Building on the Globus Python SDK

Stephen Rosen

April 11, 2017

SDK Overview

The Globus SDK is a client-side library which provides language bindings for entities and methods available via web APIs.

SDK Overview

The Globus SDK is a client-side library which provides language bindings for entities and methods available via web APIs.
In principle, an SDK could be built for any language. Today, we only support Python. (Hence references to the "Python SDK")
Everything in this presentation would apply to SDKs for other languages. They only have to maintain the same basic design.

How is the SDK Organized

How is the SDK Organized

Making API resources native parts of a language requires some way of modeling those resources in that language.

How is the SDK Organized

Making API resources native parts of a language requires some way of modeling those resources in that language.
The SDK tries to maintain a very simple, class-based model for interacting with the service. Specifically:

How is the SDK Organized

Making API resources native parts of a language requires some way of modeling those resources in that language.
The SDK tries to maintain a very simple, class-based model for interacting with the service. Specifically:

- An adapter class for each API. e.g. *TransferClient*

How is the SDK Organized

Making API resources native parts of a language requires some way of modeling those resources in that language.
The SDK tries to maintain a very simple, class-based model for interacting with the service. Specifically:

- An adapter class for each API. e.g. *TransferClient*
- Authorization objects which handle API authorization, e.g. *RefreshTokenAuthorizer*

How is the SDK Organized

Making API resources native parts of a language requires some way of modeling those resources in that language.
The SDK tries to maintain a very simple, class-based model for interacting with the service. Specifically:

- An adapter class for each API. e.g. *TransferClient*
- Authorization objects which handle API authorization, e.g. *RefreshTokenAuthorizer*
- API Responses as *GlobusResponse* objects

How is the SDK Organized

Making API resources native parts of a language requires some way of modeling those resources in that language.
The SDK tries to maintain a very simple, class-based model for interacting with the service. Specifically:

- An adapter class for each API. e.g. *TransferClient*
- Authorization objects which handle API authorization, e.g. *RefreshTokenAuthorizer*
- API Responses as *GlobusResponse* objects
- Always allow access to verbatim API response data, usually as Python *dict*s

How is the SDK Organized

Making API resources native parts of a language requires some way of modeling those resources in that language.
The SDK tries to maintain a very simple, class-based model for interacting with the service. Specifically:

- An adapter class for each API. e.g. *TransferClient*
- Authorization objects which handle API authorization, e.g. *RefreshTokenAuthorizer*
- API Responses as *GlobusResponse* objects
- Always allow access to verbatim API response data, usually as Python *dict*s
- Contain the HTTP interface without concealing it – i.e. throw Python Exceptions, but attach *http_status* to them, etc

Why Does This Model Matter?

Why am I even up here, in front of you, talking about these internal minutae of this library?

Why Does This Model Matter?

Why am I even up here, in front of you, talking about these internal minutae of this library?
This model is extensible, the objects are composable, and it is a viable basis for building tools against APIs of your own which authenticate with Globus Auth.

Why Does This Model Matter?

Why am I even up here, in front of you, talking about these internal minutae of this library?
This model is extensible, the objects are composable, and it is a viable basis for building tools against APIs of your own which authenticate with Globus Auth.
All that's necessary to support a new API is a new Client class to act as an adapter.

Is This Going to Get Technical? I'm Getting Woozy!

Please take good care of yourselves and drink lots of water.

Is This Going to Get Technical? I'm Getting Woozy!

Is This Going to Get Technical? I'm Getting Woozy!

I'm going to get fairly technical in content here. You are *not* expected to be Python experts – please just think about what *kinds* of usage we're enabling if you wish to dig in.

Is This Going to Get Technical? I'm Getting Woozy!

I'm going to get fairly technical in content here. You are *not* expected to be Python experts – please just think about what *kinds* of usage we're enabling if you wish to dig in.
Even if you are comfortable with the content, the focus is not only on How we're doing things, but also on What we're trying to do.
Id est, let's make programming with Globus (and maybe with your API too!) easy.

Is This Going to Get Technical? I'm Getting Woozy!

