Clyman Documentation

Release 2.0.0

AO

Sep 30, 2018

Contents:

CHAPTER 1

Overview

CLyman is a service which maintains Objects and Properties over time, and streams out updates to an external service which distributes them to anyone viewing the same elements.

It's primary focus is on distributed animation (ie. using programs like Blender or Maya, and having many artists collaborate on one Scene), as well as multi-user Augmented Reality applications. Clyman serves as the store for all information which is distributed amongst devices, and as the starting point for that distribution.

Clyman is built to be horizontally scalable, and performance is a top priority. In addition, Clyman uses the latest security models for running in low-trust environments, making sure your animations stay safe.

Detailed documentation can be found on ReadTheDocs.

CHAPTER 2

Features

- Storage of Renderable Objects and Properties
- HTTP and UDP API's for both Objects and Properties
- Stream updates via UDP to External Service (Crazy Ivan) for distribution
- Multi-layered security
- Scalable and Flexible Deployment Strategies

Clyman is a part of the AO Aesel Project, along with Crazy Ivan.

Stuck and need help? Have general questions about the application? Reach out to the development team at clyman@emaillist.io

2.1 Getting Started with Clyman

Go Home

2.1.1 Docker

Using the Clyman Docker image is as simple as:

```
docker run --publish=8768:8768 --publish=8762:8762/udp aostreetart/clyman:v2 clyman.
↪prod.mongo=mongodb://mongo:27017
```

However, we also need a running instance of Mongo to do anything interesting. The above command assumes that you have an instance of Mongo running on the same node with the container name 'mongo'. To get you up and running quickly, a Docker Compose file is provided. To start up a Mongo instance and a Clyman instance, simply run the following from the 'compose/min' folder:

```
docker-compose up
```

Alternatively, you can deploy the stack with Docker Swarm using:

```
docker stack deploy --compose-file compose/min/docker-compose.yml clyman-stack
```

Once the services have started, test them by hitting Clyman's healthcheck endpoint:

```
curl http://localhost:8768/health
```

The Transaction (HTTP) API is available on port 8768, and the Event (UDP) API is available on port 8762. Keep in mind that this is not a secure deployment, but is suitable for exploring the *Clyman API*.

You may also continue on to the discussion of *How to Use Clyman*.

2.1.2 Shutdown

Shutdown of Clyman can be initiated with a kill or interrupt signal to the container, or with 'docker stop'. However, at least one udp message must be received afterwards in order to successfully shut down the main event thread. You can send one with:

```
echo "kill" | nc -u $(ip addr show eth0 | grep -Po 'inet \K[\d.]+') 8762
```

Replacing 'eth0' with your network device, if necessary.

2.1.3 Latest Release

Download and unzip the latest release file from https://github.com/AO-StreetArt/CLyman/releases.

Once you have done this, you can run the easy_install script with the -d option to install dependencies and the Clyman executable. Alternatively, you can simply run the install_deps.sh script from the scripts/ folder, and then run the clyman executable from the main release folder.

```
./clyman
```

In order to run Clyman, you will need a Mongo Server installed locally. Instructions can be found at https://docs.mongodb.com/manual/administration/install-on-linux/, or Mongo can be started via a Docker image:

```
docker run --name database -d -p 27017:27017 mongo:latest
```

Either way, the default connection for Clyman will connect without authentication.

You can move on to explore the *Clyman API*, or check out the *Configuration Section* for more details on the configuration options available when starting Clyman.

You may also continue on to the discussion of *How to Use Clyman*.

2.1.4 Building from Source

The recommended system for development of Clyman is either Ubuntu 18.04 or CentOS7. You will need gcc 6.0 or greater and gnu make installed to successfully compile the program.

- Ubuntu

```
sudo apt-get install gcc-6 g++-6
export CC=gcc-6
export CXX=g++-6
```

- Redhat

https://www.softwarecollections.org/en/scls/rhscl/devtoolset-6/

Next, you'll need to clone the repository and run the build_deps script. This will install all of the required dependencies for Clyman, and may take a while to run.

```
git clone https://github.com/AO-StreetArt/CLyman.git
mkdir clyman_deps
cp CLyman/scripts/deb/build_deps.sh clyman_deps/build_deps.sh
cd clyman_deps
sudo ./build_deps.sh
```

You will also need to ensure that the POCO dependency is on the linker path, which can be done with:

```
export LD_LIBRARY_PATH="/usr/local/lib:$LD_LIBRARY_PATH"
```

Now, we can build Clyman:

```
cd ../clyman
make
```

This will result in creation of the clyman executable, which we can run with the below command:

```
./clyman
```

When not supplied with any command line parameters, Clyman will look for an app.properties file to start from.

You may also build the test executable in the tests/ directory with:

```
make tests
```

In order to run Clyman from a properties file, you will need:

- A Mongo Server installed locally. Instructions can be found at https://docs.mongodb.com/manual/administration/install-on-linux/

Neo4j can be started via a Docker image:

```
docker run --name database -d -p 27017:27017 mongo:latest
```

Either way, the default connection for Clyman will connect without authentication.

You can move on to explore the *Clyman API*, or check out the *Configuration Section* for more details on the configuration options available when starting Clyman.

You may also continue on to the discussion of *How to Use Clyman*.

2.1.5 Shutdown

Shutdown of Clyman can be initiated with a kill or interrupt signal to the main thread. However, at least one udp message must be received afterwards in order to successfully shut down the main event thread. You can send one with:

```
echo "kill" | nc -u $(ip addr show eth0 | grep -Po 'inet \K[\d.]+') 8764
```

Replacing 'eth0' with your network device, if necessary.

2.2 How to Use CLyman

2.2.1 Overview

CLyman's primary objective is to track specific attributes of 3-Dimensional scenes which then need to be distributed to interested parties. CLyman provides both an HTTP and UDP API which can be used to create, manipulate, and destroy these attributes. Required changes are then pushed over UDP to an external system for distribution to end-users.

CLyman is an extremely flexible system, but it's design is primarily based on the concepts present in 3D Animation Software (such as Blender or Maya).

2.2.2 Data Model

We start at the basic level with a 3D Object (via the *Object API*), which can move through space. We track these objects via a 4x4 matrix, assuming all users are utilizing the same coordinate system to view the objects.

Objects can be tracked in real-time, or they can be tied to specific frames. A frame is a single point in the animation, and typically animators work by assigning 'keyframes', which are then stored in by CLyman.

Animators frequently utilize a graph-based editor to then tweak the key frames, utilizing 'handles' on either side of each key frame that can be used to adjust individual parts of the movement. These are optional, but can be provided for any object.

Finally, we frequently want to track other attributes over time, which may not be associated directly to the movement of the object. Examples of this might be the value of gravity changing over time, or a keyframed parameter in a shader. Clyman can store these as 'Properties', in the *Property API*, which can have one, two, or three individual values that can each have graph handles as well.

Using CLyman is done by creating Objects and Properties, then using updates to push changes out to all interested parties. Users are expected to create and delete objects and properties throughout any real use case, while CLyman tracks the required values and pumps out an event stream.

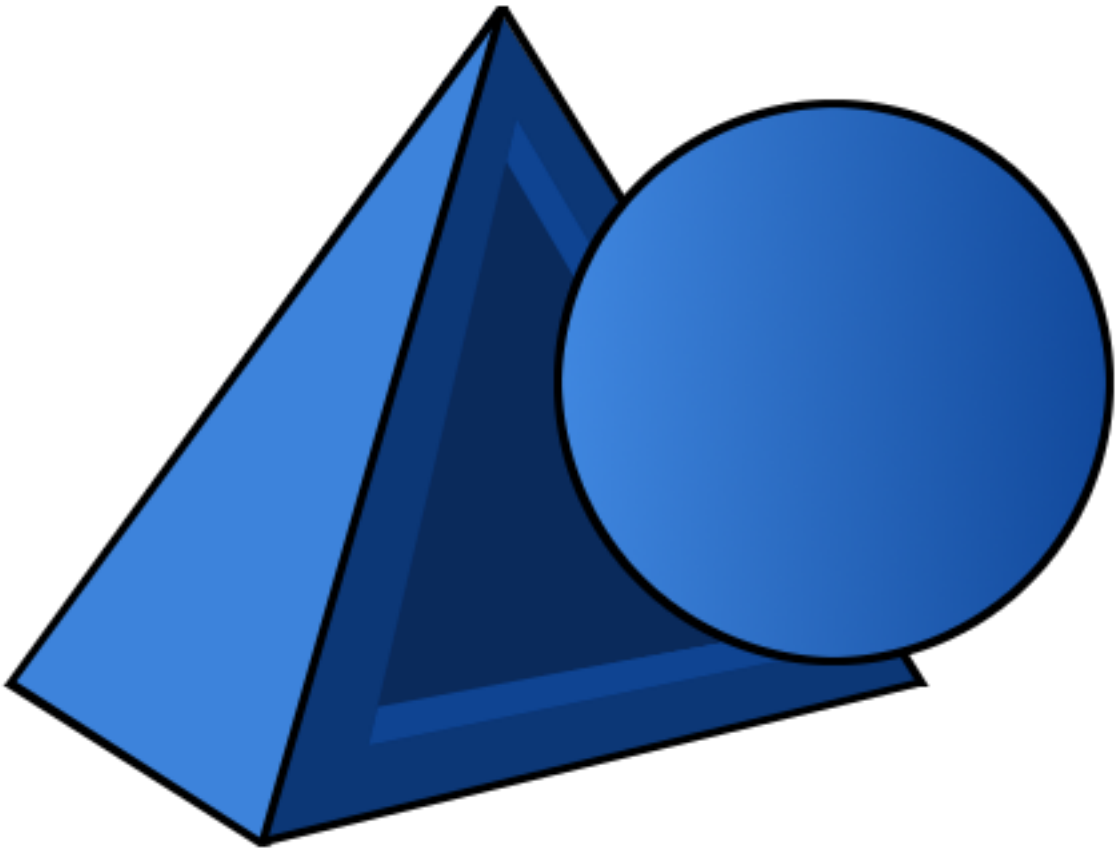
2.2.3 Event Streams

Event Streams are designed to be as fast as possible. Communication of events is limited to UDP and/or shared memory, and events can be restricted to specific clusters of CLyman and other components to ensure minimal network latency.

