
Aesel Documentation

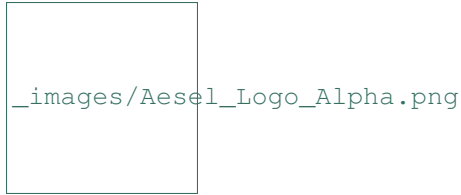
Release 2.0.0

AO

Apr 21, 2019

Contents

1	Other Aesel Repositories	3
1.1	Components	3
1.2	Clients and Integrations	3
2	Documentation	5
2.1	Getting Started	5
2.2	Aesel HTTP API	18
2.3	Design Documents	49
2.4	Advanced Topics	67
	Aesel HTTP API	77



Aesel is a project aimed at bringing together the digital and physical worlds to connect us in new, diverse ways.

Computers today and the Internet allow for an unprecedented level of communication. However, these same computers sit between their users, both metaphorically and literally. So, while we are all connected, we are also isolated.

We believe that there is a different way to interact with the digital world, one that does not isolate us and force us away from each other. We believe that there is a way to communicate digitally that can also be personal, and powerful.

Aesel is a back-end server architecture built to allow this communication by independently communicating between arbitrary devices. It tracks object locations, rotations, and scaling and streams these out to devices that need them, as well as tracking relationships between different coordinate systems. It will also ensure that 3D assets are distributed to all necessary devices.

The Documentation for Aesel can be found on [ReadTheDocs](#).

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.

Other Aesel Repositories

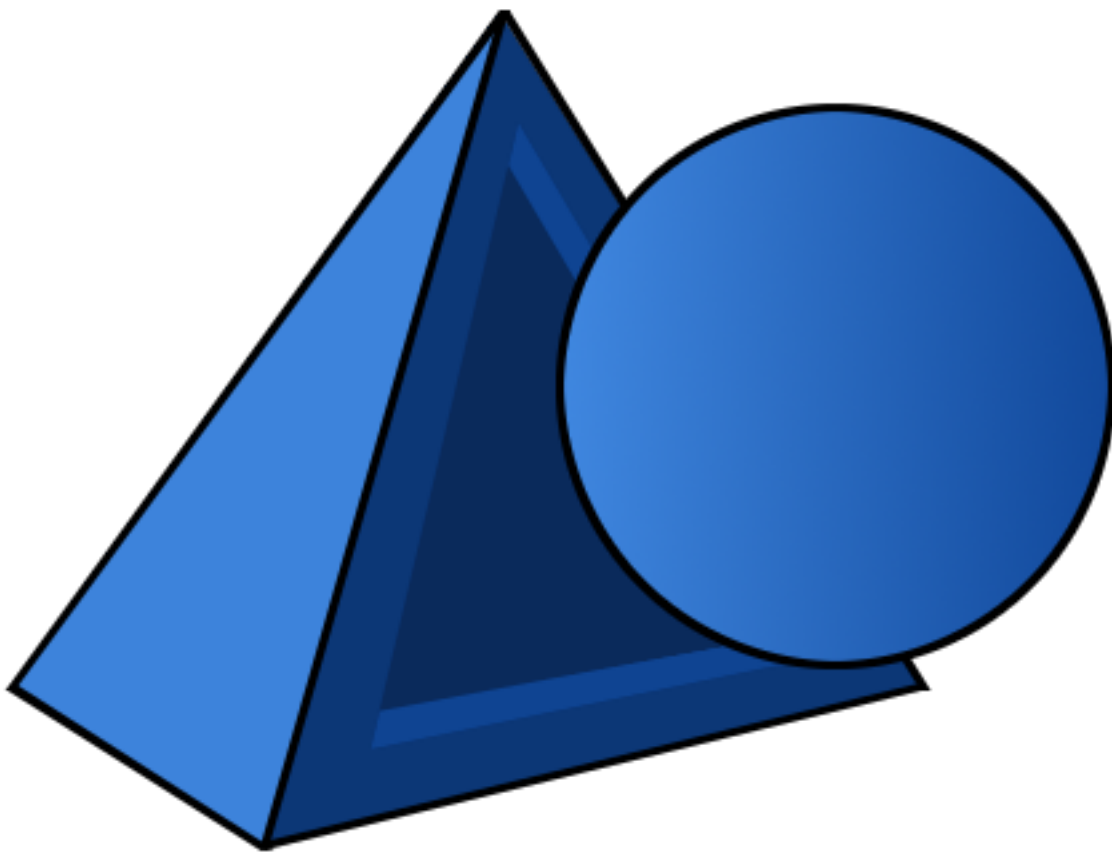
1.1 Components

- Adrestia
- Crazy Ivan
- CLyman
- Kelona
- AeselProjects

1.2 Clients and Integrations

- Blender Sync
- PyAesel

2.1 Getting Started



2.1.1 Installing Aesel

System Requirements

In order to run Aesel, you should have at least one server with a minimum of:

- 8GB RAM Available
- 8GB Hard Disk Space Available

Note that production systems will likely require significantly more resources.

In order to run Aesel on Docker, you should have at least:

- Docker CE >17.03 or Docker EE >17.06
- Docker Compose >1.12.0 for using Docker Compose scripts

Running Aesel natively is supported on the following platforms:

- Ubuntu >16.04
- Redhat/Centos >7

Docker

The easiest way to get started with Aesel is with [Docker](#) and [Docker Compose](#). Before continuing, please be sure to follow the steps in the following links:

- <https://docs.docker.com/install/>
- <https://docs.docker.com/compose/install/>

Note

- Docker Community Edition is more than sufficient, but if you already have a subscription to Docker Enterprise, that will work perfectly well too.
 - Windows users will typically have compose installed with Docker, and won't need to follow the steps in the second link.
-

Starting a Development Deployment

This is a great option if you want to play around with the Aesel API's, integrations, and web browser, without dealing with all of the complexity that a secure environment brings.

First, Download the Aesel setup files from <https://github.com/AO-StreetArt/Aesel/archive/master.zip>.

Unzip the files, and open a terminal/command prompt from the main folder.

Run the following commands:

```
./aesel.sh dev
```

Note

Windows users will need to call docker-compose manually, from the ‘deployment/min/compose’ folder. You will need the environment variable NETWORK_INTERFACE_ADDRESS set to your IP address for the development cluster UDP API to function correctly.

Congratulations, Aesel is now up and running on your computer! To make sure that everything started correctly, open up your web browser after about 30 seconds and go to the address <http://localhost:8080/portal/home>.

Advanced Deployment Guides

- *Deploying a Secure, Single-Server Setup*

Continue on to the quickstart guide

2.1.2 Getting Started with Aesel

Installing Aesel

Make sure that you have either *installed Aesel*, or have access to an existing Aesel Cloud before continuing.

Interacting with Aesel

User Interface

The Aesel user interface will be available in a development environment (if you are accessing a remote environment, then replace ‘localhost:8080’ with your Aesel address):

- Aesel Web UI - <http://localhost:8080/portal/home>

If you are accessing a secure environment, then you can access the web UI at:

- Aesel Web UI - <https://aesel-cloud-demo.com/portal/login>

with a valid username and password (you can login to the above link with the username ‘demo’, and the password ‘demo’).

API

If you setup a development environment, then Aesel’s HTTP API will be available at <http://localhost:8080>. The UDP API for CLyman is available at localhost:8762.

If you are connecting to an existing Aesel Cloud, then the HTTP URL will be provided along with your login information. You can get a bearer token for use with the *Login API*.

The UDP host, port, and encryption/decryption information will be provided in the response of a Device Registration message.

Official client libraries are available for the following languages:

- Python

Other languages are encouraged to make use of the HTTP and UDP API’s directly.

Client Applications

Some popular 3D applications have existing Aesel integrations:

Blender

Aesel supports integration with the [Blender](#) through an addon, [BlenderSync](#).

Next

Continue on to the overview to read more about interacting with Aesel

2.1.3 Using the Aesel Web Browser

The Aesel Web Browser is great for exploring and learning about the Aesel API, as well as for administering and creating applications using Aesel.

The navigation bar at the top of the page contains links to the various sections of the browser, including ‘Projects’, ‘Scenes’, and ‘Assets’.

Assets

Assets are files stored in the Aesel cloud, which can include image files, scripts, or 3D export files (such as .obj or .blend). In the Asset Browser, you can query for different assets, and view a preview of many of them.

Double-click on any record in the data grid to load it into the Asset Viewer.

Scenes

A Scene consists of a number of renderable Objects, which users can collaboratively manipulate. In the Scene Browser, you can query for various Scenes, and see the Objects and Assets associated with each one.

Double-click on any record in the Scene Grid to load the corresponding Objects into the Object Grid.

Projects

A Project consists of groups of Scenes and Assets. These can be used to organize your work in Aesel.

Double-click on any record in the data grid to load the thumbnail and description.

Next

Continue on to the overview to read more about interacting with Aesel

2.1.4 Aesel Workflow

Aesel builds on top of a traditional video game server by tracking relationships between coordinate systems. This is a critical component when supporting Augmented Reality clients who are rendering these objects on top of a view of real space. In this case, we can't assume that all of the devices are referencing the same origin point, as well as x, y, and z directions. Aesel gives a means of storing these transformations and calculating the transformations wherever possible.

Note: It is not required to utilize this functionality, meaning Aesel will function perfectly well as a traditional video game server, or in any other scenario which involves multiple users interacting with 3-dimensional objects. We will make special note of those steps in the workflow which are unnecessary for non-Augmented Reality clients.

Scene Registration

Whether we are building an Augmented Reality, Virtual Reality, traditional game console, or other application, our first step is to ask Aesel for information on scenes that are available. A scene is simply a logical grouping of objects which devices can move between, so they can correspond to different physical locations (in a typical AR application), different independent games (in a typical VR/PC/Console game)

This amounts to executing a Scene Query. We can look for scenes based on names, regions, and tags, or we can look for scenes within a specific distance of our latitude/longitude.

For example: http

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

{
  "scenes": [
    {
      "name": "test",
      "region": "US-MD",
      "latitude": 124,
      "longitude": 122,
      "assets": ["TestAsset10"],
      "tags": ["Testing2"]
    }
  ]
}
```

curl

```
curl -i -X POST http://localhost:5885/v1/scene/query -H 'Content-Type: application/
↪json' --data-raw '{"scenes": [{"name": "test", "tags": ["Testing2"], "region": "US-
↪MD", "longitude": 122, "latitude": 124, "assets": ["TestAsset10"]}]]'
```

response

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

(continues on next page)

(continued from previous page)

```
{
  "num_records":1,
  "scenes": [
    {
      "key": "jklmnop",
      "name": "TestScene10",
      "region": "US-MD",
      "latitude": 124.0,
      "longitude": 122.0,
      "tags": ["test", "test2"],
      "asset_ids": ["asset1", "asset2"]
    }
  ]
}
```

Joining a game or walking into a new area is represented by a Scene Registration. Aesel responds to this message by informing you about all of the objects in the scene, as well as any other information you need to render the scene as a whole. We download all of the data we need to actually visualize the 3D objects from the server.

This exchange of information is initiated by a Scene Registration message: <http>

```
POST /v1/scene/key/register HTTP/1.1
Host: localhost:5885
Content-Type: application/json

{
  "scenes": [
    {
      "key": "jklmnop",
      "devices": [
        {
          "key": "Ud132",
          "hostname": "localhost",
          "port": 4444,
          "connection_string": "127.0.0.1:4444",
          "transform": {
            "translation": [0, 0, 0],
            "rotation": [0, 0, 0]
          }
        }
      ]
    }
  ]
}
```

curl

```
curl -i -X POST http://localhost:5885/v1/scene/key/register -H 'Content-Type:
↪application/json' --data-raw '{"scenes": [{"devices": [{"connection_string": "127.0.
↪0.1:4444", "hostname": "localhost", "port": 4444, "key": "Ud132", "transform": {
↪"translation": [0, 0, 0], "rotation": [0, 0, 0]}}], "key": "jklmnop"}]}'
```

response

```

HTTP/1.1 200 OK
Location: http://localhost:5885/v1/scene/key/register
Content-Type: application/json