I'm going to get fairly technical in content here. You are *not* expected to be Python experts – please just think about what *kinds* of usage we're enabling if you wish to dig in.
Even if you are comfortable with the content, the focus is not only on How we're doing things, but also on What we're trying to do.
Id est, let's make programming with Globus (and maybe with your API too!) easy.
Let's make interactive experimentation with these APIs a real possibility, and a source of joy.

The Simplest Possible Example

The Simplest Possible Example

To start with, let's consider the simplest possible form of a search API. There is one, and only one, call available:

```
GET /search?q=hello
```

against the API at *search.example.com*.

The Simplest Possible Example

To start with, let's consider the simplest possible form of a search API. There is one, and only one, call available:

```
GET /search?q=hello
```

against the API at *search.example.com*.
What does the client for this API look like?

The Simplest Possible Example (cont.)

GET /*search*?*q* = *hello* against *search.example.com*

The Simplest Possible Example (cont.)

GET /search?q = hello against *search.example.com*

```
from globus_sdk.base import BaseClient


class SearchClient(BaseClient):
    def __init__(self, *args, **kwargs):
        super(SearchClient, self).__init__(self, "search", *args, **kwargs)
        self.base_url = "https://search.example.com"

    def search(self, q):
        return self.get("/search", params={"q": q})
```

The Simplest Possible Example (cont.)

GET /search?q = hello against *search.example.com*

```
from globus_sdk.base import BaseClient


class SearchClient(BaseClient):
    def __init__(self, *args, **kwargs):
        super(SearchClient, self).__init__(self, "search", *args, **kwargs)
        self.base_url = "https://search.example.com"

    def search(self, q):
        return self.get("/search", params={"q": q})
```

And one could use it as in

```
client = SearchClient()
client.search("hello")
```

What About Authorization

What About Authorization

Yeah...

What About Authorization

Yeah...
What *about* authorization?

What About Authorization

Yeah...
What *about* authorization?
Well, out of the box your brand new client will support it just fine.
Assuming you have some tokens from doing an authentication flow, then...

What About Authorization

Yeah...
What *about* authorization?
Well, out of the box your brand new client will support it just fine.
Assuming you have some tokens from doing an authentication flow, then...

```
from globus_sdk import RefreshTokenAuthorizer

# skip the details on the authorizer for now
client = SearchClient(authorizer=RefreshTokenAuthorizer(...))
client.search("hello, but authenticated")
```

Authentication for your API

Although authorizing access once a login flow is complete is pretty simple, getting that done requires a few steps. Specifically, you need

Authentication for your API

Although authorizing access once a login flow is complete is pretty simple, getting that done requires a few steps. Specifically, you need

- your API registered in Globus as a Resource Server

Authentication for your API

Although authorizing access once a login flow is complete is pretty simple, getting that done requires a few steps. Specifically, you need

- your API registered in Globus as a Resource Server
- at least one, possibly more than one, Scope for your Resource Server

Authentication for your API

Although authorizing access once a login flow is complete is pretty simple, getting that done requires a few steps. Specifically, you need

- your API registered in Globus as a Resource Server
- at least one, possibly more than one, Scope for your Resource Server
- a client application – a consumer of your API – registered in Globus, with or without credentials

Authentication for your API

Although authorizing access once a login flow is complete is pretty simple, getting that done requires a few steps. Specifically, you need

- your API registered in Globus as a Resource Server
- at least one, possibly more than one, Scope for your Resource Server
- a client application – a consumer of your API – registered in Globus, with or without credentials
- the user must complete a login flow, via your client application definition, authorizing it to access your API on his or her behalf

Authentication for your API

Although authorizing access once a login flow is complete is pretty simple, getting that done requires a few steps. Specifically, you need

- your API registered in Globus as a Resource Server
- at least one, possibly more than one, Scope for your Resource Server
- a client application – a consumer of your API – registered in Globus, with or without credentials
- the user must complete a login flow, via your client application definition, authorizing it to access your API on his or her behalf

A full discussion of all – perhaps even of any – of these is beyond the scope of this talk.

Customizing Error and Response Types

If it's your API, you may very well have your own standards and conventions for error formats. If your API is similar enough to Globus APIs, maybe you can make do with the default error handler.