Optionally, Event Streams can also utilize AES symmetric-encryption, to make sure that the live updates cannot be read by prying eyes.

Go Home

2.3 API Documentation



2.3.1 Object API

An Object is represented by a transformation matrix representing it's position in 3-space, as well as a collection of Assets (Mesh files, Texture files, Shader scripts, etc.), and potentially a frame and/or timestamp. This API exposes CRUD and Query operations for Objects.

Objects are meant to be interacted with by individual devices, and these changes will be streamed to other devices via the Events API. In addition, Create and Update messages sent to the HTTP API are converted to events and streamed out to registered devices.

Object Creation

`POST /v1/object/`

Create a new object.

Request Headers

- Content-Type – Application/json

Status Codes

- 200 OK – Success

http

```
POST /v1/object HTTP/1.1
Host: localhost:8768
Content-Type: application/json

{
  "msg_type": 0,
  "objects":[
    {
            "name": "Test Object 123464",
            "type": "Curve",
            "subtype": "Sphere",
            "owner": "",
            "frame": 0,
            "timestamp": 123456789,
            "translation": [0, 0, 1],
            "quaternion_rotation": [0, 1, 0, 0],
            "euler_rotation": [0, 0, 0],
            "scale": [1, 1, 2],
            "assets": ["Asset_5"],
            "translation_handle":[
                    {"left_type":"test","left_x":0,"left_y":0,"right_type":"","right_x
↪":0,"right_y":0},
                    {"left_type":"","left_x":0,"left_y":0,"right_type":"","right_x":0,
↪"right_y":0},
                    {"left_type":"","left_x":0,"left_y":0,"right_type":"","right_x":0,
↪"right_y":0}],
            "rotation_handle":[
                    {"left_type":"test","left_x":0,"left_y":0,"right_type":"","right_x
↪":0,"right_y":0},
                    {"left_type":"","left_x":0,"left_y":0,"right_type":"","right_x":0,
↪"right_y":0},
                    {"left_type":"","left_x":0,"left_y":0,"right_type":"","right_x":0,
↪"right_y":0},
                    {"left_type":"","left_x":0,"left_y":0,"right_type":"","right_x":0,
↪"right_y":0}],
            "scale_handle":[
                    {"left_type":"test","left_x":0,"left_y":0,"right_type":"","right_x
↪":0,"right_y":0},
                    {"left_type":"","left_x":0,"left_y":0,"right_type":"","right_x":0,
↪"right_y":0},
                    {"left_type":"","left_x":0,"left_y":0,"right_type":"","right_x":0,
↪"right_y":0}]
        }
  ]
}
```

curl

```
curl -i -X POST http://localhost:8768/v1/object -H 'Content-Type: application/json' --
↪data-raw '{"msg_type": 0, "objects": [{"rotation_handle": [{"left_type": "test",
↪"left_x": 0, "left_y": 0, "right_x": 0, "right_y": 0, "right_type": ""}, {"left_type
↪": "", "left_x": 0, "left_y": 0, "right_x": 0, "right_y": 0, "right_type": ""}, {
↪"left_type": "", "left_x": 0, "left_y": 0, "right_x": 0, "right_y": 0, "right_type
↪": ""}, {"left_type": "", "left_x": 0, "left_y": 0, "right_x": 0, "right_y": 0,
↪"right_type": ""}], "euler_rotation": [0, 0, 0], "scale": [1, 1, 2], "name": "Test␣
↪Object 123464", "timestamp": 123456789, "frame": 0, "scale_handle": [{"left_type":
↪"test", "left_x": 0, "left_y": 0, "right_x": 0, "right_y": 0, "right_type": ""}, {
↪"left_type": "", "left_x": 0, "left_y": 0, "right_x": 0, "right_y": 0, "right_type
↪": ""}, {"left_type": "", "left_x": 0, "left_y": 0, "right_x": 0, "right_y": 0,
↪"right_type": ""}], "subtype": "Sphere", "translation_handle": [{"left_type": "test
↪", "left_x": 0, "left_y": 0, "right_x": 0, "right_y": 0, "right_type": ""}, {"left_
↪type": "", "left_x": 0, "left_y": 0, "right_x": 0, "right_y": 0, "right_type": ""},
↪{"left_type": "", "left_x": 0, "left_y": 0, "right_x": 0, "right_y": 0, "right_type
↪": ""}], "quaternion_rotation": [0, 1, 0, 0], "owner": "", "translation": [0, 0, 1],
```

(continues on next page)

(continued from previous page)

wget

```
wget -S -O- http://localhost:8768/v1/object --header='Content-Type: application/json'␣
↪--post-data='{"msg_type": 0, "objects": [{"rotation_handle": [{"left_type": "test",
↪"left_x": 0, "left_y": 0, "right_x": 0, "right_y": 0, "right_type": ""}, {"left_type
↪": "", "left_x": 0, "left_y": 0, "right_x": 0, "right_y": 0, "right_type": ""}, {
↪"left_type": "", "left_x": 0, "left_y": 0, "right_x": 0, "right_y": 0, "right_type
↪": ""}, {"left_type": "", "left_x": 0, "left_y": 0, "right_x": 0, "right_y": 0,
↪"right_type": ""}], "euler_rotation": [0, 0, 0], "scale": [1, 1, 2], "name": "Test␣
↪Object 123464", "timestamp": 123456789, "frame": 0, "scale_handle": [{"left_type":
↪"test", "left_x": 0, "left_y": 0, "right_x": 0, "right_y": 0, "right_type": ""}, {
↪"left_type": "", "left_x": 0, "left_y": 0, "right_x": 0, "right_y": 0, "right_type
↪": ""}, {"left_type": "", "left_x": 0, "left_y": 0, "right_x": 0, "right_y": 0,
↪"right_type": ""}], "subtype": "Sphere", "translation_handle": [{"left_type": "test
↪", "left_x": 0, "left_y": 0, "right_x": 0, "right_y": 0, "right_type": ""}, {"left_
↪type": "", "left_x": 0, "left_y": 0, "right_x": 0, "right_y": 0, "right_type": ""},
↪{"left_type": "", "left_x": 0, "left_y": 0, "right_x": 0, "right_y": 0, "right_type
↪": ""}], "quaternion_rotation": [0, 1, 0, 0], "owner": "", "translation": [0, 0, 1],
↪ "type": "Curve", "assets": ["Asset_5"]}]}'
```

httpie

```
echo '{
  "msg_type": 0,
  "objects": [
    {
      "assets": [
        "Asset_5"
      ],
      "euler_rotation": [
        0,
        0,
        0
      ],
      "frame": 0,
      "name": "Test Object 123464",
      "owner": "",
      "quaternion_rotation": [
        0,
        1,
        0,
        0
      ],
      "rotation_handle": [
        {
          "left_type": "test",
          "left_x": 0,
          "left_y": 0,
          "right_type": "",
          "right_x": 0,
          "right_y": 0
        },
        {
          "left_type": "",
          "left_x": 0,
```

(continues on next page)

(continued from previous page)


```
        "left_y": 0,
        "right_type": "",
        "right_x": 0,
        "right_y": 0
      },
      {
        "left_type": "",
        "left_x": 0,
        "left_y": 0,
        "right_type": "",
        "right_x": 0,
        "right_y": 0
      },
      {
        "left_type": "",
        "left_x": 0,
        "left_y": 0,
        "right_type": "",
        "right_x": 0,
        "right_y": 0
      }
    ],
    "scale": [
      1,
      1,
      2
    ],
    "scale_handle": [
      {
        "left_type": "test",
        "left_x": 0,
        "left_y": 0,
        "right_type": "",
        "right_x": 0,
        "right_y": 0
      },
      {
        "left_type": "",
        "left_x": 0,
        "left_y": 0,
        "right_type": "",
        "right_x": 0,
        "right_y": 0
      },
      {
        "left_type": "",
        "left_x": 0,
        "left_y": 0,
        "right_type": "",
        "right_x": 0,
        "right_y": 0
      },
      {
        "left_type": "",
        "left_x": 0,
        "left_y": 0,
        "right_type": "",
        "right_x": 0,
        "right_y": 0
      }
    ],
    "subtype": "Sphere",
    "timestamp": 123456789,
    "translation": [
      0,
```