{
  "msg_type": 4,
  "err_code": 100,
  "num_records": 2,
  "scenes": [
    {
      "key": "20dd78a2-9224-11e8-b492-d850e6db3ad1",
      "active": true,
      "distance": 0,
      "assets": [],
      "tags": [],
      "devices": [
        {
          "key": "12345",
          "transform": {
            "translation": [
              0,
              0,
              0
            ],
            "rotation": [
              0,
              0,
              0
            ]
          }
        }
      ]
    },
    {
      "key": "123456",
      "active": true,
      "distance": 0,
      "assets": [],
      "tags": [],
      "devices": []
    }
  ]
}

```

After Registration, there may be a great deal of communication between the user's device and the Aesel server. Rather than dumping all of the necessary information directly to the device, Aesel simply responds to a registration with the keys to find both it's Assets and Objects. The client then responds with load messages for these records, so it loads each file on the server until the device has everything it needs. For a more detailed explanation of the process involved in loading a scene, please see the [Loading a Scene](#) page.

Scene Synchronization

Note: Scene Synchronization is only required for Augmented Reality applications, where each device cannot be provided on start-up with a pre-defined coordinate system for the scene (as is generally the case in standard gam-

ing/animation and Virtual Reality)

In this stage, we need to correct our view with that of everyone else. This can be done any number of ways (from a real reference object to device-device communication), but once the right adjustments are made we send this information back to Aesel.

The message to Aesel takes the form of a Scene Synchronization Message: http

```
POST /v1/scene/key/align HTTP/1.1
Host: localhost:5885
Content-Type: application/json

{
  "scenes": [
    {
      "key": "jklmnop",
      "devices": [
        {
          "key": "Ud132",
          "transform": {
            "translation": [0, 0, 0],
            "rotation": [0, 0, 0]
          }
        }
      ]
    }
  ]
}
```

curl

```
curl -i -X POST http://localhost:5885/v1/scene/key/align -H 'Content-Type:
↪application/json' --data-raw '{"scenes": [{"devices": [{"transform": {"translation
↪": [0, 0, 0], "rotation": [0, 0, 0]}, "key": "Ud132"}], "key": "jklmnop"}]}'
```

We need to perform this step to determine the difference between the default coordinate system of your device, and the coordinate system of the Scene. This is the coordinate system that all of the objects are stored in, and that your device will need to use when referencing the scene. Think of it this way: If I tell you that there is a cube located at the point (1, 2, 3), how do you know where (0, 0, 0) is? If we don't agree, then we'll end up seeing the cube in different places.

Object Updates

Now, we can move around any of the virtual objects we downloaded right in front of us. When we do, that information is sent to Aesel and, from there, to everyone else in the scene.

Here we utilize the Object Streaming API to move the object around:

Aesel provides live change feeds of Object location, rotation, and scaling for any users registered to the Scene containing that Object. These feeds are designed to be extremely high-speed, and are sent via UDP.

Note: Live Change feeds are the heart-and-soul of Aesel, and continually stream the current transformation matrix of the objects to all registered devices in real-time. Clients who are running live simulations or games will only interact

through this API until the game/simulation ends.

Cross-Registration

Note: Cross-Registration is only required for Augmented Reality applications, where each device cannot be provided on start-up with a pre-defined coordinate system for the scene (as is generally the case in standard gaming/animation and Virtual Reality)

When entering another, different physical location, we send another registration message to Aesel. And, just like before, we need to correct our view with that of everyone else.

This time, Aesel sees that you've corrected two different scenes that you've moved between, and it uses this information to calculate a general correction that can be used by anyone else going between these two scenes. This means that, if anyone else wants to change from one of these scenes to another, they will be provided with the correct transformation upon registration.

De-Registration

We finish by de-registering from the first scene, as we no longer need to receive updates on the objects in that scene. This can be done after moving out of a physical area in an Augmented Reality application, or when leaving a game in a Virtual Reality or standard gaming application.

This message to Aesel comes as a De-Registration Message: http

```
POST /v1/scene/jklmnop/deregister HTTP/1.1
Host: localhost:8080
Content-Type: application/json

{
  "scenes": [
    {
      "key": "jklmnop",
      "devices": [
        {
          "key": "Ud132"
        }
      ]
    }
  ]
}
```

curl

```
curl -i -X POST http://localhost:8080/v1/scene/jklmnop/deregister -H 'Content-Type: application/json' --data-raw '{"scenes": [{"devices": [{"key": "Ud132"}], "key": "jklmnop"}]}'
```

Notice that we only de-register after performing any corrections needed, and once we are synchronized we can leave the original scene. Also note that we do not necessarily need to leave the original scene. We may also remain registered and continue receiving updates on all objects in both scenes.

Continue on to Loading an Aesel Scene to read about the process of loading an Aesel scene

2.1.5 Loading an Aesel Scene

Once registered to a Scene, a device needs to actually load all of the data that comprises this full 3-D Environment. This can contain an enormous amount of different types of data, such as (but not limited to):

- Mesh Data (.obj)
- Animation Data (.fbx)
- Shader Data (.glsl)
- Texture Data (.png)

These are just a few examples of the multitude of types of information that may be present, and we boil these all down to one term: 'Assets'.

Assets may be present for the Scene (such as a reference image or map/level data), as well as for individual objects (which also have an associated location, rotation, and scale within the scene). So, our process of loading a scene boils down to a few simple steps:

- Load Scene
- Load Scene Assets
- Load Scene Objects
- Load Object Assets

Loading Scenes

Loading a Scene can be done either with a Scene Retrieval: `http`

```
GET /v1/scene/key HTTP/1.1
Host: localhost:5885
```

`curl`

```
curl -i http://localhost:5885/v1/scene/key
```

`response`

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

{
  "msg_type": 2,
  "err_code": 100,
  "num_records": 1,
  "start_record": 0,
  "scenes": [
    {
      "key": "123",
      "name": "testScene",
      "region": "us-ga",
      "latitude": 100,
      "active": true,
```

(continues on next page)

(continued from previous page)

```

        "longitude": 100,
        "distance": 0,
        "assets": [
            "asset1"
        ],
        "tags": [
            "tag1"
        ],
        "devices": []
    }
]
}

```

or, we can find Scenes with a Scene Query: http

```

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

{
  "scenes": [
    {
      "name": "test",
      "region": "US-MD",
      "latitude": 124,
      "longitude": 122,
      "assets": ["TestAsset10"],
      "tags": ["Testing2"]
    }
  ]
}

```

curl

```

curl -i -X POST http://localhost:5885/v1/scene/query -H 'Content-Type: application/
↪json' --data-raw '{"scenes": [{"name": "test", "tags": ["Testing2"], "region": "US-
↪MD", "longitude": 122, "latitude": 124, "assets": ["TestAsset10"]}]}

```

response

```

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

{
  "num_records": 1,
  "scenes": [
    {
      "key": "jklmnop",
      "name": "TestScene10",
      "region": "US-MD",
      "latitude": 124.0,
      "longitude": 122.0,
      "tags": ["test", "test2"],
      "asset_ids": ["asset1", "asset2"]
    }
  ]
}

```

(continues on next page)

(continued from previous page)

```
]
}
```

Loading Scene Assets

Now that we have our scene, We query for Assets related to the scene: http

```
GET /v1/relationship?type=scene&related=123 HTTP/1.1
Host: localhost:8080
```

curl

```
curl -i 'http://localhost:8080/v1/relationship?type=scene&related=123'
```

response

```
HTTP/1.1 200 OK
Location: http://localhost:8080/v1/relationship?type=scene&related=123

[
  {
    "id": "5bbd6ea100bd75575fb32caa",
    "assetId": "5bbd6ea100bd75575fb32ca8",
    "assetSubId": "meshName",
    "relationshipType": "object",
    "relationshipSubtype": "mesh",
    "relatedId": "5bbd6da600bd75575fb32ca5"
  }
]
```

we issue an Asset Retrieval Message for each asset listed in the response: http

```
GET /v1/asset/key HTTP/1.1
Host: localhost:8080
```

curl

```
curl -i http://localhost:8080/v1/asset/key
```

Loading Scene Objects

We can issue an Object Query to pull the Objects in a Scene, and if all parameters are left out of the request, than all Objects in the scene will be returned: http

```
POST /v1/scene/scene-key/object/query HTTP/1.1
Host: localhost:8080
Content-Type: application/json

{
  "name": "Test Object 123464",
  "type": "Curve",
  "subtype": "Sphere",
  "owner": "456",
```

(continues on next page)

(continued from previous page)

```

"timestamp": 123456789,
"translation": [0, 0, 1],
"quaternion_rotation": [0, 1, 0, 0],
"euler_rotation": [0, 0, 0],
"scale": [1, 1, 2],
"assets": ["Asset_5"]
}

```

curl

```

curl -i -X POST http://localhost:8080/v1/scene/scene-key/object/query -H 'Content-
↳Type: application/json' --data-raw '{"assets": ["Asset_5"], "euler_rotation": [0, 0,
↳0], "name": "Test Object 123464", "owner": "456", "quaternion_rotation": [0, 1, 0,
↳0], "scale": [1, 1, 2], "subtype": "Sphere", "timestamp": 123456789, "translation":
↳[0, 0, 1], "type": "Curve"}'

```

response

```

HTTP/1.1 200 OK
Location: http://localhost:8080/v1/scene/scene-key/object/query
Content-Type: application/json

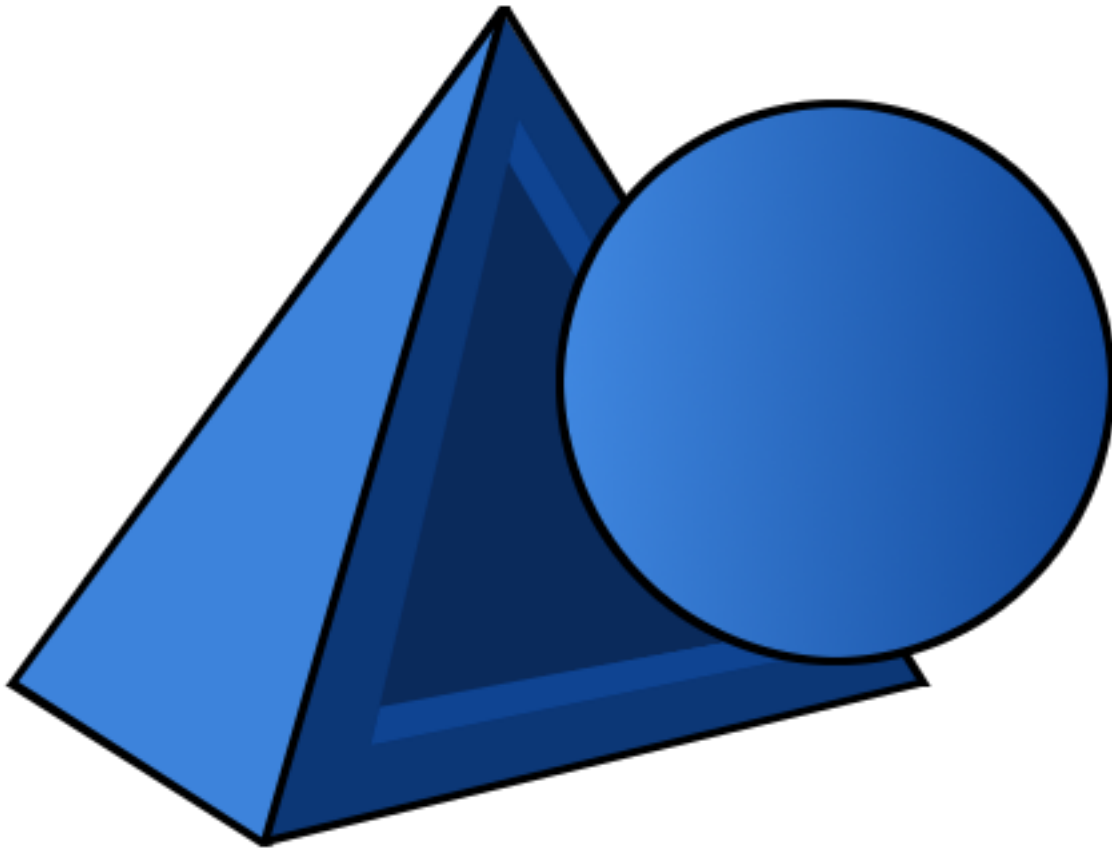
{
  "num_records":1,
  "objects":[
    {
      "key":"5951dd759af59c00015b1409",
      "name":"123",
      "scene":"DEFGHIJ123463",
      "type":"Mesh",
      "subtype":"Cube",
      "owner":"456",
      "transform":[1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1],
      "assets": ["Asset_5"]
    }
  ]
}

```

Loading Object Assets

Loading Object Assets follows exactly the same process as retrieving Scene Assets, only we query the Asset Relationship API with relationship-type=object, instead of 'scene'.

2.2 Aesel HTTP API



2.2.1 Login API

Log into Aesel with a valid username and password.

This will return values in the ‘Set-Cookie’ header, meant for use by browsers, as well as the ‘Authorization’ header, meant for use by external applications.

Login

POST /v1/login

Login to Aesel, retrieving the authentication token required for future requests.

