globus-sdk

Release 3.0.1

Globus Team

Sep 15, 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"
- **Native App**: Check this Box
- **Scopes**: "openid", "profile", "email", "urn:globus:auth:scope:transfer.api.globus.org:all"
- **Redirects**: https://auth.globus.org/v2/web/auth-code
- **Required Identity**: <Leave Unchecked>
- **Pre-select Identity Provider**: <Leave Unchecked>
- **Privacy Policy**: <Leave Blank>
- **Terms & Conditions**: <Leave Blank>

and click "Create App".

> **Warning:** The **Native App** setting cannot be changed after a client is created.

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>"

```

(continues on next page)

(continued from previous page)

```
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))

auth_code = 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()

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

auth_code = 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()))

auth_code = 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 abbreviated 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, AccessTokenAuthorizer

# you must have a valid transfer token for this to work
tc = TransferClient(
    authorizer=AccessTokenAuthorizer("TRANSFER_TOKEN_STRING")
)

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"])
```

Note: 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.

1.3.1 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: Optional[str] = None*, ***kwargs: Any*)

Bases: *globus_sdk.client.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()*
- *get_jwk()*
- *get_openid_configuration()*
- *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: Optional[Union[Iterable[str], str]] = None*, *ids: Optional[Union[Iterable[Union[uuid.UUID, str]], uuid.UUID, str]] = None*, *provision: bool = False*, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

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",
```

(continues on next page)

(continued from previous page)

```
>>>          "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(
>>>     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(*, *query_params: Optional[Dict[str, Any]] = None*) → str

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 **query_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: str*) → *globus_sdk.services.auth.response.OAuthTokenResponse*

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: str*, *, *body_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.services.auth.response.OAuthTokenResponse*

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
- **body_params** (*dict, optional*) – A dict of extra params to encode in the refresh call.

oauth2_validate_token(*token: str*, ***, *body_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

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
- **body_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: str*, ***, *body_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

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

- **body_params** (*dict, optional*) – 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: Union[dict,* globus_sdk.utils.PayloadWrapper*]*) → *globus_sdk.services.auth.response.OAuthTokenResponse*
oauth2_token(*form_data: Union[dict,* globus_sdk.utils.PayloadWrapper*]*, **, body_params: Optional[Dict[str, Any]]*) → *globus_sdk.services.auth.response.OAuthTokenResponse*
oauth2_token(*form_data: Union[dict,* globus_sdk.utils.PayloadWrapper*]*, **, response_class: Type[globus_sdk.services.auth.client.base.RT]*) → globus_sdk.services.auth.client.base.RT
oauth2_token(*form_data: Union[dict,* globus_sdk.utils.PayloadWrapper*]*, **, body_params: Optional[Dict[str, Any]]*, *response_class: Type[globus_sdk.services.auth.client.base.RT]*) → globus_sdk.services.auth.client.base.RT

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() → *globus_sdk.response.GlobusHTTPResponse*

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.

get_openid_configuration() → *globus_sdk.response.GlobusHTTPResponse*

Fetch the OpenID Connect configuration data from the well-known URI for Globus Auth.

get_jwk(*openid_configuration: Optional[Union[*globus_sdk.response.GlobusHTTPResponse*, Dict[str, Any]]]*, **, as_pem: Literal[True]*) → cryptography.hazmat.primitives.asymmetric.rsa.RSAPublicKey
get_jwk(*openid_configuration: Optional[Union[*globus_sdk.response.GlobusHTTPResponse*, Dict[str, Any]]]*, **, as_pem: Literal[False]*) → dict

Fetch the Globus Auth JWK.

Returns either a dict or an RSA Public Key object depending on `as_pem`.

Parameters

- **openid_configuration** (*dict or GlobusHTTPResponse*) – The OIDC config as a GlobusHTTPResponse or dict. When not provided, it will be fetched automatically.
- **as_pem** (*bool*) – Decode the JWK to an RSA PEM key, typically for JWT decoding

class globus_sdk.**NativeAppAuthClient**(*client_id: str*, ***kwargs: Any*)

Bases: *globus_sdk.services.auth.client.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: Optional[Union[Iterable[str], str]] = None*, ***, *redirect_uri: Optional[str] = None*, *state: str = '_default'*, *verifier: Optional[str] = None*, *refresh_tokens: bool = False*, *prefill_named_grant: Optional[str] = None*) → *globus_sdk.services.auth.flow_managers.native_app.GlobusNativeAppFlowManager*

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 modifications to the Authorization Code grant flow as The PKCE Security Protocol.

oauth2_refresh_token(*refresh_token: str*, *, *body_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.services.auth.response.OAuthTokenResponse*

`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: str*, *client_secret: str*, ***kwargs: Any*)

Bases: *globus_sdk.services.auth.client.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* 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: Optional[Union[Iterable[str], str]] = None*) → *globus_sdk.services.auth.response.OAuthTokenResponse*

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 or iterable of 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: str*, *requested_scopes: Optional[Union[Iterable[str], str]] = None*, *, *state: str = '_default'*, *refresh_tokens: bool = False*) → *globus_sdk.services.auth.flow_managers.authorization_code.GlobusAuthorizationCodeFlowManager*

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: str*, ***, *additional_params: Optional[dict] = None*) → *globus_sdk.services.auth.response.OAuthDependentTokenResponse*

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 *OAuthDependentTokenResponse*

oauth2_token_introspect(*token: str*, ***, *include: Optional[str] = None*) → *globus_sdk.response.GlobusHTTPResponse*

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:* globus_sdk.services.auth.client.base.AuthClient, *identity_ids: Optional[Iterable[str]] = None*, *, *id_batch_size: Optional[int] = 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, optional*) – 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: str*) → None
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: str*) → Any
IdentityMap supports dict-like lookups with `map[key]`

__init__(*auth_client:* globus_sdk.services.auth.client.base.AuthClient, *identity_ids: Optional[Iterable[str]] = None*, *, *id_batch_size: Optional[int] = None*)

add(*identity_id: str*) → bool
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: str*, *default: Optional[Any] = None*) → Any
A dict-like get() method which accepts a default value.

Auth Responses

class globus_sdk.**OAuthTokenResponse**(**args: Any*, ***kwargs: Any*)
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: Dict[str, dict]
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: globus_sdk.services.auth.response._ByScopesGetter
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(*openid_configuration: Optional[Union[*globus_sdk.response.GlobusHTTPResponse*, Dict[str, Any]]] = None*, *jwk: Optional[cryptography.hazmat.primitives.asymmetric.rsa.RSAPublicKey] = None*, *jwt_params: Optional[Dict] = None*) → Dict[str, Any]
Parse the included ID Token (OIDC) as a dict and return it.

If you provide the *jwk*, you must also provide *openid_configuration*.

Parameters

- **openid_configuration** (*dict or* GlobusHTTPResponse) – The OIDC config as a GlobusHTTPResponse or dict. When not provided, it will be fetched automatically.
- **jwk** (*RSAPublicKey*) – The JWK as a cryptography public key object. When not provided, it will be fetched and parsed automatically.
- **jwt_params** (*dict*) – An optional dict of parameters to pass to the jwt decode step. These are passed verbatim to the jwt library.

class globus_sdk.**OAuthDependentTokenResponse**(**args: Any*, ***kwargs: Any*)

Bases: *globus_sdk.services.auth.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(*openid_configuration: Optional[Union[*globus_sdk.response.GlobusHTTPResponse*, Dict[str, Any]]] = None*, *jwk: Optional[cryptography.hazmat.primitives.asymmetric.rsa.RSAPublicKey] = None*, *jwt_params: Optional[Dict] = None*) → Dict[str, Any]

Parse the included ID Token (OIDC) as a dict and return it.

If you provide the *jwk*, you must also provide *openid_configuration*.

Parameters

- **openid_configuration** (*dict or* GlobusHTTPResponse) – The OIDC config as a GlobusHTTPResponse or dict. When not provided, it will be fetched automatically.
- **jwk** (*RSAPublicKey*) – The JWK as a cryptography public key object. When not provided, it will be fetched and parsed automatically.
- **jwt_params** (*dict*) – An optional dict of parameters to pass to the jwt decode step. These are passed verbatim to the jwt library.

OAuth2 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.services.auth.**GlobusNativeAppFlowManager**(*auth_client:* globus_sdk.AuthClient, *requested_scopes: Optional[Union[str, Iterable[str]]] = None*, *redirect_uri: Optional[str] = None*, *state: str = '_default'*, *verifier: Optional[str] = None*, *refresh_tokens: bool = False*, *prefill_named_grant: Optional[str] = None*)

Bases: *globus_sdk.services.auth.flow_managers.base.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: str*) → *globus_sdk.services.auth.response.OAuthTokenResponse*

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(*query_params: Optional[Dict[str, Any]] = None*) → str

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

Parameters **query_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.services.auth.**GlobusAuthorizationCodeFlowManager**(*auth_client: globus_sdk.AuthClient, redirect_uri: str, requested_scopes: Optional[Union[str, Iterable[str]]] = None, state: str = '_default', refresh_tokens: bool = False*)

Bases: *globus_sdk.services.auth.flow_managers.base.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` (that is, `DEFAULT_REQUESTED_SCOPES` from `globus_sdk.services.auth.oauth2_constants`)
- **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: str*) → *globus_sdk.services.auth.response.OAuthTokenResponse*

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(*query_params: Optional[Dict[str, Any]] = None*) → str

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

Parameters **query_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.services.auth.flow_managers.**GlobusOAuthFlowManager**

Bases: `abc.ABC`

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.

abstract exchange_code_for_tokens(*auth_code: str*) → *globus_sdk.services.auth.response.OAuthTokenResponse*

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*

abstract get_authorize_url(*query_params: Optional[Dict[str, Any]] = None*) → str

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

1.3.2 Globus Groups

class globus_sdk.**GroupsClient**(*, *environment: Optional[str] = None*, *base_url: Optional[str] = None*, *authorizer: Optional[globus_sdk.authorizers.base.GlobusAuthorizer] = None*, *app_name: Optional[str] = None*, *transport_params: Optional[Dict] = None*)

Bases: *globus_sdk.client.BaseClient*

Client for the Globus Groups API.

This provides a relatively low level client to public groups API endpoints. You may also consider looking at the GroupsManager as a simpler interface to more common actions.

Methods

- *batch_membership_action()*
- *create_group()*
- *delete_group()*
- *get_group()*
- *get_group_policies()*
- *get_identity_preferences()*
- *get_membership_fields()*
- *get_my_groups()*
- *set_group_policies()*
- *set_identity_preferences()*
- *set_membership_fields()*

get_my_groups(*, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

Return a list of groups your identity belongs to.

External Documentation

See Retrieve your groups and membership in the API documentation for details.

get_group(*group_id: Union[uuid.UUID, str]*, ***, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

Get details about a specific group

External Documentation

See Get Group in the API documentation for details.

delete_group(*group_id: Union[uuid.UUID, str]*, ***, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

Delete a group.

External Documentation

See Delete a group in the API documentation for details.

create_group(*data: Dict[str, Any]*, ***, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

Create a group.

External Documentation

See Create a group in the API documentation for details.

get_group_policies(*group_id: Union[uuid.UUID, str]*, ***, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

Get policies for the given group

External Documentation

See Get the policies for the group in the API documentation for details.

set_group_policies(*group_id: Union[uuid.UUID, str]*, *data: Union[Dict[str, Any], globus_sdk.services.groups.data.GroupPolicies]*, ***, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

Set policies for the group.

External Documentation

See Set the policies for the group in the API documentation for details.

get_identity_preferences(***, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

Get identity preferences. Currently this only includes whether the user allows themselves to be added to groups.

External Documentation

See Get the preferences for your identity set in the API documentation for details.

set_identity_preferences(*data: Dict[str, Any]*, ***, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

Set identity preferences. Currently this only includes whether the user allows themselves to be added to groups.

External Documentation

See Set the preferences for your identity set in the API documentation for details.

get_membership_fields(*group_id: Union[uuid.UUID, str]*, ***, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

Get membership fields for your identities.

External Documentation

See Get the membership fields for your identity set in the API documentation for details.

set_membership_fields(*group_id: Union[uuid.UUID, str]*, *data: Dict[Any, str]*, ***, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

Get membership fields for your identities.

External Documentation

See Set the membership fields for your identity set in the API documentation for details.

batch_membership_action(*group_id: Union[uuid.UUID, str]*, *actions: Union[Dict[str, Any], globus_sdk.services.groups.data.BatchMembershipActions]*, ***, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

Execute a batch of actions against several group memberships.

External Documentation

See Perform actions on members of the group in the API documentation for details.

Helper Objects

These helper objects make it easier to create and submit data to a `GroupsClient`. Additionally, they may be used in concert with the `GroupsManager` to perform operations.

These enums define values which can be passed to other helpers:

class globus_sdk.**GroupMemberVisibility**(*value*)

An enumeration.

managers = 'managers'

members = 'members'

class globus_sdk.**GroupRequiredSignupFields**(*value*)

An enumeration.

address = 'address'

address1 = 'address1'

address2 = 'address2'

city = 'city'

country = 'country'

current_project_name = 'current_project_name'

department = 'department'

field_of_science = 'field_of_science'

institution = 'institution'

phone = 'phone'

state = 'state'

zip = 'zip'

class globus_sdk.**GroupRole**(*value*)
An enumeration.

admin = 'admin'

manager = 'manager'

member = 'member'

class globus_sdk.**GroupVisibility**(*value*)
An enumeration.

authenticated = 'authenticated'

private = 'private'

Payload Types

A `BatchMembershipActions` defines how to formulate requests to add, remove, or modify memberships in a group. It can be used to formulate multiple operations to submit in a single request to the service.

class globus_sdk.**BatchMembershipActions**(*dict=None*, */*, ***kwargs*)
An object used to represent a batch action on memberships of a group. Perform actions on group members.

accept_invites(*identity_ids: Iterable[Union[uuid.UUID, str]]*) → *globus_sdk.services.groups.data.BatchMembershipActions*
Accept invites for identities. The identities must belong to the identity set of authenticated user.

add_members(*identity_ids: Iterable[Union[uuid.UUID, str]]*, ***, *role:* globus_sdk.services.groups.data.GroupRole = *GroupRole.member*) → *globus_sdk.services.groups.data.BatchMembershipActions*
Add a list of identities to a group with the given role.

approve_pending(*identity_ids: Iterable[Union[uuid.UUID, str]]*) → *globus_sdk.services.groups.data.BatchMembershipActions*
Approve a list of identities with pending join requests.

decline_invites(*identity_ids: Iterable[Union[uuid.UUID, str]]*) → *globus_sdk.services.groups.data.BatchMembershipActions*
Decline an invitation for a given set of identities.

invite_members(*identity_ids: Iterable[Union[uuid.UUID, str]]*, ***, *role:* globus_sdk.services.groups.data.GroupRole = *GroupRole.member*) → *globus_sdk.services.groups.data.BatchMembershipActions*
Invite a list of identities to a group with the given role.

join(*identity_ids: Iterable[Union[uuid.UUID, str]]*) → *globus_sdk.services.groups.data.BatchMembershipActions*
Join a group with the given identities. The identities must be in the authenticated users identity set.

leave(*identity_ids: Iterable[Union[uuid.UUID, str]]*) → *globus_sdk.services.groups.data.BatchMembershipActions*
Leave a group that one of the identities in the authenticated user's identity set is a member of.

reject_join_requests(*identity_ids: Iterable[Union[uuid.UUID, str]]*) → *globus_sdk.services.groups.data.BatchMembershipActions*
Reject a members that have requested to join the group.

remove_members(*identity_ids: Iterable[Union[uuid.UUID, str]]*) → *globus_sdk.services.groups.data.BatchMembershipActions*
Remove members from a group. This must be done as an admin or manager of the group.

request_join(*identity_ids: Iterable[Union[uuid.UUID, str]]*) → *globus_sdk.services.groups.data.BatchMembershipActions*

Request to join a group.

A `GroupPolicies` object defines the various policies which can be set on a group. It can be used with the `GroupsClient` or the `GroupsManager`.

class globus_sdk.**GroupPolicies**(**, is_high_assurance: bool, group_visibility:* globus_sdk.services.groups.data.GroupVisibility, *group_members_visibility:* globus_sdk.services.groups.data.GroupMemberVisibility, *join_requests: bool, signup_fields: Iterable[*globus_sdk.services.groups.data.GroupRequiredSignupFields*], authentication_assurance_timeout: Optional[int] = None*)

An object used to represent the policy settings of a group. This may be used to set or modify group settings.

See also: API documentation on setting the policies for the group.

High-Level Client Wrappers

The `GroupsManager` is a high-level helper which wraps a `GroupsClient`. Many common operations which require assembling a `BatchMembershipActions` and submitting the result can be achieved with a single method-call on a `GroupsManager`.

class globus_sdk.**GroupsManager**(*client: Optional[*globus_sdk.services.groups.client.GroupsClient*] = None*)

A wrapper for the groups client with common membership and group actions wrapped in convenient methods with parameters and type hints.

Methods

- *accept_invite()*
- *add_member()*
- *approve_pending()*
- *create_group()*
- *decline_invite()*
- *invite_member()*
- *join()*
- *leave()*
- *reject_join_request()*
- *remove_member()*
- *request_join()*
- *set_group_policies()*

accept_invite(*group_id: Union[uuid.UUID, str], identity_id: Union[uuid.UUID, str]*) → *globus_sdk.response.GlobusHTTPResponse*

Accept invite for an identity. The identity must belong to the identity set of the authenticated user.

add_member(*group_id: Union[uuid.UUID, str], identity_id: Union[uuid.UUID, str], *, role:* globus_sdk.services.groups.data.GroupRole = *GroupRole.member*) → *globus_sdk.response.GlobusHTTPResponse*

Add a list of identities to a group with the given role.

approve_pending(*group_id: Union[uuid.UUID, str]*, *identity_id: Union[uuid.UUID, str]*) → *globus_sdk.response.GlobusHTTPResponse*

Approve a list of identities with pending join requests.

create_group(*name: str*, *description: str*, *, *parent_id: Optional[Union[uuid.UUID, str]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

Create a group with the given name. If a parent id is included, the group will be a subgroup of the given parent group.

decline_invite(*group_id: Union[uuid.UUID, str]*, *identity_id: Union[uuid.UUID, str]*) → *globus_sdk.response.GlobusHTTPResponse*

Decline an invitation for a given identity.

invite_member(*group_id: Union[uuid.UUID, str]*, *identity_id: Union[uuid.UUID, str]*, *, *role:* globus_sdk.services.groups.data.GroupRole *= GroupRole.member*) → *globus_sdk.response.GlobusHTTPResponse*

Invite an identity to a group with the given role.

join(*group_id: Union[uuid.UUID, str]*, *identity_id: Union[uuid.UUID, str]*) → *globus_sdk.response.GlobusHTTPResponse*

Join a group with the given identity. The identity must be in the authenticated users identity set.

leave(*group_id: Union[uuid.UUID, str]*, *identity_id: Union[uuid.UUID, str]*) → *globus_sdk.response.GlobusHTTPResponse*

Leave a group that one of the identities in the authenticated user's identity set is a member of.

reject_join_request(*group_id: Union[uuid.UUID, str]*, *identity_id: Union[uuid.UUID, str]*) → *globus_sdk.response.GlobusHTTPResponse*

Reject a member that has requested to join the group.

remove_member(*group_id: Union[uuid.UUID, str]*, *identity_id: Union[uuid.UUID, str]*) → *globus_sdk.response.GlobusHTTPResponse*

Remove members from a group. This must be done as an admin or manager of the group.

request_join(*group_id: Union[uuid.UUID, str]*, *identity_id: Union[uuid.UUID, str]*) → *globus_sdk.response.GlobusHTTPResponse*

Request to join a group.

set_group_policies(*group_id: Union[uuid.UUID, str]*, *, *is_high_assurance: bool*, *group_visibility:* globus_sdk.services.groups.data.GroupVisibility, *group_members_visibility:* globus_sdk.services.groups.data.GroupMemberVisibility, *join_requests: bool*, *signup_fields: Iterable[*globus_sdk.services.groups.data.GroupRequiredSignupFields*]*, *authentication_assurance_timeout: Optional[int] = None*) → *globus_sdk.response.GlobusHTTPResponse*

Set the group policies for the given group.

Client Errors

When an error occurs, a `GroupsClient` will raise this type of error:

class globus_sdk.**GroupsAPIError**(*r: requests.models.Response*, **args: Any*, ***kw: Any*)

Bases: *globus_sdk.exc.api.GlobusAPIError*

Error class for the Globus Groups Service.

1.3.3 Globus Search

class globus_sdk.**SearchClient**(*, *environment: Optional[str] = None*, *base_url: Optional[str] = None*, *authorizer: Optional[globus_sdk.authorizers.base.GlobusAuthorizer] = None*, *app_name: Optional[str] = None*, *transport_params: Optional[Dict] = None*)

Bases: *globus_sdk.client.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_subject()*
- *get_task()*
- *get_task_list()*
- *ingest()*
- *post_search()*
- *search()*, `paginated.search()`
- *update_entry()*

get_index(*index_id: Union[uuid.UUID, str]*, ***, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

`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: Union[uuid.UUID, str]*, *q: str*, ***, *offset: int = 0*, *limit: int = 10*, *advanced: bool = False*, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

`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)
```