(continues on next page)

(continued from previous page)

```
        0,
        1
      ],
      "translation_handle": [
        {
          "left_type": "test",
          "left_x": 0,
          "left_y": 0,
          "right_type": "",
          "right_x": 0,
          "right_y": 0
        },
        {
          "left_type": "",
          "left_x": 0,
          "left_y": 0,
          "right_type": "",
          "right_x": 0,
          "right_y": 0
        },
        {
          "left_type": "",
          "left_x": 0,
          "left_y": 0,
          "right_type": "",
          "right_x": 0,
          "right_y": 0
        }
      ],
      "type": "Curve"
    }
  ]
}' | http POST http://localhost:8768/v1/object Content-Type:application/json
```

python-requests

```
requests.post('http://localhost:8768/v1/object', headers={'Content-Type':
↪'application/json'}, json={'msg_type': 0, 'objects': [{'rotation_handle': [{'left_
↪type': 'test', 'left_x': 0, 'left_y': 0, 'right_x': 0, 'right_y': 0, 'right_type': '
↪'}, {'left_type': '', 'left_x': 0, 'left_y': 0, 'right_x': 0, 'right_y': 0, 'right_
↪type': ''}, {'left_type': '', 'left_x': 0, 'left_y': 0, 'right_x': 0, 'right_y': 0,
↪'right_type': ''}, {'left_type': '', 'left_x': 0, 'left_y': 0, 'right_x': 0, 'right_
↪y': 0, 'right_type': ''}], 'euler_rotation': [0, 0, 0], 'scale': [1, 1, 2], 'name':
↪'Test Object 123464', 'timestamp': 123456789, 'frame': 0, 'scale_handle': [{'left_
↪type': 'test', 'left_x': 0, 'left_y': 0, 'right_x': 0, 'right_y': 0, 'right_type': '
↪'}, {'left_type': '', 'left_x': 0, 'left_y': 0, 'right_x': 0, 'right_y': 0, 'right_
↪type': ''}, {'left_type': '', 'left_x': 0, 'left_y': 0, 'right_x': 0, 'right_y': 0,
↪'right_type': ''}], 'subtype': 'Sphere', 'translation_handle': [{'left_type': 'test
↪', 'left_x': 0, 'left_y': 0, 'right_x': 0, 'right_y': 0, 'right_type': ''}, {'left_
↪type': '', 'left_x': 0, 'left_y': 0, 'right_x': 0, 'right_y': 0, 'right_type': ''},
↪{'left_type': '', 'left_x': 0, 'left_y': 0, 'right_x': 0, 'right_y': 0, 'right_type
↪': ''}], 'quaternion_rotation': [0, 1, 0, 0], 'owner': '', 'translation': [0, 0, 1],
↪ 'type': 'Curve', 'assets': ['Asset_5']}]})
```

response

```
HTTP/1.1 200 OK
Location: http://localhost:8768/v1/object
Content-Type: application/json

{
  "num_records":1,
  "objects":[{"key":"jklmnop"}]
}
```

Object Update

POST /v1/object/{key}

Update an existing object.

Request Headers

- Content-Type – Application/json

Status Codes

- 200 OK – Success
- 404 Not Found – Object Not Found

http

```
POST /v1/object/{key} HTTP/1.1
Host: localhost:8768
Content-Type: application/json

{
  "msg_type": 0,
  "objects":[
    {
             "name": "Test Object 123464",
             "type": "Curve",
             "subtype": "Sphere",
             "frame": 0
    }
  ]
}
```

curl

```
curl -i -X POST 'http://localhost:8768/v1/object/{key}' -H 'Content-Type: application/
↪json' --data-raw '{"msg_type": 0, "objects": [{"subtype": "Sphere", "frame": 0,
↪"type": "Curve", "name": "Test Object 123464"}]}'
```

wget

```
wget -S -O- 'http://localhost:8768/v1/object/{key}' --header='Content-Type:␣
↪application/json' --post-data='{"msg_type": 0, "objects": [{"subtype": "Sphere",
↪"frame": 0, "type": "Curve", "name": "Test Object 123464"}]}'
```

httpie

```
echo '{
  "msg_type": 0,
  "objects": [
    {
      "frame": 0,
      "name": "Test Object 123464",
      "subtype": "Sphere",
      "type": "Curve"
    }
  ]
}' | http POST 'http://localhost:8768/v1/object/{key}' Content-Type:application/json
```

python-requests

```
requests.post('http://localhost:8768/v1/object/{key}', headers={'Content-Type':
↪'application/json'}, json={'msg_type': 0, 'objects': [{'subtype': 'Sphere', 'frame
↪': 0, 'type': 'Curve', 'name': 'Test Object 123464'}]})
```

response

```
HTTP/1.1 200 OK
Location: http://localhost:5885/v1/object/{key}
Content-Type: application/json

{
  "num_records":1,
  "objects":[{"key":"jklmnop"}]
}
```

Object Retrieval

GET /v1/object/(*key*)

Get an object details in JSON Format.

Status Codes

- 200 OK – Success
- 404 Not Found – Object Not Found

http

```
GET /v1/object/{key} HTTP/1.1
Host: localhost:8768
```

curl

```
curl -i 'http://localhost:8768/v1/object/{key}'
```

wget

```
wget -S -O- 'http://localhost:8768/v1/object/{key}'
```

httpie

```
http 'http://localhost:8768/v1/object/{key}'
```

python-requests

```
requests.get('http://localhost:8768/v1/object/{key}')
```

response

```
HTTP/1.1 200 OK
Location: http://localhost:5885/v1/object/{key}
Content-Type: application/json

{
    "msg_type": 2,
    "err_code": 100,
    "num_records": 1,
    "objects": [
        {
            "name": "Test Object 123464",
            "type": "Curve",
            "subtype": "Sphere",
            "frame": 0,
            "timestamp": 123456789,
            "translation_handle": [
                {
                    "left_type": "test",
                    "left_x": 0,
                    "left_y": 0,
                    "right_type": "",
                    "right_x": 0,
                    "right_y": 0
                },
                {
                    "left_type": "",
                    "left_x": 0,
                    "left_y": 0,
                    "right_type": "",
                    "right_x": 0,
                    "right_y": 0
                },
                {
                    "left_type": "",
                    "left_x": 0,
                    "left_y": 0,
                    "right_type": "",
                    "right_x": 0,
                    "right_y": 0
                }
            ],
            "rotation_handle": [
                {
                    "left_type": "test",
                    "left_x": 0,
                    "left_y": 0,
                    "right_type": "",
                    "right_x": 0,
                    "right_y": 0
                },
                {
                    "left_type": "",
                    "left_x": 0,
```

(continues on next page)

(continued from previous page)


```
                    "left_y": 0,
                    "right_type": "",
                    "right_x": 0,
                    "right_y": 0
                },
                {
                    "left_type": "",
                    "left_x": 0,
                    "left_y": 0,
                    "right_type": "",
                    "right_x": 0,
                    "right_y": 0
                },
                {
                    "left_type": "",
                    "left_x": 0,
                    "left_y": 0,
                    "right_type": "",
                    "right_x": 0,
                    "right_y": 0
                }
            ],
            "scale_handle": [
                {
                    "left_type": "test",
                    "left_x": 0,
                    "left_y": 0,
                    "right_type": "",
                    "right_x": 0,
                    "right_y": 0
                },
                {
                    "left_type": "",
                    "left_x": 0,
                    "left_y": 0,
                    "right_type": "",
                    "right_x": 0,
                    "right_y": 0
                },
                {
                    "left_type": "",
                    "left_x": 0,
                    "left_y": 0,
                    "right_type": "",
                    "right_x": 0,
                    "right_y": 0
                }
            ],
            "transform": [
                1,
                0,
                0,
                0,
                0,
                1,
                0,
                0,
```


(continues on next page)

(continued from previous page)

```
                    0,
                    0,
                    2,
                    1,
                    0,
                    0,
                    0,
                    1
                ],
                "assets": [
                    "Asset_5"
                ]
            }
        ]
}
```

Object Deletion

DELETE /v1/object/ (*key*)

Delete an object.

Status Codes

- 200 OK – Success
- 404 Not Found – Object Not Found

http

```
DELETE /v1/object/{key} HTTP/1.1
Host: localhost:8768
```

curl

```
curl -i -X DELETE 'http://localhost:8768/v1/object/{key}'
```

wget

```
wget -S -O- --method=DELETE 'http://localhost:8768/v1/object/{key}'
```

httpie

```
http DELETE 'http://localhost:8768/v1/object/{key}'
```

python-requests

```
requests.delete('http://localhost:8768/v1/object/{key}')
```

Object Query

GET /v1/object/query

Query for objects which match the input JSON. This will only return as many records as specified in the field 'num_records'.