Request Headers

- **Content-Type** – application/json

Status Codes

- **200 OK** – Success

http

```
POST /login HTTP/1.1
Host: localhost:8080
Content-Type: application/json

{
  "username": "aesel",
  "password": "guest"
}
```

curl

```
curl -i -X POST http://localhost:8080/login -H 'Content-Type: application/json' --
  ↳data-raw '{"password": "guest", "username": "aesel"}'
```

wget

```
wget -S -O- http://localhost:8080/login --header='Content-Type: application/json' --
  ↳post-data='{"password": "guest", "username": "aesel"}'
```

httpie

```
echo '{
  "password": "guest",
  "username": "aesel"
}' | http POST http://localhost:8080/login Content-Type:application/json
```

python-requests

```
requests.post('http://localhost:8080/login', headers={'Content-Type': 'application/
  ↳json'}, json={'password': 'guest', 'username': 'aesel'})
```

2.2.2 Users API

A user is a valid login for a secured application.

Users can also be labelled with ‘isAdmin’, which marks them as able to access back-end endpoints not required for standard users.

User Creation

POST /v1/users/sign-up

Create a new User. Please note that only administrator users can access this endpoint.

Request Headers

- Content-Type – application/json

Status Codes

- 200 OK – Success

http

```
POST /users/sign-up HTTP/1.1
Host: localhost:8080
Content-Type: application/json
```

(continues on next page)

(continued from previous page)

```
{
  "username": "aesel",
  "password": "guest",
  "email": "test@test.com",
  "isAdmin": false,
  "isActive": true,
  "favoriteProjects": [],
  "favoriteScenes": []
}
```

curl

```
curl -i -X POST http://localhost:8080/users/sign-up -H 'Content-Type: application/json'
↳ --data-raw '{"email": "test@test.com", "favoriteProjects": [], "favoriteScenes":
↳ [], "isActive": true, "isAdmin": false, "password": "guest", "username": "aesel"}'
```

User Retrieval

GET `/v1/users/` (*key*)

Get a User by ID.

Status Codes

- 200 OK – Success

http

```
GET /users/{key} HTTP/1.1
Host: localhost:8080
```

curl

```
curl -i 'http://localhost:8080/users/{key}'
```

response

```
HTTP/1.1 200 OK
Location: http://localhost:8080/users/{key}

{
  "id": "5c1aecd5728a474b669a880",
  "username": "demo2",
  "email": "test3@test.com",
  "isAdmin": false,
  "isActive": true,
  "favoriteProjects": [],
  "favoriteScenes": []
}
```

User Update

PUT `/v1/users/` (*key*)

Update an existing User's basic (String or number) attributes. This endpoint cannot update list attributes, including favorite projects and favorite scenes. These can be updated by the respective HTTP endpoints.

Request Headers

- **Content-Type** – application/json

Status Codes

- **200 OK** – Success

http

```
PUT /users/{key} HTTP/1.1
Host: localhost:8080
Content-Type: application/json

{
  "username": "guest2",
  "password": "guest2",
  "email": "test2@test.com",
  "isAdmin": false,
  "isActive": true,
  "favoriteProjects": [],
  "favoriteScenes": []
}
```

curl

```
curl -i -X PUT 'http://localhost:8080/users/{key}' -H 'Content-Type: application/json' \
  --data-raw '{"email": "test2@test.com", "favoriteProjects": [], "favoriteScenes": \
  [], "isActive": true, "isAdmin": false, "password": "guest2", "username": "guest2"}'
```

Add Favorite Project**PUT** /v1/users/ (key) /projects/*projectKey* Atomically add a Project Key to the favoriteProjects list of the user.**Request Headers**

- **Content-Type** – application/json

Status Codes

- **200 OK** – Success

http

```
PUT /users/{key}/projects/{projectKey} HTTP/1.1
Host: localhost:8080
```

curl

```
curl -i -X PUT 'http://localhost:8080/users/{key}/projects/{projectKey}'
```

Remove Favorite Project**DELETE** /v1/users/ (key) /projects/*projectKey* Atomically remove a Project Key from the favoriteProjects list of the user.**Request Headers**

- Content-Type – application/json

Status Codes

- 200 OK – Success

http

```
DELETE /users/{key}/projects/{projectKey} HTTP/1.1
Host: localhost:8080
```

curl

```
curl -i -X DELETE 'http://localhost:8080/users/{key}/projects/{projectKey}'
```

Add Favorite Scene

PUT /v1/users/ (*key*) /scenes/

sceneKey Atomically add a Scene Key to the favoriteScenes list of the user.

Request Headers

- Content-Type – application/json

Status Codes

- 200 OK – Success

http

```
PUT /users/{key}/scenes/{sceneKey} HTTP/1.1
Host: localhost:8080
```

curl

```
curl -i -X PUT 'http://localhost:8080/users/{key}/scenes/{sceneKey}'
```

Remove Favorite Scene

DELETE /v1/users/ (*key*) /scenes/

sceneKey Atomically remove a Scene Key from the favoriteScenes list of the user.

Request Headers

- Content-Type – application/json

Status Codes

- 200 OK – Success

http

```
DELETE /users/{key}/scenes/{sceneKey} HTTP/1.1
Host: localhost:8080
```

curl

```
curl -i -X DELETE 'http://localhost:8080/users/{key}/scenes/{sceneKey}'
```

Make User Admin

PUT `/v1/users/ (key) /admin`

Make a user an administrator.

Request Headers

- `Content-Type` – application/json

Status Codes

- `200 OK` – Success

http

```
PUT /users/{key}/admin HTTP/1.1
Host: localhost:8080
```

curl

```
curl -i -X PUT 'http://localhost:8080/users/{key}/admin'
```

Make User Non-Admin

DELETE `/v1/users/ (key) /admin`

Remove administrator access from a user.

Request Headers

- `Content-Type` – application/json

Status Codes

- `200 OK` – Success

http

```
DELETE /users/{key}/admin HTTP/1.1
Host: localhost:8080
```

curl

```
curl -i -X DELETE 'http://localhost:8080/users/{key}/admin'
```

Activate User

PUT `/v1/users/ (key) /active`

Activate a user.

Request Headers

- `Content-Type` – application/json

Status Codes

- `200 OK` – Success

http

```
PUT /users/{key}/active HTTP/1.1
Host: localhost:8080
```

curl

```
curl -i -X PUT 'http://localhost:8080/users/{key}/active'
```

Deactivate User

DELETE /v1/users/ (*key*) /active

Deactivate a user, revoking all access until they are reactivated.

Request Headers

- Content-Type – application/json

Status Codes

- 200 OK – Success

http

```
DELETE /users/{key}/active HTTP/1.1
Host: localhost:8080
```

curl

```
curl -i -X DELETE 'http://localhost:8080/users/{key}/active'
```

User Query

GET /v1/users/

Query for users by attribute.

Status Codes

- 200 OK – Success

http

```
GET /users?username=aesel HTTP/1.1
Host: localhost:8080
```

curl

```
curl -i 'http://localhost:8080/users?username=aesel'
```

response

```
HTTP/1.1 200 OK
Location: http://localhost:8080/users/{key}

{
  "id": "5c1aecd5728a474b669a880",
  "username": "demo2",
  "email": "test3@test.com",
  "isAdmin": false,
```

(continues on next page)

(continued from previous page)

```

    "isActive": true,
    "favoriteProjects": [],
    "favoriteScenes": []
  }

```

User Delete

DELETE `/v1/users/` (*key*)

Delete a user by ID.

Status Codes

- 200 OK – Success

http

```

DELETE /users/{key} HTTP/1.1
Host: localhost:8080

```

curl

```

curl -i -X DELETE 'http://localhost:8080/users/{key}'

```

2.2.3 Asset API

An asset is a resource that each device will need in order to accurately depict objects around it. This can vary, from an .obj file, containing mesh information, to a .glsl file, containing shader information, to a .png or .jpg file containing an image. Assets can either be associated to a Scene or an Object, or anything else by Asset Relationships.

A Scene Asset is a file that devices may need to utilize in order to synchronize views of scenes with other devices, such as a photograph of a specified object or marker. It can also include map/level mesh and shader information, for use with VR/traditional video games. This API allows for the storage and retrieval of such files.

An Object Asset is a file that is used to construct an Object. Entire collections of assets are expected to be downloaded for each object, and so registering to a scene is expected to incur a large set of downloads to collect all of the assets within that scene.

When authentication is active, assets are associated to the user that creates them and can be public or private. Users can only interact with assets that are either public, or that they own.

Asset Creation

POST `/v1/asset/`

Create a new asset from the File Data in the body of the request. If the 'related-id' and 'related-type' are also populated, then an Asset Relationship is created as well.

Query Parameters

- **content-type** (*string*) – Optional. The content type of the asset (ie. application/json).
- **file-type** (*string*) – Optional. The file type of the asset (ie. json).
- **related-id** (*string*) – Optional. Must appear with 'related-type'. Used to create a relationship to the specified object.

- **related-type** (*string*) – Optional. Must appear with ‘related-id’. Used to create a relationship of the specified type.
- **asset-type** (*string*) – Optional. Populated into the query-able Asset Metadata.
- **isPublic** (*boolean*) – Optional. Is the asset public or private

Request Headers

- **Content-Type** – multipart/*

Status Codes

- **200 OK** – Success

http

```
POST /v1/asset HTTP/1.1
Host: localhost:8080
Content-Type: multipart/form-data

"file=@testupload.txt"
```

curl

```
curl -i -X POST http://localhost:8080/v1/asset -H 'Content-Type: multipart/form-data' \
  --data-raw '"file=@testupload.txt"'
```

response

```
HTTP/1.1 200 OK
Location: http://localhost:8080/v1/asset
Content-Type: text/plain

abcdef
```

Asset Update

POST /v1/asset/{asset_key}

Update an existing Asset. This returns a new key for the asset, and adds an entry to the associated Asset History. This will also update all relationships which were associated to the old Asset, and associate them to the new Asset.

Query Parameters

- **content-type** (*string*) – Optional. The content type of the asset (ie. application/json).
- **file-type** (*string*) – Optional. The file type of the asset (ie. json).
- **asset-type** (*string*) – Optional. Populated into the query-able Asset Metadata.

Request Headers

- **Content-Type** – multipart/*

Status Codes

- **200 OK** – Success

http

```
POST /v1/asset/key HTTP/1.1
Host: localhost:8080
Content-Type: multipart/form-data

"file=@testupload.txt"
```

curl

```
curl -i -X POST http://localhost:8080/v1/asset/key -H 'Content-Type: multipart/form-
  ↪data' --data-raw '"file=@testupload.txt"'
```

response

```
HTTP/1.1 200 OK
Location: http://localhost:8080/v1/asset
Content-Type: text/plain

abcdef
```

Asset Retrieval

GET /v1/asset/ (*asset_key*)
Retrieve an asset by ID.

Status Codes

- 200 OK – Success

http

```
GET /v1/asset/key HTTP/1.1
Host: localhost:8080
```

curl

```
curl -i http://localhost:8080/v1/asset/key
```

Asset Count

GET /v1/asset/count
Count the total number of assets matching the given query.

Query Parameters

- **content-type** (*string*) – Optional. The content type of the asset (ie. application/json).
- **file-type** (*string*) – Optional. The file type of the asset (ie. json).
- **asset-type** (*string*) – Optional. Valid options are ‘standard’ (for normal assets), and ‘thumbnail’ for thumbnail assets.

Status Codes

- 200 OK – Success

http

```
GET /v1/asset/count?file-type=obj HTTP/1.1
Host: localhost:8080
```

curl

```
curl -i 'http://localhost:8080/v1/asset/count?file-type=obj'
```

Asset Metadata Query

GET /v1/asset

Query Asset Metadata based on various attributes.

Query Parameters

- **content-type** (*string*) – Optional. The content type of the asset (ie. application/json).
- **file-type** (*string*) – Optional. The file type of the asset (ie. json).
- **asset-type** (*string*) – Optional. Valid options are ‘standard’ (for normal assets), and ‘thumbnail’ for thumbnail assets.
- **limit** – Optional. The maximum number of records to return.
- **offset** – Optional. The number of records to skip, enabling pagination with the ‘limit’ parameter.

Status Codes

- 200 OK – Success

http

```
GET /v1/asset?file-type=obj HTTP/1.1
Host: localhost:8080
```

curl

```
curl -i 'http://localhost:8080/v1/asset?file-type=obj'
```

Asset Metadata Batch Query

POST /v1/bulk/asset

Get Asset Metadata by IDs in bulk.

Request Headers

- Content-Type – application/json

Status Codes

- 200 OK – Success

Asset Deletion

DELETE /v1/asset/ (*asset_key*)

Delete an asset.

Status Codes

- 200 OK – Success

http

```
DELETE /v1/asset/key HTTP/1.1
Host: localhost:8080
```

curl

```
curl -i -X DELETE http://localhost:8080/v1/asset/key
```

Asset History Retrieval

GET /v1/asset/ (*asset_key*)

An Asset History is a record of all the versions of a particular asset.