Customizing Error and Response Types

If it's your API, you may very well have your own standards and conventions for error formats. If your API is similar enough to Globus APIs, maybe you can make do with the default error handler.

You may want to do more sophisticated parsing of your errors. Maybe your errors always have a "reason" field, or a "request_id" for logging, by way of example.

Customizing Error and Response Types

If it's your API, you may very well have your own standards and conventions for error formats. If your API is similar enough to Globus APIs, maybe you can make do with the default error handler.

You may want to do more sophisticated parsing of your errors. Maybe your errors always have a "reason" field, or a "request_id" for logging, by way of example.

The same basic principle may apply to response formats – perhaps your Search API always sends back its results in an array named "hits".

Custom Response Class

```
from globus_sdk import GlobusHTTPResponse


class SearchAPIResponse(GlobusHTTPResponse):
    """Assume (parsed) JSON response data, handled by GlobusHTTPResponse""
    def __iter__(self):
        """Search responses are iterable -- iterate over search hits array"""
        return iter(self["hits"])
```

Custom Response Class

```
from globus_sdk import GlobusHTTPResponse


class SearchAPIResponse(GlobusHTTPResponse):
    """Assume (parsed) JSON response data, handled by GlobusHTTPResponse""
    def __iter__(self):
        """Search responses are iterable -- iterate over search hits array"""
        return iter(self["hits"])
```

Well, that doesn't seem too hard. But it must be really tricky to get it plugged correctly into the *SearchClient* right?

Custom Response Class (cont.)

...it must be really tricky to get it plugged correctly into the *SearchClient* right?
Obviously, I wouldn't ask such a question unless it's really, really easy.

Custom Response Class (cont.)

...it must be really tricky to get it plugged correctly into the *SearchClient* right?
Obviously, I wouldn't ask such a question unless it's really, really easy.
There are two ways: is this a default class for all responses from this API, via this client class? Or is this a specific response type for this method?

Custom Response Class (cont.)

...it must be really tricky to get it plugged correctly into the *SearchClient* right?
Obviously, I wouldn't ask such a question unless it's really, really easy.
There are two ways: is this a default class for all responses from this API, via this client class? Or is this a specific response type for this method?
Either

```
class SearchClient(BaseClient):
    default_response_class = SearchAPIResponse
    ...
```

or

```
    def search(self, q):
        return self.get("/search", params={"q": q},
                        response_class=SearchAPIResponse)
```

depending on your needs.

Why would I customize responses?

Why would I customize responses?

Start with the basics:

```
class SearchAPIResponse(GlobusHTTPResponse):
    def __iter__(self):
        return iter(self["hits"])

class SearchClient(BaseClient):
    ...
    def search(self, q):
        return self.get("/search", params={"q": q},
                        response_class=SearchAPIResponse)
```

Why would I customize responses?

Start with the basics:

```
class SearchAPIResponse(GlobusHTTPResponse):
    def __iter__(self):
        return iter(self["hits"])

class SearchClient(BaseClient):
    ...
    def search(self, q):
        return self.get("/search", params={"q": q},
                        response_class=SearchAPIResponse)
```

Now we can iterate over the “hits” in a response:

```
client = SearchClient()
for hit in client.search("hello"):
    # some helper which does pretty printing
    print_hit(hit)
```

Continuing to Custom Error Classes

Just as easily as you can set the *response_class* for a client, you can give it a custom error class. This may merely serve to ensure that you have your own exception type, distinct from *GlobusError*, or it may do complex parsing.

Continuing to Custom Error Classes

Just as easily as you can set the *response_class* for a client, you can give it a custom error class. This may merely serve to ensure that you have your own exception type, distinct from *GlobusError*, or it may do complex parsing.
Starting with

```
from globus_sdk import GlobusAPIError


class SearchAPIError(GlobusAPIError)
    """No special methods or parsing on this specific error type"""
```

it's easy enough to apply this class to a client class.

