sbg Documentation

Release 0.7.4

Seven Bridges Genomics

May 13, 2017

Contents

sevenbridges-python is a Python library that provides an interface for the Seven Bridges Platform and Cancer Genomics Cloud public APIs.

The Seven Bridges Platform is a cloud-based environment for conducting bioinformatic analyses. It is a central hub for teams to store, analyze, and jointly interpret their bioinformatic data. The Platform co-locates analysis pipelines alongside the largest genomic datasets to optimize processing, allocating storage and compute resources on demand.

The The Cancer Genomics Cloud (CGC), powered by Seven Bridges, is also a cloud-based computation environment. It was built as one of three pilot systems funded by the National Cancer Institute to explore the paradigm of colocalizing massive genomics datasets, like The Cancer Genomics Atlas (TCGA), alongside secure and scalable computational resources to analyze them. The CGC makes more than a petabyte of multi-dimensional data available immediately to authorized researchers. You can add your own data to analyze alongside TCGA using predefined analytical workflows or your own tools.

CHAPTER 1

Content

Installation

The easiest way to install sevenbridges-python is using pip.

```
$ pip install sevenbridges-python
```

Get the Code

sevenbridges-python is actively developed on GitHub, where the code is always available.

The easiest way to obtain the source is to clone the public repository:

```
$ git clone git://github.com/sbg/sevenbridges-python.git
```

Once you have a copy of the source, you can embed it in your Python package, or install it into your site-packages by invoking:

```
$ python setup.py install
```

If you are interested in reviewing this documentation locally, clone the repository and invoke:

```
$ make html
```

from the docs folder.

Quickstart

On this page, you'll find a reference for the Seven Bridges API Python client.

We encourage you to consult our other API resources:

- The Seven Bridges Github repository, okAPI, which includes Python example scripts such as recipes (which allow you to perform specific tasks) and tutorials (which will walk you through entire analyses) via the API. These recipes and tutorials make use of the sevenbridges-python bindings below.
- The Seven Bridges API documentation on our Knowledge Center, which includes a reference collection of API requests to help you get started right away.

Authentication and Configuration

In order to authenticate with the API, you should pass the following items to sevenbridges-python:

1. Your authentication token
2. The API endpoint you will be interacting with. This is either the endpoint for the Seven Bridges Platform or for the Seven Bridges Cancer Genomics Cloud (CGC).

You can find your authentication token on the respective pages:

- https://igor.sbgenomics.com/developers for the Seven Bridges Platform
- https://cgc.sbgenomics.com/developers for the CGC

The API endpoints for each environment are:

- https://api.sbgenomics.com/v2 for the Seven Bridges Platform
- https://cgc-api.sbgenomics.com/v2 for the CGC.

Note: We will see below how to supply information about your auth token and endpoint to the library.

For more information about the API, including details of the available parameters for each API call, you should check the API documentation before using this library:

- http://docs.sevenbridges.com/docs/the-api for the Seven Bridges Platform.
- http://docs.cancergenomicscloud.org/docs/the-cgc-api for the CGC.

How to use the Quickstart

We recommend that you pay particular attention to the section 'Managing Projects' of this Quickstart, since it contains general information on working with any kind of Platform or CGC resource (projects, files, tasks, etc) via the Python methods available in this library.

Initializing the library

Once you have obtained your authentication token from one of the URLs listed above, you can initialize the `Api` object defined by this library by passing in your authentication token and endpiont. There are three methods to do this. Details of each method are given below:

1. Pass the parameters `url` and `token` and optional `proxies` explicitly when initializing the API object.
2. Set the API endpoint and token to the environment variables `SB_API_ENDPOINT` and `SB_AUTH_TOKEN` respectively.
3. Use a configuration file `$HOME/.sevenbridges/credentials` with the defined credentials parameters. If config is used proxy settings will be read from `$HOME/.sevenbridges/sevenbridges-python/config` .ini like file for section `[proxies]`

Note: Keep your authentication token safe! It encodes all your credentials on the Platform or CGC. Generally, we recommend storing the token in a configuration file, which will then be stored in your home folder rather than in the code itself. This prevents the authentication token from being committed to source code repositories.

Import the library

You should begin by importing the API library sevenbridges-python to your python script that will interact with the API:

```
import sevenbridges as sbg
```

Then, use one of the following three methods to initialize the library:

1. Initialize the library explicitly

The library can be also instantiated explicitly by passing the URL and authentication token as key-value arguments into the `Api` object.

```
api = sbg.Api(url='https://api.sbgenomics.com/v2', token='<TOKEN_HERE>')
```

Note - you can initialize several API clients with different credentials or environments.

2. Initialize the library using environment variables

```
import os

# Usually these variables would be set in the shell beforehand
os.environ['SB_API_ENDPOINT'] = '<https://api.sbgenomics.com/v2 or https://cgc-api.sbgenomics.com/v2>
os.environ['SB_AUTH_TOKEN'] = '<TOKEN_HERE>'

api = sbg.Api()
```

3. Initialize the library using a configuration file

The configuration file, `$HOME/.sevenbridges/credentials`, has a simple `.ini` file format, with the environment (the Seven Bridges Platform, or the CGC) indicated in square brackets, as shown:

```
[default]
api_endpoint = https://api.sbgenomics.com/v2
auth_token = <TOKEN_HERE>

[cgc]
api_endpoint = https://cgc-api.sbgenomics.com/v2
auth_token = <TOKEN_HERE>
```

The `Api` object is the central resource for querying, saving and performing other actions on your resources on the Seven Bridges Platform or CGC. Once you have instantiated the configuration class, pass it to the API class constructor.

```
c = sbg.Config(profile='cgc')
api = sbg.Api(config=c)
```

If not profile is set it will use the default profile.

Note: if user creates the api object `api=sbg.Api()` and does not pass any information the library will first search whether the environment variables are set. If not it will check if the configuration file is present and read the `[default]` profile. If that also fails it will raise an exception

---

Proxy configuration

Proxy configuration can be supplied in three different ways.

- explicit initialization

```
api = sb.Api(url='https://api.sbgenomics.com/v2', token='<TOKEN_HERE>',
       proxies={'https_proxy':'host:port', 'http_proxy': 'host:port'})
```

- environment variables

```
os.environ['HTTP_PROXY'] = 'host:port'
os.environ['HTTPS_PROXY'] = 'host:port'
```

- *$HOME/.sevenbridges/sevenbridges-python/config* configuration file

```
[proxies]
https_proxy=host:port
http_proxy=host:port
```

- Explicit with config

```
config = sb.Config(profile='my-profile',
                   proxies={'https_proxy':'host:port', 'http_proxy': 'host:port'})
api = sb.Api(config=config)
```

Note: Once you set the proxy, all calls including upload and download will use the proxy settings.