Status Codes

- 200 OK – Success

http

```
POST /v1/object/query HTTP/1.1
Host: localhost:5885
Content-Type: application/json

{
  "objects":[
    {
      "name":"test",
      "assets":["TestAsset10"]
    }
  ]
}
```

curl

```
curl -i -X POST http://localhost:5885/v1/object/query -H 'Content-Type: application/
↪json' --data-raw '{"objects": [{"name": "test", "assets": ["TestAsset10"]}]}'
```

wget

```
wget -S -O- http://localhost:5885/v1/object/query --header='Content-Type: application/
↪json' --post-data='{"objects": [{"name": "test", "assets": ["TestAsset10"]}]}'
```

httpie

```
echo '{
  "objects": [
    {
      "assets": [
        "TestAsset10"
      ],
      "name": "test"
    }
  ]
}' | http POST http://localhost:5885/v1/object/query Content-Type:application/json
```

python-requests

```
requests.post('http://localhost:5885/v1/object/query', headers={'Content-Type':
↪'application/json'}, json={'objects': [{'name': 'test', 'assets': ['TestAsset10']}]}
↪)
```

response

```
HTTP/1.1 200 OK
Location: http://localhost:5885/v1/object/query
Content-Type: application/json

{
    "msg_type": 2,
    "err_code": 100,
    "num_records": 1,
    "objects": [
        {
```

(continues on next page)

(continued from previous page)


```
            "name": "Test Object 123464",
            "type": "Curve",
            "subtype": "Sphere",
            "frame": 0,
            "timestamp": 123456789,
            "translation_handle": [
                {
                    "left_type": "test",
                    "left_x": 0,
                    "left_y": 0,
                    "right_type": "",
                    "right_x": 0,
                    "right_y": 0
                },
                {
                    "left_type": "",
                    "left_x": 0,
                    "left_y": 0,
                    "right_type": "",
                    "right_x": 0,
                    "right_y": 0
                },
                {
                    "left_type": "",
                    "left_x": 0,
                    "left_y": 0,
                    "right_type": "",
                    "right_x": 0,
                    "right_y": 0
                }
            ],
            "rotation_handle": [
                {
                    "left_type": "test",
                    "left_x": 0,
                    "left_y": 0,
                    "right_type": "",
                    "right_x": 0,
                    "right_y": 0
                },
                {
                    "left_type": "",
                    "left_x": 0,
                    "left_y": 0,
                    "right_type": "",
                    "right_x": 0,
                    "right_y": 0
                },
                {
                    "left_type": "",
                    "left_x": 0,
                    "left_y": 0,
                    "right_type": "",
                    "right_x": 0,
                    "right_y": 0
                },
                {
```


(continues on next page)

(continued from previous page)


```
                    "left_type": "",
                    "left_x": 0,
                    "left_y": 0,
                    "right_type": "",
                    "right_x": 0,
                    "right_y": 0
                }
            ],
            "scale_handle": [
                {
                    "left_type": "test",
                    "left_x": 0,
                    "left_y": 0,
                    "right_type": "",
                    "right_x": 0,
                    "right_y": 0
                },
                {
                    "left_type": "",
                    "left_x": 0,
                    "left_y": 0,
                    "right_type": "",
                    "right_x": 0,
                    "right_y": 0
                },
                {
                    "left_type": "",
                    "left_x": 0,
                    "left_y": 0,
                    "right_type": "",
                    "right_x": 0,
                    "right_y": 0
                }
            ],
            "transform": [
                1,
                0,
                0,
                0,
                0,
                1,
                0,
                0,
                0,
                0,
                2,
                1,
                0,
                0,
                0,
                1
            ],
            "assets": [
                "Asset_5"
            ]
        }
    ]
```


(continues on next page)

(continued from previous page)

```
}
```

Object Lock

GET /v1/object/{key}/lock

A lock allows a single client to obtain 'ownership' of an object. This is an atomic operation, and is guaranteed to return a lock to one and only one client who requests it. Keep in mind, however, that no checks are performed within CLyman to verify that the object's owner is the only one updating it, it is the responsibility of the client to obtain a lock prior to making updates.

Status Codes

- 200 OK – Success
- 423 Locked – Object Lock Failed

Query Parameters

- **device** (*string*) – Required. The ID of the Device requesting the lock.

http

```
GET /v1/object/{key}/lock?device=123 HTTP/1.1
Host: localhost:8768
```

curl

```
curl -i 'http://localhost:8768/v1/object/{key}/lock?device=123'
```

wget

```
wget -S -O- 'http://localhost:8768/v1/object/{key}/lock?device=123'
```

httpie

```
http 'http://localhost:8768/v1/object/{key}/lock?device=123'
```

python-requests

```
requests.get('http://localhost:8768/v1/object/{key}/lock?device=123')
```

Object Unlock

DELETE /v1/object/{key}/lock

Unlocking allows a client to release 'ownership' of an object. This is an atomic operation, and no additional locks will be granted on a locked object until this method is called by the owner.

Status Codes

- 200 OK – Success

Query Parameters

- **device** (*string*) – Required. The ID of the Device requesting the lock.

http

```
DELETE /v1/object/{key}/lock?device=123 HTTP/1.1
Host: localhost:8768
```

curl

```
curl -i -X DELETE 'http://localhost:8768/v1/object/{key}/lock?device=123'
```

wget

```
wget -S -O- --method=DELETE 'http://localhost:8768/v1/object/{key}/lock?device=123'
```

httpie

```
http DELETE 'http://localhost:8768/v1/object/{key}/lock?device=123'
```

python-requests

```
requests.delete('http://localhost:8768/v1/object/{key}/lock?device=123')
```

Asset Addition

PUT /v1/object/{object_key}/asset/{asset_key}

We can add an asset to an object easily with this API, which requires no message body.

Status Codes

- 200 OK – Success

http

```
PUT /v1/object/{object_key}/asset/{asset_key} HTTP/1.1
Host: localhost:8768
```

curl

```
curl -i -X PUT 'http://localhost:8768/v1/object/{object_key}/asset/{asset_key}'
```

wget

```
wget -S -O- --method=PUT 'http://localhost:8768/v1/object/{object_key}/asset/{asset_
↪key}'
```

httpie

```
http PUT 'http://localhost:8768/v1/object/{object_key}/asset/{asset_key}'
```

python-requests

```
requests.put('http://localhost:8768/v1/object/{object_key}/asset/{asset_key}')
```

Asset Removal

DELETE /v1/object/{object_key}/asset/{asset_key}

We can remove an asset from an object easily with this API, which requires no message body.

Status Codes

- 200 OK – Success

http

```
DELETE /v1/object/{object_key}/asset/{asset_key} HTTP/1.1
Host: localhost:8768
```

curl

```
curl -i -X DELETE 'http://localhost:8768/v1/object/{object_key}/asset/{asset_key}'
```

wget

```
wget -S -O- --method=DELETE 'http://localhost:8768/v1/object/{object_key}/asset/
↪{asset_key}'
```

httpie

```
http DELETE 'http://localhost:8768/v1/object/{object_key}/asset/{asset_key}'
```

python-requests

```
requests.delete('http://localhost:8768/v1/object/{object_key}/asset/{asset_key}')
```

2.3.2 Property API

A Property is a set of between 1 and 4 double values, which may or not be associated to properties. Properties can also support frames and/or timestamps, just like properties, but cannot be locked and have no transformations.

Properties are meant to be interacted with by individual devices, and these changes will be streamed to other devices via the Events API. In addition, Create and Update messages sent to the HTTP API are converted to events and streamed out to registered devices.

Property Creation

POST /v1/property/

Create a new property.

Request Headers

- Content-Type – Application/json

Status Codes

- 200 OK – Success

http

```
POST /v1/property HTTP/1.1
Host: localhost:8768
Content-Type: application/json

{
  "properties":[
    {
            "key":"12345",
            "name":"testName",
```

(continues on next page)

(continued from previous page)

```
            "parent":"testParent",
            "asset_sub_id":"testAssetSubId",
            "scene":"testScene",
            "frame":1,
            "timestamp":123456789,
            "values":[
                    {
                            "value":100,
                            "left_type":"vector",
                            "left_x":10,
                            "left_y":5,
                            "right_type":"free",
                            "right_x":4,
                            "right_y":3

                    }
            ]
    }
  ]
}
```

curl

```
curl -i -X POST http://localhost:8768/v1/property -H 'Content-Type: application/json'␣
↪--data-raw '{"properties": [{"name": "testName", "parent": "testParent", "timestamp
↪": 123456789, "frame": 1, "scene": "testScene", "values": [{"left_type": "vector",
↪"value": 100, "left_x": 10, "left_y": 5, "right_x": 4, "right_y": 3, "right_type":
↪"free"}], "asset_sub_id": "testAssetSubId", "key": "12345"}]}'
```

wget

```
wget -S -O- http://localhost:8768/v1/property --header='Content-Type: application/json
↪' --post-data='{"properties": [{"name": "testName", "parent": "testParent",
↪"timestamp": 123456789, "frame": 1, "scene": "testScene", "values": [{"left_type":
↪"vector", "value": 100, "left_x": 10, "left_y": 5, "right_x": 4, "right_y": 3,
↪"right_type": "free"}], "asset_sub_id": "testAssetSubId", "key": "12345"}]}'
```

httpie

```
echo '{
  "properties": [
    {
      "asset_sub_id": "testAssetSubId",
      "frame": 1,
      "key": "12345",
      "name": "testName",
      "parent": "testParent",
      "scene": "testScene",
      "timestamp": 123456789,
      "values": [
        {
          "left_type": "vector",
          "left_x": 10,
          "left_y": 5,
          "right_type": "free",
          "right_x": 4,
          "right_y": 3,
```

(continues on next page)

(continued from previous page)

```
          "value": 100
        }
      ]
    }
  ]
}' | http POST http://localhost:8768/v1/property Content-Type:application/json
```

python-requests

```
requests.post('http://localhost:8768/v1/property', headers={'Content-Type':
↪'application/json'}, json={'properties': [{'name': 'testName', 'parent': 'testParent
↪', 'timestamp': 123456789, 'frame': 1, 'scene': 'testScene', 'values': [{'left_type
↪': 'vector', 'value': 100, 'left_x': 10, 'left_y': 5, 'right_x': 4, 'right_y': 3,
↪'right_type': 'free'}], 'asset_sub_id': 'testAssetSubId', 'key': '12345'}]})
```

response

```
HTTP/1.1 200 OK
Location: http://localhost:8768/v1/property
Content-Type: application/json

{
  "num_records":1,
  "objects":[{"key":"jklmnop"}]
}
```

Property Update

POST /v1/property/{property_key}

Update an existing property.