Continuing to Custom Error Classes

Just as easily as you can set the *response_class* for a client, you can give it a custom error class. This may merely serve to ensure that you have your own exception type, distinct from *GlobusError*, or it may do complex parsing.
Starting with

```
from globus_sdk import GlobusAPIError


class SearchAPIError(GlobusAPIError)
    """No special methods or parsing on this specific error type"""
```

it's easy enough to apply this class to a client class.
All that's needed is

```
class SearchClient(BaseClient):
    error_class = SearchAPIError
    ...
```

It's My API, Why Would I Do ANY of This?

It's My API, Why Would I Do ANY of This?

Short answer? Auth is hard. Or, at the very least, messy.

It's My API, Why Would I Do ANY of This?

As we've started to build tools and products on top of the SDK over this past year, some things have become obvious pain points. The biggest, and most significant one to address is Authorization.

It's My API, Why Would I Do ANY of This?

As we've started to build tools and products on top of the SDK over this past year, some things have become obvious pain points. The biggest, and most significant one to address is Authorization. Reminder: Authentication is "user logs in", and Authorization is "credential gets sent to API"

It's My API, Why Would I Do ANY of This?

As we've started to build tools and products on top of the SDK over this past year, some things have become obvious pain points. The biggest, and most significant one to address is Authorization.
Reminder: Authentication is "user logs in", and Authorization is "credential gets sent to API"
We provide *GlobusAuthorizer*s which handle the messy authorization activities of tracking tokens, watching their expiration times, requesting renewed credentials, and generally making things "just work".

It's My API, Why Would I Do ANY of This?

As we've started to build tools and products on top of the SDK over this past year, some things have become obvious pain points. The biggest, and most significant one to address is Authorization.
Reminder: Authentication is "user logs in", and Authorization is "credential gets sent to API"
We provide *GlobusAuthorizer*s which handle the messy authorization activities of tracking tokens, watching their expiration times, requesting renewed credentials, and generally making things "just work".
You can re-invent these wheels if you want, but perhaps, if you do, maybe take a peek at our source... Like I said, Auth is hard.

Proof of Concept? No!

We really are using this toolkit internally. Even as we work on the new Search API, we need client-side tools and scripts.

Proof of Concept? No!

We really are using this toolkit internally. Even as we work on the new Search API, we need client-side tools and scripts.
That *SearchClient* from before? Eerily similar to a *SearchClient* we really use.

Proof of Concept? No!

We really are using this toolkit internally. Even as we work on the new Search API, we need client-side tools and scripts.
That *SearchClient* from before? Eerily similar to a *SearchClient* we really use.
When we first started sharing this library last year, it was just beginning to grow beyond being a toy, a prototype, a proof of concept. Today, we have enough confidence that we're building products on it. We'd like you all to have the confidence in us to do the same.

I Already Use the SDK, and I Hate/Love It

Some of you are already using the SDK to start building applications. There are probably many things more you want it to do, plenty of things you like about it lots, and a handful which you find unsatisfactory.

I Already Use the SDK, and I Hate/Love It

Some of you are already using the SDK to start building applications. There are probably many things more you want it to do, plenty of things you like about it lots, and a handful which you find unsatisfactory.
It can only be, maximally, as good as the feedback you give us.
Tell us what you like. Tell us what you don't like.

I Already Use the SDK, and I Hate/Love It

Some of you are already using the SDK to start building applications. There are probably many things more you want it to do, plenty of things you like about it lots, and a handful which you find unsatisfactory.

It can only be, maximally, as good as the feedback you give us.

Tell us what you like. Tell us what you don't like.

Tell us the class names that you find annoying, the functions whose signatures are too big, the helpers which you think are missing, and tell us how it made one of your days – just one is enough – a little bit better.

Say it on the listhost (developer-discuss@globus.org) and tell us in GitHub issues (https://github.com/globus/globus-sdk-python).

Final Notes

I'd like to say a special thanks to everyone, both in and out of Globus, who has made a commit, filed an issue, asked or answered a question, or voiced an opinion.

Final Notes

I'd like to say a special thanks to everyone, both in and out of Globus, who has made a commit, filed an issue, asked or answered a question, or voiced an opinion.

Reminder: Give Us Your Feedback and Contributions

- Open GitHub issues and Pull Requests (`https://github.com/globus/globus-sdk-python/issues`)
- Join the discussion `developer-discuss@globus.org`