Rate limit

For API requests that require authentication (i.e. all requests, except the call to list possible API paths), you can issue a maximum of 1000 requests per 300 seconds. Note that this limit is generally subject to change, depending on API usage and technical limits. Your current rate limit, the number of remaining requests available within the limit, and the time until your limit is reset can be obtained using your `Api` object, as follows.

```
api.limit
api.remaining
api.reset_time
```

Error Handlers

Error handler is a callable that accepts the `api` and `response` objects and returns the response object. They are most useful when additional logic needs to be implemented based on request result.

Example:

```
def error_handler(api, response):
    # Do something with the response object
    return response
```

sevenbridges-python library comes bundled with several useful error handlers. The most used ones are `maintenance_sleeper` and `rate_limit_sleeper` which pause your code execution until the Seven-Bridges/CGC public API is in maintenance mode or when the rate limit is breached.

Usage:

```
from sevenbridges.http.error_handlers import rate_limit_sleeper, maintenance_sleeper
api = sb.Api(url='https://api.sbgenomics.com/v2', token='<TOKEN_HERE>',
        error_handlers=[rate_limit_sleeper, maintenance_sleeper])
```

Note: Api object instantiated in this way with error handlers attached will be resilient to server maintenance and rate limiting.

Managing users

Currently any authenticated user can access his or her information by invoking the following method:

```
me = api.users.me()
```

Once you have initialized the library by authenticating yourself, the object `me` will contain your user information. This includes:

```
me.href
me.username
me.email
me.first_name
me.last_name
me.affiliation
me.phone
me.address
me.city
me.state
me.zip_code
me.country
```

For example, to obtain your email address invoke:

```
me.email
```

Managing projects

There are several methods on the `Api` object that can help you manage your projects.

Note: If you are not familiar with the project structure of the Seven Bridges Platform and CGC, take a look at their respective documentation: projects on the CGC and projects on the Seven Bridges Platform.

List Projects - introduction to pagination and iteration

In order to list your projects, invoke the `api.projects.query` method. This method follows server pagination and therefore allows pagination parameters to be passed to it. Passing a pagination parameter controls which resources you are shown. The `offset` parameter controls the start of the pagination while the `limit` parameter controls the number of items to be retrieved.

Note: See the Seven Bridges API overview or the CGC API overview for details of how to refer to a project, and for examples of the pagination parameters.

Below is an example of how to get all your projects, using the `query` method and the pagination parameters `offset` of 0 and `limit` of 10.

```
project_list = api.projects.query(offset=0, limit=10)
```

`project_list` has now been defined to be an object of the type **collection** which acts just like a regular python list, and so supports indexing, slicing, iterating and other list functions. All collections in the sevenbridges-python library have two methods: `next_page` and `previous_page` which allow you to load the next or previous pagination pages.

There are several things you can do with a **collection** of any kind of object:

1. The generic query, e.g. `api.projects.query()`, accepts the pagination parameters `offset` and `limit` as introduced above.
2. If you wish to iterate on a complete **collection** use the `all()` method, which returns an iterator
3. If you want to manually iterate on the **collection** (page by page), use `next_page()` and `previous_page()` methods on the collection.
4. You can easily cast the **collection** to the list, so you can re-use it later by issuing the standard Python `project_list = list(api.projects.query().all())`.

```
# Get details of my first 10 projects.
project_list = api.projects.query(limit=10)
```

```
# Iterate through all my projects and print their name and id
for project in api.projects.query().all():
    print (project.id,project.name)
```

```
# Get all my current projects and store them in a list
my_projects = list(api.projects.query().all())
```

Get details of a single project

You can get details of a single project by issuing the `api.projects.get()` method with the parameter `id` set to the id of the project in question. Note that this call, as well as other calls to the API server may raise an exception which you can catch and process if required.

Note - To process errors from the library, import `SbgError` from `sevenbridges.errors`, as shown below.

```
from sevenbridges.errors import SbgError
try:
    project_id = 'doesnotexist/forsure'
    project = api.projects.get(id=project_id)
except SbgError as e:
    print (e.message)
```

Errors in `SbgError` have the properties `code` and `message` which refer to the number and text of 4-digit API status codes that are specific to the Seven Bridges Platform and API. To see all the available codes, see the documentation:

- http://docs.sevenbridges.com/docs/api-status-codes for the Seven Bridges Platform
- http://docs.cancergenomicscloud.org/docs/api-status-codes for the CGC.

Project properties

Once you have obtained the `id` of a Project instance, you can see its properties. All projects have the following properties:

`href` - Project href on the API

`id` - Id of the project

`name` - name of the project

`description` - description of the project

`billing_group` - billing group attached to the project

`type` - type of the project (v1 or v2)

`tags` - list of project tags

The property href `href` is a URL on the server that uniquely identifies the resource in question. All resources have this attribute. Each project also has a name, identifier, description indicating its use, a type, some tags and also a billing_group identifier representing the billing group that is attached to the project.

Project methods – an introduction to methods in the sevenbridges-python library

There are two types of methods in the sevenbridges-python library: static and dynamic. Static methods are invoked on the `Api` object instance. Dynamic methods are invoked from the instance of the object representing the resource (e.g. the project).

Static methods include:

1. Create a new resource: for example, `api.projects.create(name="My new project", billing_group='296a98a9-424c-43f3-aec5-306e0e41c799')` creates a new resource. The parameters used will depend on the resource in question.
2. Get a resource: the method `api.projects.get(id='user/project')` returns details of a specific resource, denoted by its id.
3. Query resources - the method `api.projects.query()` method returns a pageable list of type `collection` of projects. The same goes for other resources, so `api.tasks.query(status='COMPLETED')` returns a **collection** of completed tasks with default paging.

Dynamic methods can be generic (for all resources) or specific to a single resource. They are called on a concrete object, such as a `Project` object.