Paginated Usage

This method supports paginated access. To use the paginated variant, give the same arguments as normal, but prefix the method name with `paginated`, as in

```
>>> client.paginated.search(...)
```

For more information, see *how to make paginated calls*.

External Documentation

See GET Search Query in the API documentation for details.

post_search(*index_id: Union[uuid.UUID, str]*, *data: Union[Dict[str, Any], globus_sdk.services.search.data.SearchQuery]*) → *globus_sdk.response.GlobusHTTPResponse*

`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: Union[uuid.UUID, str]*, *data: Dict[str, Any]*) → *globus_sdk.response.GlobusHTTPResponse*

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: Union[uuid.UUID, str]*, *data: Dict[str, Any]*) → *globus_sdk.response.GlobusHTTPResponse*

POST /v1/index/<index_id>/delete_by_query

Examples

```
>>> sc = globus_sdk.SearchClient(...)
>>> query_data = {
>>>   "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: Union[uuid.UUID, str]*, *subject: str*, ***, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

`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: Union[uuid.UUID, str]*, *subject: str*, ***, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

`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: Union[uuid.UUID, str]*, *subject: str*, ***, *entry_id: Optional[str] = None*, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

`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: Union[uuid.UUID, str]*, *data: Dict[str, Any]*) → *globus_sdk.response.GlobusHTTPResponse*

`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: Union[uuid.UUID, str]*, *data: Dict[str, Any]*) → *globus_sdk.response.GlobusHTTPResponse*

`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",
```

(continues on next page)

(continued from previous page)

```
>>>     "visible_to": ["public"],
>>>     "content": {
>>>         "foo/bar": "some val"
>>>     }
>>> })
```

External Documentation

See Update Entry in the API documentation for details.

delete_entry(*index_id: Union[uuid.UUID, str]*, *subject: str*, ***, *entry_id: Optional[str] = None*, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

`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_task(*task_id: Union[uuid.UUID, str]*, ***, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

`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'])
```

External Documentation

See Get Task in the API documentation for details.

get_task_list(*index_id: Union[uuid.UUID, str]*, ***, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

`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'])
```

External Documentation

See Task List in the API documentation for details.

Helper Objects

class globus_sdk.**SearchQuery**(*q: Optional[str] = None*, ***, *limit: Optional[int] = None*, *offset: Optional[int] = None*, *advanced: Optional[bool] = None*, *additional_fields: Optional[Dict[str, Any]] = None*)

Bases: *globus_sdk.utils.PayloadWrapper*

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.**SearchAPIError**(*r: requests.models.Response*)

Bases: *globus_sdk.exc.api.GlobusAPIError*

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

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

1.3.4 Globus Transfer

Client

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

class globus_sdk.**TransferClient**(***, *environment: Optional[str] = None*, *base_url: Optional[str] = None*, *authorizer: Optional[globus_sdk.authorizers.base.GlobusAuthorizer] = None*, *app_name: Optional[str] = None*, *transport_params: Optional[Dict] = None*)

Bases: *globus_sdk.client.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.

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

Paginated Calls

Methods which support pagination can be called as paginated or unpaginated methods. If the method name is `TransferClient.foo`, the paginated version is `TransferClient.paginated.foo`. Using `TransferClient.endpoint_search` as an example:

```
from globus_sdk import TransferClient
tc = TransferClient(...)

# this is the unpaginated version
for x in tc.endpoint_search("tutorial"):
    print("Endpoint ID: {}".format(x["id"]))

# this is the paginated version
for page in tc.paginated.endpoint_search("testdata"):
    for x in page:
        print("Endpoint ID: {}".format(x["id"]))
```

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()*, paginated.endpoint_manager_task_event_list()
- *endpoint_manager_task_list()*, paginated.endpoint_manager_task_list()
- *endpoint_manager_task_pause_info()*
- *endpoint_manager_task_skipped_errors()*
- *endpoint_manager_task_successful_transfers()*, paginated.endpoint_manager_task_successful_transfers()
- *endpoint_manager_update_pause_rule()*
- *endpoint_role_list()*
- *endpoint_search()*, paginated.endpoint_search()
- *endpoint_server_list()*
- *get_bookmark()*
- *get_endpoint()*
- *get_endpoint_acl_rule()*
- *get_endpoint_role()*
- *get_endpoint_server()*
- *get_shared_endpoint_list()*, paginated.get_shared_endpoint_list()
- *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()*, paginated.task_event_list()
- *task_list()*, paginated.task_list()
- *task_pause_info()*
- *task_skipped_errors()*, paginated.task_skipped_errors()
- *task_successful_transfers()*, paginated.task_successful_transfers()
- *task_wait()*
- *update_bookmark()*
- *update_endpoint()*
- *update_endpoint_acl_rule()*
- *update_endpoint_server()*
- *update_task()*

get_endpoint(*endpoint_id: Union[uuid.UUID, str]*, ***, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

GET /endpoint/<endpoint_id>

Parameters

- **endpoint_id** (*str or UUID*) – ID of endpoint to lookup
- **query_params** (*dict, optional*) – Any additional parameters will be passed through as query params.

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 API documentation for details.

update_endpoint(*endpoint_id: Union[uuid.UUID, str]*, *data: Dict[str, Any]*, ***, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

PUT /endpoint/<endpoint_id>

Parameters

- **endpoint_id** (*str or UUID*) – ID of endpoint to lookup
- **data** (*dict*) – A partial endpoint document with fields to update
- **query_params** (*dict, optional*) – Any additional parameters will be passed through as query params.

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 API documentation for details.

create_endpoint(*data: Dict[str, Any]*) → *globus_sdk.response.GlobusHTTPResponse*

```
POST /endpoint/<endpoint_id>
```

Parameters **data** (*dict*) – An endpoint document with fields for the new endpoint

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 API documentation for details.

delete_endpoint(*endpoint_id: Union[uuid.UUID, str]*) → *globus_sdk.response.GlobusHTTPResponse*

```
DELETE /endpoint/<endpoint_id>
```

Parameters **endpoint_id** (*str or UUID*) – ID of endpoint to delete

Examples

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

External Documentation

See Delete Endpoint by ID in the API documentation for details.

endpoint_search(*filter_fulltext: Optional[str] = None*, *, *filter_scope: Optional[str] = None*, *filter_owner_id: Optional[str] = None*, *filter_host_endpoint: Optional[Union[uuid.UUID, str]] = None*, *filter_non_functional: Optional[bool] = None*, *limit: Optional[int] = None*, *offset: Optional[int] = None*, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.services.transfer.response.iterable.IterableTransferResponse*

```
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)

- **filter_owner_id** (*str, optional*) – Limit search to endpoints owned by the specified Globus Auth identity. Conflicts with scopes 'my-endpoints', 'my-gcp-endpoints', and 'shared-by-me'.
- **filter_host_endpoint** (*str, optional*) – Limit search to endpoints hosted by the specified endpoint. May cause BadRequest or PermissionDenied errors if the endpoint ID given is not valid for this operation.
- **filter_non_functional** (*bool, optional*) – Limit search to endpoints which have the 'non_functional' flag set to True or False.
- **limit** (*int, optional*) – limit the number of results
- **offset** (*int, optional*) – offset used in paging
- **query_params** (*dict, optional*) – Any additional parameters will be passed through as query params.

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']))
```

It is important to be aware that the Endpoint Search API limits you to 1000 results for any search query.

Paginated Usage

This method supports paginated access. To use the paginated variant, give the same arguments as normal, but prefix the method name with `paginated`, as in

```
>>> client.paginated.endpoint_search(...)
```

For more information, see *how to make paginated calls*.

External Documentation

See Endpoint Search in the API documentation for details.

endpoint_autoactivate(*endpoint_id: Union[uuid.UUID, str]*, *, *if_expires_in: Optional[int] = None*, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

`POST /endpoint/<endpoint_id>/autoactivate`

Parameters

- **endpoint_id** (*str or UUID*) – The ID of the endpoint to autoactivate
- **if_expires_in** (*int, optional*) – A number of seconds. Autoactivation will only be attempted if the current activation expires within this timeframe. Otherwise, autoactivation will succeed with a code of 'AlreadyActivated'

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

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:"
>>>         f"https://app.globus.org/file-manager?origin_id={ep_id}"
>>>     )
>>>     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 API documentation for details.

endpoint_deactivate(*endpoint_id: Union[uuid.UUID, str]*, *, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

`POST /endpoint/<endpoint_id>/deactivate`

Parameters

- **endpoint_id** (*str or UUID*) – The ID of the endpoint to deactivate

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

External Documentation

See Deactivate Endpoint in the API documentation for details.

endpoint_activate(*endpoint_id: Union[uuid.UUID, str]*, ***, *requirements_data: Optional[dict]*, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

`POST /endpoint/<endpoint_id>/activate`

Parameters

- **endpoint_id** (*str or UUID*) – The ID of the endpoint to activate
- **query_params** (*dict, optional*) – Any additional parameters will be passed through as query params.

Pram requirements_data Filled in activation requirements data, as can be fetched from *endpoint_get_activation_requirements()*. Only the fields for the activation type being used need to be filled in.

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

External Documentation

See Activate Endpoint in the API documentation for details.

endpoint_get_activation_requirements(*endpoint_id: Union[uuid.UUID, str]*, ***, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.services.transfer.response.activation.ActivationRequirementsRespons*

`GET /endpoint/<endpoint_id>/activation_requirements`

Parameters

- **endpoint_id** (*str or UUID*) – The ID of the endpoint whose activation requirements data is being looked up
- **query_params** (*dict, optional*) – Any additional parameters will be passed through as query params.

External Documentation

See Get Activation Requirements in the API documentation for details.

my_effective_pause_rule_list(*endpoint_id: Union[uuid.UUID, str]*, ***, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.services.transfer.response.iterable.IterableTransferResponse*

`GET /endpoint/<endpoint_id>/my_effective_pause_rule_list`

Parameters

- **endpoint_id** (*str or UUID*) – the endpoint on which the current user's effective pause rules are fetched
- **query_params** (*dict, optional*) – Additional passthrough query parameters

External Documentation

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

my_shared_endpoint_list(*endpoint_id: Union[uuid.UUID, str]*, ***, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.services.transfer.response.iterable.IterableTransferResponse*

`GET /endpoint/<endpoint_id>/my_shared_endpoint_list`

Parameters

- **endpoint_id** (*str or UUID*) – the host endpoint whose shares are listed
- **query_params** (*dict, optional*) – Additional passthrough query parameters

Get a list of shared endpoints for which the user has `administrator` or `access_manager` on a given host endpoint.

External Documentation

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

get_shared_endpoint_list(*endpoint_id: Union[uuid.UUID, str]*, ***, *max_results: Optional[int] = None*, *next_token: Optional[str] = None*, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.services.transfer.response.iterable.IterableTransferResponse*

`GET /endpoint/<endpoint_id>/shared_endpoint_list`

Parameters

- **endpoint_id** (*str or UUID*) – the host endpoint whose shares are listed
- **max_results** (*int, optional*) – cap to the number of results
- **next_token** (*str, optional*) – token used for paging
- **query_params** – Any additional parameters will be passed through as query params.

Get a list of all shared endpoints on a given host endpoint.

Paginated Usage

This method supports paginated access. To use the paginated variant, give the same arguments as normal, but prefix the method name with `paginated`, as in

```
>>> client.paginated.get_shared_endpoint_list(...)
```

For more information, see *how to make paginated calls*.

External Documentation

See Get shared endpoint list (2) in the API documentation for details.

create_shared_endpoint(*data: Dict[str, Any]*) → *globus_sdk.response.GlobusHTTPResponse*

`POST /shared_endpoint`

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

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,
>>>   # optionally specify additional endpoint fields
```

(continues on next page)

(continued from previous page)

```
>>>   "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 API documentation for details.

endpoint_server_list(*endpoint_id: Union[uuid.UUID, str]*, *, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.services.transfer.response.iterable.IterableTransferResponse*

`GET /endpoint/<endpoint_id>/server_list`

Parameters **endpoint_id** (*str or UUID*) – The endpoint whose servers are being listed

External Documentation

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

get_endpoint_server(*endpoint_id: Union[uuid.UUID, str]*, *server_id: Union[int, str]*, *, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

`GET /endpoint/<endpoint_id>/server/<server_id>`

Parameters

- **endpoint_id** (*str or UUID*) – The endpoint under which the server is registered
- **server_id** (*str or int*) – The ID of the server
- **query_params** (*dict, optional*) – Additional passthrough query parameters

External Documentation

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

add_endpoint_server(*endpoint_id: Union[uuid.UUID, str]*, *server_data: Dict[str, Any]*) → *globus_sdk.response.GlobusHTTPResponse*

`POST /endpoint/<endpoint_id>/server`

Parameters

- **endpoint_id** (*str or UUID*) – The endpoint under which the server is being registered
- **server_data** (*dict*) – Fields for the new server, as a server document

External Documentation

See Add endpoint server in the API documentation for details.

update_endpoint_server(*endpoint_id: Union[uuid.UUID, str]*, *server_id: Union[int, str]*, *server_data: Dict[str, Any]*) → *globus_sdk.response.GlobusHTTPResponse*

`PUT /endpoint/<endpoint_id>/server/<server_id>`

Parameters

- **endpoint_id** (*str or UUID*) – The endpoint under which the server is registered
- **server_id** (*str or int*) – The ID of the server to update
- **server_data** (*dict*) – Fields on the server to update, as a partial server document

External Documentation

See Update endpoint server by ID in the API documentation for details.

delete_endpoint_server(*endpoint_id: Union[uuid.UUID, str]*, *server_id: Union[int, str]*) → *globus_sdk.response.GlobusHTTPResponse*

`DELETE /endpoint/<endpoint_id>/server/<server_id>`

Parameters

- **endpoint_id** (*str or UUID*) – The endpoint under which the server is registered
- **server_id** (*str or int*) – The ID of the server to delete

External Documentation

See Delete endpoint server by ID in the API documentation for details.

endpoint_role_list(*endpoint_id: Union[uuid.UUID, str]*, ***, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.services.transfer.response.iterable.IterableTransferResponse*

`GET /endpoint/<endpoint_id>/role_list`

Parameters **endpoint_id** (*str or UUID*) – The endpoint whose roles are being listed

External Documentation

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

add_endpoint_role(*endpoint_id: Union[uuid.UUID, str]*, *role_data: Dict[str, Any]*) → *globus_sdk.response.GlobusHTTPResponse*

`POST /endpoint/<endpoint_id>/role`

Parameters

- **endpoint_id** (*str or UUID*) – The endpoint on which the role is being added
- **role_data** (*dict*) – A role document for the new role

External Documentation

See Create endpoint role in the API documentation for details.

get_endpoint_role(*endpoint_id: Union[uuid.UUID, str]*, *role_id: str*, ***, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

`GET /endpoint/<endpoint_id>/role/<role_id>`

Parameters

- **endpoint_id** (*str or UUID*) – The endpoint on which the role applies
- **role_id** (*str*) – The ID of the role
- **query_params** (*dict, optional*) – Additional passthrough query parameters

External Documentation

See Get endpoint role by ID in the API documentation for details.

delete_endpoint_role(*endpoint_id: Union[uuid.UUID, str]*, *role_id: str*) → *globus_sdk.response.GlobusHTTPResponse*

`DELETE /endpoint/<endpoint_id>/role/<role_id>`

Parameters

- **endpoint_id** (*str or UUID*) – The endpoint on which the role applies
- **role_id** (*str*) – The ID of the role to delete

External Documentation

See Delete endpoint role by ID in the API documentation for details.

endpoint_acl_list(*endpoint_id: Union[uuid.UUID, str]*, ***, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.services.transfer.response.iterable.IterableTransferResponse*

`GET /endpoint/<endpoint_id>/access_list`

Parameters

- **endpoint_id** (*str or UUID*) – The endpoint whose ACLs are being listed
- **query_params** (*dict, optional*) – Additional passthrough query parameters