Status Codes

- 200 OK – Success

http

```
GET /v1/asset-history/key HTTP/1.1
Host: localhost:8080
```

curl

```
curl -i http://localhost:8080/v1/asset-history/key
```

response

```
HTTP/1.1 200 OK
Location: http://localhost:8080/v1/asset-history/key
Content-Type: application/json

[ {
  "id": "5ab878ca98afd7729e928dd5",
  "scene": "testScene",
  "object": "testObject",
  "asset": "5ab878c998afd7729e928dd3",
  "assetIds": [
    "5ab878c998afd7729e928dd3",
    "5ab878b098afd7729e928dd1"
  ]
} ]
```

2.2.4 Asset Relationship API

An Asset Relationship is a link between an asset and any other data entity which is identifiable by a unique ID. Each relationship contains an Asset ID and a Related ID, as well as a Relationship Type. These are used to model relationships with both external sources (such as Scenes and Objects), and between assets (such as having one Asset be the thumbnail of another).

Asset Relationship Save

PUT /v1/relationship

Create or update an Asset Relationship.

Query Parameters

- **asset** (*string*) – Optional. If this and ‘type’ are specified, then this will overwrite matching Relationships.
- **related** (*string*) – Optional. If this and ‘type’ are specified, then this will overwrite matching Relationships.
- **type** (*string*) – Optional. Must appear with ‘related’ or ‘asset’. The type of Relationship to override.

Request Headers

- **Content-Type** – application/json

Status Codes

- **200 OK** – Success

http

```
PUT /v1/relationship HTTP/1.1
Host: localhost:8080
Content-Type: application/json

{
  "assetId": "asset123",
  "assetSubId": "meshName",
  "relationshipType": "scene",
  "relationshipSubtype": "mesh",
  "relatedId": "scene123"
}
```

curl

```
curl -i -X PUT http://localhost:8080/v1/relationship -H 'Content-Type: application/
↪json' --data-raw '{"assetId": "asset123", "assetSubId": "meshName", "relatedId":
↪"scene123", "relationshipSubtype": "mesh", "relationshipType": "scene"}'
```

response

```
HTTP/1.1 200 OK
Location: http://localhost:8080/v1/relationship

[
  {
    "id": "5bbec73700bd755e5e2e9630",
    "assetId": "5bbd6ea100bd75575fb32ca8",
    "relationshipType": "scene",
    "relatedId": "123"
  }
]
```

Asset Relationship Deletion

DELETE /v1/relationship

Delete an Asset Relationship.

Query Parameters

- **asset** (*string*) – Required. The Asset ID of the Relationship to delete.
- **related** (*string*) – Required. The Related ID of the Relationship to delete.
- **type** (*string*) – Required. The type of Relationship to delete.

Status Codes

- 200 OK – Success

http

```
DELETE /v1/relationship?type=scene&related=123&asset=456 HTTP/1.1
Host: localhost:8080
```

curl

```
curl -i -X DELETE 'http://localhost:8080/v1/relationship?type=scene&related=123&
asset=456'
```

Asset Relationship Query

GET /v1/relationship

Find Asset Relationships based on one or more attributes.

Query Parameters

- **asset** (*string*) – Required. The Asset ID of the Relationship to find.
- **related** (*string*) – Required. The Related ID of the Relationship to find.
- **type** (*string*) – Required. The type of Relationship to find.

Status Codes

- 200 OK – Success

http

```
GET /v1/relationship?type=scene&related=123 HTTP/1.1
Host: localhost:8080
```

curl

```
curl -i 'http://localhost:8080/v1/relationship?type=scene&related=123'
```

response

```
HTTP/1.1 200 OK
Location: http://localhost:8080/v1/relationship?type=scene&related=123

[
  {
    "id": "5bbd6ea100bd75575fb32caa",
```

(continues on next page)

(continued from previous page)

```
    "assetId": "5bbd6ea100bd75575fb32ca8",
    "assetSubId": "meshName",
    "relationshipType": "object",
    "relationshipSubtype": "mesh",
    "relatedId": "5bbd6da600bd75575fb32ca5"
  }
]
```

2.2.5 Asset Collection API

An Asset Collection is a grouping of Assets. Collections are generally used for organizational purposes, to simplify browsing and searching of large asset libraries.

Collections are associated to Assets through Relationships, allowing Collections to stay up to date with the latest versions of each Asset.

When authentication is active, collections are associated to the user that creates them and can be public or private. Users can only interact with collections that are either public, or that they own.

Asset Collection Creation

POST /v1/collection

Create a new Asset Collection.

Request Headers

- **Content-Type** – application/json

Status Codes

- **200 OK** – Success

http

```
POST /v1/collection HTTP/1.1
Host: localhost:8080
Content-Type: application/json

{
  "name": "testCollection",
  "description": "This is a test",
  "category": "test",
  "public": true,
  "tags": ["test1"],
  "isPublic": true
}
```

curl

```
curl -i -X POST http://localhost:8080/v1/collection -H 'Content-Type: application/json' \
  --data-raw '{"category": "test", "description": "This is a test", "isPublic": true, "name": "testCollection", "public": true, "tags": ["test1"]}'
```

response

```

HTTP/1.1 200 OK
Location: http://localhost:8080/v1/collection

{
  "id": "5be8cb42f5eee933213a3982",
  "name": "testCollection",
  "description": "This is a test",
  "category": "test",
  "tags": [
    "test1"
  ]
}

```

Asset Collection Retrieval

GET /v1/collection/{key}

Get a Collection by ID.

Status Codes

- 200 OK – Success

http

```

GET /v1/collection/{key} HTTP/1.1
Host: localhost:8080

```

curl

```
curl -i 'http://localhost:8080/v1/collection/{key}'
```

Asset Collection Batch Retrieval

POST /v1/bulk/collection

Get Asset Collections by IDs in bulk.

Request Headers

- Content-Type – application/json

Status Codes

- 200 OK – Success

Asset Collection Update

POST /v1/collection/{key}

Create a new Asset Collection.

Request Headers

- Content-Type – application/json

Status Codes

- 200 OK – Success

http

```
POST /v1/collection/{key} HTTP/1.1
Host: localhost:8080
Content-Type: application/json

{
  "name": "testCollection",
  "description": "This is a test",
  "category": "test",
  "tags": ["test1"]
}
```

curl

```
curl -i -X POST 'http://localhost:8080/v1/collection/{key}' -H 'Content-Type: application/json' --data-raw '{"category": "test", "description": "This is a test", "name": "testCollection", "tags": ["test1"]}'
```

response

```
HTTP/1.1 200 OK
Location: http://localhost:8080/v1/collection

{
  "id": "5be8cb42f5eee933213a3982",
  "name": "testCollection",
  "description": "This is a test",
  "category": "test",
  "tags": [
    "test1"
  ]
}
```

Asset Collection Query

GET /v1/collection

Query for Collections by attribute.

Status Codes

- 200 OK – Success

http

```
GET /v1/collection?name=test&num_records=10&page=0 HTTP/1.1
Host: localhost:8080
```

curl

```
curl -i 'http://localhost:8080/v1/collection?name=test&num_records=10&page=0'
```

response

```
HTTP/1.1 200 OK
Location: http://localhost:8080/v1/collection?name=test&num_records=10&page=0

[
  {
```

(continues on next page)

(continued from previous page)

```

    "id": "5be8cb42f5eee933213a3982",
    "name": "test",
    "description": "This is another test",
    "category": "test2",
    "tags": [
        "test3"
    ]
}
]
```

Asset Collection Delete

DELETE `/v1/collection/{key}`

Delete a Collection by ID.

Status Codes

- 200 OK – Success

http

```
DELETE /v1/collection/{key} HTTP/1.1
Host: localhost:8080
```

curl

```
curl -i -X DELETE 'http://localhost:8080/v1/collection/{key}'
```

2.2.6 Scene API

A Scene is a group of Objects associated to a particular Latitude and Longitude. Examples of Scenes include levels in a video game, rooms in a house, and shots in a movie. This API exposes CRUD and Query operations for Scenes.

When authentication is active, scenes are associated to the user that creates them and can be public or private. Users can only interact with scenes that are either public, or that they own.

Scene Creation

PUT `/v1/scene/` (*key*)

Create a new scene with key *key*.

Request Headers

- Content-Type – Application/json

Status Codes

- 200 OK – Success

http

```
PUT /v1/scene/key HTTP/1.1
Host: localhost:5885
Content-Type: application/json
```

(continues on next page)

(continued from previous page)

```
{
  "scenes": [
    {
      "name": "testScene",
      "region": "US-MD",
      "latitude": 124,
      "longitude": 122,
      "assets": ["TestAsset10"],
      "tags": ["Testing2"],
      "public": true
    }
  ]
}
```

curl

```
curl -i -X PUT http://localhost:5885/v1/scene/key -H 'Content-Type: application/json' \
--data-raw '{"scenes": [{"name": "testScene", "tags": ["Testing2"], "region": "US-MD", "longitude": 122, "latitude": 124, "public": true, "assets": ["TestAsset10"]}]}'
```

response

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

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

Scene Update

POST /v1/scene/ (*key*)

Update an existing scene with key *key*. This executes a full overwrite of any fields present in the message, and triggers Object Change Streams.

Request Headers

- Content-Type – Application/json

Status Codes

- 200 OK – Success

http

```
POST /v1/scene/key HTTP/1.1
Host: localhost:5885
Content-Type: application/json

{
  "scenes": [
    {
      "name": "testScene",
      "region": "US-MD",
      "latitude": 124,
```

(continues on next page)

(continued from previous page)

```

    "longitude":122,
    "assets":["TestAsset10"],
    "tags":["Testing2"]
  }
]
}

```

curl

```

curl -i -X POST http://localhost:5885/v1/scene/key -H 'Content-Type: application/json' \
  --data-raw '{"scenes": [{"name": "testScene", "tags": ["Testing2"], "region": "US-MD", "longitude": 122, "latitude": 124, "assets": ["TestAsset10"]}]}'
```

response

```

HTTP/1.1 200 OK
Location: http://localhost:5885/v1/scene/key
Content-Type: application/json

{
  "num_records":1,
  "scenes":[{"key":"jklmnop"}]
}

```

Scene Retrieval

GET `/v1/scene/` (*key*)Retrieve scene information for scene *key*. This includes name, region, tags, latitude, and longitude.

Status Codes

- 200 OK – Success

http

```

GET /v1/scene/key HTTP/1.1
Host: localhost:5885

```

curl

```
curl -i http://localhost:5885/v1/scene/key
```

response

```

HTTP/1.1 200 OK
Location: http://localhost:5885/v1/scene/key
Content-Type: application/json

{
  "msg_type": 2,
  "err_code": 100,
  "num_records": 1,
  "start_record": 0,
  "scenes": [
    {
      "key": "123",

```

(continues on next page)

(continued from previous page)

```
    "name": "testScene",
    "region": "us-ga",
    "latitude": 100,
    "active": true,
    "longitude": 100,
    "distance": 0,
    "assets": [
        "asset1"
    ],
    "tags": [
        "tag1"
    ],
    "devices": []
  }
]
```

Scene Deletion

DELETE `/v1/scene/` (*key*)

Delete a scene.

CAUTION: This will delete all information associated to a scene, including all registered devices.

Status Codes

- 200 OK – Success

http

```
DELETE /v1/scene/key HTTP/1.1
Host: localhost:5885
```

curl

```
curl -i -X DELETE http://localhost:5885/v1/scene/key
```

Scene Query

POST `/v1/scene/query`

Find scenes by one or more attributes, including distance.

The fields ‘latitude’, ‘longitude’, and ‘distance’ should always appear together if present. The distance provided is taken in kilometers.

Multiple scenes can be passed in the `scene_list` attribute, and the return value will be the sum of the results from each query.

Request Headers

- Content-Type – Application/json

Status Codes

- 200 OK – Success

http

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

```
{
  "scenes": [
    {
      "name": "test",
      "region": "US-MD",
      "latitude": 124,
      "longitude": 122,
      "assets": ["TestAsset10"],
      "tags": ["Testing2"]
    }
  ]
}
```

curl

```
curl -i -X POST http://localhost:5885/v1/scene/query -H 'Content-Type: application/
↪json' --data-raw '{"scenes": [{"name": "test", "tags": ["Testing2"], "region": "US-
↪MD", "longitude": 122, "latitude": 124, "assets": ["TestAsset10"]}]]'
```

response

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

```
{
  "num_records": 1,
  "scenes": [
    {
      "key": "jklmnop",
      "name": "TestScene10",
      "region": "US-MD",
      "latitude": 124.0,
      "longitude": 122.0,
      "tags": ["test", "test2"],
      "asset_ids": ["asset1", "asset2"]
    }
  ]
}
```

2.2.7 Scene Registration API

Devices need to register/de-register to scenes as they move around in the world, and Aesel uses this information to determine what object updates to stream out to that device. This API allows for registration, de-registration, and synchronization of devices to scenes.