So, suppose that `project` is an instance of `Project` object. Then, we can:

1. Delete the resource: `project.delete()` deletes the object (if deletion of this resource is supported on the API).
2. Reload the resource from server: `project.reload()` reloads the state of the object from the server.
3. Save changes to the server: `project.save()` saves all properties

The following example shows some of the methods used to manipulatate projects.

```
# Get a collection of projects
projects = api.projects.query()

# Grab the first billing group
bg = api.billing_groups.query(limit=1)[0]

# Create a project using the billing group grabbed above
new_project = api.projects.create(name="My new project", billing_group=bg.id)

# Add a new member to the project
new_project.add_member(user='newuser', permissions= {'write':True, 'execute':True})
```

Other project methods include:

1. Get members of the project and their permissions - `project.get_members()` - returns a `Collection` of members and their permissions
2. Add a member to the project - `project.add_member()`
3. Remove a member from the project - `project.remove_member()`
4. List files from the project - `project.get_files()`
5. Add files to the project - `project.add_files()` - you can add a single `File` or a `Collection` of files
6. List apps from the project - `project.get_apps()`
7. List tasks from the project - `project.get_tasks()`

Manage billing

There are several methods on the `Api` object to can help you manage your billing information. The billing resources that you can interact with are *billing groups* and *invoices*.

Manage billing groups

Querying billing groups will return a standard **collection** object.

```
# Query billing groups
bgroup_list = api.billing_groups.query(offset=0, limit=10)
```

```
# Fetch a billing group's information
bg = api.billing_groups.get(id='f1969c90-da54-4118-8e96-c3f0b49a163d')
```

Billing group properties

The following properties are attached to each billing group:

`href` - Billing group href on the API server.

`id` - Billing group identifier.

`owner` - Username of the user that owns the billing group.

`name` - Billing group name.

`type` - Billing group type (free or regular)

`pending` - True if billing group is not yet approved, False if the billing group has been approved.

`disabled` - True if billing group is disabled, False if its enabled.

`balance` - Billing group balance.

Billing group methods

There is one billing group method:

`breakdown()` fetches a cost breakdown by project and analysis for the selected billing group.

Manage invoices

Querying invoices will return an Invoices **collection** object.

```
invoices = api.invoices.query()
```

Once you have obtained the invoice identifier you can also fetch specific invoice information.

```
invoices = api.invoices.get(id='6351830069')
```

Invoice properties

The following properties are attached to each invoice.

`href` - Invoice href on the API server.

`id` - Invoice identifier.

`pending` - Set to `True` if invoice has not yet been approved by Seven Bridges, `False` otherwise.

`analysis_costs` - Costs of your analysis.

`storage_costs` - Storage costs.

`total` - Total costs.

`invoice_period` - Invoicing period (from-to)

Managing files

Files are an integral part of each analysis. As for as all other resources, the sevenbridges-python library enables you to efectively query files, in order to retreive each file's details and metadata. The request to get a file's information can be made in the same manner as for projects and billing, presented above.

The available methods for fetching specific files are `query` and `get`:

```
# Query all files in a project
file_list = api.files.query(project='user/my-project')
```

```
# Get a single file's information
file = api.files.get(id='5710141760b2b14e3cc146af')
```

File properties

Each file has the following properties:

`href` - File href on the API server.

`id` - File identifier.

`name` - File name.

`size` - File size in bytes.

`project` - Identifier of the project that file is located in.

`created_on` - Date of the file creation.

`modified_on` - Last modification of the file.

`origin` - File origin information, indicating the task that created the file.

`tags` - File tags.

`metadata` - File metadata.

File methods

Files have the following methods:

- Refresh the file with data from the server: `reload()`
- Copy the file from one project to another: `copy()`
- Download the file: `download()`
- Save modifications to the file to the server `save()`
- Delete the resource: `delete()`

See the examples below for information on the arguments these methods take:

Examples

```
# Filter files by name to find only file names containing the specified string:
files = api.files.query(project='user/my-project')
my_file = [file for file in files if 'fasta' in file.name]

# Or simply query files by name if you know their exact file name(s)
files = api.files.query(project='user/myproject', names=['SRR062634.filt.fastq.gz','SRR062635.filt.fa
my_files = api.files.query(project='user/myproject', metadata = {'sample_id': 'SRR062634'} )


# Edit a file's metadata
my_file = my_files[0]
my_file.metadata['sample_id'] = 'my-sample'
my_file.metadata['library'] = 'my-library'


# Add metadata (if you are starting with a file without metadata)
my_file = my_files[0]
my_file.metadata = {'sample_id' : 'my-sample',
                    'library' : 'my-library'
                   }

# Also set a tag on that file
my_file.tags = ['example']

# Save modifications
my_file.save()

# Copy a file between projects
new_file = my_file.copy(project='user/my-other-project', name='my-new-file')

# Download a file to the current working directory
# Optionally, path can contain a full path on local filesystem
new_file.download(path='my_new_file_on_disk')
```

Managing file upload and download

`sevenbridges-python` library provides both synchronous and asynchronous way of uploading or downloading files.

File Download

Synchronous file download:

```
file = api.files.get('file-identifier')
file.download('/home/bar/foo/file.bam')
```

Asynchronous file download:

```
file = api.files.get('file-identifier')
download = file.download('/home/bar/foo.bam', wait=False)

download.path # Gets the target file path of the download.
download.status # Gets the status of the download.
download.progress # Gets the progress of the download as percentage.
download.start_time # Gets the start time of the download.
download.duration # Gets the download elapsed time.

download.start() # Starts the download.
download.pause() # Pauses the download.
download.resume() # Resumes the download.
download.stop() # Stops the download.
download.wait() # Block the main loop until download completes.
```

You can register the callback or error callback function to the download handle: `download.add_callback(callback=my_callback, errorback=my_error_back)`

Registered callback method will be invoked on completion of the download. The errorback method will be invoked if error happens during download.

File Upload

Synchronous file upload:

```
# Get the project where we want to upload files.
project = api.projects.get('project-identifier')
api.files.upload('/home/bar/foo/file.fastq', project)
# Optionally we can set file name of the uploaded file.
api.files.upload('/home/bar/foo/file.fastq', project, file_name='new.fastq')
```

Asynchronous file upload:

```
upload = api.files.upload('/home/bar/foo/file.fastq', 'project-identifier', wait=False)

upload.file_name # Gets the file name of the upload.
upload.status # Gets the status of the upload.
upload.progress # Gets the progress of the upload as percentage.
upload.start_time # Gets the start time of the upload.
upload.duration # Gets the upload elapsed time.

upload.start() # Starts the upload.
upload.pause() # Pauses the upload.
```

```
upload.resume() # Resumes the upload.
upload.stop() # Stops the upload.
upload.wait() # Block the main loop until upload completes.
```

You can register the callback or error callback in the same manner as it was described for asynchronous file download.

