PythonAeselClient Documentation

Release 1.0.0

Alex Barry

Apr 27, 2019

Contents

CHAPTER 1

Overview

This is a Python Implementation of the Aesel Client. For more information about Aesel, please visit the homepage, https://aostreetart.com/aolabs/aesel/.

For the latest version of the documentation, please visit our ReadTheDocs site, https://pyaesel.readthedocs.io/en/latest.

To find the source code, please visit our github repository at https://github.com/AO-StreetArt/PyAesel.

CHAPTER 2

License

PyAesel is licensed under the Apache2 license. For further details, please refer to the LICENSE file.

CHAPTER 3

Contact

Stuck and need help? Have general questions about the application? We encourage you to publish your question on Stack Overflow. We regularly monitor for the tag 'aesel' in questions.

We encourage the use of Stack Overflow for a few reasons:

- Once the question is answered, it is searchable and viewable by everyone else.
- The forum format offers an easy method to get a larger community involved with a tougher question.

If you believe that you have found a bug in Aesel, or have an enhancement request, we encourage you to raise an issue on our github page.

CHAPTER 4

Guides

4.1 Installing PyAesel

4.1.1 Installing with pip

To install the latest build from pip:

```
pip install --user python-aesel-client
```

4.1.2 Using Development Versions

PyAesel requires Python and pip installed, and all other dependencies can be installed from the root project directory with:

```
git clone https://github.com/AO-StreetArt/PyAesel.git
cd PyAesel
pip install --user -r requirements.txt
```

Finally, you can use pip to install the library locally:

```
pip install --user .
```

Go Home

4.2 Getting Started with Aesel

This tutorial assumes that you have successfully *Installed PyAesel*, and have a running Aesel Server.

Reading through the Aesel Workflow and the process for loading an Aesel Scene is also recommended.

Please note that this is just intended as an overview of the functionality presented, for full documentation please refer to modindex.

4.2.1 Using the Transaction Client

The first object you will generally interact with is the AeselTransactionClient. This is where we specify the Aesel address, which we'll have on localhost for this tutorial.

```
from aesel.AeselTransactionClient import AeselTransactionClient
http_client = AeselTransactionClient("http://localhost:8080")
```

The Transaction Client gives us access to the all of the HTTP Operations in the Aesel API. PyAesel also supplies some basic classes as a data model that is passed to the client, for example creating an Asset with metadata can be accomplished by:

```
from aesel.model.AeselAssetMetadata import AeselAssetMetadata
metadata = AeselAssetMetadata()
metadata.file_type = "json"
metadata.asset_type = "test"
new_key = http_client.create_asset("testupload.json", metadata)
```

In this case, a file in the base python directory called 'testupload.json' will be sent in a multipart file upload, and the variable 'new_key' will be populated with the string key of the asset in the Aesel server. The same key can then be used to retrieve the asset back out.

If the client is unable to connect, or receives a 400 or 500 HTTP Response from the server, then it will throw an appropriate Exception.

Likewise, we can register to a Scene:

```
from aesel.model.AeselUserDevice import AeselUserDevice
ud = AeselUserDevice()
ud.key = "testDevice"
ud.hostname = "localhost"
ud.port = 8182
ud.connection_string = "http://localhost:8182"
register_resp = transaction_client.register("scene-key", ud)
```

In this case, the device will be registered to the specified scene, with the address specified being the UDP address the device is listening on. The response will be a dictionary populated with the JSON information of the response from the Aesel server.

4.2.2 Using the Event Client

In the response of a Scene Registration, you will be presented with UDP information (including IP/hostname, port, and encryption information) which can then be used by the Event Client to send UDP updates for Objects and/or Properties.

```
from aesel.AeselEventClient import AeselEventClient
event_client = AeselEventClient("localhost", 8762)
```

The same data model objects used in the Transaction Client are used by the Event Client, for example, to send an object update:

```
from aesel.model.AeselObject import AeselObject
obj = AeselObject()
obj.key = "obj-key"
obj.scene = "scene-key"
obj.transform = [2.0, 0.0, 0.0, 0.0, 0.0, 2.0, 0.0, 0.0, 0.0, 0.0, 2.0, 0.0, 0.0, 0.0,
↪ 0.0, 2.0]
event_client.send_object_update(obj)
```

4.2.3 Security

The Transaction Client accepts both HTTP and HTTPS locations, and is capable of adding an authentication token via the 'set_auth_info' method:

```
http_client = AeselTransactionClient("http://localhost:8080")
http_client.set_auth_info("auth-token")
```

After calling this method, all calls to the Aesel servers will include the provided authentication token.

The Event client accepts AES-256-cbc encryption details as part of it's constructor and the 'update_endpoint' method. These are generally provided by the registration response from the Aesel Server.

```
event_client = AeselEventClient("localhost", 8762, "encryption-key", "encryption-iv")
```

Go Home

4.3 Developer Notes

This page contains a series of notes intended to be beneficial for any contributors to the Python Aesel Client.

4.3.1 Running Tests

PyAesel tests require a running Aesel server on localhost, and can be run with:

```
python -m pytest
```

4.3.2 Generating Pip Distributions

In order to generate a PIP distribution, you'll need a ~/.pypirc file with the contents:

```
[distutils]
index-servers=pypi

[pypi]
username: username
password: password
```

Then, the following commands will generate the distribution:

```
pip install --user twine
python setup.py sdist bdist_wheel
twine upload dist/*
```

Go Home

CHAPTER 5

API Documentation

- modindex
- search