Request Headers

- Content-Type – Application/json

Status Codes

- 200 OK – Success
- 404 Not Found – Property Not Found

http

```
POST /v1/property/{key} HTTP/1.1
Host: localhost:8768
Content-Type: application/json

{
  "properties":[
    {
              "name":"anotherName",
            "parent":"anotherParent"
      }
  ]
}
```

curl

```
curl -i -X POST 'http://localhost:8768/v1/property/{key}' -H 'Content-Type:␣
↪application/json' --data-raw '{"properties": [{"name": "anotherName", "parent":
↪"anotherParent"}]}'
```

wget

```
wget -S -O- 'http://localhost:8768/v1/property/{key}' --header='Content-Type:␣
↪application/json' --post-data='{"properties": [{"name": "anotherName", "parent":
↪"anotherParent"}]}'
```

httpie

```
echo '{
  "properties": [
    {
      "name": "anotherName",
      "parent": "anotherParent"
    }
  ]
}' | http POST 'http://localhost:8768/v1/property/{key}' Content-Type:application/json
```

python-requests

```
requests.post('http://localhost:8768/v1/property/{key}', headers={'Content-Type':
↪'application/json'}, json={'properties': [{'name': 'anotherName', 'parent':
↪'anotherParent'}]})
```

response

```
HTTP/1.1 200 OK
Location: http://localhost:5885/v1/property/{key}
Content-Type: application/json

{
  "num_records":1,
  "properties":[{"key":"jklmnop"}]
}
```

Property Retrieval

GET /v1/property/ (*property_key*)
Get property details in JSON Format.

Status Codes

- 200 OK – Success
- 404 Not Found – Property Not Found

http

```
GET /v1/property/{key} HTTP/1.1
Host: localhost:8768
```

curl

```
curl -i 'http://localhost:8768/v1/property/{key}'
```

wget

```
wget -S -O- 'http://localhost:8768/v1/property/{key}'
```

httpie

```
http 'http://localhost:8768/v1/property/{key}'
```

python-requests

```
requests.get('http://localhost:8768/v1/property/{key}')
```

response

```
HTTP/1.1 200 OK
Location: http://localhost:8768/v1/property/{key}
Content-Type: application/json

{
    "msg_type": 10,
    "err_code": 100,
    "num_records": 1,
    "properties": [
        {
            "name": "testName",
            "parent": "testParent",
            "asset_sub_id": "testAssetSubId",
            "scene": "testScene",
            "frame": 1,
            "timestamp": 123456789,
            "values": [
                {
                    "value": 100,
                    "left_type": "vector",
                    "left_x": 10,
                    "left_y": 5,
                    "right_type": "free",
                    "right_x": 4,
                    "right_y": 3
                }
            ]
        }
    ]
}
```

Property Deletion

DELETE /v1/property/ (*property_key*)

Delete an property.

Status Codes

- 200 OK – Success
- 404 Not Found – Property Not Found

http

```
DELETE /v1/property/{key} HTTP/1.1
Host: localhost:8768
```

curl

```
curl -i -X DELETE 'http://localhost:8768/v1/property/{key}'
```

wget

```
wget -S -O- --method=DELETE 'http://localhost:8768/v1/property/{key}'
```

httpie

```
http DELETE 'http://localhost:8768/v1/property/{key}'
```

python-requests

```
requests.delete('http://localhost:8768/v1/property/{key}')
```

Property Query

GET /v1/property/query
Query for properties which match the input JSON. This will only return as many records as specified in the field 'num_records'.

Status Codes

- 200 OK – Success

http

```
POST /v1/property/query HTTP/1.1
Host: localhost:5885
Content-Type: application/json

{
  "properties":[
    {
      "name":"test"
    }
  ]
}
```

curl

```
curl -i -X POST http://localhost:5885/v1/property/query -H 'Content-Type: application/
↪json' --data-raw '{"properties": [{"name": "test"}]}'
```

wget

```
wget -S -O- http://localhost:5885/v1/property/query --header='Content-Type:␣
↪application/json' --post-data='{"properties": [{"name": "test"}]}'
```

httpie

```
echo '{
  "properties": [
    {
      "name": "test"
    }
  ]
}' | http POST http://localhost:5885/v1/property/query Content-Type:application/json
```

python-requests

```
requests.post('http://localhost:5885/v1/property/query', headers={'Content-Type':
↪'application/json'}, json={'properties': [{'name': 'test'}]})
```

response

```
HTTP/1.1 200 OK
Location: http://localhost:5885/v1/property/query
Content-Type: application/json

{
    "msg_type": 10,
    "err_code": 100,
    "num_records": 1,
    "properties": [
        {
            "name": "testName",
            "parent": "testParent",
            "asset_sub_id": "testAssetSubId",
            "scene": "testScene",
            "frame": 1,
            "timestamp": 123456789,
            "values": [
                {
                    "value": 100,
                    "left_type": "vector",
                    "left_x": 10,
                    "left_y": 5,
                    "right_type": "free",
                    "right_x": 4,
                    "right_y": 3
                }
            ]
        }
    ]
}
```

2.3.3 Event API

An event is a high-speed update with overwrites to the current state of an object. Upon receiving an event from a client via UDP, CLyman sends the event via UDP to an event forwarder (typically Crazy Ivan), whose job it is to pass along that event to any interested parties.

If configured, the event may be encrypted with an AES-256 symmetric key and salt. The event is a JSON message, which can take one of two formats but always includes the field "msg_type". The first format is an Object Overwrite, which follows the same JSON-schema as the *Object Update HTTP API*. The second format is a Property Overwrite, which follows the same JSON-schema as the *Property Update HTTP API*

Here is an example Object Update:

```
{
  "msg_type": 1,
  "key": "5b98880a270698496c36e392",
  "transform": [1.0, 1.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0,␣
↪1.0, 1.0]
}
```

Sending this update would overwrite the transform matrix of the object to:

[[1.0, 1.0, 1.0, 0.0], [1.0, 1.0, 0.0, 1.0], [1.0, 0.0, 1.0, 1.0], [0.0, 1.0, 1.0, 1.0]]

The key is the key of the object, and the "msg_type" = 1 let's CLyman know that this particular update is for an Object and not a property. We can, of course, update a property with an event as well:

```
{
  "msg_type": 9,
  "key": "5b98880a270698496c36e392",
  "values": [{"value": 100.0}]
}
```

This will overwrite the current value of this property to 100.0.

Keep in mind that Events work on the premise of overwrites for any field in the event. You should, therefore, include any information (such as graph handles) onto the objects that you are actually utilizing. For a framed property using graph handles, this might look a bit more like:

```
{
  "msg_type": 9,
  "key": "5b98880a270698496c36e392",
  "frame": 100,
  "values": [{"value": 100.0,
              "left_type": "vector",
              "left_x": 10.0,
              "left_y":10.0,
              "right_type": "vector",
              "right_x": 10.0,
              "right_y":10.0}]
}
```

2.4 Configuration

CLyman can be configured from one or more sources:

- Environment Variables
- Command Line Arguments
- Consul KV Store
- Vault KV Store
- Properties File

The application gives priority to the values retrieved in the above order. This means that an environment variable setting will override any other setting.

Command Line arguments and Properties File keys are lower case, and separated by periods (ie. 'section.key='). Environment Variables, Vault, and Consul keys are all upper case, and are separated by underscores (ie. 'SECTION_KEY=').

All arguments are prefixed with the application name and profile name (ie. 'section.key' becomes 'clyman.prod.section.key'). The profile name can be changed by providing the command line argument 'profile':

```
./clyman profile=dev
```

You can store multiple profiles in your configuration sources, and then specify which one to use on startup of each instance.

2.4.1 Cluster Name

The 'cluster' option on the command line or in a properties file, or the 'AOSSL_CLUSTER_NAME' environment variable, will set the name of the cluster. A cluster is a grouping of CLyman instances, which have been assigned particular scenes to manage. Each CLyman instance is designed to manage a set number of scenes, and this allows for highly optimized streaming of object updates.

The cluster name will affect both how CLyman registers with Consul, if provided, as well as the names of cluster-specific security properties.