Managing volumes: connecting cloud storage to the Platform

Volumes authorize the Platform to access and query objects on a specified cloud storage (Amazon Web Services or Google Cloud Storage) on your behalf. As for as all other resources, the sevenbridges-python library enables you to effectively query volumes, import files from a volume to a project or export files from a project to the volume.

The available methods for listing volumes, imports and exports are `query` and `get`, as for other objects:

```
# Query all volumes
volume_list = api.volumes.query()
# Query all imports
all_imports = api.imports.query()
# Query failed exports
failed_exports = api.exports.query(state='FAILED')
```

```
# Get a single volume's information
volume = api.volumes.get(id='user/volume')
# Get a single import's information
i = api.imports.get(id='08M4ywDZkQuJOb3L5M8mMSvzoeGezTdh')
# Get a single export's information
e = api.exports.get(id='0C7T8sBDP6aiNbwvXv12QZFPW55wJ3GJ')
```

Volume properties

Each volume has the following properties:

`href` - Volume href on the API server.

`id` - Volume identifier in format owner/name.

`name` - Volume name. Learn more about this in our Knowledge Center.

`access_mode` - Whether the volume was created as read-only (RO) or read-write (RW). Learn more about this in our Knowledge Center.

`active` - Whether or not this volume is active.

`created_on` - Time when the volume was created.

`modified_on` - Time when the volume was last modified.

`description` - An optional description of this volume.

`service` - This object contains the information about the cloud service that this volume represents.

Volume methods

Volumes have the following methods:

- Refresh the volume with data from the server: `reload()`
- Get imports for a particular volume `get_imports()`

- Get exports for a particular volume `get_exports()`
- Create a new volume based on the AWS S3 service - `create_s3_volume()`
- Create a new volume based on Google Cloud Storage service - `create_google_volume()`
- Save modifications to the volume to the server `save()`
- Unlink the volume `delete()`

See the examples below for information on the arguments these methods take:

Examples

```
# Create a new volume based on AWS S3 for importing files
volume_import = api.volumes.create_s3_volume(name='my_input_volume', bucket='my_bucket',access_key_i

# Create a new volume based on AWS S3 for exporting files
volume_export = api.volumes.create_s3_volume(name='my_output_volume', bucket='my_bucket', access_key_
# List all volumes available
volumes = api.volumes.query()
```

Import properties

When you import a file from a volume into a project on the Platform, you are importing a file from your cloud storage provider (Amazon Web Services or Google Cloud Storage) via the volume onto the Platform.

If successful, an alias will be created on the Platform. Aliases appear as files on the Platform and can be copied, executed, and modified as such. They refer back to the respective file on the given volume.

Each import has the following properties:

`href` - Import href on the API server.

`id` - Import identifier.

`source` - Source of the import, object of type `VolumeFile`, contains info on volume and file location on the volume

`destination` - Destination of the import, object of type `ImportDestination`, containing info on project where the file was imported to and name of the file in the project

`state` - State of the import. Can be *PENDING*, *RUNNING*, *COMPLETED* and *FAILED*.

`result` - If the import was completed, contains the result of the import - a `File` object.

`error` - Contains the `Error` object if the import failed.

`overwrite` - Whether the import was set to overwrite file at destination or not.

`started_on` - Contains the date and time when the import was started.

`finished_on` - Contains the date and time when the import was finished.

Import methods

Imports have the following methods:

- Refresh the import with data from the server: `reload()`
- Start an import by specifying the source and the destination of the import - `submit_import()`
- Delete the import - `delete()`

See the examples below for information on the arguments these methods take:

Examples

```
# Import a  file to a project
my_project = api.projects.get(id='my_project')
bucket_location = 'fastq/my_file.fastq'
imp = api.imports.submit_import(volume=volume_import, project=my_project, location=bucket_location)
# Wait until the import finishes
while True:
      import_status = imp.reload().state
      if import_status in (ImportExportState.COMPLETED, ImportExportState.FAILED):
           break
      time.sleep(10)
# Continue with the import
if imp.state == ImportExportState.COMPLETED:
      imported_file = imp.result
```

Export properties

When you export a file from a project on the Platform into a volume, you are essentially writing to your cloud storage bucket on Amazon Web Services or Google Cloud Storage via the volume.

Note that the file selected for export must not be a public file or an alias. Aliases are objects stored in your cloud storage bucket which have been made available on the Platform.

The volume you are exporting to must be configured for read-write access. To do this, set the `access_mode` parameter to `RW` when creating or modifying a volume. Learn more about this from our Knowledge Center.

Each export has the following properties:

`href` - Export href on the API server.

`id` - Export identifier.

`source` - Source of the export, object of type `File`

`destination` - Destination of the export, object of type `VolumeFile`, containing info on project where the file was imported to and name of the file in the project

`state` - State of the export. Can be *PENDING*, *RUNNING*, *COMPLETED* and *FAILED*.

`result` - If the export was completed, this contains the result of the import - a `File` object.

`error` - Contains the `Error` object if the export failed.

`overwrite` - Whether or not the export was set to overwrite the file at the destination.

`started_on` - Contains the date and time when the export was started.

`finished_on` - Contains the date and time when the export was finished.

Export methods

Exports have the following methods:

- Refresh the export with data from the server: `reload()`
- Submit export, by specifying source and destination of the import: `submit_import()`

- Delete the export: `delete()`

See the examples below for information on the arguments these methods take:

Examples

```
# Export a set of files to a volume
# Get files from a project
files_to_export = api.files.query(project=my_project).all()
# And export all the files to the output bucket
exports = []
for f in files_to_export:
      export = api.exports.submit_export(file=f, volume = volume_export, location=f.name)
      exports.append(export)
# Wait for exports to finish:
num_exports = len(exports)
done = False

while not done:
      done_len = 0
      for e in exports:
            if e.reload().state in (ImportExportState.COMPLETED, ImportExportState.FAILED):
                  done_len += 1
            time.sleep(10)
      if done_len == num_exports:
            done = True
```

Managing apps

Managing apps (tools and workfows) with the sevenbridges-python library is simple. Apps on the Seven Bridges Platform and CGC are implemented using the Common Workflow Language (CWL) specification https://github.com/common-workflow-language/common-workflow-language. The sevenbridges-python currenty supports only Draft 2 format of the CWL. Each app has a CWL description, expressed in JSON.

Querying all apps or getting the details of a single app can be done in the same way as for other resources, using the `query()` and `get` methods. You can also invoke the following class-specific methods:

- `get_revision()` - Returns a specific app revision.
- `install_app()` - Installs your app on the server, using its CWL desription.
- `create_revision()` - Creates a new revision of the specified app.