External Documentation

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

get_endpoint_acl_rule(*endpoint_id: Union[uuid.UUID, str]*, *rule_id: str*, ***, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

`GET /endpoint/<endpoint_id>/access/<rule_id>`

Parameters

- **endpoint_id** (*str or UUID*) – The endpoint on which the access rule applies
- **rule_id** (*str*) – The ID of the rule to fetch
- **query_params** (*dict, optional*) – Additional passthrough query parameters

External Documentation

See Get access rule by ID in the API documentation for details.

add_endpoint_acl_rule(*endpoint_id: Union[uuid.UUID, str]*, *rule_data: Dict[str, Any]*) → *globus_sdk.response.GlobusHTTPResponse*

`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

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 API documentation for details.

update_endpoint_acl_rule(*endpoint_id: Union[uuid.UUID, str]*, *rule_id: str*, *rule_data: Dict[str, Any]*) → *globus_sdk.response.GlobusHTTPResponse*

`PUT /endpoint/<endpoint_id>/access/<rule_id>`

Parameters

- **endpoint_id** (*str or UUID*) – The endpoint on which the access rule applies
- **rule_id** (*str*) – The ID of the access rule to modify
- **rule_data** (*dict*) – A partial `access` document containing fields to update

External Documentation

See Update access rule in the API documentation for details.

delete_endpoint_acl_rule(*endpoint_id: Union[uuid.UUID, str]*, *rule_id: str*) → *globus_sdk.response.GlobusHTTPResponse*

`DELETE /endpoint/<endpoint_id>/access/<rule_id>`

Parameters

- **endpoint_id** (*str or UUID*) – The endpoint on which the access rule applies
- **rule_id** (*str*) – The ID of the access rule to remove

External Documentation

See Delete access rule in the API documentation for details.

bookmark_list(*, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.services.transfer.response.iterable.IterableTransferResponse*

`GET /bookmark_list`

Parameters **query_params** (*dict, optional*) – Additional passthrough query parameters

External Documentation

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

create_bookmark(*bookmark_data: Dict[str, Any]*) → *globus_sdk.response.GlobusHTTPResponse*

`POST /bookmark`

Parameters **bookmark_data** (*dict*) – A bookmark document for the bookmark to create

External Documentation

See Create bookmark in the API documentation for details.

get_bookmark(*bookmark_id: Union[uuid.UUID, str]*, *, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

`GET /bookmark/<bookmark_id>`

Parameters

- **bookmark_id** (*str or UUID*) – The ID of the bookmark to lookup
- **query_params** (*dict, optional*) – Additional passthrough query parameters

External Documentation

See Get bookmark by ID in the API documentation for details.

update_bookmark(*bookmark_id: Union[uuid.UUID, str]*, *bookmark_data: Dict[str, Any]*) → *globus_sdk.response.GlobusHTTPResponse*

`PUT /bookmark/<bookmark_id>`

Parameters

- **bookmark_id** (*str or UUID*) – The ID of the bookmark to modify
- **bookmark_data** (*dict*) – A partial bookmark document with fields to update

External Documentation

See Update bookmark in the API documentation for details.

delete_bookmark(*bookmark_id: Union[uuid.UUID, str]*) → *globus_sdk.response.GlobusHTTPResponse*

`DELETE /bookmark/<bookmark_id>`

Parameters **bookmark_id** (*str or UUID*) – The ID of the bookmark to delete

External Documentation

See Delete bookmark by ID in the API documentation for details.

operation_ls(*endpoint_id: Union[uuid.UUID, str]*, *path: Optional[str] = None*, ***, *show_hidden: Optional[bool] = None*, *orderby: Optional[Union[str, List[str]]] = None*, *filter: Optional[str] = None*, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.services.transfer.response.iterable.IterableTransferResponse*

`GET /operation/endpoint/<endpoint_id>/ls`

Parameters

- **endpoint_id** (*str or UUID*) – The ID of the endpoint on which to do a dir listing
- **path** (*str, optional*) – Path to a directory on the endpoint to list
- **show_hidden** (*bool, optional*) – Show hidden files (names beginning in dot). Defaults to true.
- **orderby** (*str, optional*) – One or more order-by options. Each option is either a field name or a field name followed by a space and 'ASC' or 'DESC' for ascending or descending.
- **filter** (*str, optional*) – Only return file documents that match these filter clauses. For the filter syntax, see the **External Documentation** linked below.
- **query_params** (*dict, optional*) – Additional passthrough query parameters

Examples

List with a path:

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

List with explicit ordering:

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

External Documentation

See List Directory Contents in the API documentation for details.

operation_mkdir(*endpoint_id: Union[uuid.UUID, str]*, *path: str*, ***, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

`POST /operation/endpoint/<endpoint_id>/mkdir`

Parameters

- **endpoint_id** (*str or UUID*) – The ID of the endpoint on which to create a directory
- **path** (*str*) – Path to the new directory to create
- **query_params** (*dict, optional*) – Additional passthrough query parameters

Examples

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

External Documentation

See Make Directory in the API documentation for details.

operation_rename(*endpoint_id: Union[uuid.UUID, str]*, *oldpath: str*, *newpath: str*, ***, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

`POST /operation/endpoint/<endpoint_id>/rename`

Parameters

- **endpoint_id** (*str or UUID*) – The ID of the endpoint on which to rename a file
- **oldpath** (*str*) – Path to the old filename
- **newpath** (*str*) – Path to the new filename
- **query_params** (*dict, optional*) – Additional passthrough query parameters

Examples

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

External Documentation

See Rename in the API documentation for details.

operation_symlink(*endpoint_id: Union[uuid.UUID, str]*, *symlink_target: str*, *path: str*, ***, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

`POST /operation/endpoint/<endpoint_id>/symlink`

Parameters

- **endpoint_id** (*str or UUID*) – The ID of the endpoint on which to create a symlink
- **symlink_target** (*str*) – The path referenced by the new symlink
- **path** (*str*) – The name of (path to) the new symlink
- **query_params** (*dict, optional*) – Additional passthrough query parameters

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 API documentation for details.

get_submission_id(*, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

`GET /submission_id`

Parameters **query_params** (*dict, optional*) – Additional passthrough query parameters

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 API documentation for details.

submit_transfer(*data: Union[Dict[str, Any], globus_sdk.services.transfer.data.transfer_data.TransferData]*) → *globus_sdk.response.GlobusHTTPResponse*

`POST /transfer`

Parameters **data** (*dict or TransferData*) – A transfer task document listing files and directories, and setting various options. See *TransferData* for details

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 API documentation for details.

submit_delete(*data: Union[Dict[str, Any], globus_sdk.services.transfer.data.delete_data.DeleteData]*) → *globus_sdk.response.GlobusHTTPResponse*

`POST /delete`

Parameters **data** (*dict or DeleteData*) – A delete task document listing files and directories, and setting various options. See *DeleteData* for details

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 API documentation for details.

task_list(*, *limit: Optional[int] = None*, *offset: Optional[int] = None*, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.services.transfer.response.iterable.IterableTransferResponse*

`GET /task_list`

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

Parameters

- **limit** (*int, optional*) – limit the number of results
- **offset** (*int, optional*) – offset used in paging
- **query_params** (*dict, optional*) – Additional passthrough query parameters

Examples

Fetch 10 tasks and print some basic info:

```
>>> tc = TransferClient(...)
>>> for task in tc.task_list(limit=10):
>>>     print("Task({}): {} -> {}".format(
>>>         task["task_id"], task["source_endpoint"],
>>>         task["destination_endpoint"]))
```

Paginated Usage

This method supports paginated access. To use the paginated variant, give the same arguments as normal, but prefix the method name with `paginated`, as in

```
>>> client.paginated.task_list(...)
```

For more information, see *how to make paginated calls*.

External Documentation

See Task list in the API documentation for details.

task_event_list(*task_id: Union[uuid.UUID, str]*, *, *limit: Optional[int] = None*, *offset: Optional[int] = None*, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.services.transfer.response.iterable.IterableTransferResponse*

`GET /task/<task_id>/event_list`

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

Parameters

- **task_id** (*str or UUID*) – The ID of the task to inspect
- **limit** (*int, optional*) – limit the number of results
- **offset** (*int, optional*) – offset used in paging
- **query_params** (*dict, optional*) – Additional passthrough query parameters

Examples

Fetch 10 events and print some basic info:

```
>>> tc = TransferClient(...)
>>> task_id = ...
>>> for event in tc.task_event_list(task_id, limit=10):
>>>     print("Event on Task({}) at {}:\n{}".format(
>>>         task_id, event["time"], event["description"])
```

Paginated Usage

This method supports paginated access. To use the paginated variant, give the same arguments as normal, but prefix the method name with `paginated`, as in

```
>>> client.paginated.task_event_list(...)
```

For more information, see *how to make paginated calls*.

External Documentation

See Get event list in the API documentation for details.

get_task(*task_id: Union[uuid.UUID, str]*, ***, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

`GET /task/<task_id>`

Parameters

- **task_id** (*str or UUID*) – The ID of the task to inspect
- **query_params** (*dict, optional*) – Additional passthrough query parameters

External Documentation

See Get task by ID in the API documentation for details.

update_task(*task_id: Union[uuid.UUID, str]*, *data: Dict[str, Any]*, ***, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

`PUT /task/<task_id>`

Modify a task. Only tasks which are still running can be modified, and only the `label` and `deadline` fields can be updated.

Parameters

- **task_id** (*str or UUID*) – The ID of the task to modify
- **data** (*dict*) – A partial task document with fields to update
- **query_params** (*dict, optional*) – Additional passthrough query parameters

External Documentation

See Update task by ID in the API documentation for details.

cancel_task(*task_id: Union[uuid.UUID, str]*) → *globus_sdk.response.GlobusHTTPResponse*

`POST /task/<task_id>/cancel`

Cancel a task which is still running.

Parameters **task_id** (*str or UUID*) – The ID of the task to cancel

External Documentation

See Cancel task by ID in the API documentation for details.

task_wait(*task_id: Union[uuid.UUID, str]*, *, *timeout: int = 10*, *polling_interval: int = 10*) → bool

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 or UUID*) – 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: Union[uuid.UUID, str]*, *, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

`GET /task/<task_id>/pause_info`

Get info about why a task is paused or about to be paused.

Parameters

- **task_id** (*str or UUID*) – The ID of the task to inspect
- **query_params** (*dict, optional*) – Additional passthrough query parameters

External Documentation

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

task_successful_transfers(*task_id: Union[uuid.UUID, str]*, ***, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.services.transfer.response.iterable.IterableTransferResponse*

`GET /task/<task_id>/successful_transfers`

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.

Parameters

- **task_id** (*str or UUID*) – The ID of the task to inspect
- **query_params** (*dict, optional*) – Additional passthrough query parameters

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"]))
```

Paginated Usage

This method supports paginated access. To use the paginated variant, give the same arguments as normal, but prefix the method name with `paginated`, as in

```
>>> client.paginated.task_successful_transfers(...)
```

For more information, see *how to make paginated calls*.

External Documentation

See Get Task Successful Transfer in the API documentation for details.

task_skipped_errors(*task_id: Union[uuid.UUID, str]*, ***, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.services.transfer.response.iterable.IterableTransferResponse*

`GET /task/<task_id>/skipped_errors`

Get path and error information for all paths that were skipped due to skip_source_errors being set on a completed transfer Task.

Parameters

- **task_id** (*str or UUID*) – The ID of the task to inspect
- **query_params** (*dict, optional*) – Additional passthrough query parameters

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

Paginated Usage

This method supports paginated access. To use the paginated variant, give the same arguments as normal, but prefix the method name with `paginated`, as in

```
>>> client.paginated.task_skipped_errors(...)
```

For more information, see *how to make paginated calls*.

External Documentation

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

endpoint_manager_monitored_endpoints(*, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.services.transfer.response.iterable.IterableTransferResponse*

`GET endpoint_manager/monitored_endpoints`

Get endpoints the current user is a monitor or manager on.

Parameters **query_params** (*dict, optional*) – Additional passthrough query parameters

External Documentation

See Get monitored endpoints in the API documentation for details.

endpoint_manager_hosted_endpoint_list(*endpoint_id: Union[uuid.UUID, str]*, *, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.services.transfer.response.iterable.IterableTransferResponse*

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

Get shared endpoints hosted on the given endpoint.

Parameters

- **endpoint_id** (*str or UUID*) – The ID of the host endpoint
- **query_params** (*dict, optional*) – Additional passthrough query parameters

External Documentation

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

endpoint_manager_get_endpoint(*endpoint_id: Union[uuid.UUID, str]*, *, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

`GET /endpoint_manager/endpoint/<endpoint_id>`

Get endpoint details as an admin.

Parameters

- **endpoint_id** (*str or UUID*) – The ID of the endpoint
- **query_params** (*dict, optional*) – Additional passthrough query parameters

External Documentation

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

endpoint_manager_acl_list(*endpoint_id: Union[uuid.UUID, str]*, ***, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.services.transfer.response.iterable.IterableTransferResponse*

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

Get a list of access control rules on specified endpoint as an admin.

Parameters

- **endpoint_id** (*str or UUID*) – The ID of the endpoint
- **query_params** (*dict, optional*) – Additional passthrough query parameters

External Documentation

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

endpoint_manager_task_list(***, *filter_status: Optional[Union[Iterable[str], str]] = None*, *filter_task_id: Optional[Union[Iterable[Union[uuid.UUID, str]], uuid.UUID, str]] = None*, *filter_owner_id: Optional[Union[uuid.UUID, str]] = None*, *filter_endpoint: Optional[Union[uuid.UUID, str]] = None*, *filter_is_paused: Optional[bool] = None*, *filter_completion_time: Union[None, str, Tuple[Union[str, datetime.datetime], Union[str, datetime.datetime]]] = None*, *filter_min_faults: Optional[int] = None*, *filter_local_user: Optional[str] = None*, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.services.transfer.response.iterable.IterableTransferResponse*

`GET endpoint_manager/task_list`

Get a list of tasks visible via `activity_monitor` role, as opposed to tasks owned by the current user.

For any query that doesn't specify a `filter_status` that is a subset of `("ACTIVE", "INACTIVE")`, at least one of `filter_task_id` or `filter_endpoint` is required.

Parameters

- **filter_status** (*str or iterable of str, optional*) – Return only tasks with any of the specified statuses Note that in-progress tasks will have status `"ACTIVE"` or `"INACTIVE"`, and completed tasks will have status `"SUCCEEDED"` or `"FAILED"`.
- **filter_task_id** (*str, UUID, or iterable of str or UUID, optional*) – Return only tasks with any of the specified ids. If any of the specified tasks do not involve an endpoint the user has an appropriate role for, a `PermissionDenied` error will be returned. This filter can't be combined with any other filter. If another filter is passed, a `BadRequest` will be returned. (limit: 50 task IDs)
- **filter_owner_id** (*str or UUID, optional*) – A Globus Auth identity id. Limit results to tasks submitted by the specified identity, or linked to the specified identity, at submit time. Returns `UserNotFound` if the identity does not exist or has never used the Globus Transfer service. If no tasks were submitted by this user to an endpoint the current user has an appropriate role on, an empty result set will be returned. Unless filtering for running tasks (i.e. `filter_status` is a subset of `("ACTIVE", "INACTIVE")`, `filter_endpoint` is required when using `filter_owner_id`.
- **filter_endpoint** (*str or UUID, optional*) – Single endpoint id. Return only tasks with a matching source or destination endpoint or matching source or destination host endpoint.
- **filter_is_paused** (*bool, optional*) – Return only tasks with the specified `is_paused` value. Requires that `filter_status` is also passed and contains a subset of `"ACTIVE"` and `"INACTIVE"`. Completed tasks always have `is_paused` equal to `False`

and filtering on their paused state is not useful and not supported. Note that pausing is an async operation, and after a pause rule is inserted it will take time before the `is_paused` flag is set on all affected tasks. Tasks paused by id will have the `is_paused` flag set immediately.

- **filter_completion_time** (*str, tuple of str, or tuple of datetime, optional*) – Start and end date-times separated by a comma, or provided as a tuple of strings or datetime objects. Returns only completed tasks with `completion_time` in the specified range. Date strings should be specified in one of the following ISO 8601 formats: `YYYY-MM-DDTHH:MM:SS`, `YYYY-MM-DDTHH:MM:SS+/-HH:MM`, or `YYYY-MM-DDTHH:MM:SSZ`. If no timezone is specified, UTC is assumed. A space can be used between the date and time instead of `T`. A blank string may be used for either the start or end (but not both) to indicate no limit on that side. If the end date is blank, the filter will also include all active tasks, since they will complete some time in the future.
- **filter_min_faults** (*int, optional*) – Minimum number of cumulative faults, inclusive. Return only tasks with `faults >= N`, where `N` is the filter value. Use `filter_min_faults=1` to find all tasks with at least one fault. Note that many errors are not fatal and the task may still be successful even if `faults >= 1`.
- **filter_local_user** (*str, optional*) – A valid username for the target system running the endpoint, as a utf8 encoded string. Requires that `filter_endpoint` is also set. Return only tasks that have successfully fetched the local user from the endpoint, and match the values of `filter_endpoint` and `filter_local_user` on the source or on the destination.
- **query_params** (*dict, optional*) – Additional passthrough query parameters

Examples

Fetch some tasks and print some basic info:

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

Paginated Usage

This method supports paginated access. To use the paginated variant, give the same arguments as normal, but prefix the method name with `paginated`, as in

```
>>> client.paginated.endpoint_manager_task_list(...)
```

For more information, see *how to make paginated calls*.

External Documentation

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

endpoint_manager_get_task(*task_id: Union[uuid.UUID, str]*, ***, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

`GET /endpoint_manager/task/<task_id>`

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

Parameters

- **task_id** (*str or UUID*) – The ID of the task to inspect
- **query_params** (*dict, optional*) – Additional passthrough query parameters

External Documentation

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

endpoint_manager_task_event_list(*task_id: Union[uuid.UUID, str]*, ***, *limit: Optional[int] = None*, *offset: Optional[int] = None*, *filter_is_error: Optional[bool] = None*, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.services.transfer.response.iterable.IterableTransferResponse*

`GET /task/<task_id>/event_list`

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.

Parameters

- **task_id** (*str or UUID*) – The ID of the task to inspect
- **limit** (*int, optional*) – limit the number of results
- **offset** – offset used in paging
- **filter_is_error** (*bool, optional*) – Return only events that are errors. A value of `False` (returning only non-errors) is not supported. By default all events are returned.
- **query_params** (*dict, optional*) – Additional passthrough query parameters

Paginated Usage

This method supports paginated access. To use the paginated variant, give the same arguments as normal, but prefix the method name with `paginated`, as in

