/abc')
```

External Documentation

See Delete Subject in the API documentation for details.

get_entry (*index_id*, *subject*, *entry_id=None*, ***params*)
`GET /v1/index/<index_id>/entry`

Examples

Lookup the entry with a subject of `https://example.com/foo/bar` and a null entry_id:

```
>>> sc = globus_sdk.SearchClient(...)
>>> entry_data = sc.get_entry(index_id, 'http://example.com/foo/bar')
```

Lookup the entry with a subject of `https://example.com/foo/bar` and an entry_id of `foo/bar`:

```
>>> sc = globus_sdk.SearchClient(...)
>>> entry_data = sc.get_entry(index_id, 'http://example.com/foo/bar',
>>>                           entry_id='foo/bar')
```

External Documentation

See Get Entry in the API documentation for details.

create_entry(*index_id*, *data*)

`POST /v1/index/<index_id>/entry`

Examples

Create an entry with a subject of `https://example.com/foo/bar` and a null entry_id:

```
>>> sc = globus_sdk.SearchClient(...)
>>> sc.create_entry(index_id, {
>>>     "subject": "https://example.com/foo/bar",
>>>     "visible_to": ["public"],
>>>     "content": {
>>>         "foo/bar": "some val"
>>>     }
>>> })
```

Create an entry with a subject of `https://example.com/foo/bar` and an entry_id of `foo/bar`:

```
>>> sc = globus_sdk.SearchClient(...)
>>> sc.create_entry(index_id, {
>>>     "subject": "https://example.com/foo/bar",
>>>     "visible_to": ["public"],
>>>     "id": "foo/bar",
>>>     "content": {
>>>         "foo/bar": "some val"
>>>     }
>>> })
```

External Documentation

See Create Entry in the API documentation for details.

update_entry(*index_id*, *data*)

`PUT /v1/index/<index_id>/entry`

Examples

Update an entry with a subject of `https://example.com/foo/bar` and a null entry_id:

```
>>> sc = globus_sdk.SearchClient(...)
>>> sc.update_entry(index_id, {
>>>     "subject": "https://example.com/foo/bar",
>>>     "visible_to": ["public"],
>>>     "content": {
>>>         "foo/bar": "some val"
>>>     }
>>> })
```

External Documentation

See Update Entry in the API documentation for details.

delete_entry(*index_id*, *subject*, *entry_id=None*, ***params*)
DELETE /v1/index/<index_id>/entry

Examples

Delete an entry with a subject of `https://example.com/foo/bar` and a null entry_id:

```
>>> sc = globus_sdk.SearchClient(...)
>>> sc.delete_entry(index_id, "https://example.com/foo/bar")
```

Delete an entry with a subject of `https://example.com/foo/bar` and an entry_id of "foo/bar":

```
>>> sc = globus_sdk.SearchClient(...)
>>> sc.delete_entry(index_id, "https://example.com/foo/bar",
>>>                 entry_id="foo/bar")
```

External Documentation

See Delete Entry in the API documentation for details.

get_query_template(*index_id*, *template_name*)
GET /v1/index/<index_id>/query_template/<template_name>

External Documentation

See Get Query Template in the API documentation for details.

get_query_template_list(*index_id*)
GET /v1/index/<index_id>/query_template

External Documentation

See Get Query Template List in the API documentation for details.

get_task(*task_id*, ***params*)
GET /v1/task/<task_id>

Examples

```
>>> sc = globus_sdk.SearchClient(...)
>>> task = sc.get_task(task_id)
>>> assert task['index_id'] == known_index_id
>>> print(task["task_id"] + " | " + task['state'])
```

get_task_list(*index_id*, ***params*)
GET /v1/task_list/<index_id>

Examples

```
>>> sc = globus_sdk.SearchClient(...)
>>> task_list = sc.get_task_list(index_id)
>>> for task in task_list['tasks']:
>>>     print(task["task_id"] + " | " + task['state'])
```

Helper Objects

class globus_sdk.**SearchQuery**
Bases: dict

A specialized dict which has helpers for creating and modifying a Search Query document.

Example usage:

```
>>> from globus_sdk import SearchClient, SearchQuery
>>> sc = SearchClient(...)
>>> index_id = ...
>>> query = (SearchQuery(q='example query')
>>>          .set_limit(100).set_offset(10)
>>>          .add_filter('path.to.field1', ['foo', 'bar']))
>>> result = sc.post_search(index_id, query)
```

Client Errors

When an error occurs, a SearchClient will raise this specialized type of error, rather than a generic GlobusAPIError.

class globus_sdk.exc.**SearchAPIError**(*r*)
Bases: globus_sdk.exc.GlobusAPIError

Error class for the Search API client. In addition to the inherited code and message instance variables, provides:

Variables **error_data** – Additional object returned in the error response. May be a dict, list, or None.

1.4 Exceptions

All Globus SDK errors inherit from GlobusError, and all SDK error classes are importable from globus_sdk.

You can therefore capture *all* errors thrown by the SDK by looking for GlobusError, as in:

```
import logging
from globus_sdk import TransferClient, GlobusError

try:
    tc = TransferClient(...)
    # search with no parameters will throw an exception
    eps = tc.endpoint_search()
except GlobusError:
    logging.exception("Globus Error!")
    raise
```

In most cases, it's best to look for specific subclasses of GlobusError. For example, to write code which is distinguishes between network failures and unexpected API conditions, you'll want to look for NetworkError and GlobusAPIError:

```
import logging
from globus_sdk import (TransferClient,
                        GlobusError, GlobusAPIError, NetworkError)
```

(continues on next page)

(continued from previous page)

```
try:
    tc = TransferClient(...)

    eps = tc.endpoint_search(filter_fulltext="myendpointsearch")

    for ep in eps:
        print(ep["display_name"])

    ...
except GlobusAPIError as e:
    # Error response from the REST service, check the code and message for
    # details.
    logging.error(("Got a Globus API Error\n"
                   "Error Code: {}\n"
                   "Error Message: {}").format(e.code, e.message))
    raise e
except NetworkError:
    logging.error(("Network Failure. "
                   "Possibly a firewall or connectivity issue"))
    raise
except GlobusError:
    logging.exception("Totally unexpected GlobusError!")
    raise
else:
    ...
```

Of course, if you want to learn more information about the response, you should inspect it more than this.

All errors raised by the SDK should be instances of `GlobusError`. Malformed calls to Globus SDK methods typically raise `GlobusSDKUsageError`, but, in rare cases, may raise standard python exceptions (`ValueError`, `OSError`, etc.)

1.4.1 Error Classes

class globus_sdk.**GlobusError**
Bases: `Exception`

Root of the Globus Exception hierarchy. Stub class.

class globus_sdk.**GlobusSDKUsageError**
Bases: `globus_sdk.exc.GlobusError`, `ValueError`

A `GlobusSDKUsageError` may be thrown in cases in which the SDK detects that it is being used improperly.

These errors typically indicate that some contract regarding SDK usage (e.g. required order of operations) has been violated.

class globus_sdk.**GlobusAPIError**(*r*, **args*, ***kw*)
Bases: `globus_sdk.exc.GlobusError`

Wraps errors returned by a REST API.

Variables

- **http_status** – HTTP status code (int)
- **code** – Error code from the API (str), or "Error" for unclassified errors

- **message** – Error message from the API. In general, this will be more useful to developers, but there may be cases where it's suitable for display to end users.

property raw_json
Get the verbatim error message received from a Globus API, interpreted as a JSON string and evaluated as a *dict*

If the body cannot be loaded as JSON, this is None

property raw_text
Get the verbatim error message receved from a Globus API as a *string*

class globus_sdk.**NetworkError**(*msg*, *exc*, **args*, ***kw*)
Bases: globus_sdk.exc.GlobusError

Error communicating with the REST API server.

Holds onto original exception data, but also takes a message to explain potentially confusing or inconsistent exceptions passed to us

class globus_sdk.**GlobusConnectionError**(*msg*, *exc*, **args*, ***kw*)
Bases: globus_sdk.exc.NetworkError

A connection error occured while making a REST request.

class globus_sdk.**GlobusTimeoutError**(*msg*, *exc*, **args*, ***kw*)
Bases: globus_sdk.exc.NetworkError

The REST request timed out.

class globus_sdk.**GlobusConnectionTimeoutError**(*msg*, *exc*, **args*, ***kw*)
Bases: globus_sdk.exc.GlobusTimeoutError

The request timed out during connection establishment. These errors are safe to retry.

1.5 Local Endpoints

Unlike SDK functionality for accessing Globus APIs, the locally available Globus Endpoints require special treatment. These accesses are not authenticated via Globus Auth, and may rely upon the state of the local filesystem, running processes, and the permissions of local users.

1.5.1 Globus Connect Server

There are no SDK methods for accessing an installation of Globus Connect Server.

1.5.2 Globus Connect Personal

Globus Connect Personal endpoints belonging to the current user may be accessed via instances of the following class:

class globus_sdk.**LocalGlobusConnectPersonal**
A LocalGlobusConnectPersonal object represents the available SDK methods for inspecting and controlling a running Globus Connect Personal installation.

These objects do *not* inherit from BaseClient and do not provide methods for interacting with any Globus Service APIs.

property endpoint_id

Type string

The endpoint ID of the local Globus Connect Personal endpoint installation.

This value is loaded whenever it is first accessed, but saved after that.

Usage:

```
>>> from globus_sdk import TransferClient, LocalGlobusConnectPersonal
>>> local_ep = LocalGlobusConnectPersonal()
>>> ep_id = local_ep.endpoint_id
>>> tc = TransferClient(...)  # needs auth details
>>> for f in tc.operation_ls(ep_id):
>>>     print("Local file: ", f["name"])
```

You can also reset the value, causing it to load again on next access, with `del local_ep.endpoint_id`

1.6 API Authorization

Authorizing calls against Globus can be a complex process. In particular, if you are using Refresh Tokens and short-lived Access Tokens, you may need to take particular care managing your Authorization state.

Within the SDK, we solve this problem by using *GlobusAuthorizers*, which are attached to clients. These are a very simple class of generic objects which define a way of getting an up-to-date `Authorization` header, and trying to handle a 401 (if that header is expired).

Whenever using the *Service Clients*, you should be passing in an authorizer when you create a new client unless otherwise specified.

The type of authorizer you will use depends very much on your application, but if you want examples you should look at the *examples section*. It may help to start with the examples and come back to the full documentation afterwards.

1.6.1 The Authorizer Interface

We define the interface for `GlobusAuthorizer` objects in terms of an Abstract Base Class:

class globus_sdk.authorizers.base.**GlobusAuthorizer**

A `GlobusAuthorizer` is a very simple object which generates valid Authorization headers. It may also have handling for responses that indicate that it has provided an invalid Authorization header.

abstract set_authorization_header()

Takes a dict of headers, and adds to it a mapping of `{"Authorization": "..."}` per this object's type of Authorization. Importantly, if an `Authorization` header is already set, this method is expected to overwrite it.

handle_missing_authorization(**args*, ***kwargs*)

This operation should be called if a request is made with an Authorization header generated by this object which returns a 401 (HTTP Unauthorized). If the `GlobusAuthorizer` thinks that it can take some action to remedy this, it should update its state and return `True`. If the Authorizer cannot do anything in the event of a 401, this *may* update state, but importantly returns `False`.

By default, this always returns `False` and takes no other action.

`GlobusAuthorizer` objects that fetch new access tokens when their existing ones expire or a 401 is received implement the RenewingAuthorizer class

class globus_sdk.authorizers.renewing.**RenewingAuthorizer**(*access_token=None*, *expires_at=None*, *on_refresh=None*)

Bases: *globus_sdk.authorizers.base.GlobusAuthorizer*

A `RenewingAuthorizer` is an abstract superclass to any authorizer that needs to get new Access Tokens in order to form Authorization headers.

It may be passed an initial Access Token, but if so must also be passed an expires_at value for that token.

It provides methods that handle the logic for checking and adjusting expiration time, callbacks on renewal, and 401 handling.

To make an authorizer that implements this class implement the _get_token_response and _extract_token_data methods for that authorization type,

Parameters

- **access_token** (*str, optional*) – Initial Access Token to use, only used if `expires_at` is also set
- **expires_at** (*int, optional*) – Expiration time for the starting `access_token` expressed as a POSIX timestamp (i.e. seconds since the epoch)
- **on_refresh** (*callable, optiona*) – A callback which is triggered any time this authorizer fetches a new access_token. The `on_refresh` callable is invoked on the *OAuthTokenResponse* object resulting from the token being refreshed. It should take only one argument, the token response object. This is useful for implementing storage for Access Tokens, as the `on_refresh` callback can be used to update the Access Tokens and their expiration times.

set_authorization_header(*header_dict*)

Checks to see if a new access token is needed. Once that's done, sets the `Authorization` header to "Bearer <access_token>"

handle_missing_authorization(**args*, ***kwargs*)

The renewing authorizer can respond to a service 401 by immediately invalidating its current Access Token. When this happens, the next call to `set_authorization_header()` will result in a new Access Token being fetched.

1.6.2 Authorizer Types

All of these types of authorizers can be imported from `globus_sdk.authorizers`.

class globus_sdk.**NullAuthorizer**

Bases: *globus_sdk.authorizers.base.GlobusAuthorizer*

This Authorizer implements No Authentication – as in, it ensures that there is no Authorization header.

set_authorization_header(*header_dict*)

Removes the Authorization header from the given header dict if one was present.

class globus_sdk.**BasicAuthorizer**(*username*, *password*)

Bases: *globus_sdk.authorizers.base.GlobusAuthorizer*

This Authorizer implements Basic Authentication. Given a "username" and "password", they are sent base64 encoded in the header.

Parameters

- **username** (*str*) – Username component for Basic Auth

- **password** (*str*) – Password component for Basic Auth

set_authorization_header(*header_dict*)
Sets the `Authorization` header to "Basic <base64 encoded username:password>"

class globus_sdk.**AccessTokenAuthorizer**(*access_token*)
Bases: *globus_sdk.authorizers.base.GlobusAuthorizer*

Implements Authorization using a single Access Token with no Refresh Tokens. This is sent as a Bearer token in the header – basically unadorned.

Parameters **access_token** (*str*) – An access token for Globus Auth

set_authorization_header(*header_dict*)
Sets the `Authorization` header to "Bearer <access_token>"

class globus_sdk.**RefreshTokenAuthorizer**(*refresh_token*, *auth_client*, *access_token=None*, *expires_at=None*, *on_refresh=None*)
Bases: *globus_sdk.authorizers.renewing.RenewingAuthorizer*

Implements Authorization using a Refresh Token to periodically fetch renewed Access Tokens. It may be initialized with an Access Token, or it will fetch one the first time that `set_authorization_header()` is called.

Example usage looks something like this:

```
>>> import globus_sdk
>>> auth_client = globus_sdk.AuthClient(client_id=..., client_secret=...)
>>> # do some flow to get a refresh token from auth_client
>>> rt_authorizer = globus_sdk.RefreshTokenAuthorizer(
>>>     refresh_token, auth_client)
>>> # create a new client
>>> transfer_client = globus_sdk.TransferClient(authorizer=rt_authorizer)
```

anything that inherits from `BaseClient`, so at least `TransferClient` and `AuthClient` will automatically handle usage of the `RefreshTokenAuthorizer`.

Parameters

- **refresh_token** (*str*) – Refresh Token for Globus Auth
- **auth_client** (*AuthClient*) – `AuthClient` capable of using the `refresh_token`
- **access_token** (*str, optional*) – Initial Access Token to use, only used if `expires_at` is also set
- **expires_at** (*int, optional*) – Expiration time for the starting `access_token` expressed as a POSIX timestamp (i.e. seconds since the epoch)
- **on_refresh** (*callable, optional*) – A callback which is triggered any time this authorizer fetches a new access_token. The `on_refresh` callable is invoked on the *OAuthTokenResponse* object resulting from the token being refreshed. It should take only one argument, the token response object. This is useful for implementing storage for Access Tokens, as the `on_refresh` callback can be used to update the Access Tokens and their expiration times.

class globus_sdk.**ClientCredentialsAuthorizer**(*confidential_client*, *scopes*, *access_token=None*, *expires_at=None*, *on_refresh=None*)
Bases: *globus_sdk.authorizers.renewing.RenewingAuthorizer*

Implementation of a RenewingAuthorizer that renews confidential app client Access Tokens using a ConfidentialAppAuthClient and a set of scopes to fetch a new Access Token when the old one expires.

Example usage looks something like this:

```
>>> import globus_sdk
>>> confidential_client = globus_sdk.ConfidentialAppAuthClient(
    client_id=..., client_secret=...)
>>> scopes = "..."
>>> cc_authorizer = globus_sdk.ClientCredentialsAuthorizer(
>>>     confidential_client, scopes)
>>> # create a new client
>>> transfer_client = globus_sdk.TransferClient(authorizer=cc_authorizer)
```

any client that inherits from `BaseClient` should be able to use a ClientCredentialsAuthorizer to act as the client itself.

Parameters

- **confidential_client** (*ConfidentialAppAuthClient*) – client object with a valid id and client secret
- **scopes** (*str*) – A string of space-separated scope names being requested for the access tokens that will be used for the Authorization header. These scopes must all be for the same resource server, or else the token response will have multiple access tokens.
- **access_token** (*str*) – Initial Access Token to use, only used if `expires_at` is also set. Must be requested with the same set of scopes passed to this authorizer.
- **expires_at** (*int, optional*) – Expiration time for the starting `access_token` expressed as a POSIX timestamp (i.e. seconds since the epoch)
- **on_refresh** (*callable, optiona*) – A callback which is triggered any time this authorizer fetches a new access_token. The `on_refresh` callable is invoked on the *OAuthTokenResponse* object resulting from the token being refreshed. It should take only one argument, the token response object. This is useful for implementing storage for Access Tokens, as the `on_refresh` callback can be used to update the Access Tokens and their expiration times.

1.7 Globus SDK Configuration

There are three standard, canonical locations from which the Globus SDK will attempt to load configuration.

There are two config file locations:

```
/etc/globus.cfg # system config, shared by all users
~/.globus.cfg # personal config, specific to your user
```

additionally, the shell environment variables loaded into Python's *os.environ* will be searched for configuration.

The precedence rules are very simply

1. Environment
2. `~/.globus.cfg`
3. `/etc/globus.cfg`

1.7.1 Config Format

Config files are INI formatted, so they take the general form

```
[SectionName]
key1 = value1
key2 = value2
```

At present, there are no configuration parameters which you should set in config files.

The Globus CLI uses the `[cli]` section to store configuration information.

1.7.2 Environment Variables

`GLOBUS_SDK_ENVIRONMENT` is a shell variable that can be used to point the SDK to an alternate set of Globus Servers.

For example, if you have an integration with Globus you may be asked to test against a preview of upcoming changes. To point the SDK at the Preview environment `GLOBUS_SDK_ENVIRONMENT=preview` can be used.

1.8 Versioning Policy

The Globus SDK follows Semantic Versioning.

That means that we use version numbers of the form **MAJOR.MINOR.PATCH**.

When the SDK needs to make incompatible API changes, the **MAJOR** version number will be incremented. **MINOR** and **PATCH** version increments indicate new features or bugfixes.

1.8.1 Public Interfaces

Features documented here are public and all other components of the SDK should be considered private. Undocumented components may be subject to backwards incompatible changes without increments to the **MAJOR** version.

1.8.2 Recommended Pinning

We recommend that users of the SDK pin only to the major version which they require. e.g. specify `globus-sdk>=1.7,<2.0` in your package requirements.

1.8.3 Upgrade Caveat

It is always possible for new features or bugfixes to cause issues.

If you are installing the SDK into mission-critical production systems, we strongly encourage you to establish a method of pinning the exact version used and testing upgrades.

1.9 Globus SDK Examples

Each of these pages contains an example of a piece of SDK functionality.

1.9.1 API Authorization

Using a `GlobusAuthorizer` is hard to grasp without a few examples to reference. The basic usage should be to create these at client instantiation time.

Access Token Authorization on AuthClient and TransferClient

Perhaps you're in a part of your application that only sees Access Tokens. Access Tokens are used to directly authenticate calls against Globus APIs, and are limited-lifetime credentials. You have distinct Access Tokens for each Globus service which you want to access.

With the tokens in hand, it's just a simple matter of wrapping the tokens in *`AccessTokenAuthorizer`* objects.