App properties

Each app has the following available properties:

`href` - The URL of the app on the API server.

`id` - App identifier.

`name` - App name.

`project` - Identifier of the project that app is located in.

`revision` - App revision.

`raw` - Raw CWL description of the app.

App methods

- App only has class methods that were mentioned above.

Managing tasks

Tasks (pipeline executions) are easy to handle using the sevenbridges-python library. As with all resources you can `query()` your tasks, and `get()` a single task instance. You can also do much more. We will outline task properties and methods and show in the examples how easy is to run your first analysis using Python.

Task properties

`href` - Task URL on the API server.

`id` - Task identifier.

`name` - Task name.

`status` - Task status.

`project` - Identifier of the project that the task is located in.

`app` - The identifier of the app that was used for the task.

`type` - Task type.

`created_by` - Username of the task creator.

`executed_by`- Username of the task executor.

`batch` - Boolean flag: `True` for batch tasks, `False` for regular & child tasks.

`batch_by` - Batching criteria.

`batch_group` - Batch group assigned to the child task calculated from the `batch_by` criteria.

`batch_input` - Input identifier on to which to apply batching.

`parent` - Parent task for a batch child.

`end_time` - Task end time.

`execution_status` - Task execution status.

`price` - Task cost.

`inputs` - Inputs that were subbmited to the task.

`outputs` - Generated outputs from the task.

Note: Check the documentation on the Seven Bridges API and the CGC API for more details on batching criteria.

Task methods

The following class and instance methods are available for tasks:

- Create a task on the server and, optionally, run it: `create()`.
- Query tasks: `query()`.
- Get single task's information: `get()`.

- Abort a running task: `abort()`.
- Run a draft task: `run()`
- Delete a draft task from the server: `delete()`.
- Refresh the task object information with the date from the server: `refresh()`.
- Save task modifications to the sever: `save()`.
- Get task exection datails: `get_execution_details()`.
- Get batch children if the task is a batch task: `get_batch_children()`.

Task creation hints

- Both input files and parameters are passed the same way together in a single dictionary to `inputs`.
- `api.files.query` always return an array of files. For single file inputs, use `api.files.query(project='my-project', names=["one_file.fa"])[0]`.

Task Examples

Single task

```
# Task name
task_name = 'my-first-task'

# Project in which I want to run a task.
project_id = 'my-username/my-project'

# App I want to use to run a task
app = 'my-username/my-project/my-app'

# Inputs
inputs = {}
inputs['FastQC-Reads'] = api.files.query(project='my-project', metadata={'sample': 'some-sample'})

try:
    task = api.tasks.create(name=name, project=project, app=app, inputs=inputs, run=True)
except SbError:
    print('I was unable to run the task.')

# Task can also be ran by invoking .run() method on the draft task.
task.run()
```

Batch task

```
# Task name
task_name = 'my-first-task'

# Project in which to run the task.
project_id = 'my-username/my-project'

# App to use to run the task
app = 'my-username/my-project/my-app'
```

```
# Inputs
inputs = {}
inputs['FastQC-Reads'] = api.files.query(project='my-project', metadata={'sample': 'some-sample'})

# Specify that one task should be created per file (i.e. batch tasks by file).
batch_by = {'type': 'item'}


# Specify that the batch input is FastQC-Reads
batch_input = 'FastQC-Reads'

try:
    task = api.tasks.create(name=name, project=project, app=app,
                            inputs=inputs, batch_input=batch_input, batch_by=batch_by, run=True)
except SbError:
    print('I was unable to run a batch task.')
```

sevenbridges package

sevenbridges-python

copyright 2016 Seven Bridges Genomics Inc.

license Apache 2.0

Subpackages

sevenbridges.http package

Submodules

sevenbridges.http.client module

class sevenbridges.http.client.**HttpClient**(*url=None*, *token=None*, *oauth_token=None*, *config=None*, *timeout=None*, *proxies=None*, *error_handlers=None*)

Bases: object

Implementation of all low-level API stuff, creating and sending requests, returning raw responses, authorization, etc.

add_error_handler(*handler*)

delete(*url*, *headers=None*, *params=None*, *append_base=True*)

get(*url*, *headers=None*, *params=None*, *data=None*, *append_base=True*, *stream=False*)

limit

patch(*url*, *headers=None*, *params=None*, *data=None*, *append_base=True*)

post(*url*, *headers=None*, *params=None*, *data=None*, *append_base=True*)

put(*url*, *headers=None*, *params=None*, *data=None*, *append_base=True*)

remaining

remove_error_handler(*handler*)

request_id

reset_time

session

sevenbridges.http.client.**config_vars**(*profiles*)
Utility method to fetch config vars using ini section profile :param profiles: profile name. :return:

sevenbridges.http.client.**generate_session**(*proxies=None*)
Utility method to generate request sessions. :param proxies: Proxies dictionary. :return: requests.Session object.

sevenbridges.http.error_handlers module

sevenbridges.http.error_handlers.**general_error_sleeper**(*api*, *response*)
Pauses the execution if response status code is > 500. :param api: Api instance. :param response: requests.Response object

sevenbridges.http.error_handlers.**maintenance_sleeper**(*api*, *response*)
Pauses the execution if sevenbridges api is under maintenance. :param api: Api instance. :param response: requests.Response object.

sevenbridges.http.error_handlers.**rate_limit_sleeper**(*api*, *response*)
Pauses the execution if rate limit is breached. :param api: Api instance. :param response: requests.Response object

sevenbridges.meta package

Submodules

sevenbridges.meta.collection module

class sevenbridges.meta.collection.**Collection**(*resource*, *href*, *total*, *items*, *links*, *api*)
Bases: list

Wrapper for SevenBridges pageable resources. Among the actual collection items it contains information regarding the total number of entries available in on the server and resource href.

all()
Fetches all available items. :return: Collection object.

next_page()
Fetches next result set. :return: Collection object.

previous_page()
Fetches previous result set. :return: Collection object.

resource = None

total

sevenbridges.meta.comp_mutable_dict module

class sevenbridges.meta.comp_mutable_dict.**CompoundMutableDict**(***kwargs*)
Bases: dict

Resource used for mutable compound dictionaries.

equals(*other*)

items()

update(*E=None*, ***F*)

sevenbridges.meta.data module

class sevenbridges.meta.data.**DataContainer**(*urls*, *api*)
Bases: object

Utility for fetching data from the API server using, resource identifier or href.