2.4.2 Vault

Vault Address - Starts CLyman against a Vault instance. Specified by a collection of arguments:

- vault (Environment Variable VAULT) - the address of the vault instance

```
vault=http://localhost:8200
```

- vault.cert (Environment Variable VAULT_CERT) - the location of the SSL certificate

to use when communicating with Vault. You may also leave this blank to enable SSL encryption without providing a client certificate.

```
vault.cert=
```

- vault.authtype (Environment Variable VAULT_AUTHTYPE) - the authentication type

used by Vault, currently supported options are 'APPROLE' and 'BASIC'

```
vault.authtype=BASIC
```

- vault.un (Environment Variable VAULT_UN) - The Username/Role Id for

authenticating with Vault

```
vault.un=test
```

- vault.pw (Environment Variable VAULT_PW) - The Password/Secret Id for

authenticating with Vault

```
vault.pw=test
```

In addition, the Vault UN and PW can be loaded from files on disk, 'vault_un.txt' and 'vault_pw.txt'. This is the recommended method to set authentication info in CI/CD processes within an application container.

2.4.3 Secure Properties

Secure Properties can be loaded from a properties file for development purposes, but in a Production scenario should always be loaded from a Vault instance. Once CLyman is connected to a Vault instance, the following properties can be loaded:

- CONSUL_SSL_CERT - The SSL Certificate to use when communicating with Consul
- CONSUL_ACL_TOKEN - The ACL Token to use when communicating with Consul
- {cluster-name}_TRANSACTION_SECURITY_AUTH_USER - The username which will authenticate with CLyman over HTTP(s)
- {cluster-name}_TRANSACTION_SECURITY_AUTH_PASSWORD - The password which will authenticate with CLyman over HTTP(s)
- {cluster-name}_TRANSACTION_SECURITY_HASH_PASSWORD - The password for the hashing algorithm used to hash the password prior to storage.
- {cluster-name}_EVENT_SECURITY_OUT_AES_KEY - The key for the AES-256 encryption used for sending UDP messages.
- {cluster-name}_EVENT_SECURITY_OUT_AES_SALT - The salt used for the AES-256 encryption used for sending UDP messages.
- {cluster-name}_EVENT_SECURITY_IN_AES_KEY - The key for the AES-256 encryption used for receiving UDP messages.
- {cluster-name}_EVENT_SECURITY_IN_AES_SALT - The salt used for the AES-256 encryption used for receiving UDP messages.

Secure properties can be loaded from any configuration source, but when loaded from Vault they should be present at the default path ('secret/') in the v2 KV Store.

2.4.4 Consul

Consul Address - Starts CLyman against a Consul instance. Specified by either the *consul* command line argument or the *AOSSL_CONSUL_ADDRESS* environment variable.

```
./clyman consul=http://127.0.0.1:8500
```

We may also include the arguments:

- consul.cert (Environment Variable AOSSL_CONSUL_SSL_CERT) - The location of the

SSL Certificate to use when communicating with Consul. You may also leave this blank to enable SSL encryption without providing a client certificate.

```
consul.cert=
```

- consul.token (Environment Variable AOSSL_CONSUL_ACL_TOKEN) - The ACL Token to use when communicating with Consul

This will enable property retrieval from Consul KV Store & registering with Consul on start up.

The Consul ACL Token can alternatively be generated from the Consul Secret Store in Vault.

- consul.token.role - The role configured in Vault to use to generate the Consul ACL Token.

```
consul.token.role=consul-role
```

2.4.5 Properties File

Properties File - Starts CLyman against a Properties File. Specified by either the *props* command line argument or the *AOSSL_PROPS_FILE* environment variable. For example:

```
./clyman props=app.properties
```

If no properties file is specified, CLyman will look for one named *app.properties* in both the current working folder, and in /etc/clyman/.

The consul address can also be specified within the properties file, with the key *consul*.

2.4.6 HTTPS Setup

SSL Context Configuration is performed on startup, if enabled. If the following properties are set, then SSL Certs for CLyman can be generated dynamically from Vault:

- transaction.security.ssl.ca.vault.active - 'true' or 'false'

```
transaction.security.ssl.ca.vault.active=true
```

- transaction.security.ssl.ca.vault.role_name - the name of the role to use to generate the SSL Cert

```
transaction.security.ssl.ca.vault.role_name=test-role
```

- transaction.security.ssl.ca.vault.common_name - The Common-Name to use on the Certificate

```
transaction.security.ssl.ca.vault.common_name=local
```

Otherwise, SSL Certificate Generation can be configured from a file in the current working directory called 'ssl.properties'.

HTTPS must be enabled with the following parameter:

- transaction.security.ssl.enabled - 'true' or 'false'

```
transaction.security.ssl.enabled=true
```

2.4.7 Mongo Connection

- mongo - A full connection string may be supplied here.

```
mongo=mongodb://localhost:27017
```

In Production Scenarios it is recommended to use Mongo Discovery. If it is set to true, then CLyman will use Consul to find a Mongo instance, and will dynamically find new instances when it encounters many consecutive failures. To enable this, just do not provide a pre-existing connection string.

- mongo.db - The name of the Database to store CLyman information in

```
mongo.db=CLyman
```

- mongo.obj.collection - The name of the Collection in which CLyman will store Object Documents

```
mongo.obj.collection=obj3
```

- mongo.prop.collection - The name of the Collection in which CLyman will store Property Documents

```
mongo.prop.collection=prop
```

- mongo.ssl.active - Activate SSL Encryption on Mongo Connections, may be used with other SSL options in the config file.

```
mongo.ssl.active=true
```

2.4.8 Other Values

There are a number of other options that CLyman can be provided on startup. Below is an overview of the remaining properties:

- Log File - Path on disk to write logs to

```
log.file=clyman.log
```

- Log Level - Debug, Info, Warning, Error

```
log.level=Debug
```

- HTTP host to register with Consul

```
http.host=127.0.0.1
```

- HTTP Port

```
http.port=8766
```

- UDP Port

```
udp.port=8764
```

- Enable Event (UDP) Encryption

```
event.security.aes.enabled=false
```

- Transaction ID's active or inactive. If active, CLyman will ensure a Transaction Id is stamped on each message.

```
transaction.id.stamp=True
```

- Format for transactions (HTTP traffic). Currently only json is supported.

```
transaction.format=json
```

- Method for streaming events. Currently only udp is supported.

```
event.stream.method=udp
```

- Format for streaming events. Currently only json is supported

```
event.format=json
```

Go Home

2.5 Security

CLyman has several forms of security, with one form for transactions (HTTP API), and another form for events (UDP API).

2.5.1 Transactions

Transactional Security utilizes SSL and Basic Auth over HTTP (HTTPS). The username/password can be configured in the application configuration, and SSL will require a valid server key and certificate. The locations can then be entered into the ssl.properties file.

The following commands can be used to generate a self-signed SSL cert, along with a client cert. This can be used to test the secured transactional setup.

openssl req -x509 -newkey rsa:4096 -keyout caKey.key -out caCert.pem -days 365

openssl genrsa -out clientKey.key 2048

openssl req -new -key clientKey.key -out clientCert.csr

openssl x509 -req -in clientCert.csr -CA caCert.pem -CAkey caKey.key -CAcreateserial -out MyClient1.crt -days 1024 -sha256

2.5.2 Events

UDP Events utilize AES encryption, with the key and salt set in the application configuration.

Keep in mind that AES encryption is symmetrical, meaning that the encryption keys must be distributed to the clients in order to encrypt traffic between them and CLyman. The key and salt are delivered to end user devices after a registration transaction, which is both authenticated and encrypted.

2.5.3 Configuration

Secure configuration values should stored in Hashicorp Vault, with full encryption and authentication enabled. Connecting and authenticating to any service requires accessing at least one secure property in Vault, ensuring that any malicious entities must go through Vault to get into any system in the network.

This does mean that your Vault instance should be carefully guarded: it has all of the keys to the castle. However, it is a system designed specifically to guard these secrets, so when used properly it is one of the best safeguards available, along with a healthy dose of common-sense.

2.6 Deployment

This page includes an overview and notes on full production deployment of Clyman. A step-by-step walkthrough for setting up a secured, single-node deployment is also available in the *Advanced Walkthrough*.

A full deployment of Clyman involves several steps:

- Consul Setup
- Mongo Setup
- Vault Setup
- Clyman Setup

Each component has it's own encryption and authentication layers.

2.6.1 Clyman Setup

Clyman instances can be deployed in clusters, along with Crazy Ivan. Each cluster can utilize the same or different databases.

Clyman can load configuration values from Consul and/or Vault, and uses SSL encryption with HTTP Basic Authentication for transactions. Events (sent via UDP) utilize AES symmetric encryption.

Many configuration values are cluster-specific. This allows us to set, for example, separate encryption keys by cluster.

2.6.2 Consul Setup

Deploying a Consul Cluster is covered in detail on the Consul webpage.

Clyman uses the Consul KV Store for unsecured configuration values, as well as using Consul for Service Discovery. It can utilize SSL encryption, as well as the ACL layer.

2.6.3 Mongo Setup

Deploying a Mongo Cluster is covered in detail here.