Scene Registration

POST /v1/scene/ (*key*) /register

Devices are expected to register to scenes as they move through space. This tells Aesel what objects that device needs to receive information on. If the specified scene is not present, then it will be created.

Request Headers

- `Content-Type` – `Application/json`

Status Codes

- `200 OK` – Success

http

```
POST /v1/scene/key/register HTTP/1.1
Host: localhost:5885
Content-Type: application/json

{
  "scenes": [
    {
      "key": "jklmnop",
      "devices": [
        {
          "key": "Ud132",
          "hostname": "localhost",
          "port": 4444,
          "connection_string": "127.0.0.1:4444",
          "transform": {
            "translation": [0, 0, 0],
            "rotation": [0, 0, 0]
          }
        }
      ]
    }
  ]
}
```

curl

```
curl -i -X POST http://localhost:5885/v1/scene/key/register -H 'Content-Type:
↪application/json' --data-raw '{"scenes": [{"devices": [{"connection_string": "127.0.
↪0.1:4444", "hostname": "localhost", "port": 4444, "key": "Ud132", "transform": {
↪"translation": [0, 0, 0], "rotation": [0, 0, 0]}]}, {"key": "jklmnop"}]}'
```

response

```
HTTP/1.1 200 OK
Location: http://localhost:5885/v1/scene/key/register
Content-Type: application/json

{
  "msg_type": 4,
  "err_code": 100,
  "num_records": 2,
  "scenes": [
    {
      "key": "20dd78a2-9224-11e8-b492-d850e6db3ad1",
      "active": true,
      "distance": 0,
      "assets": [],
      "tags": [],
      "devices": [
```

(continues on next page)

(continued from previous page)

```

        {
            "key": "12345",
            "transform": {
                "translation": [
                    0,
                    0,
                    0
                ],
                "rotation": [
                    0,
                    0,
                    0
                ]
            }
        }
    ],
    {
        "key": "123456",
        "active": true,
        "distance": 0,
        "assets": [],
        "tags": [],
        "devices": []
    }
]
}

```

Scene De-Registration

POST /v1/scene/(key)/deregister

Devices are expected to register to scenes as they move through space. This tells Aesel what objects that device needs to receive information on. De-Registration occurs after a device has left the scene and joined others, and is now ready to stop receiving updates on objects within the old scene.

Note that devices are expected to de-register only after registering with a new scene and performing any necessary corrections. This allows a network of transformations to be created, which can be used to pre-calculate those needed for future registrations.

Status Codes

- 200 OK – Success

http

```

POST /v1/scene/jklmnop/deregister HTTP/1.1
Host: localhost:8080
Content-Type: application/json

{
  "scenes": [
    {
      "key": "jklmnop",
      "devices": [
        {
          "key": "Ud132"
        }
      ]
    }
  ]
}

```

(continues on next page)

(continued from previous page)

```

    }
  ]
}
}

```

curl

```

curl -i -X POST http://localhost:8080/v1/scene/jklmnop/deregister -H 'Content-Type:
↪application/json' --data-raw '{"scenes": [{"devices": [{"key": "Ud132"}], "key":
↪"ijklmnop"}]}'

```

Scene Synchronization

PUT /v1/scene/ (*key*) /align

Aesel will not always be able to supply a device with an accurate transformation upon registering to a scene. In particular, this will happen when the device first registers to a scene with no prior registrations, as well as when the network of transformations is first being built and collected. In these cases, the Device will need to supply Aesel with a correction in order to correct the transformation.

Request Headers

- **Content-Type** – Application/json

Status Codes

- 200 OK – Success

http

```

POST /v1/scene/key/align HTTP/1.1
Host: localhost:5885
Content-Type: application/json

{
  "scenes": [
    {
      "key": "ijklmnop",
      "devices": [
        {
          "key": "Ud132",
          "transform": {
            "translation": [0, 0, 0],
            "rotation": [0, 0, 0]
          }
        }
      ]
    }
  ]
}

```

curl

```

curl -i -X POST http://localhost:5885/v1/scene/key/align -H 'Content-Type:
↪application/json' --data-raw '{"scenes": [{"devices": [{"transform": {"translation
↪": [0, 0, 0], "rotation": [0, 0, 0]}, "key": "Ud132"}], "key": "ijklmnop"}]}'

```

2.2.8 Project API

A Project contains groups of scenes, as well as Asset Collections. It is primarily used for organization, and helps manage a full-scale animation production.

When authentication is active, projects are associated to the user that creates them and can be public or private. Users can only interact with projects that are either public, or that they own.

Project Creation

POST /v1/project

Create a new Project.

Request Headers

- **Content-Type** – application/json

Status Codes

- **200 OK** – Success

http

```
POST /v1/project HTTP/1.1
Host: localhost:8080
Content-Type: application/json

{
  "name": "Test",
  "description": "This is a test",
  "category": "test",
  "tags": ["testTag"],
  "isPublic": true,
  "sceneGroups": [
    {
      "name": "testGroup",
      "description": "This is a test group",
      "category": "test",
      "scenes": ["1234"]
    }
  ],
  "assetCollectionIds": ["4321"]
}
```

curl

```
curl -i -X POST http://localhost:8080/v1/project -H 'Content-Type: application/json' -
↳-data-raw '{"assetCollectionIds": ["4321"], "category": "test", "description":
↳"This is a test", "isPublic": true, "name": "Test", "sceneGroups": [{"category":
↳"test", "scenes": ["1234"], "name": "testGroup", "description": "This is a test_
↳group"}], "tags": ["testTag"]}'
```

response

```
HTTP/1.1 200 OK
Location: http://localhost:8080/v1/project

{
```

(continues on next page)

(continued from previous page)

```
{
  "id": "5be8eeb4f5eee94951e553a9",
  "name": "Test",
  "description": "This is a test",
  "category": "test",
  "tags": [
    "testTag"
  ],
  "sceneGroups": [
    {
      "name": "testGroup",
      "description": "This is a test group",
      "category": "test",
      "scenes": [
        "1234"
      ]
    }
  ],
  "assetCollectionIds": [
    "4321"
  ]
}
```

Project Retrieval

GET `/v1/project/` (*key*)

Get a project by ID.

Status Codes

- 200 OK – Success

http

```
GET /v1/project/{key} HTTP/1.1
Host: localhost:8080
```

curl

```
curl -i 'http://localhost:8080/v1/project/{key}'
```

Project Update

POST `/v1/project/` (*key*)

Update the basic (String and numeric) attributes of an existing Project.

Request Headers

- Content-Type – application/json

Status Codes

- 200 OK – Success

http


```
POST /v1/project/{key} HTTP/1.1
Host: localhost:8080
Content-Type: application/json

{
  "name": "AnotherName",
  "description": "This is a second test",
  "category": "testing"
}
```

curl

```
curl -i -X POST 'http://localhost:8080/v1/project/{key}' -H 'Content-Type:
↪application/json' --data-raw '{"category": "testing", "description": "This is a
↪second test", "name": "AnotherName"}'
```

response

```
HTTP/1.1 200 OK
Location: http://localhost:8080/v1/project

{
  "id": "5be8eeb4f5eee94951e553a9",
  "name": "Test",
  "description": "This is a test",
  "category": "test",
  "tags": [
    "testTag"
  ],
  "sceneGroups": [
    {
      "name": "testGroup",
      "description": "This is a test group",
      "category": "test",
      "scenes": [
        "1234"
      ]
    }
  ],
  "assetCollectionIds": [
    "4321"
  ]
}
```

Add Scene Group

POST /v1/project/(key)/groups

Create a new Scene Group associated to the specified Project.

Request Headers

- **Content-Type** – application/json

Status Codes

- **200 OK** – Success

http

```
POST /v1/project/{key}/groups HTTP/1.1
Host: localhost:8080
Content-Type: application/json

{
  "name": "AnotherName",
  "description": "This is a second test",
  "category": "testing"
}
```

curl

```
curl -i -X POST 'http://localhost:8080/v1/project/{key}/groups' -H 'Content-Type:
↪application/json' --data-raw '{"category": "testing", "description": "This is a
↪second test", "name": "AnotherName"}'
```

Update Scene Group

POST /v1/project/ (key) /groups/{groupName}

Update the basic attributes (String and numeric) of an existing Scene Group associated to the specified Project.

Request Headers

- Content-Type – application/json

Status Codes

- 200 OK – Success

http

```
POST /v1/project/{key}/groups/{groupName} HTTP/1.1
Host: localhost:8080
Content-Type: application/json

{
  "name": "AnotherName",
  "description": "This is a second test",
  "category": "testing"
}
```

curl

```
curl -i -X POST 'http://localhost:8080/v1/project/{key}/groups/{groupName}' -H
↪'Content-Type: application/json' --data-raw '{"category": "testing", "description":
↪"This is a second test", "name": "AnotherName"}'
```

Add Scene to Scene Group

PUT /v1/project/ (key) /groups/{groupName}/scenes/{sceneKey}

Add a Scene by ID to an existing Scene Group, which is associated to the specified Project.

Status Codes

- 200 OK – Success

http

```
PUT /v1/project/{key}/groups/{groupName}/scenes/{sceneKey} HTTP/1.1
Host: localhost:8080
```

curl

```
curl -i -X PUT 'http://localhost:8080/v1/project/{key}/groups/{groupName}/scenes/
↳{sceneKey}'
```

Remove Scene from Scene Group

DELETE /v1/project/ (*key*) /groups/{groupName}/scenes/{sceneKey}

Remove a Scene by ID from an existing Scene Group, which is associated to the specified Project.

Status Codes

- 200 OK – Success

http

```
DELETE /v1/project/{key}/groups/{groupName}/scenes/{sceneKey} HTTP/1.1
Host: localhost:8080
```

curl

```
curl -i -X DELETE 'http://localhost:8080/v1/project/{key}/groups/{groupName}/scenes/
↳{sceneKey}'
```

Remove Scene Group

DELETE /v1/project/ (*key*) /groups/{groupName}

Remove an existing Scene Group from the specified Project.

Status Codes

- 200 OK – Success

http

```
DELETE /v1/project/{key}/groups/{groupName} HTTP/1.1
Host: localhost:8080
```

curl

```
curl -i -X DELETE 'http://localhost:8080/v1/project/{key}/groups/{groupName}'
```

Project Query

GET /v1/project

Query for projects by attribute.

Status Codes

- 200 OK – Success

http

```
GET /v1/project?name=test&num_records=10&page=0 HTTP/1.1
Host: localhost:8080
```

curl

```
curl -i 'http://localhost:8080/v1/project?name=test&num_records=10&page=0'
```

response

```
HTTP/1.1 200 OK
Location: http://localhost:8080/v1/project?name=AnotherName&num_records=10&page=0

[
  {
    "id": "5be8eeb4f5eee94951e553a9",
    "name": "AnotherName",
    "description": "This is a second test",
    "category": "testing",
    "tags": [
      "testTag2"
    ],
    "sceneGroups": [
      {
        "name": "testGroup2",
        "description": "This is another test group",
        "category": "testing",
        "scenes": [
          "12345"
        ]
      }
    ],
    "assetCollectionIds": [
      "43212"
    ]
  }
]
```

Project Delete

DELETE /v1/project/ (*key*)
Delete a project by ID.