```
>>> client.paginated.endpoint_manager_task_event_list(...)
```

For more information, see *how to make paginated calls*.

External Documentation

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

endpoint_manager_task_pause_info(*task_id: Union[uuid.UUID, str]*, ***, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

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

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

Parameters

- **task_id** (*str or UUID*) – The ID of the task to inspect
- **query_params** (*dict, optional*) – Additional passthrough query parameters

External Documentation

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

endpoint_manager_task_successful_transfers(*task_id: Union[uuid.UUID, str]*, ***, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.services.transfer.response.iterable.IterableTransferResponse*

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

Get the successful file transfers for a completed Task as an admin.

Parameters

- **task_id** (*str or UUID*) – The ID of the task to inspect
- **query_params** (*dict, optional*) – Additional passthrough query parameters

Paginated Usage

This method supports paginated access. To use the paginated variant, give the same arguments as normal, but prefix the method name with `paginated`, as in

```
>>> client.paginated.endpoint_manager_task_successful_transfers(...)
```

For more information, see *how to make paginated calls*.

External Documentation

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

endpoint_manager_task_skipped_errors(*task_id: Union[uuid.UUID, str]*, ***, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.services.transfer.response.iterable.IterableTransferResponse*

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

Get skipped errors for a completed Task as an admin.

Parameters

- **task_id** (*str*) – The ID of the task to inspect
- **query_params** (*dict, optional*) – Additional passthrough query parameters

External Documentation

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

endpoint_manager_cancel_tasks(*task_ids: Iterable[Union[uuid.UUID, str]]*, *message: str*, ***, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

`POST /endpoint_manager/admin_cancel`

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

Parameters

- **task_ids** (*iterable of str or UUID*) – List of task ids to cancel
- **message** (*str*) – Message given to all users whose tasks have been canceled
- **query_params** (*dict, optional*) – Additional passthrough query parameters

External Documentation

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

endpoint_manager_cancel_status(*admin_cancel_id: Union[uuid.UUID, str]*, ***, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

`GET /endpoint_manager/admin_cancel/<admin_cancel_id>`

Get the status of an an admin cancel (result of endpoint_manager_cancel_tasks).

Parameters

- **admin_cancel_id** (*str or UUID*) – The ID of the the cancel job to inspect
- **query_params** (*dict, optional*) – Additional passthrough query parameters

External Documentation

See Get cancel status by ID in the API documentation for details.

endpoint_manager_pause_tasks(*task_ids: Iterable[Union[uuid.UUID, str]]*, *message: str*, ***, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

`POST /endpoint_manager/admin_pause`

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

Parameters

- **task_ids** (*iterable of str or UUID*) – List of task ids to pause
- **message** (*str*) – Message given to all users whose tasks have been paused
- **query_params** (*dict, optional*) – Additional passthrough query parameters

External Documentation

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

endpoint_manager_resume_tasks(*task_ids: Iterable[Union[uuid.UUID, str]]*, ***, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

`POST /endpoint_manager/admin_resume`

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

Parameters

- **task_ids** (*iterable of str or UUID*) – List of task ids to resume
- **query_params** (*dict, optional*) – Additional passthrough query parameters

External Documentation

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

endpoint_manager_pause_rule_list(***, *filter_endpoint: Optional[Union[uuid.UUID, str]] = None*, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.services.transfer.response.iterable.IterableTransferResponse*

`GET /endpoint_manager/pause_rule_list`

Get a list of pause rules on endpoints that the current user has the activity monitor effective role on.

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.

- **query_params** (*dict, optional*) – Additional passthrough query parameters

External Documentation

See Get pause rules in the API documentation for details.

endpoint_manager_create_pause_rule(*data: Optional[Dict[str, Any]]*) → *globus_sdk.response.GlobusHTTPResponse*

`POST /endpoint_manager/pause_rule`

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

Parameters **data** (*dict*) – A pause rule document describing the rule to create

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 API documentation for details.

endpoint_manager_get_pause_rule(*pause_rule_id: Union[uuid.UUID, str]*, *, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

`GET /endpoint_manager/pause_rule/<pause_rule_id>`

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

Parameters

- **pause_rule_id** (*str*) – ID of pause rule to get
- **query_params** (*dict, optional*) – Additional passthrough query parameters

External Documentation

See Get pause rule in the API documentation for details.

endpoint_manager_update_pause_rule(*pause_rule_id: Union[uuid.UUID, str]*, *data: Optional[Dict[str, Any]]*) → *globus_sdk.response.GlobusHTTPResponse*

`PUT /endpoint_manager/pause_rule/<pause_rule_id>`

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.

Parameters

- **pause_rule_id** (*str*) – The ID of the pause rule to update
- **data** (*dict*) – A partial pause rule document with fields to update

Examples

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

endpoint_manager_delete_pause_rule(*pause_rule_id: Union[uuid.UUID, str]*, ***, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

`DELETE /endpoint_manager/pause_rule/<pause_rule_id>`

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

Parameters

- **pause_rule_id** (*str*) – The ID of the pause rule to delete
- **query_params** (*dict, optional*) – Additional passthrough query parameters

External Documentation

See Delete pause rule in the API 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: globus_sdk.TransferClient*, *source_endpoint: Union[uuid.UUID, str]*, *destination_endpoint: Union[uuid.UUID, str]*, ***, *label: Optional[str] = None*, *submission_id: Optional[Union[uuid.UUID, str]] = None*, *sync_level: Optional[str] = None*, *verify_checksum: bool = False*, *preserve_timestamp: bool = False*, *encrypt_data: bool = False*, *deadline: Optional[Union[str, datetime.datetime]] = None*, *skip_source_errors: bool = False*, *fail_on_quota_errors: bool = False*, *recursive_symlinks: str = 'ignore'*, *delete_destination_extra: bool = False*, *additional_fields: Optional[Dict[str, Any]] = None*)

Bases: *globus_sdk.utils.PayloadWrapper*

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 or UUID*) – The endpoint ID of the source endpoint

- **destination_endpoint** (*str or UUID*) – The endpoint ID of the destination endpoint
- **label** (*str, optional*) – A string label for the Task
- **submission_id** (*str or UUID, 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`]
- **delete_destination_extra** – Delete files, directories, and symlinks on the destination endpoint which don't exist on the source endpoint or are a different type. Only applies for recursive directory transfers. [default: `False`]
- **delete_destination_extra** – bool, optional
- **additional_fields** (*dict, optional*) – additional fields to be added to the transfer document. Mostly intended for internal use

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 modified 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: str*, *destination_path: str*, ***, *recursive: bool = False*, *external_checksum: Optional[str] = None*, *checksum_algorithm: Optional[str] = None*, *additional_fields: Optional[Dict[str, Any]] = None*) → None

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

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 transferred
- **destination_path** (*str*) – Path to the source directory or file will be transferred to
- **recursive** (*bool*) – Set to True if the target at source path is a directory
- **external_checksum** (*str, optional*) – A checksum to verify both source file and destination file integrity. The checksum will be verified after the data transfer and a failure will cause the entire task to fail. 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: str*, *destination_path: str*) → None

Add a symlink to be transferred 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 transferred

class globus_sdk.**DeleteData**(*transfer_client:* globus_sdk.TransferClient, *endpoint: Union[uuid.UUID, str]*, ***, *label: Optional[str] = None*, *submission_id: Optional[Union[uuid.UUID, str]] = None*, *recursive: bool = False*, *deadline: Optional[Union[str, datetime.datetime]] = None*, *additional_fields: Optional[Dict[str, Any]] = None*)

Bases: *globus_sdk.utils.PayloadWrapper*

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 or UUID*) – The endpoint ID which is targeted by this deletion Task
- **label** (*str, optional*) – A string label for the Task
- **submission_id** (*str or UUID, 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`
- **additional_fields** (*dict, optional*) – additional fields to be added to the delete document. Mostly intended for internal use

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: str*, ***, *additional_fields: Optional[Dict[str, Any]] = None*) → None

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.**TransferAPIError**(*r: requests.models.Response*)

Bases: *globus_sdk.exc.api.GlobusAPIError*

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

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

Transfer Responses

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

Bases: *globus_sdk.response.GlobusHTTPResponse*

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: int*, *relative_time: bool = True*) → bool

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: bool

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

Return type bool

property supports_auto_activation: bool

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: bool

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.services.transfer.response.**IterableTransferResponse**(*response: Union[requests.models.Response, GlobusHTTPResponse]*, *client: Optional[globus_sdk.BaseClient] = None*, ***, *iter_key: Optional[str] = None*)

Bases: *globus_sdk.response.IterableResponse*

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 a specified key:

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

1.3.5 Globus Connect Server API

The Globus Connect Server Manager API (GCS Manager API) runs on a Globus Connect Server Endpoint and allows management of the Endpoint, Storage Gateways, Collections, and other resources.

Unlike other Globus services, there is no single central API used to contact GCS Manager instances. Therefore, the `GCSClient` is always initialized with the FQDN (DNS name) of the GCS Endpoint. e.g. `gcs = GCSClient("abc.def.data.globus.org")`

Client

The primary interface for the GCS Manager API is the `GCSClient` class.

class globus_sdk.**GCSClient**(*gcs_address: str*, ***, *environment: Optional[str] = None*, *authorizer: Optional[globus_sdk.authorizers.base.GlobusAuthorizer] = None*, *app_name: Optional[str] = None*, *transport_params: Optional[Dict] = None*)

Bases: *globus_sdk.client.BaseClient*

A GCSClient provides communication with the GCS Manager API of a Globus Connect Server instance. For full reference, see the documentation for the GCS Manager API.

Unlike other client types, this must be provided with an address for the GCS Manager. All other arguments are the same as those for *~globus_sdk.BaseClient*.

Parameters **gcs_address** (*str*) – The FQDN (DNS name) or HTTPS URL for the GCS Manager API.

Methods

- *create_collection()*
- *delete_collection()*
- *get_collection()*
- *get_collection_list()*
- *get_gcs_collection_scopes()*
- *get_gcs_endpoint_scopes()*
- *update_collection()*

static get_gcs_endpoint_scopes(*endpoint_id: Union[uuid.UUID, str]*) → *globus_sdk.scopes.GCSEndpointScopeBuilder*

Given a GCS Endpoint ID, this helper constructs an object containing the scopes for that Endpoint.

Parameters **endpoint_id** (*UUID or str*) – The ID of the Endpoint

See documentation for *globus_sdk.scopes.GCSEndpointScopeBuilder* for more information.

static get_gcs_collection_scopes(*collection_id: Union[uuid.UUID, str]*) → *globus_sdk.scopes.GCSCollectionScopeBuilder*

Given a GCS Collection ID, this helper constructs an object containing the scopes for that Collection.

Parameters **collection_id** (*UUID or str*) – The ID of the Collection

See documentation for *globus_sdk.scopes.GCSCollectionScopeBuilder* for more information.

get_collection_list(**, include: Optional[Union[Iterable[str], str]] = None, query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.services.gcs.response.IterableGCSResponse*

`GET /collections`

Parameters

- **include** (*str or iterable of str, optional*) – Names of additional documents to include in the response
- **query_params** (*dict, optional*) – Additional passthrough query parameters

List the Collections on an Endpoint

External Documentation

See List Collections in the API documentation for details.

get_collection(*collection_id: Union[uuid.UUID, str], *, query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.services.gcs.response.UnpackingGCSResponse*

`GET /collections/{collection_id}`

Parameters

- **collection_id** (*str or UUID*) – The ID of the collection to lookup
- **query_params** (*dict, optional*) – Additional passthrough query parameters

Lookup a Collection on an Endpoint

External Documentation

See Get Collection in the API documentation for details.

create_collection(*collection_data: Union[Dict[str, Any],* globus_sdk.services.gcs.data.CollectionDocument*]*) → *globus_sdk.services.gcs.response.UnpackingGCSResponse*

`POST /collections`

Create a collection. This is used to create either a mapped or a guest collection. When created, a `collection:administrator` role for that collection will be created using the caller's identity.

In order to create a guest collection, the caller must have an identity that matches the Storage Gateway policies.

In order to create a mapped collection, the caller must have an `endpoint:administrator` or `endpoint:owner` role.

Parameters **collection_data** (*dict or CollectionDocument*) – The collection document for the new collection

External Documentation

See Create Collection in the API documentation for details.

update_collection(*collection_id: Union[uuid.UUID, str]*, *collection_data: Union[Dict[str, Any], globus_sdk.services.gcs.data.CollectionDocument]*, ***, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.services.gcs.response.UnpackingGCSResponse*

`PATCH /collections/{collection_id}`

Parameters

- **collection_id** (*str or UUID*) – The ID of the collection to update
- **collection_data** (*dict or CollectionDocument*) – The collection document for the modified collection
- **query_params** (*dict, optional*) – Additional passthrough query parameters

External Documentation

See Update Collection in the API documentation for details.

delete_collection(*collection_id: Union[uuid.UUID, str]*, ***, *query_params: Optional[Dict[str, Any]] = None*) → *globus_sdk.response.GlobusHTTPResponse*

`DELETE /collections/{collection_id}`

Parameters

- **collection_id** (*str or UUID*) – The ID of the collection to delete
- **query_params** (*dict, optional*) – Additional passthrough query parameters

External Documentation

See Delete Collection in the API documentation for details.

Helper Objects

class globus_sdk.services.gcs.data.**CollectionDocument**(***, *data_type: Optional[str] = None*, *collection_base_path: Optional[str] = None*, *contact_email: Optional[str] = None*, *contact_info: Optional[str] = None*, *default_directory: Optional[str] = None*, *department: Optional[str] = None*, *description: Optional[str] = None*, *display_name: Optional[str] = None*, *identity_id: Optional[Union[uuid.UUID, str]] = None*, *info_link: Optional[str] = None*, *organization: Optional[str] = None*, *user_message: Optional[str] = None*, *user_message_link: Optional[str] = None*, *keywords: Optional[Iterable[str]] = None*, *disable_verify: Optional[bool] = None*, *enable_https: Optional[bool] = None*, *force_encryption: Optional[bool] = None*, *force_verify: Optional[bool] = None*, *public: Optional[bool] = None*, *additional_fields: Optional[Dict[str, Any]] = None*)

Bases: *globus_sdk.utils.PayloadWrapper*, abc.ABC

This is the base class for *MappedCollectionDocument* and *GuestCollectionDocument*.

Parameters common to both of those are defined and documented here.

Parameters

- **data_type** (*str, optional*) – Explicitly set the DATA_TYPE value for this collection. Normally DATA_TYPE is deduced from the provided parameters and should not be set. To maximize compatibility with different versions of GCS, only set this value when necessary.
- **collection_base_path** (*str, optional*) – The location of the collection on its underlying storage. For a mapped collection, this is an absolute path on the storage system named by the storage_gateway_id. For a guest collection, this is a path relative to the value of the root_path attribute on the mapped collection identified by the mapped_collection_id. This parameter is optional for updates but required when creating a new collection.
- **contact_email** (*str, optional*) – Email address of the support contact for the collection
- **contact_info** (*str, optional*) – Other contact information for the collection, e.g. phone number or mailing address
- **default_directory** (*str, optional*) – Default directory when using the collection
- **department** (*str, optional*) – The department which operates the collection
- **description** (*str, optional*) – A text description of the collection
- **display_name** (*str, optional*) – Friendly name for the collection
- **identity_id** (*str or UUID, optional*) – The Globus Auth identity which acts as the owner of the collection
- **info_link** (*str, optional*) – Link for more info about the collection
- **organization** (*str, optional*) – The organization which maintains the collection
- **user_message** (*str, optional*) – A message to display to users when interacting with this collection
- **user_message_link** (*str, optional*) – A link to additional messaging for users when interacting with this collection
- **keywords** (*iterable of str, optional*) – A list of keywords used to help searches for the collection
- **disable_verify** (*bool, optional*) – Disable verification checksums on transfers to and from this collection
- **enable_https** (*bool, optional*) – Enable or disable HTTPS support (requires a managed endpoint)
- **force_encryption** (*bool, optional*) – When set to True, all transfers to and from the collection are always encrypted
- **force_verify** (*bool, optional*) – Force verification checksums on transfers to and from this collection
- **public** (*bool, optional*) – If True, the collection will be visible to other Globus users
- **additional_fields** (*dict, optional*) – Additional data for inclusion in the collection document

class globus_sdk.services.gcs.data.**GuestCollectionDocument**(*, *data_type: Optional[str] = None*, *collection_base_path: Optional[str] = None*, *contact_email: Optional[str] = None*, *contact_info: Optional[str] = None*, *default_directory: Optional[str] = None*, *department: Optional[str] = None*, *description: Optional[str] = None*, *display_name: Optional[str] = None*, *identity_id: Optional[Union[uuid.UUID, str]] = None*, *info_link: Optional[str] = None*, *organization: Optional[str] = None*, *user_message: Optional[str] = None*, *user_message_link: Optional[str] = None*, *keywords: Optional[Iterable[str]] = None*, *disable_verify: Optional[bool] = None*, *enable_https: Optional[bool] = None*, *force_encryption: Optional[bool] = None*, *force_verify: Optional[bool] = None*, *public: Optional[bool] = None*, *mapped_collection_id: Optional[Union[uuid.UUID, str]] = None*, *user_credential_id: Optional[Union[uuid.UUID, str]] = None*, *additional_fields: Optional[Dict[str, Any]] = None*)

Bases: *globus_sdk.services.gcs.data.CollectionDocument*

An object used to represent a Guest Collection for creation or update operations. The initializer supports all writable fields on Guest Collections but does not include read-only fields like `id`.

Because these documents may be used for updates, no fields are strictly required. However, GCS will require the following fields for creation:

- `mapped_collection_id`
- `user_credential_id`
- `collection_base_path`

All parameters for *CollectionDocument* are supported in addition to the parameters below.

Parameters

- **mapped_collection_id** (*str or UUID*) – The ID of the mapped collection which hosts this guest collection
- **user_credential_id** (*str or UUID*) – The ID of the User Credential which is used to access data on this collection. This credential must be owned by the collection's `identity_id`.

class globus_sdk.services.gcs.data.**MappedCollectionDocument**(*, *data_type: Optional[str] = None*, *collection_base_path: Optional[str] = None*, *contact_email: Optional[str] = None*, *contact_info: Optional[str] = None*, *default_directory: Optional[str] = None*, *department: Optional[str] = None*, *description: Optional[str] = None*, *display_name: Optional[str] = None*, *identity_id: Optional[Union[uuid.UUID, str]] = None*, *info_link: Optional[str] = None*, *organization: Optional[str] = None*, *user_message: Optional[str] = None*, *user_message_link: Optional[str] = None*, *keywords: Optional[Iterable[str]] = None*, *disable_verify: Optional[bool] = None*, *enable_https: Optional[bool] = None*, *force_encryption: Optional[bool] = None*, *force_verify: Optional[bool] = None*, *public: Optional[bool] = None*, *storage_gateway_id: Optional[Union[uuid.UUID, str]] = None*, *domain_name: Optional[str] = None*, *sharing_users_allow: Optional[Iterable[str]] = None*, *sharing_users_deny: Optional[Iterable[str]] = None*, *sharing_restrict_paths: Optional[Dict[str, Any]] = None*, *allow_guest_collections: Optional[bool] = None*, *disable_anonymous_writes: Optional[bool] = None*, *policies: Optional[Dict[str, Any]] = None*, *additional_fields: Optional[Dict[str, Any]] = None*)

Bases: *globus_sdk.services.gcs.data.CollectionDocument*

An object used to represent a Mapped Collection for creation or update operations. The initializer supports all writable fields on Mapped Collections but does not include read-only fields like `id`.

Because these documents may be used for updates, no fields are strictly required. However, GCS will require the following fields for creation:

- `storage_gateway_id`
- `collection_base_path`

All parameters for *CollectionDocument* are supported in addition to the parameters below.

Parameters

- **storage_gateway_id** (*str or UUID, optional*) – The ID of the storage gateway which hosts this mapped collection. This parameter is required when creating a collection.
- **domain_name** (*str, optional*) – DNS name of the virtual host serving this collection

- **sharing_users_allow** (*iterable of str, optional*) – Connector-specific usernames allowed to create guest collections
- **sharing_users_deny** (*iterable of str, optional*) – Connector-specific usernames forbidden from creating guest collections
- **allow_guest_collections** (*bool, optional*) – Enable or disable creation and use of Guest Collections on this Mapped Collection
- **disable_anonymous_writes** (*bool, optional*) – Allow anonymous write ACLs on Guest Collections attached to this Mapped Collection. This option is only usable on non high-assurance collections
- **policies** (*dict, optional*) – Connector-specific collection policies
- **sharing_restrict_paths** (*dict, optional*) – A PathRestrictions document

Client Errors

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

class globus_sdk.**GCSAPIError**(*r: requests.models.Response*)

Bases: *globus_sdk.exc.api.GlobusAPIError*

Error class for the GCS Manager API client

GCS Responses

class globus_sdk.services.gcs.response.**IterableGCSResponse**(*response: Union[requests.models.Response, GlobusHTTPResponse], client: Optional[*globus_sdk.BaseClient*] = None*, **, iter_key: Optional[str] = None*)

Bases: *globus_sdk.response.IterableResponse*

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.services.gcs.response.**UnpackingGCSResponse**(*response: globus_sdk.response.GlobusHTTPResponse, match: Union[str, Callable[[dict], bool]]*)

Bases: *globus_sdk.response.GlobusHTTPResponse*

An "unpacking" response looks for a "data" array in the response data, which is expected to have dict elements. The "data" is traversed until the first matching object is found, and this is presented as the `data` property of the response.

The full response data is available as `full_data`.

If the expected datatype is not found in the array, or the array is missing, the `data` will be the full response data (identical to `full_data`).

Parameters **match** (*str or callable*) – Either a string containing a DATA_TYPE prefix, or an arbitrary callable which does the matching

property full_data: Any
The full, parsed JSON response data. None if the data cannot be parsed as JSON.

1.4 Scopes and ScopeBuilders

OAuth2 Scopes for various Globus services are represented by ScopeBuilder objects.

A number of pre-set scope builders are provided and populated with useful data, and they are also accessible via the relevant client classes.

1.4.1 Direct Use (as constants)

To use the scope builders directly, import from globus_sdk.scopes.

For example, one might use the Transfer "all" scope during a login flow like so:

```
import globus_sdk
from globus_sdk.scopes import TransferScopes

CLIENT_ID = "<YOUR_ID_HERE>"

client = globus_sdk.NativeAppAuthClient(CLIENT_ID)
client.oauth2_start_flow(requested_scopes=[TransferScopes.all])
...
```

1.4.2 As Client Class Attributes

Because the scopes for a token are associated with some concrete client which will use that token, it makes sense to associate a scope with a client class.

The Globus SDK does this by providing the ScopeBuilder for a service as an attribute of the client. For example,

```
import globus_sdk

CLIENT_ID = "<YOUR_ID_HERE>"

client = globus_sdk.NativeAppAuthClient(CLIENT_ID)
client.oauth2_start_flow(requested_scopes=[globus_sdk.TransferClient.scopes.all])
...

# or, potentially, after there is a concrete client
_tc = globus_sdk.TransferClient()
client.oauth2_start_flow(requested_scopes=[_tc.scopes.all])
```

1.4.3 Using a Scope Builder to Get Matching Tokens

A `ScopeBuilder` contains the resource server name used to get token data from a token response. To elaborate on the above example:

```
import globus_sdk
from globus_sdk.scopes import TransferScopes

CLIENT_ID = "<YOUR_ID_HERE>"

client = globus_sdk.NativeAppAuthClient(CLIENT_ID)
client.oauth2_start_flow(requested_scopes=[TransferScopes.all])
authorize_url = client.oauth2_get_authorize_url()
print("Please go to this URL and login:", authorize_url)
auth_code = input("Please enter the code you get after login here: ").strip()
token_response = client.oauth2_exchange_code_for_tokens(auth_code)

# use the `resource_server` of a ScopeBuilder to grab the associated token
# data from the response
tokendata = token_response.by_resource_server[TransferScopes.resource_server]
```

1.4.4 ScopeBuilder Types and Constants

class globus_sdk.scopes.**ScopeBuilder**(*resource_server: str*, *, *known_scopes: Optional[List[str]] = None*)

Bases: `object`

Utility class for creating scope strings for a specified resource server.

Parameters

- **resource_server** (*str*) – The identifier, usually a domain name or a UUID, for the resource server to return scopes for.
- **known_scopes** (*list, optional*) – A list of scope names to pre-populate on this instance. This will set attributes on the instance using the URN scope format.

url_scope_string(*scope_name: str*) → str

Return a complete string representing the scope with a given name for this client, in URL format.

Examples

```
>>> sb = ScopeBuilder("actions.globus.org")
>>> sb.url_scope_string("actions.globus.org", "hello_world")
"https://auth.globus.org/scopes/actions.globus.org/hello_world"
```

Parameters **scope_name** (*str*) – The short name for the scope involved.

urn_scope_string(*scope_name: str*) → str

Return a complete string representing the scope with a given name for this client, in the Globus Auth URN format.

Note that this module already provides many such scope strings for use with Globus services.

Examples

```
>>> sb = ScopeBuilder("transfer.api.globus.org")
>>> sb.urn_scope_string("transfer.api.globus.org", "all")
"urn:globus:auth:scope:transfer.api.globus.org:all"
```

Parameters **scope_name** (*str*) – The short name for the scope involved.

class globus_sdk.scopes.**GCSEndpointScopeBuilder**(*resource_server: str*, *, *known_scopes: Optional[List[str]] = None*)

Bases: *globus_sdk.scopes.ScopeBuilder*

A ScopeBuilder with a named property for the GCS manage_collections scope. "manage_collections" is a scope on GCS Endpoints. The resource_server string should be the GCS Endpoint ID.

Examples

```
>>> sb = GCSEndpointScopeBuilder("xyz")
>>> mc_scope = sb.manage_collections
```

class globus_sdk.scopes.**GCSCollectionScopeBuilder**(*resource_server: str*, *, *known_scopes: Optional[List[str]] = None*)

Bases: *globus_sdk.scopes.ScopeBuilder*

A ScopeBuilder with a named property for the GCS data_access scope. "data_access" is a scope on GCS Collections. The resource_server string should be the GCS Collection ID.

Examples

```
>>> sb = GCSCollectionScopeBuilder("xyz")
>>> da_scope = sb.data_access
```

globus_sdk.scopes.**AuthScopes**

Globus Auth scopes.

Various scopes are available as attributes of this object. For example, access the `view_identity_set` scope with

```
>>> AuthScopes.view_identity_set
```

Supported Scopes

- `openid`
- `email`
- `profile`
- `view_authentications`
- `view_clients`
- `view_clients_and_scopes`
- `view_identities`
- `view_identity_set`

globus_sdk.scopes.**GroupsScopes**

Groups scopes.

Various scopes are available as attributes of this object. For example, access the `all` scope with

```
>>> GroupsScopes.all
```

Supported Scopes

- `all`
- `view_my_groups_and_memberships`

`globus_sdk.scopes.`**`NexusScopes`**
Nexus scopes (internal use only).

Various scopes are available as attributes of this object. For example, access the `groups` scope with

```
>>> NexusScopes.groups
```

Supported Scopes

- `groups`

`globus_sdk.scopes.`**`SearchScopes`**
Globus Search scopes.

Various scopes are available as attributes of this object. For example, access the `all` scope with

```
>>> SearchScopes.all
```

Supported Scopes

- `all`
- `globus_connect_server`
- `ingest`
- `search`

`globus_sdk.scopes.`**`TransferScopes`**
Globus Transfer scopes.

Various scopes are available as attributes of this object. For example, access the `all` scope with

```
>>> TransferScopes.all
```

Supported Scopes

- `all`
- `gcp_install`
- `monitor_ongoing`

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**(*, *config_dir: Optional[str] = None*)

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.

Parameters **config_dir** (*str, optional*) – Path to a non-default configuration directory. On Linux, this is the same as the value passed to Globus Connect Personal's *-dir* flag (i.e. the default value is `~/.globusonline`).

property config_dir: str

The `config_dir` for this endpoint.

If no directory was given during initialization, this will be computed based on the current platform and environment.

get_owner_info() → Optional[*globus_sdk.GlobusConnectPersonalOwnerInfo*]
get_owner_info(*auth_client: None*) → Optional[*globus_sdk.GlobusConnectPersonalOwnerInfo*]
get_owner_info(*auth_client:* globus_sdk.AuthClient) → Optional[Dict[str, Any]]

Look up the local GCP information, returning a *GlobusConnectPersonalOwnerInfo* object. The result may have an `id` or `username` set (depending on the underlying data).

If you pass an AuthClient, this method will return a dict from the Get Identities API instead of the info object. This can fail (e.g. with network errors if there is no connectivity), so passing this value should be coupled with additional error handling.

In either case, the result may be `None` if the data is missing or cannot be parsed.

Note: The data returned by this method is not checked for accuracy. It is possible for a user to modify the files used by GCP to list a different user.

Parameters **auth_client** (globus_sdk.AuthClient) – An AuthClient to use to lookup the full identity information for the GCP owner

Examples

Getting a username:

```
>>> from globus_sdk import LocalGlobusConnectPersonal
>>> local_gcp = LocalGlobusConnectPersonal()
>>> local_gcp.get_owner_info()
GlobusConnectPersonalOwnerInfo(username='foo@globusid.org')
```

or you may get back an ID:

```
>>> local_gcp = LocalGlobusConnectPersonal()
>>> local_gcp.get_owner_info()
GlobusConnectPersonalOwnerInfo(id='7deda7cc-077b-11ec-a137-67523ecffd4b')
```

Check the result easily by looking to see if these values are `None`:

```
>>> local_gcp = LocalGlobusConnectPersonal()
>>> info = local_gcp.get_owner_info()
>>> has_username = info.username is not None
```

property endpoint_id: Optional[str]

Type str

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

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

Note: This attribute is not checked for accuracy. It is possible for a user to modify the files used by GCP to list a different `endpoint_id`.

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`

class globus_sdk.**GlobusConnectPersonalOwnerInfo**(*, *config_dn: str*)

Information about the owner of the local Globus Connect Personal endpoint.

Users should never create these objects directly, but instead rely upon *LocalGlobusConnectPersonal.get_owner_info()*.

The info object contains ether `id` or `username`. Parsing an info object from local data cannot guarantee that the `id` or `username` value will be found. Whichever one is present will be set and the other attribute will be `None`.

Variables

- **id** (*str or None*) – The Globus Auth ID of the endpoint owner
- **username** (*str or None*) – The Globus Auth Username of the endpoint owner

Parameters **config_dn** (*str*) – A DN value from GCP configuration, which will be parsed into username or 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.**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 get_authorization_header() → Optional[str]

Get the value for the `Authorization` header from this authorizer. If this method returns `None`, then no `Authorization` header should be used.

handle_missing_authorization() → bool

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.**RenewingAuthorizer**(*access_token: Optional[str] = None*, *expires_at: Optional[int] = None*, *on_refresh: Optional[Callable] = 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, 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.

get_authorization_header() → str
Check to see if a new token is needed and return "Bearer <access_token>"

handle_missing_authorization() → bool
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.

`GlobusAuthorizer` objects which have a static authorization header are all implemented using the static authorizer class:

class globus_sdk.authorizers.**StaticGlobusAuthorizer**
Bases: *globus_sdk.authorizers.base.GlobusAuthorizer*

A static authorizer has some static string as its header val which it always returns as the authz header.

get_authorization_header() → str
Get the value for the `Authorization` header from this authorizer. If this method returns `None`, then no `Authorization` header should be used.

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.

get_authorization_header() → None
Get the value for the `Authorization` header from this authorizer. If this method returns `None`, then no `Authorization` header should be used.

class globus_sdk.**BasicAuthorizer**(*username: str*, *password: str*)
Bases: *globus_sdk.authorizers.base.StaticGlobusAuthorizer*

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

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

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

class globus_sdk.**RefreshTokenAuthorizer**(*refresh_token: str*, *auth_client:* globus_sdk.services.auth.client.base.AuthClient, *, *access_token: Optional[str] = None*, *expires_at: Optional[int] = None*, *on_refresh: Optional[Callable] = 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 `get_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:* globus_sdk.services.auth.client.confidential_client.ConfidentialAppAuthClie *scopes: str*, *, *access_token: Optional[str] = None*, *expires_at: Optional[int] = None*, *on_refresh: Optional[Callable] = 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)
```

(continues on next page)

(continued from previous page)

```
>>> # 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, 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.

1.7 TokenStorage

The TokenStorage component provides a way of storing and loading the tokens received from authentication and token refreshes.

1.7.1 Usage

TokenStorage is available under the name `globus_sdk.tokenstorage`.

Storage adapters are the main objects of this subpackage. Primarily, usage should revolve around creating a storage adapter, potentially loading data from it, and using it as the `on_refresh` handler for an authorizer.

For example:

```
import os
import globus_sdk
from globus_sdk.tokenstorage import SimpleJSONFileAdapter

my_file_adapter = SimpleJSONFileAdapter(os.path.expanduser("~/mytokens.json"))

if not my_file_adapter.file_exists():
    # ... do a login low, getting back initial tokens
    # elided for simplicity here
    token_response = ...
    # now store the tokens, and pull out the tokens for the
    # resource server we want
```

(continues on next page)

(continued from previous page)

```
    my_file_adapter.store(token_response)
    by_rs = token_response.by_resource_server
    tokens = by_rs["transfer.api.globus.org"]
else:
    # otherwise, we already did this whole song-and-dance, so just
    # load the tokens from that file
    tokens = my_file_adapter.get_token_data("transfer.api.globus.org")


# RereshTokenAuthorizer and ClientCredentialsAuthorizer both use
# `on_refresh` callbacks
# this feature is therefore only relevant for those auth types
#
# auth_client is the internal auth client used for refreshes,
# and which was used in the login flow
# note that this is all normal authorizer usage wherein
# my_file_adapter is providing the on_refresh callback
auth_client = ...
authorizer = globus_sdk.RefreshTokenAuthorizer(
    tokens["refresh_token"],
    auth_client,
    access_token=tokens["access_token"],
    expires_at=tokens["access_token_expires"],
    on_refresh=my_file_adapter.on_refresh,
)

# or, for client credentials
authorizer = globus_sdk.ClientCredentialsAuthorizer(
    auth_client,
    ["urn:globus:auth:transfer.api.globus.org:all"],
    access_token=tokens["access_token"],
    expires_at=tokens["access_token_expires"],
    on_refresh=my_file_adapter.on_refresh,
)