fetch()

sevenbridges.meta.fields module

class sevenbridges.meta.fields.**BasicListField**(*name=None*, *read_only=False*, *max_length=None*)
Bases: *sevenbridges.meta.fields.Field*

validate(*value*)

class sevenbridges.meta.fields.**BooleanField**(*name=None*, *read_only=False*)
Bases: *sevenbridges.meta.fields.Field*

validate(*value*)

class sevenbridges.meta.fields.**CompoundField**(*cls*, *name=None*, *read_only=False*, *validator=None*)
Bases: *sevenbridges.meta.fields.Field*

class sevenbridges.meta.fields.**CompoundListField**(*cls*, *name=None*, *read_only=True*)
Bases: *sevenbridges.meta.fields.Field*

class sevenbridges.meta.fields.**DateTimeField**(*name=None*, *read_only=True*)
Bases: *sevenbridges.meta.fields.Field*

class sevenbridges.meta.fields.**DictField**(*name=None*, *read_only=False*)
Bases: *sevenbridges.meta.fields.Field*, dict

class sevenbridges.meta.fields.**Field**(*name=None*, *read_only=True*, *validator=None*)
Bases: object

validate(*value*)

class sevenbridges.meta.fields.**FloatField**(*name=None*, *read_only=False*)
Bases: *sevenbridges.meta.fields.Field*

validate(*value*)

class sevenbridges.meta.fields.**HrefField**(*name=None*)
Bases: *sevenbridges.meta.fields.Field*

class sevenbridges.meta.fields.**IntegerField**(*name=None*, *read_only=False*)
Bases: *sevenbridges.meta.fields.Field*

validate(*value*)

class sevenbridges.meta.fields.**ObjectIdField**(*name=None*, *read_only=True*)
Bases: *sevenbridges.meta.fields.Field*

class sevenbridges.meta.fields.**StringField**(*name=None*, *read_only=False*, *max_length=None*)
Bases: *sevenbridges.meta.fields.Field*

validate(*value*)

class sevenbridges.meta.fields.**UuidField**(*name=None*, *read_only=True*)
Bases: *sevenbridges.meta.fields.Field*

validate(*value*)

sevenbridges.meta.resource module

sevenbridges.meta.resource.**Resource**
Resource is base class for all resources, hiding implementation details of magic of injecting instance of API and common operations (like generic query).

class sevenbridges.meta.resource.**ResourceMeta**
Bases: type

Metaclass for all resources, knows how to inject instance of API from class that contains classes with this meta. Class that contains this class has to have 'api' property which will be injected into class level API property of Resource class.

Creates constructors for all resources and manages instantiation of resource fields.

sevenbridges.meta.transformer module

class sevenbridges.meta.transformer.**Transform**
Bases: object

static to_app(*app*)

static to_billing_group(*billing_group*)

static to_datestring(*d*)

static to_file(*file_*)

static to_project(*project*)

static to_task(*task*)

static to_user(*user*)

static to_volume(*volume*)

sevenbridges.models package

Subpackages

sevenbridges.models.compound package

Subpackages

sevenbridges.models.compound.billing package

Submodules

sevenbridges.models.compound.billing.invoice_period module

sevenbridges.models.compound.billing.invoice_period.**InvoicePeriod**

Invoice period resource contains datetime information about the invoice. It has from and to fields which represent the interval period for this invoice.

sevenbridges.models.compound.billing.project_breakdown module

sevenbridges.models.compound.billing.project_breakdown.**ProjectBreakdown**

Project breakdown resource contains information regarding billing group project breakdown costs.

sevenbridges.models.compound.billing.task_breakdown module

sevenbridges.models.compound.billing.task_breakdown.**TaskBreakdown**

Task breakdown resource contains information regarding billing group analysis breakdown costs.

sevenbridges.models.compound.files package

Submodules

sevenbridges.models.compound.files.download_info module

sevenbridges.models.compound.files.download_info.**DownloadInfo**

Download info resource contains download url for the file.

sevenbridges.models.compound.files.file_origin module

sevenbridges.models.compound.files.file_origin.**FileOrigin**

File origin resource contains information about origin of a file. Among others it contains information about the task if the file was produced during executions of a analysis.

sevenbridges.models.compound.files.file_storage module

sevenbridges.models.compound.files.file_storage.**FileStorage**

File storage resource contains information about the storage location of the file if the file is imported on or exported to an external volume.

sevenbridges.models.compound.files.metadata module

sevenbridges.models.compound.files.metadata.**Metadata**

File metadata resource.

sevenbridges.models.compound.jobs package

Submodules

sevenbridges.models.compound.jobs.job module

sevenbridges.models.compound.jobs.job.**Job**

Job resource contains information for a single executed node in the analysis.

sevenbridges.models.compound.jobs.job_docker module

sevenbridges.models.compound.jobs.job_docker.**JobDocker**

JobDocker resource contains information for a docker image that was used for execution of a single job.

sevenbridges.models.compound.jobs.job_instance module

sevenbridges.models.compound.jobs.job_instance.**Instance**

Instance resource contains information regarding the instance on which the job was executed.

sevenbridges.models.compound.jobs.job_log module

sevenbridges.models.compound.jobs.job_log.**Logs**

Task output resource.

sevenbridges.models.compound.projects package

Submodules

sevenbridges.models.compound.projects.permissions module

sevenbridges.models.compound.projects.permissions.**Permissions**

Members permissions resource.

sevenbridges.models.compound.projects.settings module

sevenbridges.models.compound.projects.settings.**Settings**

Project settings resource.

sevenbridges.models.compound.tasks package

Submodules

sevenbridges.models.compound.tasks.batch_by module

sevenbridges.models.compound.tasks.batch_by.**BatchBy**

Task batch by resource.

sevenbridges.models.compound.tasks.batch_group module

sevenbridges.models.compound.tasks.batch_group.**BatchGroup**

Batch group for a batch task. Represents the group that is assigned to the child task from the batching criteria that was used when the task was started.

sevenbridges.models.compound.tasks.execution_status module

sevenbridges.models.compound.tasks.execution_status.**ExecutionStatus**

Task execution status resource.

Contains information about the number of completed task steps, total number of task steps, current execution message and information regarding computation limits.