Status Codes

- 200 OK – Success

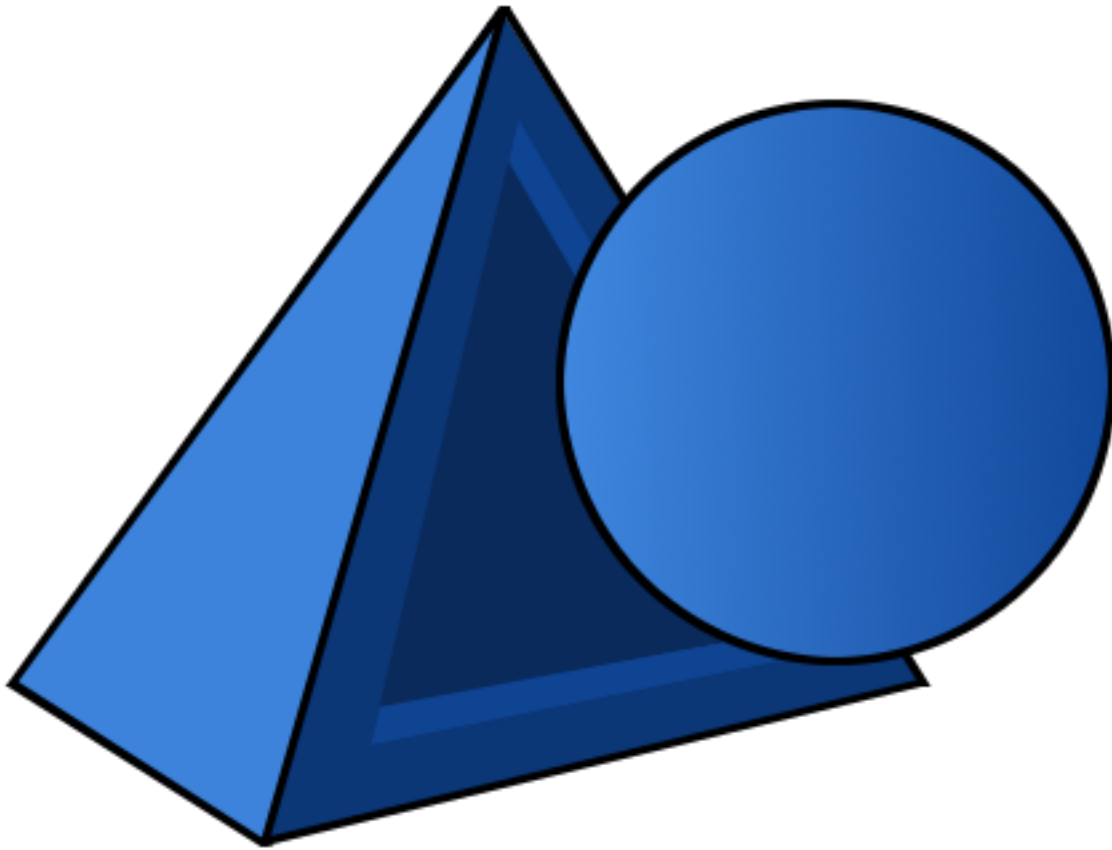
http

```
DELETE /v1/project/{key} HTTP/1.1
Host: localhost:8080
```

curl

```
curl -i -X DELETE 'http://localhost:8080/v1/project/{key}'
```

2.3 Design Documents



2.3.1 Aesel Design

Section 1: Abstract

This document outlines the design for Phase 1 of the Distributed Visualization Server. This is a system for allowing many different user devices to visualize the same 3-Dimensional objects in real-time. While additional workflows are planned, they will not be outlined within this document.

This consists of a number of different components, so this document will focus on integration-driven flows. The document will proceed on a workflow-by-workflow basis and provide the basis for a solid API design.

Workflows

The following workflows will be outlined in the below document:

- Device-Scene Registration & Exit
- Device Scene Transfer
- Device-Scene Coordinate Synchronization

- Scene-Scene Coordinate Synchronization
- Object Creation & Destruction
- Object Update (Location/Rotation/Scale)
- Object Update (Assets)
- Object Retrieval
- Object Lock & Unlock

Components

The following custom components will be utilized within Aesel:

Type	Name	Description
Gateway	Adres-tia	Gateway for Client communications to be translated into Protocol Buffer or JSON Messages
Data Storage	Mongo	Storage of text-based data to hold Mesh data, Shader data, etc
Transformation Updates	Clyman	Storage for Object Transformations with real-time change feeds
Scene Data Storage	Neo4j	Stores Scene information, and is the primary database for CrazyIvan
Scenes & Coordinate Systems	CrazyIvan	Tracks sets of objects & user devices which form scenes to know what to load. Tracks coordinate systems to resolve the differences between devices.
Asset Management	Kelona	Manages Assets and tracks different versions of Assets over time.

Vocabulary

Term	Description
User Device	A Unique Hardware device utilized by an end-user to render the objects received from Aesel
Object	An Object in 3-space that is being rendered and/or interacted with by user devices
Scene	A logical grouping of objects which devices can register to. Each scene is associated to a particular coordinate system
Coordinate System	A set of 3 Axis (X, Y, & Z), as well as an origin. Coordinate Systems are stored in relative terms, using matrix transformations to move from one to another.
Transaction	A single interaction with Aesel by a client through the HTTP interface. Strong consistency and atomicity are guaranteed, with slower performance profiles than events. Confirmation is received synchronously.
Event	A single interaction with Aesel by a client through the UDP interface. Weak consistency and atomicity is not guaranteed, with much faster performance profiles than transactions. Confirmation is received asynchronously.
Scene Cluster	A cluster consisting of one or more instances (and/or clusters) of CLyman, Crazy Ivan, Mongo, and Consul Agents. Responsible for serving up Object information for one or more particular scenes.

Section 3: Individual Workflows

Device-Scene Registration

Device-Scene Registration occurs when a User Device either:

- First joins the overall network
- Changes scenes within the network

Input

The following are necessary inputs for this workflow:

- Device Location – Current Latitude/Longitude of the device.
- Group Name – The name of the device group to be joined
- Device ID – If we are changing scenes, then we include a device ID.

Processing

Upon receiving the request from the client, we send a message to the Scene module requesting registration. When registration is successful, we return the Scene ID and the Obj3 ID's to be pulled.

Output

The following are the outputs for this workflow:

- Device ID – If we are registering within the network for the first time, the device ID is returned as an output
- Scene ID – The Unique Identifier for the scene the device is now registered to
- Coordinate System – The Coordinate System information of the new scene (transformation from scene coordinate system to device coordinate system)
- Object ID's – Unique Identifiers for the Objects to be retrieved by the device

Device Scene Transfer

A Scene Transfer occurs when an object is removed from it's current scene, and is registered to a new scene.

Input

The following are necessary inputs for this workflow:

- Initial Scene
- New Scene

Processing

If a calculable coordinate system transform is available, then it is returned to the device. Otherwise, it is approximated and a flag is returned notifying the user device that Scene-Scene coordinate synchronization is needed.

Output

The following are the outputs for this workflow:

- Coordinate System – The Coordinate System information of the new scene (transformation from old scene to new)

Device-Scene Coordinate Synchronization

Device-Coordinate Synchronization occurs when a User Device wishes to re-calibrate it's current scene transformation relative to it's local coordinate system.

Input

The following are necessary inputs for this workflow:

- Coordinate System – A Transformation from the scene coordinate system to the user device local coordinate system

Processing

This message is used to dial-in scene transformations. Upon receiving the request from the client, we send a message to the Scene module and pass back the confirmation.

Output

Confirmation or Error

Scene-Scene Coordinate Synchronization

Scene-Scene Coordinate Synchronization occurs when a User Device wishes to re-calibrate it's current scene transformation relative to it's previous scene.

Input

The following are necessary inputs for this workflow:

- Coordinate System – A Transformation from the scene coordinate system to the user device local coordinate system

Processing

This message is used to dial-in scene-scene transformations. Upon receiving the request from the client, we send a message to the Scene module and pass back the confirmation.

Output

Confirmation or Error

Device-Scene Exit

Device-Scene Exit occurs whenever a user device leaves a scene

Input

The following are the inputs for this workflow:

- Device ID – The Unique Identifier for the device
- Scene ID – The Unique Identifier for the scene the device is currently registered to

Processing

Upon receiving the request from the client, we send a message to the Scene module requesting de-registration. When de-registration is successful, we return the confirmation.

Output

Confirmation or Error

Object Creation

Object Creation is the act of making a new Object across all User Devices.

Input

- Object Information
- Mesh Information
- Any other information (shader, etc)
- Scene ID

Processing

Upon receiving the request from the client, a message is sent to the Scene Module with all the information. The object is added to the scene within the Scene Module. Then, a message is sent to Ceph to save the Mesh, Shader, and other information. Then, the resulting keys are added and the rest of the information is saved to Clyman, which generates an outbound message on the response streams in the gateway.

Output

- Outbound message to User Devices of newly created object and all associated assets that need to be downloaded
- Confirmation or Error

Object Destruction

Object Destruction is called to remove an Object from all User Devices

Input

- Object ID – The Unique Identifier for the Object (from CLyman)

Processing

Upon receiving the request from the client, a message is sent to the Scene Module with all the information. The object is removed from the scene within the Scene Module. Next, the Asset ID's are retrieved from Clyman. Then, a message is sent to Ceph to remove the Mesh, Shader, and other information. Then, the rest of the information is removed from Clyman, which generates an outbound message on the response streams in the gateway.

Output

- Outbound message to User Devices of newly deleted object and all associated assets that need to be removed
- Confirmation or Error

Object Update (Location/Rotation/Scale)

An Object Update, or Transformation, is a special flow in that it only hits Clyman. These are designed for high-speed processing, and generate outbound messages to update all devices of the changes.

Input

- Object ID – The Unique Identifier for the Object (from Clyman)
- Transformation – The actual transformation to be applied (Translation, Rotation, Scale)

Processing

Upon receiving the request from the client, a message is sent to Clyman to apply the transformations specified. This generates an outbound message on the response streams in the gateway.

Output

- Outbound message to User Devices of newly created object and all associated assets that need to be downloaded
- Confirmation or Error

Object Update (Assets)

Object Asset Update is called to edit the assets that make up an Object, and push the update to all User Devices.

Input

- Object ID – The Unique Identifier for the Object
- Asset Information – The new information to use for the asset

Processing

Upon receiving the request from the client, a message is sent to Ceph to overwrite the data there. Then, a mesh update message is sent to Clyman, which generates an outbound message on the response streams in the gateway.

Output

- Outbound message to User Devices of newly created object and all associated assets that need to be downloaded
- Confirmation or Error

Object Retrieval

Object Retrieval comes when a Device needs to load the assets and transform information for an Object.

Input

- Object ID – The Unique Identifier for the Object

Processing

Upon receiving the request from the client, a message is sent to Clyman to retrieve the Transformation information and Asset ID's. Then, the Asset ID's are used in messages to Ceph to retrieve the asset information. All of this information is assembled and passed back to the Device.

Output

- Transformation Information – The current location, rotation, scaling for the Object
- Asset Information – The assets needed to re-create the object

Object Lock

A User Device Lock is used to prevent other User Devices from updating an Object, until it is released.

Input

- Object ID – The Unique Identifier for the Object
- Device ID – The Unique Identifier for the device

Processing

Upon receiving the request from the client, a message is sent to Clyman to establish the transformation lock.

Output

- Confirmation or Error

Object Unlock

Releasing a User Device Lock allows other Devices to establish locks or update the Object.

Input

- Object ID – The Unique Identifier for the Object
- Device ID – The Unique Identifier for the device

Processing

Upon receiving the request from the client, a message is sent to Clyman to release the transformation lock.

Output

- Confirmation or Error

Section 4: Configuration Options

Several Configuration Options will be available within the DVS Server, each with different objectives.

Transaction ID

Each Service will either accept or generate a transaction ID, such that any individual transaction with a User Device can be traced through each service that it hits.

2.3.2 Aesel High Speed Streaming Design

Section 1: Abstract

This document outlines the design for Phase 2 of the Distributed Visualization Server. The primary focus of this component is one thing and one thing only: speed. Updates on object location, rotation, and scaling need to be streamed out to devices with absolute minimal overhead, and must meet a minimum of 24 updates against a single object per second. A more realistic goal would be upwards of 40 updates against a single object per second.

Workflows

The following workflows will be outlined in the below document:

- Object Update High Speed Input
- Object Update Output Streams

Components

The following custom components will be utilized within Aesel:

Type	Name	Description
Gateway	Adres-tia	Gateway for Client communications to be translated into Protocol Buffer or JSON Messages
Data Storage	Mongo	Storage of text-based data to hold Mesh data, Shader data, etc
Transformation Updates	Clyman	Storage for Object Transformations with real-time change feeds
Scene Data Storage	Neo4j	Stores Scene information, and is the primary database for CrazyIvan
Scenes & Coordinate Systems	CrazyIvan	Tracks sets of objects & user devices which form scenes to know what to load. Tracks coordinate systems to resolve the differences between devices.
Asset Management	Kelona	Manages Assets and tracks different versions of Assets over time.

Section 2: Vocabulary

Vocabulary

Term	Description
User Device	A Unique Hardware device utilized by an end-user to render the objects received from Aesel
Object	An Object in 3-space that is being rendered and/or interacted with by user devices
Scene	A logical grouping of objects which devices can register to. Each scene is associated to a particular coordinate system
Coordinate System	A set of 3 Axis (X, Y, & Z), as well as an origin. Coordinate Systems are stored in relative terms, using matrix transformations to move from one to another.
Transaction	A single interaction with Aesel by a client through the HTTP interface. Strong consistency and atomicity are guaranteed, with slower performance profiles than events. Confirmation is received synchronously.
Event	A single interaction with Aesel by a client through the UDP interface. Weak consistency and atomicity is not guaranteed, with much faster performance profiles than transactions. Confirmation is received asynchronously.
Scene Cluster	A cluster consisting of one or more instances (and/or clusters) of CLyman, Crazy Ivan, Mongo, and Consul Agents. Responsible for serving up Object information for one or more particular scenes.