# and then use the authorizer on a client!
tc = globus_sdk.TransferClient(authorizer=authorizer)
```

1.7.2 Adapter Types

`globus_sdk.tokenstorage` provides base classes for building your own storage adapters, and two complete adapters.

The `SimpleJSONFileAdapter` is good for the "simplest possible" storage, using a JSON file to store token data.

The `SQLiteAdapter` is the next step up in complexity, for applications like the globus-cli which need to store various tokens and additional configuration. In addition to basic token storage, the `SQLiteAdapter` provides for namespacing of the token data, and for additional configuration storage.

1.7.3 Reference

class globus_sdk.tokenstorage.**StorageAdapter**
Bases: object

abstract get_token_data(*resource_server: str*) → Optional[Dict]
Lookup token data for a resource server

Either returns a dict with the access token, refresh token (optional), and expiration time, or returns None, indicating that there was no data for that resource server.

on_refresh(*token_response:* globus_sdk.services.auth.response.OAuthTokenResponse) → None
By default, the on_refresh handler for a token storage adapter simply stores the token response.

class globus_sdk.tokenstorage.**FileAdapter**
Bases: *globus_sdk.tokenstorage.base.StorageAdapter*

File adapters are for single-user cases, where we can assume that there's a simple file-per-user and users are only ever attempting to read their own files.

file_exists() → bool
Check if the file used by this file storage adapter exists.

user_only_umask() → Iterator[None]
a context manager to deny rwx to Group and World, x to User

this does not create a file, but ensures that if a file is created while in the context manager, its permissions will be correct on unix systems

class globus_sdk.tokenstorage.**SimpleJSONFileAdapter**(*filename: str*)
Bases: *globus_sdk.tokenstorage.base.FileAdapter*

Parameters **filename** – the name of the file to write to and read from

A storage adapter for storing tokens in JSON files.

store(*token_response:* globus_sdk.services.auth.response.OAuthTokenResponse) → None
By default, self.on_refresh is just an alias for this function.

Given a token response, extract all the token data and write it to self.filename as JSON data. Additionally will write the version of globus_sdk.tokenstorage which was in use.

Under the assumption that this may be running on a system with multiple local users, this sets the umask such that only the owner of the resulting file can read or write it.

get_by_resource_server() → Dict
Read only the by_resource_server formatted data from the file, discarding any other keys.

This returns a dict in the same format as OAuthTokenResponse.by_resource_server

get_token_data(*resource_server: str*) → Optional[Dict]
Lookup token data for a resource server

Either returns a dict with the access token, refresh token (optional), and expiration time, or returns None, indicating that there was no data for that resource server.

class globus_sdk.tokenstorage.**SQLiteAdapter**(*dbname: str*, *, *namespace: str = 'DEFAULT'*)
Bases: *globus_sdk.tokenstorage.base.FileAdapter*

Parameters

- **dbname** – The name of the DB file to write to and read from. If the string ":memory:" is used, an in-memory database will be used instead.

- **namespace** – A "namespace" to use within the database. All operations will be performed indexed under this string, so that multiple distinct sets of tokens may be stored in the database. You might use usernames as the namespace to implement a multi-user system, or profile names to allow multiple Globus accounts to be used by a single user.

A storage adapter for storing tokens in sqlite databases.

SQLite adapters are for more complex cases, where there may be multiple users or "profiles" in play, and additionally a dynamic set of resource servers which need to be stored in an extensible way.

The `namespace` is a user-supplied way of partitioning data, and any token responses passed to the storage adapter are broken apart and stored indexed by *resource_server*. If you have a more complex use-case in which this scheme will be insufficient, you should encode that in your choice of `namespace` values.

store_config(*config_name: str*, *config_dict: Mapping*) → None

Parameters

- **config_name** – A string name for the configuration value
- **config_dict** – A dict of config which will be stored serialized as JSON

Store a config dict under the current namespace in the config table. Allows arbitrary configuration data to be namespaced under the namespace, so that application config may be associated with the stored tokens.

Uses sqlite "REPLACE" to perform the operation.

read_config(*config_name: str*) → Optional[Dict]

Parameters **config_name** – A string name for the configuration value

Load a config dict under the current namespace in the config table. If no value is found, returns None

remove_config(*config_name: str*) → bool

Parameters **config_name** – A string name for the configuration value

Delete a previously stored configuration value.

Returns True if data was deleted, False if none was found to delete.

store(*token_response:* globus_sdk.services.auth.response.OAuthTokenResponse) → None

Parameters **token_response** – a globus_sdk.OAuthTokenResponse object containing token data to store

By default, `self.on_refresh` is just an alias for this function.

Given a token response, extract the token data for the resource servers and write it to `self.dbname`, stored under the adapter's namespace

get_token_data(*resource_server: str*) → Optional[Dict[str, Any]]
Load the token data JSON for a specific resource server.

In the event that the server cannot be found in the DB, return None.

Parameters **resource_server** – The name of a resource server to lookup in the DB, as one would use as a key in OAuthTokenResponse.by_resource_server

get_by_resource_server() → Dict[str, Any]
Load the token data JSON and return the resulting dict objects, indexed by resource server.

This should look identical to an OAuthTokenResponse.by_resource_server in format and content. (But it is not attached to a token response object.)

remove_tokens_for_resource_server(*resource_server: str*) → bool

Given a resource server to target, delete tokens for that resource server from the database (limited to the current namespace). You can use this as part of a logout command implementation, loading token data as a dict, and then deleting the data for each resource server.

Returns True if token data was deleted, False if none was found to delete.

Parameters **resource_server** – The name of the resource server to remove from the DB, as one would use as a key in OAuthTokenResponse.by_resource_server

1.8 Globus SDK Configuration

The behaviors of the SDK can be controlled either through environment variables, or by passing parameters to clients and other objects.

Note: SDK v1.x and v2.x supported the use of `/etc/globus.cfg` and `~/.globus.cfg` to set certain values. This feature was removed in v3.0 in favor of new environment variables for setting these values.

1.8.1 Environment Variables

Each of these environment variables will be read automatically by the SDK.

Environment variables have lower precedence than explicit values set in the interpreter. If `GLOBUS_SDK_VERIFY_SSL="false"` is set and a client is created with `verify_ssl=True`, the resulting client will have SSL verification turned on.

GLOBUS_SDK_VERIFY_SSL Used to disable SSL verification, typically to handle SSL-intercepting firewalls. By default, all connections to servers are verified. Set `GLOBUS_SDK_VERIFY_SSL="false"` to disable verification.

GLOBUS_SDK_HTTP_TIMEOUT Adjust the timeout when HTTP requests are made. By default, requests have a 60 second read timeout – for slower responses, try setting `GLOBUS_SDK_HTTP_TIMEOUT=120`

GLOBUS_SDK_ENVIRONMENT The name of the environment to use. Set `GLOBUS_SDK_ENVIRONMENT="preview"` to use the Globus Preview environment.

GLOBUS_SDK_SERVICE_URL_* Override the URL used for a given service. The suffix of this environment variable must match the service name string used by the SDK in all caps (`SEARCH`, `TRANSFER`, etc). For example, set `GLOBUS_SDK_SERVICE_URL_TRANSFER="https://proxy-device.example.org/"` to direct the SDK to use a custom URL when contacting the Globus Transfer service.

1.9 Globus SDK Core

Underlying components of the Globus SDK.

1.9.1 BaseClient

All service clients support the low level interface, provided by the `BaseClient`, from which all client types inherit.

A client object contains a `transport`, an object responsible for sending requests, encoding data, and handling potential retries. It also may include an optional `authorizer`, an object responsible for handling token authentication for requests.

BaseClient

class globus_sdk.**BaseClient**(*, *environment: Optional[str] = None*, *base_url: Optional[str] = None*, *authorizer: Optional[globus_sdk.authorizers.base.GlobusAuthorizer] = None*, *app_name: Optional[str] = None*, *transport_params: Optional[Dict] = None*)

Abstract base class for clients with error handling for Globus APIs.

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
- **transport_params** (*dict*) – Options to pass to the transport for this client

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

scopes: Optional[*globus_sdk.scopes.ScopeBuilder*] = None

the scopes for this client may be present as a `ScopeBuilder`

get(*path: str*, *, *query_params: Optional[Dict[str, Any]] = None*, *headers: Optional[Dict] = None*) → *globus_sdk.response.GlobusHTTPResponse*

Make a GET request to the specified path.

See *request()* for details on the various parameters.

Returns *GlobusHTTPResponse* object

post(*path: str*, *, *query_params: Optional[Dict[str, Any]] = None*, *data: Union[None, Dict, globus_sdk.utils.PayloadWrapper] = None*, *headers: Optional[Dict] = None*, *encoding: Optional[str] = None*) → *globus_sdk.response.GlobusHTTPResponse*

Make a POST request to the specified path.

See *request()* for details on the various parameters.

Returns *GlobusHTTPResponse* object

delete(*path: str*, *, *query_params: Optional[Dict[str, Any]] = None*, *headers: Optional[Dict] = None*) → *globus_sdk.response.GlobusHTTPResponse*

Make a DELETE request to the specified path.

See *request()* for details on the various parameters.

Returns *GlobusHTTPResponse* object

put(*path: str*, *, *query_params: Optional[Dict[str, Any]] = None*, *data: Union[None, Dict, globus_sdk.utils.PayloadWrapper] = None*, *headers: Optional[Dict] = None*, *encoding: Optional[str] = None*) → *globus_sdk.response.GlobusHTTPResponse*

Make a PUT request to the specified path.

See *request()* for details on the various parameters.

Returns *GlobusHTTPResponse* object

patch(*path: str*, ***, *query_params: Optional[Dict[str, Any]] = None*, *data: Union[None, Dict, globus_sdk.utils.PayloadWrapper] = None*, *headers: Optional[Dict] = None*, *encoding: Optional[str] = None*) → *globus_sdk.response.GlobusHTTPResponse*

Make a PATCH request to the specified path.

See *request()* for details on the various parameters.

Returns *GlobusHTTPResponse* object

request(*method: str*, *path: str*, ***, *query_params: Optional[Dict[str, Any]] = None*, *data: Union[None, Dict, globus_sdk.utils.PayloadWrapper] = None*, *headers: Optional[Dict] = None*, *encoding: Optional[str] = None*) → *globus_sdk.response.GlobusHTTPResponse*

Send an HTTP request

Parameters

- **method** (*str*) – HTTP request method, as an all caps string
- **path** (*str*) – Path for the request, with or without leading slash
- **query_params** (*dict, optional*) – Parameters to be encoded as a query string
- **headers** (*dict*) – HTTP headers to add to the request
- **data** (*dict or str*) – Data to send as the request body. May pass through encoding.
- **encoding** (*str*) – A way to encode request data. "json", "form", and "text" are all valid values. Custom encodings can be used only if they are registered with the transport. By default, strings get "text" behavior and all other objects get "json".

Returns *GlobusHTTPResponse* object

1.9.2 Transport Layer

The transport consists of a transport object (*RequestsTransport*), but also tooling for handling retries. It is possible to either register custom retry check methods, or to override the Transport used by a client in order to customize this behavior.

Transport

class globus_sdk.transport.**RequestsTransport**(*verify_ssl: Optional[bool] = None*, *http_timeout: Optional[float] = None*, *retry_backoff: Callable[[globus_sdk.transport.retry.RetryContext], float] = <function _exponential_backoff>*, *retry_checks: Optional[List[Callable[[globus_sdk.transport.retry.RetryContext], globus_sdk.transport.retry.RetryCheckResult]]] = None*, *max_sleep: int = 10*, *max_retries: Optional[int] = None*)

The RequestsTransport handles HTTP request sending and retries.

It receives raw request information from a client class, and then performs the following steps - encode the data in a prepared request - repeatedly send the request until no retry is requested - return the last response or reraise the last exception

Retry checks are registered as hooks on the Transport. Additional hooks can be passed to the constructor via *retry_checks*. Or hooks can be added to an existing transport via a decorator.

If the maximum number of retries is reached, the final response or exception will be returned or raised.

Parameters

- **verify_ssl** (*bool, optional*) – Explicitly enable or disable SSL verification. This parameter defaults to True, but can be set via the GLOBUS_SDK_VERIFY_SSL environment variable. Any non-None setting via this parameter takes precedence over the environment variable.
- **http_timeout** (*int, optional*) – Explicitly set an HTTP timeout value in seconds. This parameter defaults to 60s but can be set via the GLOBUS_SDK_HTTP_TIMEOUT environment variable. Any value set via this parameter takes precedence over the environment variable.
- **retry_backoff** (*callable, optional*) – A function which determines how long to sleep between calls based on the RetryContext. Defaults to exponential backoff with jitter based on the context attempt number.
- **retry_checks** (*list of callable, optional*) – A list of initial retry checks. Any hooks registered, including the default hooks, will run after these checks.
- **max_sleep** (*int, optional*) – The maximum sleep time between retries (in seconds). If the computed sleep time or the backoff requested by a retry check exceeds this value, this amount of time will be used instead
- **max_retries** (*int, optional*) – The maximum number of retries allowed by this transport

request(*method: str*, *url: str*, *query_params: Optional[Dict[str, Any]] = None*, *data: Optional[dict] = None*, *headers: Optional[dict] = None*, *encoding: Optional[str] = None*, *authorizer: Optional[*globus_sdk.authorizers.base.GlobusAuthorizer*] = None*) → requests.models.Response

Send an HTTP request

Parameters

- **url** – URL for the request
- **method** (*str*) – HTTP request method, as an all caps string
- **query_params** (*dict, optional*) – Parameters to be encoded as a query string
- **headers** (*dict*) – HTTP headers to add to the request
- **data** (*dict or str*) – Data to send as the request body. May pass through encoding.
- **encoding** (*str*) – A way to encode request data. "json", "form", and "text" are all valid values. Custom encodings can be used only if they are registered with the transport. By default, strings get "text" behavior and all other objects get "json".

Returns requests.Response object

Retries

These are the components used by the RequestsTransport to implement retry logic.

class globus_sdk.transport.**RetryContext**(*attempt: int*, ***, *authorizer: Optional[*globus_sdk.authorizers.base.GlobusAuthorizer*] = None*, *response: Optional[requests.models.Response] = None*, *exception: Optional[Exception] = None*)

The RetryContext is an object passed to retry checks in order to determine whether or not a request should be retried. The context is constructed after each request, regardless of success or failure.

If an exception was raised, the context will contain that exception object. Otherwise, the context will contain a response object. Exactly one of response or exception will be present.

Parameters

- **attempt** (*int*) – The request attempt number, starting at 0.
- **response** (*requests.Response*) – The response on a successful request
- **exception** (*Exception*) – The error raised when trying to send the request
- **authorizer** (*GlobusAuthorizer*) – The authorizer object from the client making the request

class globus_sdk.transport.**RetryCheckResult**(*value*)

An enumeration.

do_retry = 1

yes, retry the request

do_not_retry = 2

no, do not retry the request

no_decision = 3

"I don't know", ask other checks for an answer

globus_sdk.transport.**RetryCheck**

The type for a retry check, a callable which takes a `RetryContext` and returns a `RetryCheckResult`. Equivalent to `Callable[[globus_sdk.transport.RetryContext], globus_sdk.transport.RetryCheckResult]`

class globus_sdk.transport.**RetryCheckRunner**(*checks: List[Callable[[*globus_sdk.transport.retry.RetryContext*], *globus_sdk.transport.retry.RetryCheckResult*]]*)

A RetryCheckRunner is an object responsible for running retry checks over the lifetime of a request. Unlike the checks or the retry context, the runner persists between retries. It can therefore implement special logic for checks like "only try this check once".

Its primary responsibility is to answer the question "should_retry(context)?" with a boolean.

It takes as its input a list of checks. Checks may be paired with flags to indicate their configuration options. When not paired with flags, the flags are taken to be "NONE".

Supported flags:

RUN_ONCE The check will run at most once for a given request. Once it has run, it is recorded as "has_run" and will not be run again on that request.

class globus_sdk.transport.**RetryCheckFlags**(*value*)

An enumeration.

NONE = 1

no flags (default)

RUN_ONCE = 2

only run this check once per request

@globus_sdk.transport.**set_retry_check_flags**(*flag:* globus_sdk.transport.retry.RetryCheckFlags) → Callable[[Callable], Callable]

A decorator for setting retry check flags on a retry check function. Usage:

```
>>> @set_retry_check_flags(RetryCheckFlags.RUN_ONCE)
>>> def foo(ctx): ...
```

Data Encoders

class globus_sdk.transport.**RequestEncoder**
A RequestEncoder takes input parameters and outputs a requests.Requests object.

The default encoder requires that the data is text and is a no-op. It can also be referred to as the `"text"` encoder.

class globus_sdk.transport.**JSONRequestEncoder**
This encoder prepares the data as JSON. It also ensures that content-type is set, so that APIs requiring a content-type of "application/json" are able to read the data.

class globus_sdk.transport.**FormRequestEncoder**
This encoder formats data as a form-encoded body. It requires that the input data is a dict – any other datatype will result in errors.

1.9.3 Responses

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

To customize client methods with additional detail, the SDK uses subclasses of `GlobusHTTPResponse`.

class globus_sdk.response.**GlobusHTTPResponse**(*response: Union[requests.models.Response, GlobusHTTPResponse]*, *client: Optional[globus_sdk.BaseClient] = None*)
Bases: `object`

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.

The most common response data is a JSON dictionary. To make handling this type of response as seamless as possible, the `GlobusHTTPResponse` object implements the immutable mapping protocol for dict-style access. This is just an alias for access to the underlying data.

If `data` is not a dictionary, item access will raise `TypeError`.

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

Variables

- **http_status** – HTTP status code returned by the server (int)
- **content_type** – Content-Type header returned by the server (str)
- **client** – The client instance which made the request

get(*key: str*, *default: Optional[Any] = None*) → Any
`get` is just an alias for `data.get(key, default)`, but with the added check that if `data` is `None`, it returns the default.

property text: str
The raw response data as a string.

class globus_sdk.response.**IterableResponse**(*response: Union[requests.models.Response, GlobusHTTPResponse]*, *client: Optional[globus_sdk.BaseClient] = None*, ***, *iter_key: Optional[str] = None*)
Bases: *globus_sdk.response.GlobusHTTPResponse*

This response class adds an __iter__ method on an 'iter_key' variable. The assumption is that iter produces dicts or dict-like mappings.

1.9.4 Paging and Paginators

Globus SDK Client objects have paginated methods which return paginators.

A paginated API is one which returns data in multiple API calls. This is used in cases where the the full set of results is too large to return all at once, or where getting all results is slow and a few results are wanted faster.

A good example of paginated data would be search results: the first "page" of data may be the first 10 results, and the next "page" consists of the next 10 results.

The number of results per call is the page size. Each page is an API response with a number of results equal to the page size.

Paging in the Globus SDK can be done by iterating over pages (responses) or by iterating over items (individual results).

Paginators

A *Paginator* object is an iterable provided by the Globus SDK. Paginators support iteration over pages with the method `pages()` and iteration over items with the method `items()`.

Paginators have fixed parameters which are set when the paginator is created. Once a method returns a paginator, you don't need to pass it any additional data – `pages()` or `items()` will operate based on the original parameters to the paginator.

Making Paginated Calls

Globus SDK client objects define paginated variants of methods. The normal method is said to be "unpaginated", and returns a single page of results. The paginated variant, prefixed with `paginated.`, returns a paginated.

For example, *TransferClient* has a paginated method, *endpoint_search()*. Once you have a client object, calls to the unpaginated method are done like so:

```
import globus_sdk

# for information on getting an authorizer, see the SDK Tutorial
tc = globus_sdk.TransferClient(authorizer=...)

# unpaginated calls can still return iterable results!
# endpoint_search() returns an iterable response
for endpoint_info in tc.endpoint_search("tutorial"):
    print("got endpoint_id:", endpoint_info["id"])
```

The paginated variant of this same method is accessed nearly identically. But instead of calling `endpoint_search(...)`, we'll invoke `paginated.endpoint_search(...)`.

Here are three variants of code with the same basic effect:

```
# note the call to `items()` at the end of this line!
for endpoint_info in tc.paginated.endpoint_search("tutorial").items():
    print("got endpoint_id:", endpoint_info["id"])

# equivalently, call `pages()` and iterate over the items in each page
for page in tc.paginated.endpoint_search("tutorial").pages():
    for endpoint_info in page:
        print("got endpoint_id:", endpoint_info["id"])
```

(continues on next page)

(continued from previous page)

```
# iterating on a paginator without calling `pages()` or `items()` is
# equivalent to iterating on `pages()`
for page in tc.paginated.endpoint_search("tutorial"):
    for endpoint_info in page:
        print("got endpoint_id:", endpoint_info["id"])
```

Do I need to use pages()? What is it for?

If your use-case is satisfied with `items()`, then stick with `items()`!

`pages()` iteration is important when there is useful data in the page other than the individual items.

For example, *~globus_sdk.TransferClient.endpoint_search <TransferClient.endpoint_search>* returns the total number of results for the search as a field on each page.

Most use-cases can be solved with `items()`, and `pages()` will be available to you if or when you need it.

Paginator Types

`globus_sdk.paging` defines several paginator classes and methods. For the most part, you do not need to interact with these classes or methods except through `pages()` or `items()`.

The `paging` subpackage also defines the `PaginatorTable`, which is used to define the `paginated` attribute on client objects.

globus_sdk.paging.**has_paginator**(*paginator_class: Type[globus_sdk.paging.base.Paginator]*, *items_key: Optional[str] = None*, ***paginator_params: Any*) → Callable[[Callable], Callable]

Mark a callable – typically a client method – as having pagination parameters. Usage:

```
>>> class MyClient(BaseClient):
>>>     @has_paginator(MarkerPaginator)
>>>     def foo(...): ...
```

This will mark `MyClient.foo` as paginated with marker style pagination. It will then be possible to get a paginator for `MyClient.foo` via

```
>>> c = MyClient(...)
>>> paginator = c.paginated.foo()
```

class globus_sdk.paging.**Paginator**(*method: Callable*, ***, *items_key: Optional[str] = None*, *client_args: List[Any]*, *client_kwargs: Dict[str, Any]*)

Bases: `Iterable[globus_sdk.response.GlobusHTTPResponse]`

Base class for all paginators. This guarantees is that they have generator methods named `pages` and `items`.

Iterating on a Paginator is equivalent to iterating on its `pages`.

Parameters

- **method** (*callable*) – A bound method of an SDK client, used to generate a paginated variant
- **items_key** (*str*) – The key to use within pages of results to get an array of items

- **client_args** (*tuple*) – Arguments to the underlying method which are passed when the paginator is instantiated. i.e. given `client.paginated.foo(a, b, c=1)`, this will be `(a, b)`. The paginator will pass these arguments to each call of the bound method as it pages.
- **client_kwargs** (*dict*) – Keyword arguments to the underlying method, like `client_args` above. `client.paginated.foo(a, b, c=1)` will pass this as `{"c": 1}`. As with `client_args`, it's passed to each paginated call.

items() → Iterator
`items()` of a paginator is a generator which yields each item in each page of results.

`items()` may raise a `ValueError` if the paginator was constructed without identifying a key for use within each page of results. This may be the case for paginators whose pages are not primarily an array of data.

abstract pages() → Iterator[*globus_sdk.response.GlobusHTTPResponse*]
`pages()` yields GlobusHTTPResponse objects, each one representing a page of results.

class globus_sdk.paging.**PaginatorTable**(*client: Any*)
Bases: `object`

A PaginatorTable maps multiple methods of an SDK client to paginated variants. Given a method, client.foo annotated with the *has_paginator* decorator, the table will gain a function attribute *foo* (name matching is automatic) which returns a Paginator.

Clients automatically build and attach paginator tables under the `paginated` attribute. That is, if *client* has two methods *foo* and *bar* which are marked as paginated, that will let us call

```
>>> client.paginated.foo()
>>> client.paginated.bar()
```

where `client.paginated` is a `PaginatorTable`.

Paginators are iterables of response pages, so ultimate usage is like so:

```
>>> paginator = client.paginated.foo()  # returns a paginator
>>> for page in paginator:  # a paginator is an iterable of pages (response objects)
>>>     print(json.dumps(page.data))  # you can handle each response object in turn
```

A `PaginatorTable` is built automatically as part of client instantiation. Creation of `PaginatorTable` objects is considered a private API.

class globus_sdk.paging.**MarkerPaginator**(*method: Callable*, *, *items_key: Optional[str] = None*, *client_args: List[Any]*, *client_kwargs: Dict[str, Any]*)
Bases: `Iterable`[*globus_sdk.response.GlobusHTTPResponse*]

A paginator which uses *has_next_page* and *marker* from payloads, sets the *marker* query param to page.

This is the default method for GCS pagination, so it's very simple.

pages() → Iterator[*globus_sdk.response.GlobusHTTPResponse*]
`pages()` yields GlobusHTTPResponse objects, each one representing a page of results.

class globus_sdk.paging.**NextTokenPaginator**(*method: Callable*, *, *items_key: Optional[str] = None*, *client_args: List[Any]*, *client_kwargs: Dict[str, Any]*)
Bases: `Iterable`[*globus_sdk.response.GlobusHTTPResponse*]

A paginator which uses *next_token* from payloads to set the *next_token* query param to page.

Very similar to GCS's marker paginator, but only used for Transfer's get_shared_endpoint_list

pages() → Iterator[*globus_sdk.response.GlobusHTTPResponse*]
pages() yields GlobusHTTPResponse objects, each one representing a page of results.

class globus_sdk.paging.**LastKeyPaginator**(*method: Callable*, ***, *items_key: Optional[str] = None*, *client_args: List[Any]*, *client_kwargs: Dict[str, Any]*)
Bases: Iterable[*globus_sdk.response.GlobusHTTPResponse*]

pages() → Iterator[*globus_sdk.response.GlobusHTTPResponse*]
pages() yields GlobusHTTPResponse objects, each one representing a page of results.

class globus_sdk.paging.**HasNextPaginator**(*method: Callable*, ***, *items_key: Optional[str] = None*, *get_page_size: Callable[[dict], int]*, *max_total_results: int*, *page_size: int*, *client_args: List[Any]*, *client_kwargs: Dict[str, Any]*)
Bases: Iterable[*globus_sdk.response.GlobusHTTPResponse*]

pages() → Iterator[*globus_sdk.response.GlobusHTTPResponse*]
pages() yields GlobusHTTPResponse objects, each one representing a page of results.

class globus_sdk.paging.**LimitOffsetTotalPaginator**(*method: Callable*, ***, *items_key: Optional[str] = None*, *get_page_size: Callable[[dict], int]*, *max_total_results: int*, *page_size: int*, *client_args: List[Any]*, *client_kwargs: Dict[str, Any]*)
Bases: Iterable[*globus_sdk.response.GlobusHTTPResponse*]

pages() → Iterator[*globus_sdk.response.GlobusHTTPResponse*]
pages() yields GlobusHTTPResponse objects, each one representing a page of results.

1.9.5 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

try:
    tc = TransferClient(...)

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

(continues on next page)

(continued from previous page)

```
    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"
        f"Error Code: {e.code}\n"
        f"Error Message: {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.)

Error Classes

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

Root of the Globus Exception hierarchy. Stub class.

class globus_sdk.**GlobusSDKUsageError**
Bases: *globus_sdk.exc.base.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: requests.models.Response*, **args: Any*, ***kw: Any*)
Bases: *globus_sdk.exc.base.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 info: *globus_sdk.exc.err_info.ErrorInfoContainer*
An `ErrorInfoContainer` with parsed error data. The `info` of an error is guaranteed to be present, but all of its contents may be falsey if the error could not be parsed.

property raw_json: **Optional[Dict[str, Any]]**
Get the verbatim error message received from a Globus API, interpreted as JSON data

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

property raw_text: **str**
Get the verbatim error message received from a Globus API as a *string*

class globus_sdk.**NetworkError**(*msg: str*, *exc: Exception*, **args: Any*, ***kw: Any*)
Bases: *globus_sdk.exc.base.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: str*, *exc: Exception*, **args: Any*, ***kw: Any*)
Bases: *globus_sdk.exc.convert.NetworkError*

A connection error occured while making a REST request.

class globus_sdk.**GlobusTimeoutError**(*msg: str*, *exc: Exception*, **args: Any*, ***kw: Any*)
Bases: *globus_sdk.exc.convert.NetworkError*

The REST request timed out.

class globus_sdk.**GlobusConnectionTimeoutError**(*msg: str*, *exc: Exception*, **args: Any*, ***kw: Any*)
Bases: *globus_sdk.exc.convert.GlobusTimeoutError*

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

ErrorInfo

`GlobusAPIError` and its subclasses all support an `info` property which may contain parsed error data. The `info` is guaranteed to be there, but its attributes should be tested before use, as in

```
# if 'err' is an API error, then 'err.info' is an 'ErrorInfoContainer',
# a wrapper which holds ErrorInfo objects
# 'err.info.consent_required' is a 'ConsentRequiredInfo', which should be
# tested for truthy/falsey-ness before use
if err.info.consent_required:
    print(
        "Got a ConsentRequired error with scopes:",
        err.info.consent_required.required_scopes,
    )
```

class globus_sdk.exc.**ErrorInfoContainer**(*error_data: Optional[Dict[str, Any]]*)
This is a wrapper type which contains various error info objects for parsed error data. It is attached to API errors as the `.info` attribute.

Variables

- **authorization_parameters** – A parsed AuthorizationParameterInfo object
- **consent_required** – A parsed ConsentRequiredInfo object

class globus_sdk.exc.**ErrorInfo**
Errors may contain "containers" of data which are testable (define `__bool__`). When they have data, they should `bool()` as `True`

class globus_sdk.exc.**AuthorizationParameterInfo**(*error_data: Dict[str, Any]*)
Bases: *globus_sdk.exc.err_info.ErrorInfo*

AuthorizationParameterInfo objects may contain information about the 'authorization_parameters' of an error. They test as truthy when the error has valid 'authorization_parameters' data.

Variables

- **session_message** (*str, optional*) – A message from the server
- **session_required_identities** (*list of str, optional*) – A list of identity IDs as strings which are being requested by the server
- **session_required_single_domain** (*list of str, optional*) – A list of domains which are being requested by the server ("single domain" because the user should choose one)

Examples

```
>>> try:
>>>     ...  # something
>>> except GlobusAPIError as err:
>>>     # get a parsed AuthorizationParamaterInfo object, and check if it's truthy
>>>     authz_params = err.info.authorization_parameters
>>>     if not authz_params:
>>>         raise
>>>     # whatever handling code is desired...
>>>     print("got authz params:", authz_params)
```

class globus_sdk.exc.**ConsentRequiredInfo**(*error_data: Dict[str, Any]*)
Bases: *globus_sdk.exc.err_info.ErrorInfo*

ConsentRequiredInfo objects contain required consent information for an error. They test as truthy if the error was marked as a ConsentRequired error.

Variables **required_scopes** (*list of str, optional*) – A list of scopes requested by the server

1.9.6 Utilities

Warning: The components in this module are *not* intended for outside use, but are internal to the Globus SDK.

They may change in backwards-incompatible ways in minor or patch releases of the SDK.

This documentation is included here for completeness.

PayloadWrapper

The `PayloadWrapper` class is used as a base class for all Globus SDK payload datatypes to provide nicer interfaces for payload construction.

The objects are a type of `UserDict` with no special methods.

class globus_sdk.utils.**PayloadWrapper**(*dict=None*, */*, ***kwargs*)

A class for defining helper objects which wrap some kind of "payload" dict. Typical for helper objects which formulate a request payload, e.g. as JSON.

Payload types inheriting from this class can be passed directly to the client `post()`, `put()`, and `patch()` methods instead of a dict. These methods will recognize a `PayloadWrapper` and convert it to a dict for serialization with the requested encoder (e.g. as a JSON request body).

1.10 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.

1.10.1 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.

1.10.2 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.

1.10.3 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.

1.11 License

1.12 CHANGELOG

See *Versioning Policy* for our versioning policy.

The *upgrading* doc is a good reference if you are upgrading to a major new version of the SDK.

1.12.1 Unreleased

1.12.2 v3.0.1

- *ScopeBuilder* objects now define the type of *__getattr__* for *mypy* to know that dynamic attributes are strings (#472)
- Fix type remaining ignore usages in globus-sdk (#473)
- Fix malformed PEP508 *python_version* bound in dev dependencies (#474)

1.12.3 v3.0.0

- Add `filter_is_error` parameter to advanced task list (#467)
- Add a `LocalGlobusConnectPersonal.get_owner_info()` for looking up local user information from gridmap (#466)
- Add support for GCS collection create and update. This includes new data helpers, `MappedCollectionDcoument` and `GuestCollectionDocument` (#468)
- Remove support for `bytes` values for fields consuming UUIDs (#471)
- Add support for specifying `config_dir` to `LocalGlobusConnectPersonal` (#470)

1.12.4 v3.0.0b4

- Minor fix to wheel builds: do not declare wheels as universal (#444)
- Add a new `GCSClient` class for interacting with GCS Manager APIs (#447)
- Rename `GCSScopeBuilder` to `GCSCollectionScopeBuilder` and add `GCSEndpointScopeBuilder`. The `GCSClient` includes helpers for instantiating these scope builders (#448)
- `GCSClient` supports `get_collection` and `delete_collection`. `get_collection` uses a new `UnpackingGCSResponse` response type (#451, #464)
- Remove `BaseClient.qjoin_path` (#452)
- Enforce keyword-only arguments for most SDK-provied APIs (#453)
- Fix annotations for `server_id` on `TransferClient` methods (#455)
- Add `delete_destination_extra` param to `TransferData` (#456)
- Ensure all `TransferClient` method parameters are documented (#449, #454, #457, #458, #459, #461, #462)
- `TransferClient.endpoint_manager_task_list` now takes filters as named keyword arguments, not only in `query_params` (#460)
- Fix `visibility` typo in `GroupsClient` (#463)
- All type annotations for `Sequence` which could be relaxed to `Iterable` have been updated (#465)

1.12.5 v3.0.0b3

- Flesh out the `GroupsClient` and add helpers for interacting with the Globus Groups service, including enumerated constants, payload builders, and a high-level client for doing non-batch operations called the `GroupsManager` (#435, #443)
- globus-sdk now provides much more complete type annotations coverage, allowing type checkers like `mypy` to catch a much wider range of usage errors (#442)

1.12.6 v3.0.0b2

- Add scope constants and scope construction helpers. See new documentation on *scopes and ScopeBuilders* for details (#437, #440)
- Improve the rendering of API exceptions in stack traces to include the method, URI, and authorization scheme (if recognized) (#439)
- Payload helper objects (`TransferData`, `DeleteData`, and `SearchQuery`) now inherit from a custom object, not `dict`, but they are still dict-like in behavior (#438)
- API Errors now have an attached `info` object with parsed error data where applicable. See the *ErrorInfo documentation* for details (#441)

1.12.7 v3.0.0b1

- Add support for `TransferClient.get_shared_endpoint_list` (#434)
- Passthrough parameters to SDK methods for query params and body params are no longer accepted as extra keyword arguments. Instead, they must be passed explicitly in a `query_params`, `body_params`, or `additional_fields` dictionary, depending on the context (#433)
- The interface for retry parameters has been simplified. `RetryPolicy` objects have been merged into the transport object, and retry parameters like `max_retries` may now be supplied directly as `transport_params` (#430)

1.12.8 v3.0.0a4

- Fix several paginators which were broken in `3.0.0a3` (#431)
- Add `BaseClient` to the top-level exports of `globus_sdk`, so it can now be accessed under the name `globus_sdk.BaseClient`
- Autodocumentation of paginated methods (#432)

1.12.9 v3.0.0a3

- Pagination has changed significantly. (#418)

** **Methods which support pagination like `TransferClient.endpoint_search` no** longer return an iterable `PaginatedResource` type. Instead, these client methods return `GlobusHTTPResponse` objects with a single page of results.

** **Paginated variants of these methods are available by renaming a call from** `client.<method>` to `client.paginated.<method>`. So, for example, a `TransferClient` now supports `client.paginated.endpoint_search()`. The arguments to this function are the same as the original method.

** **client.paginated.<method> calls return Paginator objects, which** support two types of iteration: by `pages()` and by `items()`. To replicate the same behavior as SDK v1.x and v2.x `PaginatedResource` types, use `items()`, as in `client.paginated.endpoint_search("query").items()`

1.12.10 v3.0.0a2

- Refactor response classes (#425)
- A new subpackage is available for public use, `globus_sdk.tokenstorage` (#405)
- Add client for Globus Groups API, `globus_sdk.GroupsClient`. Includes a dedicated error class, `globus_sdk.GroupsAPIError`

1.12.11 v3.0.0a1

- Update documentation site style and layout (#423)
- The interface for `GlobusAuthorizer` now defines `get_authorization_header` instead of `set_authorization_header`, and additional keyword arguments are not allowed (#422)
- New Transport layer handles HTTP details, variable payload encodings, and automatic request retries (#417)
- Instead of `json_body=...` and `text_body=...`, use `data=...` combined with `encoding="json"`, `encoding="form"`, or `encoding="text"` to format payload data. `encoding="json"` is the default when `data` is a dict.
- By default, requests are retried automatically on potentially transient error codes (e.g. `http_status=500`) and network errors with exponential backoff
- `globus_sdk.BaseClient` and its subclasses define `retry_policy` and `transport_class` class attributes which can be used to customize the retry behavior used
- `globus-sdk` now provides PEP561 typing data (#420)
- The JWT dependency has been updated to `pyjwt>=2,<3` (#416)
- The config files in `~/.globus.cfg` and `/etc/globus.cfg` are no longer used. Configuration can now be done via environment variables (#409)
- `BaseClient.app_name` is a property with a custom setter, replacing `set_app_name` (#415)
- `OAuthTokenResponse.decode_id_token` can now be provided a JWK and openid configuration as parameters. `AuthClient` implements methods for fetching these data, so that they can be fetched and stored outside of this call. There is no automatic caching of these data. (#403)
- Remove `allowed_authorizer_types` restriction from `BaseClient` (#407)
- Remove `auth_client=...` parameter to `OAuthTokenResponse.decode_id_token` (#400)

1.12.12 v2.0.1

- Remove support for python2 (#396, #397, #398)

Note: globus-sdk version 2.0.0 was yanked due to a release issue. Version 2.0.1 is the first 2.x version.