In case of a batch task it also contains the number of queued, running, completed, failed and aborted tasks.

sevenbridges.models.compound.tasks.input module

sevenbridges.models.compound.tasks.input.**Input**

Task input resource.

sevenbridges.models.compound.tasks.output module

sevenbridges.models.compound.tasks.output.**Output**

Task output resource.

sevenbridges.models.compound.volumes package

Submodules

sevenbridges.models.compound.volumes.credentials module

sevenbridges.models.compound.volumes.credentials.**VolumeCredentials**

Volume permissions resource.

sevenbridges.models.compound.volumes.import_destination module

sevenbridges.models.compound.volumes.import_destination.**ImportDestination**

ImportDestination resource describes the location of the file imported on to SevenBridges platform or related product.

sevenbridges.models.compound.volumes.properties module

sevenbridges.models.compound.volumes.properties.**VolumeProperties**

Volume permissions resource.

sevenbridges.models.compound.volumes.service module

sevenbridges.models.compound.volumes.service.**VolumeService**

Volume Service resource. Contains information about external storage provider.

sevenbridges.models.compound.volumes.volume_file module

sevenbridges.models.compound.volumes.volume_file.**VolumeFile**

VolumeFile resource describes the location of the file on the external volume.

Submodules

sevenbridges.models.compound.error module

sevenbridges.models.compound.error.**Error**

Error resource describes the error that happened and provides http status, custom codes and messages as well as the link to online resources.

sevenbridges.models.compound.price module

sevenbridges.models.compound.price.**Price**

Price resource contains an information regarding the currency and the monet value of a certain resource.

Submodules

sevenbridges.models.app module

sevenbridges.models.app.**App**
Central resource for managing apps.

sevenbridges.models.billing_breakdown module

sevenbridges.models.billing_breakdown.**BillingGroupBreakdown**
Central resource for managing billing group breakdowns.

sevenbridges.models.billing_group module

sevenbridges.models.billing_group.**BillingGroup**
Central resource for managing billing groups.

sevenbridges.models.endpoints module

sevenbridges.models.endpoints.**Endpoints**
Central resource for managing Endpoints.

sevenbridges.models.enums module

class sevenbridges.models.enums.**FileStorageType**
Bases: object

PLATFORM = 'PLATFORM'

VOLUME = 'VOLUME'

class sevenbridges.models.enums.**ImportExportState**
Bases: object

COMPLETED = 'COMPLETED'

FAILED = 'FAILED'

PENDING = 'PENDING'

RUNNING = 'RUNNING'

class sevenbridges.models.enums.**PartSize**
Bases: object

DOWNLOAD_MINIMUM_PART_SIZE = 5242880

GB = 1073741824

KB = 1024

MAXIMUM_OBJECT_SIZE = 5497558138880

MAXIMUM_TOTAL_PARTS = 10000

MAXIMUM_UPLOAD_SIZE = 5368709120

MB = 1048576

TB = 1099511627776

UPLOAD_MINIMUM_PART_SIZE = 5242880

class sevenbridges.models.enums.**TaskStatus**
Bases: object

ABORTED = 'ABORTED'

COMPLETED = 'COMPLETED'

CREATING = 'CREATING'

DRAFT = 'DRAFT'

FAILED = 'FAILED'

QUEUED = 'QUEUED'

RUNNING = 'RUNNING'

class sevenbridges.models.enums.**TransferState**
Bases: object

ABORTED = 'ABORTED'

COMPLETED = 'COMPLETED'

FAILED = 'FAILED'

PAUSED = 'PAUSED'

PREPARING = 'PREPARING'

RUNNING = 'RUNNING'

STOPPED = 'STOPPED'

class sevenbridges.models.enums.**VolumeAccessMode**
Bases: object

READ_ONLY = 'RO'

READ_WRITE = 'RW'

class sevenbridges.models.enums.**VolumeType**
Bases: object

GOOGLE = 'GCS'

S3 = 'S3'

sevenbridges.models.execution_details module

sevenbridges.models.execution_details.**ExecutionDetails**
Task execution details.

sevenbridges.models.file module

sevenbridges.models.file.**File**
Central resource for managing files.

sevenbridges.models.invoice module

sevenbridges.models.invoice.**Invoice**
Central resource for managing invoices.

sevenbridges.models.link module

sevenbridges.models.link.**Link**
Pagination links.

sevenbridges.models.member module

sevenbridges.models.member.**Member**
Central resource for managing project members.

sevenbridges.models.project module

sevenbridges.models.project.**Project**
Central resource for managing projects.

sevenbridges.models.storage_export module

sevenbridges.models.storage_export.**Export**
Central resource for managing exports.

sevenbridges.models.storage_import module

sevenbridges.models.storage_import.**Import**
Central resource for managing imports.

sevenbridges.models.task module

sevenbridges.models.task.**Task**
Central resource for managing tasks.

sevenbridges.models.user module

sevenbridges.models.user.**User**
Central resource for managing tasks.

sevenbridges.models.volume module

sevenbridges.models.volume.**Volume**
Central resource for managing volumes.

sevenbridges.transfer package

Submodules

sevenbridges.transfer.download module

class sevenbridges.transfer.download.**DPartedFile**(*file_path*, *session*, *url*, *file_size*, *part_size*, *retry*, *timeout*, *pool*)

Bases: object

done()

get_parts()
Partitions the file and saves the part information in memory.

submit()
Partitions the file into chunks and submits them into group of 4 for download on the api download pool.

class sevenbridges.transfer.download.**Download**(*url*, *file_path*, *part_size=5242880*, *retry_count=5*, *timeout=60*, *api=None*)

Bases: threading.Thread

add_callback(*callback=None*, *errorback=None*)
Adds a callback that will be called when the download finishes successfully or when error is raised.

add_progress_callback(*callback=None*)
Adds a progress callback that will be called each time a get_parts is successfully downloaded. The first argument of the progress callback will be a progress object described in sevenbridges.transfer.utils

Parameters **callback** – Callback function

duration

path

pause()
Pauses the download. :raises SbgError: If upload is not in RUNNING state.

progress

resume()
Resumes the download. :raises SbgError: If download is not in RUNNING state.

run()
Runs the thread! Should not be used use start() method instead.

start()
Starts the download. :raises SbgError: If download is not in PREPARING state.

start_time

status

stop()
Stops the download. :raises SbgError: If download is not in PAUSED or RUNNING state.

wait()
Blocks until download is completed.

sevenbridges.transfer.upload module

class sevenbridges.transfer.upload.**UPartedFile**(*fp*, *file_size*, *part_size*, *upload*, *retry_count*, *timeout*, *storage_session*, *api*)

Bases: object

done()

get_parts()
Partitions the file and saves the parts to be uploaded in memory.

submit()
Partitions the file into chunks and submits them into group of 4 for upload on the api upload pool. :return: Futures

class sevenbridges.transfer.upload.**Upload**(*file_path*, *project*, *file_name=None*, *overwrite=False*, *part_size=5242880*, *retry_count=5*, *timeout=60*, *api=None*)