Section 3: Workflow Overviews

Object Update High Speed Input

The high speed input exposure will allow devices registered to a scene to send in an update to object location, rotation, and scaling with minimal overhead and handling. Upon receipt of the message, Aesel should initiate it's Object Update Output Streams, as described in the next section.

Input

The high speed input will accept only an update on location, rotation, or scaling against an object in a particular scene. No other object attributes can be updated this way.

Processing

The high speed input should both initiate the Object Update Output Streams, as well as persist the update into CLyman.

Output

The high speed input should have little to no output, with a fire-and-forget methodology.

Object Update Output Streams

The Object Update Output Streams will push any object update made to location, rotation, or scaling out to all the devices registered to that object's Scene. This will be a joint effort between CLyman, Crazy Ivan, and an intermediary queue

Input

The output streams will be initiated by the high speed input workflow, and so do not require any explicit user input directly.

Processing

After being initiated by the high speed input flow, the following workflow should occur:

- CLyman places an update onto the intermediary queue
- Crazy Ivan picks update off of intermediary queue
- Crazy Ivan identifies other devices registered to same scene

Output

The output streams will send a UDP message containing the updated object state from Crazy Ivan to all devices registered to the object's scene.

Section 4: Required Changes

Object Update High Speed Input

- CLyman needs a new message type which allows for an update without first retrieving the object from Mongo. This means the user needs to send in the current position of the object, not the change from the last position.
- Adrestia needs a new API exposure which allows for an update directly against the object key. We only communicate with CLyman when this is called, and we do so asynchronously.
- Start Outbound Change Feeds (Documented below) asynchronously upon receipt of message

Object Update Output Streams

- Add support and dependencies for Apache Kafka (Intermediate Queue) into both CLyman and Crazy Ivan (<https://github.com/mfontanini/cppkafka>)
- Add Kafka instance to Docker file(s) and Travis CI for CLyman, Crazy Ivan, Adrestia (<https://hub.docker.com/r/spotify/kafka/>)
- Add support and dependencies for UDP transmission into Crazy Ivan (http://www.boost.org/doc/libs/1_62_0/doc/html/boost_asio/overview/networking/protocols.html)
- Add Change Stream Logic to CLyman, upon update produce a new message with the new object attributes and send to Kafka
- Add Change Stream Logic to Crazy Ivan, continually monitor Kafka cluster for new messages. Upon receiving, send UDP messages to all registered devices.

2.3.3 Aesel Zero Latency Streaming Design

Section 1: Abstract

This document outlines the design for Phase 3 of the Distributed Visualization Server. This effort has multiple focuses:

1. Allow for deployment configurations which support zero-latency event processing.
2. Allow end-to-end HTTPS encryption of transactions.
3. Enable transaction network to utilize existing load balancers.

One of the primary benefits that Aesel provides is as a back-end server that can be deployed across cloud networks, with enough speed to power a traditional multi-player video game. A single-server deployment should be capable of meeting the performance requirements, but not scalability. However, due to the expected network latencies present within the network of a cloud provider, we are not able to guarantee that such a deployment will meet the performance requirements. We need to support a cloud-native, zero-latency solution.

As a part of this enhancement, we will be introducing several new concepts, outlined in the Vocabulary section below.

Section 2: Vocabulary

Vocabulary

Term	Description
User Device	A Unique Hardware device utilized by an end-user to render the objects received from Aesel
Object	An Object in 3-space that is being rendered and/or interacted with by user devices
Scene	A logical grouping of objects which devices can register to. Each scene is associated to a particular coordinate system
Coordinate System	A set of 3 Axis (X, Y, & Z), as well as an origin. Coordinate Systems are stored in relative terms, using matrix transformations to move from one to another.
Transaction	A single interaction with Aesel by a client through the HTTP interface. Strong consistency and atomicity are guaranteed, with slower performance profiles than events. Confirmation is received synchronously.
Event	A single interaction with Aesel by a client through the UDP interface. Weak consistency and atomicity is not guaranteed, with much faster performance profiles than transactions. Confirmation is received asynchronously.
Scene Cluster	A cluster consisting of one or more instances (and/or clusters) of CLyman, Crazy Ivan, Mongo, and Consul Agents. Responsible for serving up Object information for one or more particular scenes.

The following components are utilized within phase 3 of development:

Type	Name	Description
Cluster Manager	Adrestia	Tracks available Scene Clusters, deciding which instance manages which Scene(s). Handles routing of requests for Objects and Scene Registrations.
Data Storage	Mongo	Storage of text-based data to hold Mesh data, Shader data, etc
Transformation Updates	Clyman	Storage for Object Transformations with real-time change feeds
Scenes & Coordinate Systems	CrazyIvan	Tracks sets of objects & user devices which form scenes to know what to load. Tracks coordinate systems to resolve the differences between devices.

Section 3: Events & Transactions

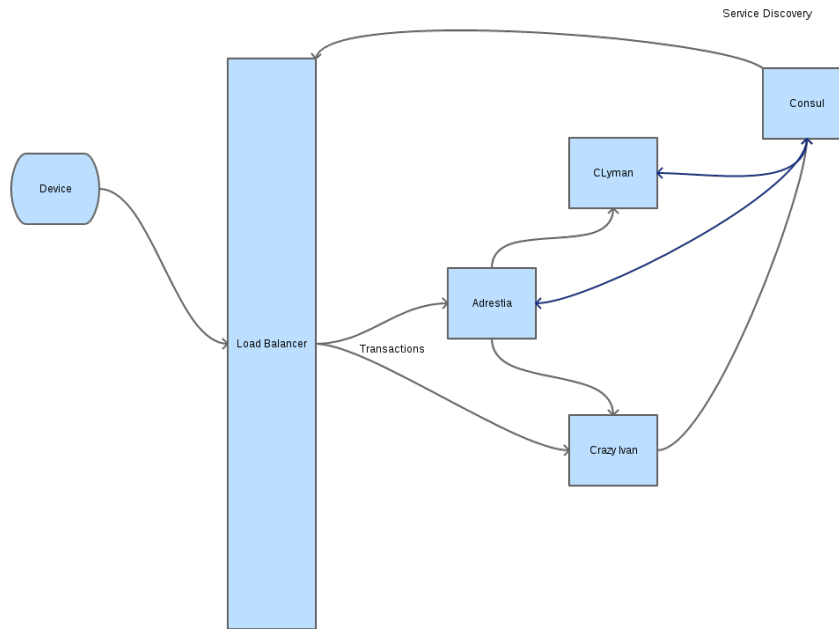
Events and Transactions are the means by which clients make changes to the state of an Aesel Scene. Generally, events are used to update an active scene (ie. live updates during games or simulations), while transactions are used for inactive scenes. This is not, however, a hard rule.

The difference between events and transactions comes down to one of consistency vs speed. In cases where speed is critical (such as live updates during games/simulations), events become the preferred method of communication. In other cases, transactions offer stronger consistency guarantees at the cost of more time spent making the request. Each client is expected to make use of both APIs, at various times throughout their lifecycle.

Section 3.1: Transactions

A transaction generally passes through an external load balancer, and is routed to either the Crazy Ivan or Adrestia. Scene API requests are passed directly to Crazy Ivan. In the case of Object updates, Adrestia forwards the message on to the correct instance of CLyman.

In the case of Scene Registration messages, Adrestia does the same but may need to decide which cluster is acting as a primary for the given scene. Either way, it then responds with the address of the cluster to which a device has been assigned. The device can use this interface to send in events to the cluster.



Section 3.2: Events

An event is sent directly from the client to a CLyman instance, via an exposed UDP port (more details in section 4). Events are currently limited to a simple Object update, overwriting the transformation, location, and scale of the object. Future development work will expand on the Event API to provide configurable event types and event stream logic.

Because we are only supporting single-object events in the implementation, we do not have to worry about cross-scene and cross-cluster events until the next phase of development.

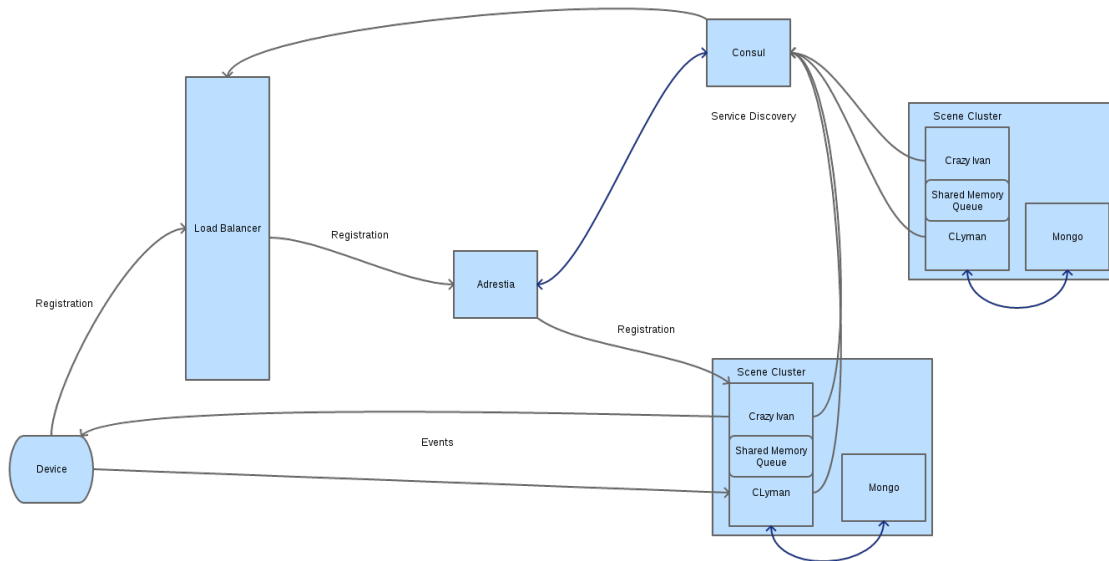
Section 4: Zero-Latency Deployments

Zero-latency means that there can be submillisecond network latency incurred from receipt of an Object Update (as detailed in the Aesel High Speed Streaming Design), to output of the Change Stream to other registered devices.

To accomplish this, CLyman and CrazyIvan can be deployed with several types of event streams: in memory, udp, and kafka. These strategies vary in terms of speed vs reliability.

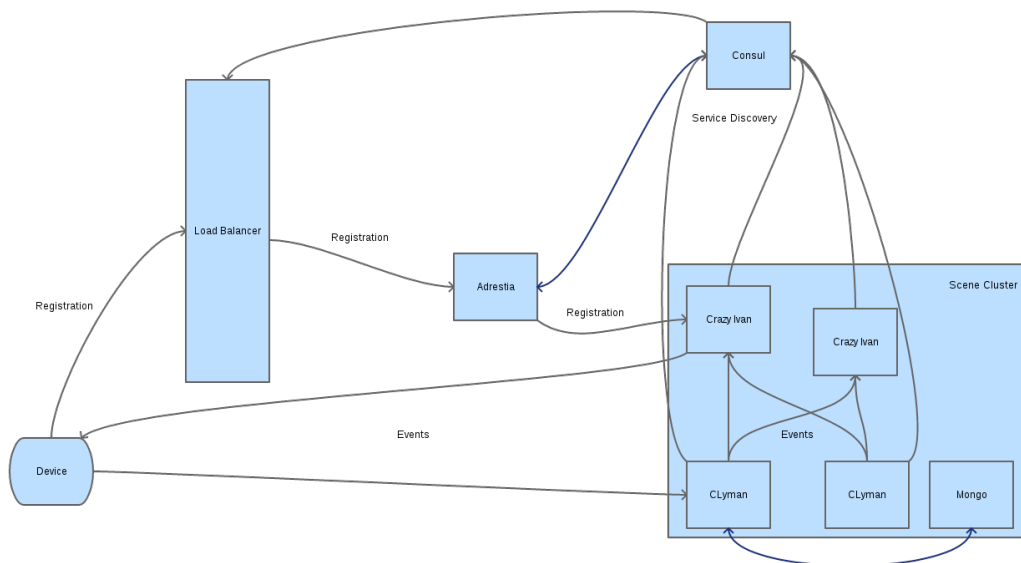
Section 4.1: In-Memory Streaming

In-memory streaming involves a block of shared memory for CLyman and Crazy Ivan, that is used as a queue. This is the fastest method of deployment, but also the least flexible. Each physical server should have one pair of CLyman and Crazy Ivan running, and each pair can manage separate scenes.