Mongo in containers is also supported. Either way, once Mongo servers are active, they need to be registered with Consul in order to be picked up by Service Discovery. This can be done with curl, for example:

curl -X PUT -d '{"ID": "mongo", "Name": "mongo", "Tags": ["Primary"], "Address": "localhost", "Port": 27017}' http://127.0.0.1:8500/v1/agent/service/register

2.6.4 Vault Setup

Deploying a Vault Cluster is covered in detail on the Vault webpage.

Clyman can utilize the following Secret Stores:

- Consul - Generate Consul ACL tokens
- PKI - Generate SSL Certs/Keys
- KV - Store secure configuration options

Advanced Walkthrough

Go Home

2.7 Secured Deployment Walkthrough

2.7.1 Overview

A full deployment of Clyman involves several steps:

- Consul Setup
- Mongo Setup
- Vault Setup

- Deploy Clyman

Here, we'll go through each step and deploy a Clyman instance which uses encryption and authentication for all communications, and stores sensitive configuration values securely in vault. We will focus on configuration and startup of the above applications, and it is assumed that you have either installed all of the above either from their latest official release, or have running Docker Images of each. You'll also need openssl installed, in order to generate SSL certs.

For those using a containerized infrastructure (ie. Docker Containers), there are a few additional steps you will need to take.

- You may need to bind volumes to each container in order to provide each container

with the correct certificates/keys for SSL/TLS encryption. In a full production deployment, the best way to provide these to each container is via orchestration architecture, such as Kubernetes, Ansible, etc. For the case of this walk-through, however, no such architecture is needed. * If you are going to network your containers together, you'll need to provide SSL Certificates with Common Names that match to each container name for Neo4j, Consul, and Vault. Otherwise, you may get certificate validation errors.

2.7.2 SSL Setup

We will have to generate SSL Certificates for every service, and in this walk-through we'll be self-signing them. This is not a good idea for a production environment, where you should be getting your certificates signed by a valid CA.

We're going to start by adding an entry to the /etc/hosts file. This is to ensure that the hostname we use resolves to only 127.0.0.1, and not ::1. Add the following line to the file:

You will need to enter 'local' as the Common Name during Certificate Generation, this will prevent certificate errors from occurring for the tutorial. Keep in mind that you will need to use your actual host and domain names here for a production deployment.

In order to generate the CA certs we'll use to self-sign the server certificates, run the following:

```
sudo mkdir /var/ssl
sudo mkdir /var/ssl/consul
sudo mkdir /var/ssl/vault
sudo mkdir /var/ssl/neo4j
sudo openssl genrsa -des3 -out /var/ssl/ca.key 4096
sudo openssl req -new -x509 -days 365 -key /var/ssl/ca.key -out /var/ssl/ca.crt
```

Next, we'll add the CA Certificate to the system trusted certificates, to prevent certificate errors during the tutorial. On Redhat/CentOS:

```
sudo cp /var/ssl/ca.crt /etc/pki/ca-trust/source/anchors/
sudo update-ca-trust
```

Ubuntu users can follow the steps here: https://askubuntu.com/questions/73287/how-do-i-install-a-root-certificate

2.7.3 Consul Setup

Before we do anything else, we should go ahead and generate the SSL certificate that Consul will use:

```
sudo openssl genrsa -out /var/ssl/consul/clientKey.key 2048
sudo openssl req -new -key /var/ssl/consul/clientKey.key -out /var/ssl/consul/
↪clientCert.csr
sudo openssl x509 -req -in /var/ssl/consul/clientCert.csr -CA /var/ssl/ca.crt -CAkey /
↪var/ssl/ca.key -CAcreateserial -out /var/ssl/consul/MyClient1.crt -days 1024 -sha256
```

Now, generate an encryption key for Consul gossip:

```
consul keygen
```

Then, take this value and save it in the file 'consul_config.json':

```
{
      "acl_datacenter": "dc1",
      "acl_master_token": "as3cr3t",
      "acl_default_policy": "deny",
      "acl_down_policy": "extend-cache"
  "encrypt": "your-encryption-key-here",
  "encrypt_verify_incoming": true,
  "encrypt_verify_outgoing": true
}
```

Now, we can startup the agent:

```
mkdir consul_data
consul agent -server -bootstrap -data-dir consul_data/ -bind=127.0.0.1 -config-file␣
↪consul_config.json -ui``
```

After this, we'll need to generate an Agent ACL token:

```
curl --request PUT --header "X-Consul-Token: b1gs33cr3t" --data '{"Name": "Agent Token
↪", "Type": "client", "Rules": "{\"key\":{\"\":{\"policy\":\"write\"}},\"node\":{\"\
↪":{\"policy\":\"write\"}},\"service\":{\"\":{\"policy\":\"write\"}},\"agent\":{\"\":
↪{\"policy\":\"write\"}},\"session\":{\"\":{\"policy\":\"write\"}}}"}' http://127.0.
↪0.1:8500/v1/acl/create
```

This will generate a token, that needs to be added into the Consul config file. We'll also go ahead and add our HTTPS information to enable encryption:

```
{
      "acl_datacenter": "dc1",
      "acl_master_token": "b1gs33cr3t",
      "acl_default_policy": "deny",
      "acl_down_policy": "extend-cache"
  "acl_agent_token": "agent-token-here"
  "encrypt": "encryption-key-here",
  "encrypt_verify_incoming": true,
  "encrypt_verify_outgoing": true,
  "addresses": {
    "https": "0.0.0.0"
  },
  "ports": {
    "https": 8289
  },
  "key_file": "/var/ssl/consul/clientKey.key",
  "cert_file": "/var/ssl/consul/MyClient1.crt",
  "ca_file": "/var/ssl/ca.crt"
}
```

Once the agent is restarted with the new configuration, both encryption and authentication fully enabled.

2.7.4 Mongo Setup

Full Documentation for Mongo TLS/SSL configuration can be found at https://docs.mongodb.com/manual/tutorial/configure-ssl/.

2.7.5 Vault Setup

Now, let's generate our SSL Certificate for Vault:

```
sudo openssl genrsa -out /var/ssl/vault/clientKey.key 2048
sudo openssl req -new -key /var/ssl/vault/clientKey.key -out /var/ssl/vault/
↪clientCert.csr``
sudo openssl x509 -req -in /var/ssl/vault/clientCert.csr -CA /var/ssl/ca.crt -CAkey /
↪var/ssl/ca.key -CAcreateserial -out /var/ssl/vault/MyClient1.crt -days 1024 -
↪sha256``
```

We'll be configuring Vault to use the Consul Storage backend, which means we are going to need an ACL token for Vault to use:

```
curl --request PUT --header "X-Consul-Token: b1gs33cr3t" --data '{"Name": "Agent Token
↪", "Type": "client", "Rules": "{\"key\":{\"vault/\":{\"policy\":\"write\"}},\"node\
↪":{\"\":{\"policy\":\"write\"}},\"service\":{\"vault\":{\"policy\":\"write\"}},\
↪"agent\":{\"\":{\"policy\":\"write\"}},\"session\":{\"\":{\"policy\":\"write\"}}}"}
↪' http://127.0.0.1:8500/v1/acl/create
```

Copy the resulting token, then save the below as a file 'vault_config.hcl':

Before starting the Vault server, you may need to add the CA certificate you generated to your system chain. On CentOS/Redhat, this can be done by copying the CA certificate into the /etc/pki/ca-trust/source/anchors directory, and then refreshing the certificate chain:

```
sudo cp /var/ssl/ca.crt /etc/pki/ca-trust/source/anchors
sudo update-ca-trust
```

You may need to reference the documentation for your particular OS otherwise.

Now, we can start the Vault server:

```
vault server -config=vault_config.hcl
```

In a separate terminal, we'll need to configure the Vault.

```
export VAULT_ADDR='https://local:8200'
vault operator init``
```

Save the unseal keys and root key output when we initialize the vault above.

Next, we will unseal the Vault. We'll need to run this operation 3 times, with 3 unique unseal keys.

```
vault operator unseal
```

Before we continue configuring the Vault, we need to login. Be sure to enter the root key you saw during Vault Initialization.

```
vault login root-key-here
```

Our next step is enabling authentication in Vault. Save the following to a file 'vault_admin_policy.hcl':

Now we can enable userpass authentication, and create a user and policy.

```
vault auth enable userpass
vault write auth/userpass/users/test password=test policies=admins
vault policy write admins vault_admin_policy.hcl
```

Now, we can enable our other secrets engines:

```
vault secrets enable -version=2 kv
vault secrets enable pki
vault secrets enable consul
vault secrets tune -max-lease-ttl=8760h pki
```

We'll need to setup Vault to use a management token from Consul:

```
curl --header "X-Consul-Token: b1gs33cr3t" --request PUT --data '{"Name": "sample",
↪"Type": "management"}' http://127.0.0.1:8500/v1/acl/create
```

Copy the resulting token, and pass it to Vault to use:

```
vault write consul/config/access address=127.0.0.1:8500 token=your-token-here
```

To complete the Consul Secrets Engine configuration, we can add a role which Clyman can use to generate consul ACL tokens.

```
vault write consul/roles/new-role policy=$(base64 <<< 'key "" {policy="read"} service
↪"" {policy="write"}')
```

Next, let's finish the PKI Secrets Engine configuration, which will allow Clyman to generate SSL Certificates from Vault on startup.