1.12.13 v1.11.0

- Add support for task skipped errors via `TransferClient.task_skipped_errors` and `TransferClient.endpoint_manager_task_skipped_errors` (#393)
- Internal maintenance (#389, #390, #391, #392)

1.12.14 v1.10.0

- Add support for pyinstaller installation of globus-sdk (#387)

1.12.15 v1.9.1

- Fix `GlobusHTTPResponse` to handle responses with no `Content-Type` header (#375)

1.12.16 v1.9.0

- Add `globus_sdk.IdentityMap`, a mapping-like object for Auth ID lookups (#367)
- Minor documentation and build improvements (#369, #362)
- Don't append trailing slashes when no path is given to a low-level client method like `get()` (#364)
- Add `external_checksum` and `checksum_algorithm` to `TransferData.add_item()` named arguments (#365)

1.12.17 v1.8.0

- Add a property to paginated results which shows if more results are available (#346)
- Update docs to state that Globus SDK uses semver (#357)
- Fix `RefreshTokenAuthorizer` to handle a new `refresh_token` being sent back by Auth (#359)
- Fix typo in endpoint_search log message (#355)
- Fix Globus Web App activation links in docs (#356)

1.12.18 v1.7.1

- Allow arbitrary keyword args to `TransferData.add_item()` and `DeleteData.add_item()`, which passthrough to the item bodies (#339)
- Minor internal improvements (#342, #343)

1.12.19 v1.7.0

- Add `get_task` and `get_task_list` to `SearchClient` (#335, #336)
- Internal maintenance and testing improvements (#331, #334, #333)

1.12.20 v1.6.1

- Replace egg distribution format with wheels (#314)
- Internal maintenance

1.12.21 v1.6.0

- Correct handling of environment=”production” as an argument to client construction (#307)
- RenewingAuthorizer and its subclasses now expose the check_expiration_time method (#309)
- Allow parameters to be passed to customize the request body of ConfidentialAppAuthClient.oauth2_get_dependent_tokens (#308)
- Use sha256 hashes of tokens (instead of last 5 chars) in debug logging (#305)
- Add the patch() method to BaseClient and its subclasses, sending an HTTP PATCH request (#302)
- Officially add support for python 3.7 (#300)
- Make pickling SDK objects safer (but still not officially supported!) (#284)
- Malformed SDK usage may now raise GlobusSDKUsageError instead of ValueError. GlobusSDKUsageError inherits from ValueError (#281)
- Numerous documentation improvements (#279, #294, #296, #297)

1.12.22 v1.5.0

- Add support for retrieving a local Globus Connect Personal endpoint’s UUID (#276)
- Fix bug in search client parameter handling (#274)

1.12.23 v1.4.1

- Send `Content-Type: application/json` on requests with JSON request bodies (#266)
- Support connection timeouts. Default timeout of 60 seconds (#264)

1.12.24 v1.4.0

- Access token response data by way of scope name (#261)
- Make `cryptography` a strict requirement, globus-sdk[jwt] is no longer necessary (#257, #260)
- Simplify OAuthTokenResponse.decode_id_token to not require the client as an argument (#255)
- Add (beta) SearchClient class (#259)

1.12.25 v1.3.0

- Improve error message when installation onto python2.6 is attempted (#245)
- Raise errors on client instantiation when `GLOBUS_SDK_ENVIRONMENT` appears to be invalid, support `GLOBUS_SDK_ENVIRONMENT=preview` (#247)

1.12.26 v1.2.2

- Allow client classes to accept `base_url` as an argument to `__init__()` (#241)
- Fix packaging to not include testsuite (#232)
- Improve docs on `TransferClient` helper classes (#231, #233)

1.12.27 v1.2.1

- Use PyJWT instead of python-jose for JWT support (#227)

1.12.28 v1.2.0

- Add Transfer symlink support (#218)
- Better handle UTF-8 inputs (#208)
- Fix endpoint manager resume (#224)
- Doc Updates & Minor Improvements

1.12.29 v1.1.1

- Use correct paging style when making `endpoint_manager_task_list` calls (#210)

1.12.30 v1.1.0

- Add endpoint_manager methods to TransferClient (#191, #199, #200, #201, #203)
- Change "identities_set" to "identity_set" for token introspection (#163)
- Fix docs references to `oauth2_start_flow_*` (#190)
- Support iterable requested_scopes everywhere (#185)
- Add python 3.6 to supported platforms (#180)
- Remove "Beta" from docs (#179)
- Update dev status classifier to 5, prod (#178)
- Numerous improvements to testsuite

1.12.31 v1.0.0

- Adds `AuthAPIError` with more flexible error payload handling (#175)

1.12.32 v0.7.2

- Add `AuthClient.validate_token` (#172)
- Bugfix for `on_refresh` users of `RefreshTokenAuthorizer` and `ClientCredentialsAuthorizer` (#173)

1.12.33 v0.7.1

- Remove deprecated `oauth2_start_flow_*` methods (#170)
- Add the `ClientCredentialsAuthorizer` (#164)
- Add `jwt` extra install target. `pip install "globus_sdk[jwt]"` installs `python-jose` (#169)

1.12.34 v0.7.0

- Make `OAuthTokenResponse.decode_id_token()` respect `ssl_verify=no` configuration (#161)
- Remove all properties of `OAuthTokenResponse` other than `by_resource_server` (#162)

1.12.35 v0.6.0

- Opt out of the Globus Auth behavior where a `GET` of an identity username will provision that identity (#145)
- Fixup OAuth2 PKCE to be spec-compliant (#154)
- Wrap some `requests` network-related errors in custom exceptions (#155)
- Add `deadline` support to `TransferData` and `DeleteData` (#159)

1.12.36 v0.5.1

- Add support for the `prefill_named_grant` option to the Native App authorization flow (#143)
- Unicode string improvements (#129)
- Better handle unexpected error payloads (#135)

1.13 Upgrading

This guide covers upgrading and migration between Globus SDK versions. It is meant to help explain and resolve incompatibilities and breaking changes, and does not cover all new features.

When upgrading, you should also read the relevant section of the *CHANGELOG*. The changelog can also be a source of information about new features between major releases.

Many explanations are written in terms of `TransferClient` for consistency, but apply to all client classes, including `AuthClient`, `NativeAppAuthClient`, `ConfidentialAppAuthClient`, `SearchClient`, and `GroupsClient`.

1.13.1 Version Parsing

In the event that a codebase must support multiple versions of the globus-sdk at the same time, consider adding this snippet:

```
from distutils.version import LooseVersion
import globus_sdk

GLOBUS_SDK_VERSION = tuple(LooseVersion(globus_sdk.__version__).version)
GLOBUS_SDK_MAJOR_VERSION = GLOBUS_SDK_VERSION[0]
```

This will parse the Globus SDK version information into a tuple and grab the first element (the major version number) as an integer.

Then, code can dispatch with

```
if GLOBUS_SDK_MAJOR_VERSION < 3:
    pass  # do one thing
else:
    pass  # do another
```

1.13.2 From 1.x or 2.x to 3.0

The *v3 changelog* covers the full list of changes made in version 3 of the Globus SDK.

Because version 2 did not introduce any changes to the SDK code other than supported python versions, you may also want to view this section when upgrading from version 1.

Type Annotations

The Globus SDK now provides PEP 561 type annotation data.

This means that codebases which use `mypy` or similar tools to check type annotations may see new warnings or errors when using version 3 of the SDK.

Note: If you believe an annotation in the SDK is incorrect, please visit our issue tracker to file a bug report!

Automatic Retries

Globus SDK client methods now automatically retry failing requests when encountering network errors and certain classes of server errors (e.g. rate limiting).

For most users, retry logic can be removed. Change:

```
import globus_sdk

# globus-sdk v1 or v2
tc = globus_sdk.TransferClient(...)

response = None
count, max_retries = 0, 10
while response is None and count < max_retries:
    count += 1
    try:  # any operation, just an example
        response = tc.get_endpoint(foo)
    except globus_sdk.NetworkError:
        pass

# globus-sdk v3
tc = globus_sdk.TransferClient(...)
response = tc.get_endpoint(foo)  # again, just an example operation
```

Import BaseClient from globus_sdk

You may be using the globus-sdk `BaseClient` object to implement a custom client or for type annotations.

Change:

```
# globus-sdk v1 or v2
from globus_sdk.base import BaseClient

# globus-sdk v3
from globus_sdk import BaseClient
```

Import exceptions from globus_sdk

Several exceptions which were available in v2 under `globus_sdk.exc` are now only available from the `globus_sdk` namespace.

Change:

```
# globus-sdk v1 or v2
from globus_sdk.exc import SearchAPIError, TransferAPIError, AuthAPIError

# globus-sdk v3
from globus_sdk import SearchAPIError, TransferAPIError, AuthAPIError
```

Note that this also may appear in your exception handling, as in:

```
# globus-sdk v1 or v2
from globus_sdk import exc

try:
    ...
except exc.TransferAPIError:  # by way of example, any error here
    ...

# globus-sdk v3
import globus_sdk

try:
    ...
except globus_sdk.TransferAPIError:
    ...
```

Low Level API for Passing Data is Improved

In version 2 of the SDK, passing data to client `post()`, `put()`, and `patch()` methods required the use of either `json_body` or `text_body`. Furthermore, `text_body` would (confusingly!) send a FORM body if it were passed a dictionary.

Now, these behaviors are described by `data` (a body for these HTTP methods) and `encoding` (an explicit data format parameter). If the `encoding` is not set, the default behavior is that if `data` is a dictionary, it will be sent as JSON. If `data` is a string, it will be sent as text.

`encoding` can be set to `"json"` or `"form"` to explicitly format the data.

Change code for a JSON PUT like so:

```
# globus-sdk v1 or v2
from globus_sdk import TransferClient

tc = TransferClient(...)
tc.put("/some/custom/path", json_body={"a": "dict", "of": "data"})

# globus-sdk v3
from globus_sdk import TransferClient
```

(continues on next page)

(continued from previous page)

```
tc = TransferClient(...)
tc.put("/some/custom/path", data={"a": "dict", "of": "data"})
```

Or a FORM POST like so:

```
# globus-sdk v1 or v2
from globus_sdk import TransferClient

tc = TransferClient(...)
tc.post("/some/custom/path", text_body={"a": "dict", "of": "data"})

# globus-sdk v3
from globus_sdk import TransferClient

tc = TransferClient(...)
tc.put("/some/custom/path", data={"a": "dict", "of": "data"}, encoding="form")
```

Passthrough Parameters are Explicit

Many methods in version 2 accepted arbitrary keyword arguments which were then transformed into query or body parameters based on the context. This is no allowed, but methods can still be passed additional query parameters in the form of a `query_params` dict.

For example, if the Transfer API is known to support a query param `foo=bar` for `GET Endpoint`, but the SDK does not include this parameter, the way that it can be added to a request has changed as follows:

```
# globus-sdk v1 or v2
from globus_sdk import TransferClient

tc = TransferClient(...)
tc.get_endpoint(epid, foo="bar")

# globus-sdk v3
from globus_sdk import TransferClient

tc = TransferClient(...)
tc.get_endpoint(epid, query_params={"foo": "bar"})
```

Note: If a parameter which you need is not supported by the Globus SDK, use `query_params` to work around it! But also, feel free to visit our issue tracker to request an improvement.

Responses are always GlobusHTTPResponse

In version 2, *GlobusHTTPResponse* inherited from a base class, *GlobusResponse*. In version 3, the distinction has been eliminated and responses are only *GlobusHTTPResponse*.

This may appear in contexts where you type annotate or use `isinstance` checks to check the type of an object.

Change:

```
# globus-sdk v1 or v2
from globus_sdk.response import GlobusResponse

data = some_complex_func()
if isinstance(data, GlobusResponse):
    ...

# globus-sdk v3
from globus_sdk import GlobusHTTPResponse

data = some_complex_func()
if isinstance(data, GlobusHTTPResponse):
    ...
```

Pagination is now explicit

In version 2, paginated methods of `TransferClient` returned a `PaginatedResource` iterable type. In version 3, no methods return paginators by default, and pagination is always opt-in. See also *doc on making paginated calls*.

Change:

```
# globus-sdk v1 or v2
from globus_sdk import TransferClient

tc = TransferClient(...)
for endpoint_info in tc.endpoint_search("query"):
    ...

# globus-sdk v3
from globus_sdk import TransferClient

tc = TransferClient(...)
for endpoint_info in tc.paginated.endpoint_search("query").items():
    ...
```

Authorizer Methods

`GlobusAuthorizer` objects have had their methods modified.

In particular, in version 2, authorizers have a method `set_authorization_header` for modifying a dict.

This has been replaced in version 3 with a method `get_authorization_header` which returns an `Authorization` header value.

Configuration has Changed

The Globus SDK no longer reads configuration data from `/etc/globus.cfg` or `~/.globus.cfg`.

If you are using these files to customize the behavior of the SDK, see *the configuration documentation*.

Internal Changes to components including Config

Several modules and components which are considered mostly or entirely internal have been reorganized.

In particular, if you are using undocumented methods from `globus_sdk.config`, note that this has been largely rewritten. (These are not considered public APIs.)

1.13.3 From 1.x to 2.0

Also see the *v2 changelog*.

When upgrading from version 1 to version 2 of the Globus SDK, no code changes should be necessary.

Version 2 removed support for python2 but made no other changes.

Simply ensure that you are running python 3.6 or later and update version specifications to `globus_sdk>=2,<3`.

1.14 Globus SDK Examples

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

1.14.1 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)

# 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
```

(continues on next page)

(continued from previous page)

```
# 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`.

1.14.2 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))

auth_code = 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)
```

1.14.3 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:".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.

1.14.4 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"]))
```

1.14.5 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 definition 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.
    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"]))
```

1.14.6 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
        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.

1.14.7 Recursive `ls` via TransferClient

The Globus Transfer API does not offer a recursive variant of the `ls` operation. There are several reasons for this, but most obviously: `ls` is synchronous, and a recursive listing may be very slow.

This example demonstrates how to write a breadth-first traversal of a dir tree using a local deque to implement recursive `ls`. You will need a properly authenticated *TransferClient*.

```
from collections import deque


def _recursive_ls_helper(tc, ep, queue, max_depth):
    while queue:
        abs_path, rel_path, depth = queue.pop()
        path_prefix = rel_path + "/" if rel_path else ""
```

(continues on next page)

(continued from previous page)

```
        res = tc.operation_ls(ep, path=abs_path)

        if depth < max_depth:
            queue.extend(
                (
                    res["path"] + item["name"],
                    path_prefix + item["name"],
                    depth + 1,
                )
                for item in res["DATA"]
                if item["type"] == "dir"
            )
        for item in res["DATA"]:
            item["name"] = path_prefix + item["name"]
            yield item


# tc: a TransferClient
# ep: an endpoint ID
# path: the path to list recursively
def recursive_ls(tc, ep, path, max_depth=3):
    queue = deque()
    queue.append((path, "", 0))
    yield from _recursive_ls_helper(tc, ep, queue, max_depth)
```

This acts as a generator function, which you can then use for iteration, or evaluate with `list()` or other expressions which will iterate over values from the generator.

adding sleep

One of the issues with the above recursive listing tooling is that it can easily run into rate limits on very large dir trees with a fast filesystem.

To avoid issues, simply add a periodic sleep. For example, we could add a `sleep_frequency` and `sleep_duration`, then count the number of `ls` calls that have been made. Every `sleep_frequency` calls, sleep for `sleep_duration`.

The modifications in the helper would be something like so:

```
import time


def _recursive_ls_helper(tc, ep, queue, max_depth, sleep_frequency, sleep_duration):
    call_count = 0
    while queue:
        abs_path, rel_path, depth = queue.pop()
        path_prefix = rel_path + "/" if rel_path else ""

        res = tc.operation_ls(ep, path=abs_path)

        call_count += 1
        if call_count % sleep_frequency == 0:
            time.sleep(sleep_duration)
```

(continues on next page)

(continued from previous page)

```
        # as above
        ...
```

parameter passthrough

What if you want to pass parameters to the `ls` calls? Accepting that some behaviors – like order-by – might not behave as expected if passed to the recursive calls, you can still do-so. Add `ls_params`, a dictionary of additional parameters to pass to the underlying `operation_ls` invocations.

The helper can assume that a dict is passed, and the wrapper would just initialize it as `{}` if nothing is passed.

Something like so:

```
def _recursive_ls_helper(tc, ep, queue, max_depth, ls_params):
    call_count = 0
    while queue:
        abs_path, rel_path, depth = queue.pop()
        path_prefix = rel_path + "/" if rel_path else ""

        res = tc.operation_ls(ep, path=abs_path, **ls_params)

        # as above
        ...


# importantly, the params should default to `None` and be rewritten to a
# dict in the function body (parameter default bindings are modifiable)
def recursive_ls(tc, ep, path, max_depth=3, ls_params=None):
    ls_params = ls_params or {}
    queue = deque()
    queue.append((path, "", 0))
    yield from _recursive_ls_helper(
        tc, ep, queue, max_depth, sleep_frequency, sleep_duration, ls_params
    )
```

What if we want to have different parameters to the top-level `ls` call from any of the recursive calls? For example, maybe we want to filter the items found in the initial directory, but not in subdirectories.

In that case, we just add on another layer: `top_level_ls_params`, and we only use those parameters on the initial call.

```
def _recursive_ls_helper(
    tc,
    ep,
    queue,
    max_depth,
    ls_params,
    top_level_ls_params,
):
    first_call = True
    while queue:
        abs_path, rel_path, depth = queue.pop()
```

(continues on next page)

(continued from previous page)

```
        path_prefix = rel_path + "/" if rel_path else ""

        use_params = ls_params
        if first_call:
            # on modern pythons, dict expansion can be used to easily
            # combine dicts
            use_params = {**ls_params, **top_level_ls_params}
            first_call = False
        res = tc.operation_ls(ep, path=abs_path, **use_params)

        # again, the rest of the loop is the same
        ...


def recursive_ls(
    tc,
    ep,
    path,
    max_depth=3,
    ls_params=None,
    top_level_ls_params=None,
):
    ls_params = ls_params or {}
    top_level_ls_params = top_level_ls_params or {}
    ...
```

With Sleep and Parameter Passing

We can combine sleeps and parameter passing into one final, complete example:

```
import time
from collections import deque


def _recursive_ls_helper(
    tc,
    ep,
    queue,
    max_depth,
    sleep_frequency,
    sleep_duration,
    ls_params,
    top_level_ls_params,
):
    call_count = 0
    while queue:
        abs_path, rel_path, depth = queue.pop()
        path_prefix = rel_path + "/" if rel_path else ""

        use_params = ls_params
        if call_count == 0:
```

(continues on next page)

(continued from previous page)

```
            use_params = {**ls_params, **top_level_ls_params}

        res = tc.operation_ls(ep, path=abs_path, **use_params)

        call_count += 1
        if call_count % sleep_frequency == 0:
            time.sleep(sleep_duration)

        if depth < max_depth:
            queue.extend(
                (
                    res["path"] + item["name"],
                    path_prefix + item["name"],
                    depth + 1,
                )
                for item in res["DATA"]
                if item["type"] == "dir"
            )
        for item in res["DATA"]:
            item["name"] = path_prefix + item["name"]
            yield item


def recursive_ls(
    tc,
    ep,
    path,
    max_depth=3,
    sleep_frequency=10,
    sleep_duration=0.5,
    ls_params=None,
    top_level_ls_params=None,
):
    ls_params = ls_params or {}
    top_level_ls_params = top_level_ls_params or {}
    queue = deque()
    queue.append((path, "", 0))
    yield from _recursive_ls_helper(
        tc,
        ep,
        queue,
        max_depth,
        sleep_frequency,
        sleep_duration,
        ls_params,
        top_level_ls_params,
    )
```

PYTHON MODULE INDEX

g

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

Z