```
from globus_sdk import AuthClient, TransferClient, AccessTokenAuthorizer

AUTH_ACCESS_TOKEN = '...'
TRANSFER_ACCESS_TOKEN = '...'

# note that we don't provide the client ID in this case
# if you're using an Access Token you can't do the OAuth2 flows
auth_client = AuthClient(
    authorizer=AccessTokenAuthorizer(AUTH_ACCESS_TOKEN))

transfer_client = TransferClient(
    authorizer=AccessTokenAuthorizer(TRANSFER_ACCESS_TOKEN))
```

Refresh Token Authorization on AuthClient and TransferClient

Refresh Tokens are long-lived credentials used to get new Access Tokens whenever they expire. However, it would be very awkward to create a new client instance every time your credentials expire!

Instead, use a *`RefreshTokenAuthorizer`* to automatically re-up your credentials whenever they near expiration.

Re-upping credentials is an operation that requires having client credentials for Globus Auth, so creating the authorizer is more complex this time.

```
from globus_sdk import (AuthClient, TransferClient, ConfidentialAppAuthClient,
                        RefreshTokenAuthorizer)

# for doing the refresh
CLIENT_ID = '...'
CLIENT_SECRET = '...'

# the actual tokens
AUTH_REFRESH_TOKEN = '...'
TRANSFER_REFRESH_TOKEN = '...'

# making the authorizer requires that we have an AuthClient which can talk
# OAuth2 to Globus Auth
internal_auth_client = ConfidentialAppAuthClient(CLIENT_ID, CLIENT_SECRET)
```

(continues on next page)

(continued from previous page)

```
# now let's bake a couple of authorizers
auth_authorizer = RefreshTokenAuthorizer(AUTH_REFRESH_TOKEN,
                                         internal_auth_client)
transfer_authorizer = RefreshTokenAuthorizer(TRANSFER_REFRESH_TOKEN,
                                             internal_auth_client)

# auth_client here is totally different from "internal_auth_client" above
# the former is being used to request new tokens periodically, while this
# one represents a user authenticated with those tokens
auth_client = AuthClient(authorizer=auth_authorizer)
# transfer_client doesn't have to contend with this duality -- it's always
# representing a user
transfer_client = TransferClient(authorizer=transfer_authorizer)
```

Basic Auth on an AuthClient

If you're using an *AuthClient* to do OAuth2 flows, you likely want to authenticate it using your client credentials – the client ID and client secret.

The preferred method is to use the `AuthClient` subclass which automatically specifies its authorizer. Internally, this will use a `BasicAuthorizer` to do Basic Authentication.

By way of example:

```
from globus_sdk import ConfidentialAppAuthClient

CLIENT_ID = '...'
CLIENT_SECRET = '...'

client = ConfidentialAppAuthClient(CLIENT_ID, CLIENT_SECRET)
```

and you're off to the races!

Under the hood, this is implicitly running

```
AuthClient(authorizer=BasicAuthorizer(CLIENT_ID, CLIENT_SECRET))
```

but don't do this yourself – `ConfidentialAppAuthClient` has different methods from the base `AuthClient`.

1.9.2 Native App Login

This is an example of the use of the Globus SDK to carry out an OAuth2 Native App Authentication flow.

The goal here is to have a user authenticate in Globus Auth, and for the SDK to procure tokens which may be used to authenticate SDK calls against various services for that user.

Get a Client

In order to complete an OAuth2 flow to get tokens, you must have a client definition registered with Globus Auth. To do so, follow the relevant documentation for the Globus Auth Service or go directly to developers.globus.org to do the registration.

Make sure, when registering your application, that you enter `https://auth.globus.org/v2/web/auth-code` into the "Redirect URIs" field. This is necessary to leverage the default behavior of the SDK, and is typically sufficient for this type of application.

Do the Flow

If you want to copy-paste an example, you'll need at least a `client_id` for your `AuthClient` object. You should also specifically use the *`NativeAppAuthClient`* type of `AuthClient`, as it has been customized to handle this flow.

The shortest version of the flow looks like this:

```
import globus_sdk

# you must have a client ID
CLIENT_ID = '...'

client = globus_sdk.NativeAppAuthClient(CLIENT_ID)
client.oauth2_start_flow()

authorize_url = client.oauth2_get_authorize_url()
print('Please go to this URL and login: {0}'.format(authorize_url))

# or just input() on python3
auth_code = raw_input(
    'Please enter the code you get after login here: ').strip()
token_response = client.oauth2_exchange_code_for_tokens(auth_code)