First, we have Vault generate an internal CA certificate (Note that this is not advised in Production scenarios), and signing information:

```
vault write pki/root/generate/internal common_name=my-website.com ttl=8760h
vault write pki/config/urls issuing_certificates="http://127.0.0.1:8200/v1/pki/ca"␣
↪crl_distribution_points="http://127.0.0.1:8200/v1/pki/crl"``
```

Finally, we'll set up another role that allows for generation of SSL Certificates

```
vault write pki/roles/pki-role allowed_domains=local allow_subdomains=true max_ttl=72h
```

2.7.6 Clyman Setup

Before starting Clyman, we'll want to populate some configuration values.

Non-secure configuration options can be set in Consul. Most of the defaults will work for us here, so we'll just go ahead and enable authentication in Crazy Ivan HTTPS requests:

```
curl --header "X-Consul-Token: b1gs33cr3t" --request PUT --data 'single' https://
↪local:8500/v1/kv/clyman/prod/CLYMAN_PROD_TRANSACTION_SECURITY_AUTH_TYPE
```

Secure configuration options can be set in Vault. Let's setup our core encryption information in Vault. First, we enter Event (UDP) encryption settings:

```
vault kv put secret/IVAN_PROD_TEST_EVENT_SECURITY_IN_AES_SALT IVAN_PROD_TEST_EVENT_
↪SECURITY_IN_AES_SALT=test
vault kv put secret/IVAN_PROD_TEST_EVENT_SECURITY_IN_AES_KEY IVAN_PROD_TEST_EVENT_
↪SECURITY_IN_AES_KEY=test
vault kv put secret/IVAN_PROD_TEST_EVENT_SECURITY_OUT_AES_SALT IVAN_PROD_TEST_EVENT_
↪SECURITY_OUT_AES_SALT=test
vault kv put secret/IVAN_PROD_TEST_EVENT_SECURITY_OUT_AES_KEY IVAN_PROD_TEST_EVENT_
↪SECURITY_OUT_AES_KEY=test
```

Next, we setup our authentication information for Neo4j:

```
vault kv put secret/IVAN_PROD_NEO4J_AUTH_UN IVAN_PROD_NEO4J_AUTH_UN=neo4j
vault kv put secret/IVAN_PROD_NEO4J_AUTH_PW IVAN_PROD_NEO4J_AUTH_PW=neo4j
```

Finally, we provide the authentication options for Transactions (HTTP(s)):

```
vault kv put secret/IVAN_PROD_TRANSACTION_SECURITY_AUTH_USER IVAN_PROD_TRANSACTION_
↪SECURITY_AUTH_USER=test
vault kv put secret/IVAN_PROD_TRANSACTION_SECURITY_AUTH_PASSWORD IVAN_PROD_
↪TRANSACTION_SECURITY_AUTH_PASSWORD=test
vault kv put secret/IVAN_PROD_TRANSACTION_SECURITY_HASH_PASSWORD IVAN_PROD_
↪TRANSACTION_SECURITY_HASH_PASSWORD=test
```

Full details on configuration options can be found in the Configuration section of the documentation. Finally, you can start Crazy Ivan with:

```
./clyman clyman.prod.vault=https://local:8200 clyman.prod.vault.cert= clyman.prod.
↪vault.authtype=BASIC clyman.prod.vault.un=test clyman.prod.vault.pw=test clyman.
↪prod.consul.token.role=new-role clyman.prod.consul=https://local:8289 clyman.prod.
↪consul.cert= clyman.prod.cluster=test clyman.prod.mongo.ssl.active=true clyman.prod.
↪transaction.security.ssl.ca.vault.active=true clyman.prod.transaction.security.ssl.
↪ca.vault.role_name=pki-role clyman.prod.transaction.security.ssl.ca.vault.common_
↪name=local.local
```

Several files will be created on startup, with the extensions '.key' and '.pem'. These are all of the certificates and keys that Crazy Ivan is using to encrypt the HTTPS connection.

Make sure your server is up using the health check endpoint:

```
curl --user test:test https://local.local/health
```

Go Home

2.8 Architecture

CLyman serves two primary purposes within the Aesel architecture:

1. Serves as the system of reference for animation/game data which is updated by clients.
2. Serves as the origin point for Events, high-speed updates streamed out to connected devices.

In other words, CLyman takes input from clients moving objects and/or adjusting properties, streams that out to other devices, and maintains the current state of all objects and properties throughout all changes. It is the core service which maintains the 3D objects that clients are expected to interact with.

2.8.1 Technical Overview

CLyman is designed to be used as a service within a larger architecture. It will take in CRUD messages for 3D Objects and Properties (both over HTTP).

Running CLyman requires an instance of Mongo to connect to in order to perform most functions. Mongo serves as the back-end database for CLyman.

CLyman can also be deployed with Consul as a Service Discovery and Distributed Configuration architecture. This requires the Consul Agent to be deployed that Crazy Ivan can connect to.

CLyman can be deployed securely using Vault as a secret store and/or intermediate CA.

2.8.2 Object Change Streams (Events)

Object Change Streams ensure that all registered User Devices remain up to date about objects within their scenes. CLyman generates UDP messages to downstream services whenever updates are received (either over HTTP or UDP).

The changes streams are designed to be high-speed and high-volume. CLyman can process many messages in parallel, and database persistence is performed after streaming updates to downstream services.

2.8.3 Clustering

Scene-specific clustering is a central idea in CLyman. This is an idea borrowed from large-scale MMORPG's, in which large maps are broken apart and each piece is run by separate servers. This allows for horizontal scaling of the system to cover additional real-estate, physical or digital.

A cluster name can be provided by CLyman on startup, and other applications should use this cluster name to identify the appropriate CLyman to send messages to.

Go Home

2.9 Dependencies

Go Home

CLyman is built on top of the work of many others, and here you will find information on all of the libraries and components that CLyman uses to be successful.

Licenses for all dependencies can be found in the licenses folder within the repository.

2.9.1 RapidJson

RapidJson is a very fast JSON parsing/writing library.

RapidJson is released under an MIT License.

2.9.2 AO Shared Service Library

AOSSL is a collection of C++ wrappers on many of the C libraries listed here.

AOSSL is released under an MIT License.

2.9.3 MongoCxx

MongoCxx is the official C++ client for Mongodb, the database behind CLyman.

MongoCxx is released under an Apache 2 License.

2.9.4 LibUUID

LibUUID is a linux utility for generating Universally Unique ID's.

LibUUID is released under a BSD License.

2.9.5 POCO

The POCO Project is a set of libraries for building networked C++ applications.

It is released under the Boost Software License.

2.9.6 Boost

The Boost Project is a set of C++ libraries, that are primarily used for UDP Processing.

It is released under the Boost Software License.

2.9.7 Automatic Dependency Resolution

For Ubuntu 16.04 or Centos7, the build_deps.sh scripts should allow for automatic resolution of dependencies.

Developers can utilize the Vagrant image, which will install dependencies in the VM.

End-Users can run the Docker image, which will install dependencies in the container.

2.9.8 Other Acknowledgements

Here we will try to list authors of other public domain code that has been used:

```
René Nyffenegger - Base64 Decoding Methods
```

2.10 Developer Notes

This page contains a series of notes intended to be beneficial for any contributors to CLyman.

2.10.1 Vagrant

We provide a Vagrantfile to setup a development environment, but this requires that you install Vagrant. Once you have Vagrant installed, cd into the main directory and run:

```
vagrant up
```

Once the box starts, you can enter it with:

```
vagrant ssh
```

The Project folder on your machine is synced to the /vagrant folder in the VM, so you will need to move there before building. Once in that folder, you can build the executable and tests:

```
make && make test
```

2.10.2 Packer

A Packer file is provided, which can be used with Hashicorp Packer. Configuration is provided for building a Docker Image, which can be executed with:

```
packer build packer.json
```

This will create a tagged image, which can then be pushed with

```
docker push aostreetart/clyman:v2
```

2.10.3 Docker

The CLyman Docker Hub Repository contains the latest Docker images for CLyman.

2.10.4 Running Test Cases

Building the tests can be done with:

```
make test
```

Tests cases are run using Catch2 (https://github.com/catchorg/Catch2), a few examples are shown below:

Run all tests:

```
./tests/tests
```

Run only the unit tests:

```
./tests/tests [unit]
```

Run only the integration tests:

```
./tests/tests [integration]
```

2.10.5 Continuous Integration

Travis CI is used to run automated tests against CLyman each time a commit or pull request is submitted against the main repository. The configuration for this can be updated via the .travis.yml file in the main folder of the project repository.

Latest CI Runs

2.10.6 Documentation

Documentation is built using Sphinx and hosted on Read the Docs.

Updates to documentation can be made in the docs/ folder of the project repository, with files being in the .rst format.

2.10.7 Generating Releases

The release_gen.sh script is utilized to generate releases for various systems. It accepts three command line arguments: * the name of the release: clyman-*os_name-os_version* * the version of the release: we follow semantic versioning * the location of the dependency script: current valid paths are linux/deb (uses apt-get) and linux/rhel (uses yum)

Go Home

/v1