PySpeckle

Jun 29, 2019

Contents:

CHAPTER 1

PySpeckle

A Python Speckle Client

Speckle.Works

> Speckle: open digital infrastructure for designing, making and operating the built environment. We reimagine the design process from the Internet up: Speckle is an open source (MIT) initiative for developing an extensible Design & AEC data communication and collaboration platform.

1.1 Installation

PySpeckle can be installed through `pip`: `pip install speckle`

1.2 Disclaimer

This code is WIP and as such should be used with caution, on non-sensitive projects.

1.3 Description

PySpeckle is a light Python wrapper / interface for the Speckle framework. It can be used independently through Python scripts, or as a base for building various plug-ins, such as SpeckleBlender.

At the moment, it copies the same method names from the .NET `SpeckleApiClient`, for consistency's sake. Although the functions are mostly labelled 'Async', they are not yet. This could eventually be implemented with `requests_futures` or `grequests` or similar.

1.4 Quick Start

Here is how you initialise a client, authenticate and start speckling:

```
from speckle import SpeckleApiClient

client = SpeckleApiClient('hestia.speckle.works')

client.login(
    email='test@test.com',
    password='Speckle<3Python'
)

stream_id = 'HjenwS2s'

objects = client.streams.list_objects(stream_id)

for object in objects:
  print(object.dict())
```

Usage documentation can be found here.

1.5 Maintainers

SpeckleBlender is written and maintained by Tom Svilans (Github), Izzy Lyseggen and Antoine Dao.

1.6 Updates

1.6.1 January 15, 2019

Pip distribution updated to version 0.2.5. Remember to `pip install speckle --upgrade` to stay up-to-date.

Installation

PySpeckle can be installed through `pip`:

```
$ pip install speckle
```

The Pypi repository can be found here

QuickStart

Playing with a new library or API client can be daunting. We feel you! This is why we have compiled a bunch of example usage scenarios to walk you through how to use the python speckle client.

In the following set of examples we assume you don't have your own Speckle server up and running so you can test all of the steps below by connecting to the `hestia` speckle server.

First we must instantiate the `SpeckleApiClient`. You can name it whatever you want, we will use `client` for simplicity. As we are connecting to the `hestia` test server we will input the full domain name as an argument: `hestia.speckle.works`

```
from speckle import SpeckleApiClient

client = SpeckleApiClient('hestia.speckle.works')
```

Registering and Logging in

Before you can do anything with Speckle you need to register yourself to the server you are going to be using. If you have already registered then you can skip the first code section below and look at the one right after where we explain how to login.

Note: When you register using a PySpeckle client this client is also authenticated and you can start using it to make API calls immediately without needing to login.

```
client.register(
    email='test@test.com',
    password='Speckle<3Python',
    company='SnakeySnakes',
    name='Snakey',
    surname='Snake'
)
```

If you are already registered then you can simply login your client as follows:

```
client.login(
    email='test@test.com',
    password='Speckle<3Python'
)
```

Sending and Retrieving Things

At the moment the api client takes dictionary payloads as data inputs (a `Speckle Object` for example) and returns data classes in most cases (essentially a class instance). If a data class is not returned then the object return is a dictionary.

Warning: Whether the client returns a data class or a dictionary is not yet very well segmented. To be sure you are getting back what you expect from the python client we reccomend reading through the *API referemce*

Attention: PySpeckle doesn't yet support any of the *websockets* data transfer methods that the SpeckleServer offers. We're working on it. Feel free to contribute if you've got an idea on how to implement it.

Common Methods

Speckle mostly follows a bog standard REST API setup where a user can `POST`, `GET`, `PUT` and `DELETE` different resources. As such the following verbs can be used to make standard REST calls to Speckle.

Warning: For most resources that support a *get*, *update*, *delete* call the *id* value to use to retrieve them is the *object.id* or Mongo *ObjectId*. The exception to this rule is the *Stream* resource which can only be retrieved by using the *StreamId* property of that stream.

Create

Equivalent of `POST / -d '{"key1":"value1", "key2":"value2"}'`

```
project = {
    'name': 'test project',
    'description': 'a project made for testing purposes',
    'tags': [],
    'streams': [],
}

project = client.projects.create(data=project)

print(project.id)
```

List

Equivalent of `GET /`

```
projects = client.projects.list()

for project in projects:
  print(project.dict())
```

Get

Equivalent of `GET /{id}`

```
project_id = '507f1f77bcf86cd799439011'

project = client.projects.get(project_id)

print(project.dict())
```

Update

Equivalent of `PUT /{id} -d '{"key1":"value1", "key2":"value2"}'`

Note: The update method returns a payload response dictionary rather than a resource data class. The response payload should look something like this

```
{
  "success": True,
  "message": "Patched ['description'] for 507f1f77bcf86cd799439011"
}
```

Warning: The *update* method will replace all the values in the input dictionary payload even if they are *None*. Be careful if you are trying to run *upsert* commands with dictionaries that contain *None* values

```
project_id = '507f1f77bcf86cd799439011'
project_update = {
    'description': 'a project updated for testing purposes',
}

response = client.projects.update(id=project_id, data=project_update)

assert response['success'], response['message']
```

Delete

Equivalent of `DELETE /{id}`

Note: The delete method returns a payload response dictionary rather than a resource data class. The response payload should look something like this

```
{
  "success": True,
  "message": "Project was deleted."
}
```

```
project_id = '507f1f77bcf86cd799439011'

response = client.projects.delete(id=project_id)

assert response['success'], response['message']
```

API Objects

If you've made it this far down you're probably thinking to yourself:

> Great I can login and mess with a project... anything else I can do?
>
> —Some Ambitious User

Glad you asked! There are a bunch of different types of Speckle objects or resources you can mess with using this python client. Here is a list of them with links to their specific documentation:

- Accounts
- Projects
- Streams
- API Clients
- Comments
- SpeckleObjects

Note: Each resource supports some or all of the common methods described above. Additionally each resource might support some extra methods which are specific to that resource. A good example is the *stream* resource which supports the *clone* method

```
streamId = 'GBexG1'

clone_stream, parent_stream = client.streams.clone(streamId)
```

Parting Notes

Hope you've found this quickstart guide useful. Please do contribute any proposed changes/improvements on the main projects github repo.

Client API Reference

This section offers an in-depth description of SpeckleClient API calls.

The `SpeckleClient` class

Accounts Methods

Projects Methods

Streams Methods

Api Client Methods

Comments methods

Objects Methods

CHAPTER 2

Indices and tables

- genindex
- modindex
- search