# the useful values that you want at the end of this
globus_auth_data = token_response.by_resource_server['auth.globus.org']
globus_transfer_data = token_response.by_resource_server['transfer.api.globus.org']
globus_auth_token = globus_auth_data['access_token']
globus_transfer_token = globus_transfer_data['access_token']
```

Do It With Refresh Tokens

The flow above will give you access tokens (short-lived credentials), good for one-off operations. However, if you want a persistent credential to access the logged-in user's Globus resources, you need to request a long-lived credential called a Refresh Token.

`refresh_tokens` is a boolean option to the `oauth2_start_flow` method. When False, the flow will terminate with a collection of Access Tokens, which are simple limited lifetime credentials for accessing services. When True, the flow will terminate not only with the Access Tokens, but additionally with a set of Refresh Tokens which can be used **indefinitely** to request new Access Tokens. The default is False.

Simply add this option to the example above:

```
client.oauth2_start_flow(refresh_tokens=True)
```

1.9.3 Client Credentials Authentication

This is an example of the use of the Globus SDK to carry out an OAuth2 Client Credentials Authentication flow.

The goal here is to have an application authenticate in Globus Auth directly, as itself. Unlike many other OAuth2 flows, the application does not act on behalf of a user, but on its own behalf.

This flow is suitable for automated cases in which an application, even one as simple as a `cron` job, makes use of Globus outside of the context of a specific end-user interaction.

Get a Client

In order to complete an OAuth2 flow to get tokens, you must have a client definition registered with Globus Auth. To do so, follow the relevant documentation for the Globus Auth Service or go directly to developers.globus.org to do the registration.

During registration, make sure that the "Native App" checkbox is unchecked. You will typically want your scopes to be `openid`, `profile`, `email`, and `urn:globus:auth:scope:transfer.api.globus.org:all`.

Once your client is created, expand it on the Projects page and click "Generate Secret". Save the secret in a secure location accessible from your code.

Do the Flow

You should specifically use the *`ConfidentialAppAuthClient`* type of `AuthClient`, as it has been customized to handle this flow.

The shortest version of the flow looks like this:

```
import globus_sdk

# you must have a client ID
CLIENT_ID = '...'
# the secret, loaded from wherever you store it
CLIENT_SECRET = '...'

client = globus_sdk.ConfidentialAppAuthClient(CLIENT_ID, CLIENT_SECRET)
token_response = client.oauth2_client_credentials_tokens()

# the useful values that you want at the end of this
globus_auth_data = token_response.by_resource_server['auth.globus.org']
globus_transfer_data = token_response.by_resource_server['transfer.api.globus.org']
globus_auth_token = globus_auth_data['access_token']
globus_transfer_token = globus_transfer_data['access_token']
```

Use the Resulting Tokens

The Client Credentials Grant will only produce Access Tokens, not Refresh Tokens, so you should pass its results directly to the *`AccessTokenAuthorizer`*.

For example, after running the code above,

```
authorizer = globus_sdk.AccessTokenAuthorizer(globus_transfer_token)
tc = globus_sdk.TransferClient(authorizer=authorizer)
print("Endpoints Belonging to {}@clients.auth.globus.org:"
```

(continues on next page)

(continued from previous page)

```
      .format(CLIENT_ID))
for ep in tc.endpoint_search(filter_scope="my-endpoints"):
    print("[{}] {}".format(ep["id"], ep["display_name"]))
```

Note that we're doing a search for "my endpoints", but we refer to the results as belonging to `<CLIENT_ID>@clients.auth.globus.org`. The "current user" is not any human user, but the client itself.

Handling Token Expiration

When you get access tokens, you also get their expiration time in seconds. You can inspect the `globus_transfer_data` and `globus_auth_data` structures in the example to see.

Tokens should have a long enough lifetime for any short-running operations (less than a day).

When your tokens are expired, you should just request new ones by making another Client Credentials request. Depending on your needs, you may need to track the expiration times along with your tokens.

1.9.4 Using ClientCredentialsAuthorizer

The SDK also provides a specialized Authorizer which can be used to automatically handle token expiration.

Use it like so:

```
import globus_sdk

# you must have a client ID
CLIENT_ID = '...'
# the secret, loaded from wherever you store it
CLIENT_SECRET = '...'

confidential_client = globus_sdk.ConfidentialAppAuthClient(
    client_id=CLIENT_ID, client_secret=CLIENT_SECRET)
scopes = "urn:globus:auth:scope:transfer.api.globus.org:all"
cc_authorizer = globus_sdk.ClientCredentialsAuthorizer(
    confidential_client, scopes)
# create a new client
transfer_client = globus_sdk.TransferClient(authorizer=cc_authorizer)