Bases: threading.Thread

add_callback(*callback=None*, *errorback=None*)
Adds a callback that will be called when the upload finishes successfully or when error is raised.

add_progress_callback(*callback=None*)
Adds a progress callback that will be called each time a get_parts is successfully uploaded. The first argument of the progress callback will be a progress object described in sevenbridges.transfer.utils

Parameters **callback** – Callback function

duration

file_name

pause()
Pauses the upload. :raises SbgError: If upload is not in RUNNING state.

progress

result()

resume()
Resumes the upload that was paused. :raises SbgError: If upload is not in PAUSED state.

run()
Runs the thread! Should not be used use start() method instead.

start()
Starts the upload. :raises SbgError: If upload is not in PREPARING state.

start_time

status

stop()
Stops the upload. :raises SbgError: If upload is not in PAUSED or RUNNING state.

wait()
Blocks until upload is completed.

sevenbridges.transfer.utils module

class sevenbridges.transfer.utils.**Part**(*start=None*, *size=None*)

Bases: object

size

start

class sevenbridges.transfer.utils.**Progress**(*num_of_parts*, *parts_done*, *bytes_done*, *file_size*, *duration*)

Bases: object

bandwidth

bytes_done

duration

file_size

num_of_parts

parts_done

progress

sevenbridges.transfer.utils.**simple_progress_bar**(*progress*)

sevenbridges.transfer.utils.**total_parts**(*file_size*, *part_size*)

Submodules

sevenbridges.api module

class sevenbridges.api.**Api**(*url=None*, *token=None*, *oauth_token=None*, *config=None*, *timeout=None*, *download_max_workers=16*, *upload_max_workers=16*, *proxies=None*, *error_handlers=None*)

Bases: *sevenbridges.http.client.HttpClient*

Api aggregates all resource classes into single place

apps
Central resource for managing apps.

billing_groups
Central resource for managing billing groups.

endpoints
Central resource for managing Endpoints.

exports
Central resource for managing exports.

files
Central resource for managing files.

imports
Central resource for managing imports.

invoices
Central resource for managing invoices.

projects
Central resource for managing projects.

rate_limit
Rate limit resource contains info regarding request and computation rate limits.

tasks
Central resource for managing tasks.

users
Central resource for managing tasks.

volumes
Central resource for managing volumes.

sevenbridges.config module

class sevenbridges.config.**Config**(*profile=None*, *proxies=None*)
Bases: object

Utility configuration class.

class sevenbridges.config.**Profile**(*profile*)
Bases: object

CREDENTIALS = '/home/docs/.sevenbridges/credentials'

PROXIES = '/home/docs/.sevenbridges/sevenbridges-python/config'

api_endpoint

auth_token

proxies

sevenbridges.config.**format_proxies**(*proxies*)
Helper method for request proxy key compatibility. :param proxies: Proxies dictionary :return: Dict compatible with request proxy format.

sevenbridges.decorators module

sevenbridges.decorators.**check_for_error**(*func*)
Executes the wrapped function and inspects the response object for specific errors.

sevenbridges.decorators.**inplace_reload**(*method*)
Executes the wrapped function and reloads the object with data returned from the server.

sevenbridges.decorators.**retry**(*retry_count*)
Retry decorator used during file upload and download.

sevenbridges.decorators.**retry_on_excs**(*excs*, *retry_count=3*, *delay=1*)
Retry decorator used to retry callables on for specific exceptions.

Parameters

- **excs** – Exceptions tuple.
- **retry_count** – Retry count.
- **delay** – Delay in seconds between retries.

Returns Wrapped function object.

sevenbridges.errors module

exception sevenbridges.errors.**BadRequest**(*code=None*, *message=None*, *more_info=None*)
Bases: *sevenbridges.errors.SbgError*

exception sevenbridges.errors.**Conflict**(*code=None*, *message=None*, *more_info=None*)
Bases: *sevenbridges.errors.SbgError*

exception sevenbridges.errors.**ExecutionDetailsInvalidTaskType**(*code=None*, *message=None*, *more_info=None*)
Bases: *sevenbridges.errors.SbgError*

exception sevenbridges.errors.**Forbidden**(*code=None*, *message=None*, *more_info=None*)
Bases: *sevenbridges.errors.SbgError*

exception sevenbridges.errors.**LocalFileAlreadyExists**(*code=None*, *message=None*, *more_info=None*)
Bases: *sevenbridges.errors.SbgError*

exception sevenbridges.errors.**MethodNotAllowed**(*code=None*, *message=None*, *more_info=None*)
Bases: *sevenbridges.errors.SbgError*

exception sevenbridges.errors.**NotFound**(*code=None*, *message=None*, *more_info=None*)
Bases: *sevenbridges.errors.SbgError*

exception sevenbridges.errors.**PaginationError**(*message*)
Bases: *sevenbridges.errors.SbgError*

exception sevenbridges.errors.**ReadOnlyPropertyError**(*message*)
Bases: *sevenbridges.errors.SbgError*

exception sevenbridges.errors.**RequestTimeout**(*code=None*, *message=None*, *more_info=None*)
Bases: *sevenbridges.errors.SbgError*

exception sevenbridges.errors.**ResourceNotModified**
Bases: *sevenbridges.errors.SbgError*

exception sevenbridges.errors.**SbgError**(*message=None*, *code=None*, *status=None*, *more_info=None*)
Bases: Exception

Base class for SBG errors.

Provides a base exception for all errors that are thrown by sevenbridges-python library.

exception sevenbridges.errors.**ServerError**(*code=None*, *message=None*, *more_info=None*)
Bases: *sevenbridges.errors.SbgError*

exception sevenbridges.errors.**ServiceUnavailable**(*code=None*, *message=None*, *more_info=None*)
Bases: *sevenbridges.errors.SbgError*

exception sevenbridges.errors.**TaskValidationError**(*message*, *task=None*)
Bases: *sevenbridges.errors.SbgError*

exception sevenbridges.errors.**TooManyRequests**(*code=None*, *message=None*, *more_info=None*)
Bases: *sevenbridges.errors.SbgError*

exception sevenbridges.errors.**Unauthorized**(*code=None*, *message=None*, *more_info=None*)
Bases: *sevenbridges.errors.SbgError*

exception sevenbridges.errors.**ValidationError**(*message*)
Bases: *sevenbridges.errors.SbgError*

S

A

B

C

D

E

F

G

H

I

J

S

T

U

V

W