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

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
▶ Contain the HTTP interface without concealing it – i.e. throw Python Exceptions, but attach http status to them, etc
How is the SDK Organized

Making API resources native parts of a language requires some way of modeling those resources in that language.
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`.
- Contain the HTTP interface without concealing it – i.e. throw Python Exceptions, but attach `http` status to them, etc.
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`
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`
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 `dicts`
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 `dicts`
- Contain the HTTP interface without concealing it – i.e. throw Python Exceptions, but attach `http_status` to them, etc
Why am I even up here, in front of you, talking about these internal minutiae of this library?
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 am I even up here, in front of you, talking about these internal minutiae 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.
Please take good care of yourselves and drink lots of water.
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.
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.
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.
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

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 example.com.

What does the client for this API look like?
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?
GET /search?q = hello against search.example.com
The Simplest Possible Example (cont.)

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

```python
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
```

```python
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

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

```python
from globus_sdk import RefreshTokenAuthorizer

# skip the details on the authorizer for now
client = SearchClient(authorizer=RefreshTokenAuthorizer(...))

client.search("hello, but authenticated")
```
What About Authorization

Yeah...
Yeah...
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...
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...

```python
from globus_sdk import RefreshTokenAuthorizer

# skip the details on the authorizer for now
client = SearchClient(authorizer=RefreshTokenAuthorizer(....))
client.search("hello, but authenticated")
```
Although authorizing access once a login flow is complete is pretty simple, getting that done requires a few steps. Specifically, you need:

1. Your API registered in Globus as a Resource Server
2. At least one, possibly more than one, Scope for your Resource Server
3. A client application – a consumer of your API – registered in Globus, with or without credentials
4. 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.
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
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

A full discussion of all – perhaps even of any – of these is beyond the scope of this talk.
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.
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.
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.
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”.
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"])
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?
...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.
...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?
...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

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

or

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

depending on your needs.
Why would I customize responses?

Start with the basics:

```python
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)
```
Why would I customize responses?

Start with the basics:

```python
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:

```python
client = SearchClient()
for hit in client.search("hello"):
    # some helper which does pretty printing
    print_hit(hit)
```
Why would I customize responses?

Start with the basics:

```python
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:

```python
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.
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

```python
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

```python
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

```python
class SearchClient(BaseClient):
    error_class = SearchAPIError
    ...
```
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 GlobusAuthorizers 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.
Short answer? Auth is hard. Or, at the very least, messy.
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.
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”
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 *GlobusAuthorizers* which handle the messy authorization activities of tracking tokens, watching their expiration times, requesting renewed credentials, and generally making things “just work”.

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 *GlobusAuthorizers* 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.
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.
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.
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.
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).
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.
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