# usage is still the same
print("Endpoints Belonging to {}@clients.auth.globus.org:"
      .format(CLIENT_ID))
for ep in tc.endpoint_search(filter_scope="my-endpoints"):
    print("[{}] {}".format(ep["id"], ep["display_name"]))
```

1.9.5 Three Legged OAuth with Flask

This type of authorization is used for web login with a server-side application. For example, a Django app or other application server handles requests.

This example uses Flask, but should be easily portable to other application frameworks.

Components

There are two components to this application: login and logout.

Login sends a user to Globus Auth to get credentials, and then may act on the user's behalf. Logout invalidates server-side credentials, so that the application may no longer take actions for the user, and the client-side session, allowing for a fresh login if desired.

Register an App

In order to complete an OAuth2 flow to get tokens, you must have a client definition registered with Globus Auth. To do so, follow the relevant documentation for the Globus Auth Service or go directly to developers.globus.org to do the registration.

Make sure that the "Native App" checkbox is unchecked, and list `http://localhost:5000/login` in the "Redirect URIs".

Set the Scopes to `openid`, `profile`, `email`, `urn:globus:auth:scope:transfer.api.globus.org:all`.

On the projects page, expand the client description and click "Generate Secret". Save the resulting secret a file named `example_app.conf`, along with the client ID:

```
SERVER_NAME = 'localhost:5000'
# this is the session secret, used to protect the Flask session. You should
# use a longer secret string known only to your application
# details are beyond the scope of this example
SECRET_KEY = 'abc123!'

APP_CLIENT_ID = '<CLIENT_ID>'
APP_CLIENT_SECRET = '<CLIENT_SECRET>'
```

Shared Utilities

Some pieces that are of use for both parts of this flow.

First, you'll need to install `Flask` and the `globus-sdk`. Assuming you want to do so into a fresh virtualenv:

```
$ virtualenv example-venv
...
$ source example-venv/bin/activate
$ pip install Flask==0.11.1 globus-sdk
...
```

You'll also want a shared function for loading the SDK `AuthClient` which represents your application, as you'll need it in a couple of places. Create it, along with the defintiion for your Flask app, in `example_app.py`:

```
from flask import Flask, url_for, session, redirect, request
import globus_sdk

app = Flask(__name__)
app.config.from_pyfile('example_app.conf')


# actually run the app if this is called as a script
if __name__ == '__main__':
    app.run()


def load_app_client():
    return globus_sdk.ConfidentialAppAuthClient(
        app.config['APP_CLIENT_ID'], app.config['APP_CLIENT_SECRET'])
```

Login

Let's add login functionality to the end of `example_app.py`, along with a basic index page:

```
@app.route('/')
def index():
    """
    This could be any page you like, rendered by Flask.
    For this simple example, it will either redirect you to login, or print
    a simple message.
    """
    if not session.get('is_authenticated'):
        return redirect(url_for('login'))
    return "You are successfully logged in!"

@app.route('/login')
def login():
    """
    Login via Globus Auth.
    May be invoked in one of two scenarios:

      1. Login is starting, no state in Globus Auth yet
      2. Returning to application during login, already have short-lived
         code from Globus Auth to exchange for tokens, encoded in a query
         param
    """
    # the redirect URI, as a complete URI (not relative path)
    redirect_uri = url_for('login', _external=True)

    client = load_app_client()
    client.oauth2_start_flow(redirect_uri)

    # If there's no "code" query string parameter, we're in this route
    # starting a Globus Auth login flow.
    # Redirect out to Globus Auth
    if 'code' not in request.args:
        auth_uri = client.oauth2_get_authorize_url()
        return redirect(auth_uri)
    # If we do have a "code" param, we're coming back from Globus Auth
    # and can start the process of exchanging an auth code for a token.
```

(continues on next page)

(continued from previous page)

```
    else:
        code = request.args.get('code')
        tokens = client.oauth2_exchange_code_for_tokens(code)

        # store the resulting tokens in the session
        session.update(
            tokens=tokens.by_resource_server,
            is_authenticated=True
        )
        return redirect(url_for('index'))
```

Logout

Logout is very simple – it's just a matter of cleaning up the session. It does the added work of cleaning up any tokens you fetched by invalidating them in Globus Auth beforehand:

```
@app.route('/logout')
def logout():
    """
    - Revoke the tokens with Globus Auth.
    - Destroy the session state.
    - Redirect the user to the Globus Auth logout page.
    """
    client = load_app_client()

    # Revoke the tokens with Globus Auth
    for token in (token_info['access_token']
                  for token_info in session['tokens'].values()):
        client.oauth2_revoke_token(token)

    # Destroy the session state
    session.clear()

    # the return redirection location to give to Globus AUth
    redirect_uri = url_for('index', _external=True)

    # build the logout URI with query params
    # there is no tool to help build this (yet!)
    globus_logout_url = (
        'https://auth.globus.org/v2/web/logout' +
        '?client={}'.format(app.config['PORTAL_CLIENT_ID']) +
        '&redirect_uri={}'.format(redirect_uri) +
        '&redirect_name=Globus Example App')

    # Redirect the user to the Globus Auth logout page
    return redirect(globus_logout_url)
```

Using the Tokens

Using the tokens thus acquired is a simple matter of pulling them out of the session and putting one into an `AccessTokenAuthorizer`. For example, one might do the following:

```
authorizer = globus_sdk.AccessTokenAuthorizer(
    session['tokens']['transfer.api.globus.org']['access_token'])
transfer_client = globus_sdk.TransferClient(authorizer=authorizer)

print("Endpoints belonging to the current logged-in user:")
for ep in transfer_client.endpoint_search(filter_scope="my-endpoints"):
    print("[{}] {}".format(ep["id"], ep["display_name"]))
```

1.9.6 Advanced Transfer Client Usage

This is a collection of examples of advanced usage patterns leveraging the *TransferClient*.

Relative Task Deadlines

One of the lesser-known features of the Globus Transfer service is the ability for users to set a `deadline` by which a Transfer or Delete task must complete. If the task is still in progress when the `deadline` is reached, it is aborted.

You can use this, for example, to enforce that a Transfer Task which takes too long results in errors (even if it is making slow progress).

Because the `deadline` is accepted as an ISO 8601 date, you can use python's built-in `datetime` library to compute a timestamp to pass to the service.

Start out by computing the current time as a `datetime`:

```
import datetime
now = datetime.datetime.utcnow()
```

Then, compute a relative timestamp using `timedelta`:

```
future_1minute = now + datetime.timedelta(minutes=1)
```

This value can be passed to a *TransferData*, as in

```
import globus_sdk
# get various components needed for a Transfer Task
# beyond the scope of this example
transfer_client = globus_sdk.TransferClient(...)
source_endpoint_uuid = ...
dest_endpoint_uuid = ...

# note how `future_1minute` is used here
submission_data = globus_sdk.TransferData(
    transfer_client, source_endpoint_uuid, dest_endpoint_uuid,
    deadline=str(future_1minute))
```

Retrying Task Submission

Globus Transfer and Delete Tasks are often scheduled and submitted by automated systems and scripts. In these scenarios, it's often desirable to retry submission in the event of network or service errors to ensure that the job is really submitted.

There are two key pieces to doing this correctly: Once and Only Once Submission, and logging captured errors.

For once-and-only-once task submission, you can explicitly invoke *TransferClient.get_submission_id()*, which is a unique ID used to ensure exactly this. However, *TransferData* and *DeleteData* both implicitly invoke this method if they are initialized without an explicit `submission_id`.

For proper logging, we'll rely on the standard library `logging` package.

In this example, we'll retry task submission 5 times, and we'll want to separate retry logic from the core task submission logic.

```
import logging
from globus_sdk import GlobusAPIError, NetworkError

# putting logger objects named by the module name into the module-level
# scope is a common best practice -- for more details, you should look
# into the python logging documentation
logger = logging.getLogger(__name__)


def retry_globus_function(func, retries=5, func_name='<func>'):
    """
    Define what it means to retry a "Globus Function", some function or
    method which produces Globus SDK errors on failure.
    """
    def actually_retry():
        """
        Helper: run the next retry
        """
        return retry_globus_function(func, retries=(retries - 1),
                                     func_name=func_name)

    def check_for_reraise():
        """
        Helper: check if we should reraise an error
                logs an error message on reraise
                must be run inside an exception handler
        """
        if retries < 1:
            logger.error('Retried {} too many times.'
                         .format(func_name))
            raise

    try:
        return func()
    except NetworkError:
        # log with exc_info=True to capture a full stacktrace as a
        # debug-level log
        logger.debug(('Globus func {} experienced a network error'
                      .format(func_name)), exc_info=True)
        check_for_reraise()
    except GlobusAPIError:
        # again, log with exc_info=True to capture a full stacktrace
```

(continues on next page)

(continued from previous page)

```
        logger.warn(('Globus func {} experienced a network error'
                     .format(func_name)), exc_info=True)
        check_for_reraise()

    # if we reach this point without returning or erroring, retry
    return actually_retry()
```

The above is a fairly generic tool for retrying any function which throws `globus_sdk.NetworkError` and `globus_sdk.GlobusAPIError` errors. It is not even specific to task resubmission, so you could use it against other retry-safe Globus APIs.

Now, moving on to creating a retry-safe function to put into it, things get a little bit tricky. The retry handler above requires a function which takes no arguments, so we'll have to define a function dynamically which fits that constraint:

```
def submit_transfer_with_retries(transfer_client, transfer_data):
    # create a function with no arguments, for our retry handler
    def locally_bound_func():
        return transfer_client.submit_transfer(transfer_data)
    return retry_globus_function(locally_bound_func,
                                 func_name='submit_transfer')
```

Now we're finally all-set to create a `TransferData` and submit it:

```
from globus_sdk import TransferClient, TransferData
# get various components needed for a Transfer Task
# beyond the scope of this example
transfer_client = TransferClient(...)
source_endpoint_uuid = ...
dest_endpoint_uuid = ...

submission_data = TransferData(
    transfer_client, source_endpoint_uuid, dest_endpoint_uuid)

# add any number of items to the submission data
submission_data.add_item('/source/path', 'dest/path')
...

# do it!
submit_transfer_with_retries(transfer_client, submission_data)
```

The same exact approach can be applied to `TransferClient.submit_delete`, and a wide variety of other SDK methods.

1.10 License

PYTHON MODULE INDEX

g

INDEX

Symbols

A

B

C

D

H

I

L

M

N

O