Section 4.2: UDP Streaming

UDP Streaming involves CLyman instances sending UDP messages to Crazy Ivan instances within the same cluster. This provides a good mix of speed and scalability, as CLyman and Crazy Ivan can be scaled horizontally within a cluster.



Section 4.3: In-Memory Caching

In-memory caching can be used to prevent repeated network calls within Crazy Ivan

- On receipt of kafka update, add scene to time-expired cache of scene ID's
- Check Scene-Device cache (seperate, time-expired cache of scene ID's and device info) for scene
- If not present, execute query & populate. If configured to still send updates, do so
- If present, send update
- On background thread, periodically query Neo4j and populate Scene-Device cache
- Crazy Ivan should always keep in memory those scenes in it's region (per region-specific server pairs)

Section 4.4: CLyman UDP Endpoint

CLyman needs to be able to accept Object Overwrite messages via UDP, mimicking the UDP API exposed by Adrestia.

Section 4.5: Region-Specific Server Pairs

Clyman/CrazyIvan pairs will need to be able to register as pairs with Consul, making them distinct enough to register the pairing within Adrestia. They will also need to register to the specified region.

Adrestia will need to provide the address of particular pairs of CLyman/CrazyIvan to devices upon registration. The devices can then send UDP updates directly to CLyman. Adrestia will also need logic to provide alternate server pairs upon request when devices detect that the server is down.

Section 5: Required Changes

- Update CLyman and CrazyIvan to use HTTP(S) interfaces rather than ZMQ
- Add UDP Endpoint to CLyman
- Add configurable options for event stream in CLyman and Crazy Ivan (Kafka, UDP, Shared Memory)
- Add logic in Adrestia to maintain mapping of Scenes-clusters
- Add configurable logic in Adrestia to pass UDP endpoints to devices
- Add Scene-Device Caching logic in CrazyIvan so no DB calls need to be made
- Logic in Adrestia to provide an updated UDP endpoint to devices on failure

2.3.4 Aesel Collaborative Animation Clients

Section 1: Abstract

This document outlines the design for Collaborative Animation clients which utilize Aesel. This is intended as a functional design, and will not delve too far into technical details as it is intended to be implemented as an add-on for multiple 3rd party tools, such as Blender or Maya.

Workflows

The following workflows will be outlined in the below document:

- Administration & Project Management
- Scene Discovery and Management
- Asset Discovery and Management
- Object Management & Replication
- Property Management & Replication
- Keyframing
- Animation Graph Handle Replication

Components

The following custom components will be utilized within Aesel:

Type	Name	Description
Gateway	Adres-tia	Gateway for Client communications to be translated into Protocol Buffer or JSON Messages
Data Storage	Mongo	Storage of text-based data to hold Mesh data, Shader data, etc
Transformation Updates	Clyman	Storage for Object Transformations with real-time change feeds
Scene Data Storage	Neo4j	Stores Scene information, and is the primary database for CrazyIvan
Scenes & Coordinate Systems	CrazyIvan	Tracks sets of objects & user devices which form scenes to know what to load. Tracks coordinate systems to resolve the differences between devices.
Asset Management	Kelona	Manages Assets and tracks different versions of Assets over time.

Section 2: Vocabulary

Vocabulary

Term	Description
User Device	A Unique Hardware device utilized by an end-user to render the objects received from Aesel
Object	An Object in 3-space that is being rendered and/or interacted with by user devices
Scene	A logical grouping of objects which devices can register to. Each scene is associated to a particular coordinate system
Coordinate System	A set of 3 Axis (X, Y, & Z), as well as an origin. Coordinate Systems are stored in relative terms, using matrix transformations to move from one to another.
Transaction	A single interaction with Aesel by a client through the HTTP interface. Strong consistency and atomicity are guaranteed, with slower performance profiles than events. Confirmation is received synchronously.
Event	A single interaction with Aesel by a client through the UDP interface. Weak consistency and atomicity is not guaranteed, with much faster performance profiles than transactions. Confirmation is received asynchronously.
Scene Cluster	A cluster consisting of one or more instances (and/or clusters) of CLyman, Crazy Ivan, Mongo, and Consul Agents. Responsible for serving up Object information for one or more particular scenes.

Section 3: Administration & Project Management

Administration & Project Management (Web UI) includes:

- Projects, which contain collections of scenes
- Integrations: CI/render-farms/project-management/email/slack/etc
- Remotely setup developer nodes

Section 4: Scene Discovery and Management

When discussing Collaborative Animation, a Scene is interpreted to mean a single ‘shot’ within the final animation.

Scene Discovery and Management then boils down to managing the shots within the animation. The Animation Client will provide several interfaces for managing these shots:

- List - List view (preferably with thumbnails)
- Detail - Detail view of the scene, to view and update scene details
- Filter - Ability to filter and query for desired scenes in the list view

Section 5: Asset Discovery and Management

Assets can come in many forms, from models to shaders to animations. Each asset is stored as a file, meaning that it’s filename extension is kept intact to tell the difference between these asset types.

Users can browser Asset Collections and select assets to import directly into a scene. Users can filter and sort the assets, as well as view thumbnails for each one.

This primarily takes the form of a list view, with the focus on thumbnails of each asset. If no scene has been loaded into the Animation Client, then assets can be imported in order to update them. Once they have been updated, they can be re-uploaded as an update to the original, which will update all corresponding scenes.

Section 6: Object Management & Replication

When a Scene is loaded, all of the objects from the scene should be loaded as well. From that point forward, objects in a scene are generally managed by normal user actions within the user interface of the 3rd party animation software. Creating a new object, however, is done via a button/hotkey/etc. Users are expected to spend some time setting up their object assets prior to saving the object for the first time, when those assets are replicated to live devices.

Object-level locking is a feature which ensures that animators do not overwrite each other's changes. The client will require a successful lock response prior to updating an object. Locked object will appear significantly visually distinct in the viewport, and will require user action (via button, hotkey, etc) to release the locks.

Object Replication

Users listen via UDP for updates to objects in real-time after registering to a scene. Users can also choose to have updates sent periodically of all locked objects in the scene. In this way, multiple users can work on multiple objects, within the same scene, all at the same time.

In addition to processing updates, clients are also expected to process object creation and delete messages. In each case, additional asset downloads may be required.

Section 7: Property Management & Replication

Properties are utilized to store a variety of non-object related values, which are somewhat dependent on the system in question. Properties can also belong to objects, and can model anything from shape keys and drivers to rig elements.

Properties are stored as one, two, or three double values, and are replicated in the same fashion as objects, but generally do not require an asset download.

Section 8: Keyframing

Keyframes in animation software are critical points within a shot, from which the rest of the animation is derived. Clients will store a base Object with a keyframe equal to -100, and all operations will default to this until a keyframe is entered. Upon creating each keyframe for an object, a new entry is made with a reference to the parent, and a new keyframe value. For each keyframe, a new transformation matrix can be stored.

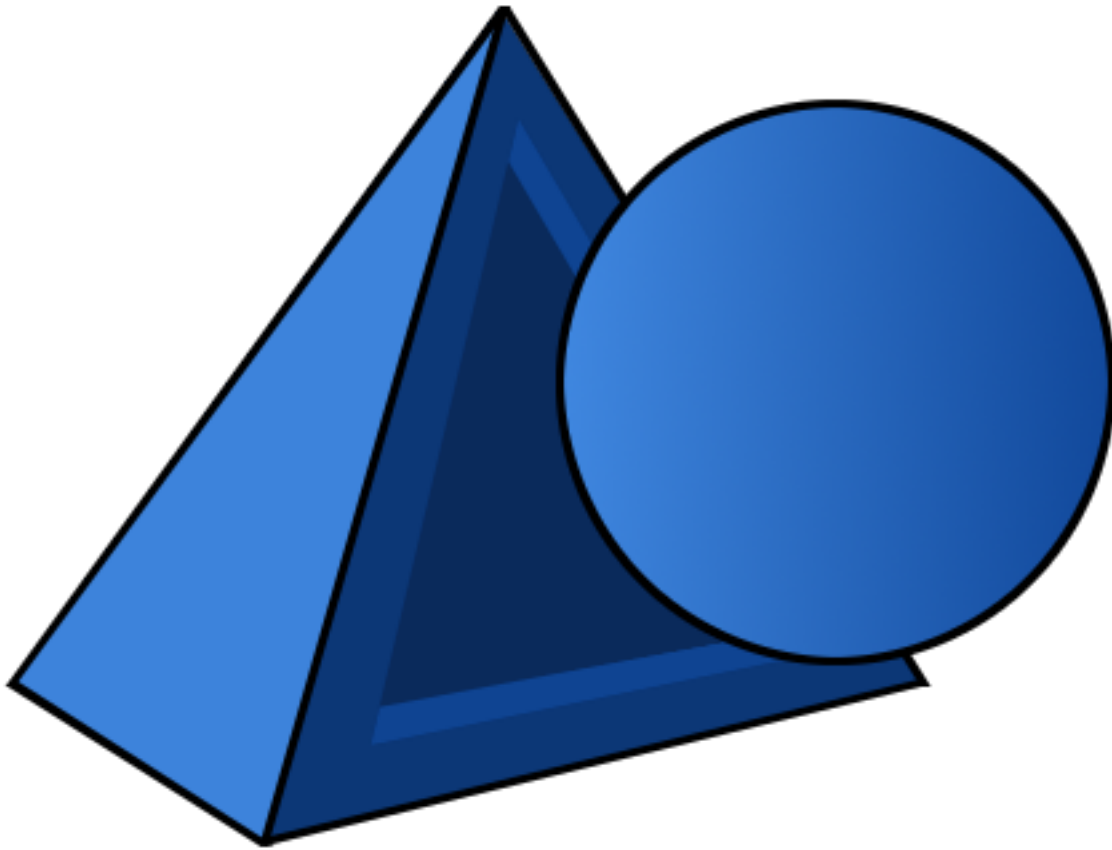
In addition to Objects, Keyframes can also be applied to Properties.

Section 9: Animation Graph Handle Replication

Animation Graph Handles are frequently used in animation software to fine-tune animations. Where keyframes set the major points of an animation graph over time, the handles set the type of curve and gradient between points. These are replicated with projects by default, but this option can be turned off.

Turning off Animation Graph Handle replication does result in a performance boost, but you should only turn it off if you're sure you don't need it.

2.4 Advanced Topics



2.4.1 Authenticating with Aesel

[Go Home](#)

Aesel utilizes [Auth0](#) for its authentication flows. This provides a number of integrations, and I suggest referring to their latest documentation to obtain a better understanding of how this all functions.

Aesel provides JWT authentication with Spring Security, so is capable of authenticating from User Databases, social log-ins, etc. To test out authentication, you can visit <http://aesel-address:8080/login>, making sure to replace 'aesel-address' with the address of your instance.

You can follow [this guide](#) to setup your Auth0 account to integrate with Aesel correctly. The configuration properties required can be set in application.properties or a separate file called 'auth0.properties'.

2.4.2 Aesel Components

[Go Home](#)

Aesel is not actually a single program, but a number of programs all running independently and communicating with each other. This allows for built-in redundancy to achieve both high-availability, and scalability to respond quickly to large changes in volume.

Below you can find the components that comprise the Aesel Architecture:

Adrestia

Adrestia acts as the HTTP Gateway for external clients into Aesel. It abstracts away the various clusters of event-based components into a single transactional plane for both users and other services to utilize.

- [Home Page](#)
- [Documentation](#)

Kelona

Kelona stores graphics assets, and tracks various versions of those assets over time. Effectively, it acts as a Version Control System for large-scale, binary files for large-scale userbases.

- [Home Page](#)
- [Documentation](#)

AeselProjects

AeselProjects stores project management information for collaborative 3-D projects (ie. Animation, Product Design, etc). This provides organizational capabilities and a rich User Experience on large projects.

- [Home Page](#)
- [Documentation](#)

CLyman

CLyman is a service working in the domain of Renderable Objects. It exposes CRUD operations, and is the starting point of the Outbound Streaming API's.

- [Home Page](#)
- [Documentation](#)

Crazy Ivan

Crazy Ivan is a service working in the domain of Scenes. It exposes CRUD operations for scenes, as well as registration operations. It also acts as the sender for the Outbound Streaming API's.

- [Home Page](#)
- [Documentation](#)

Mongo

Mongo serves as the primary data store behind CLyman, storing all Object information.

Mongo GridFS also serves as the default data store for Assets.

- [Home Page](#)

Neo4j

Neo4j serves as the primary data store behind Crazy Ivan, storing all Scene information.

- [Home Page](#)

Consul

Consul serves as both a service registry and a distributed configuration store.

- [Home Page](#)

2.4.3 Deploying Aesel

[Go Home](#)

System Requirements

In order to run Aesel, you should have at least one server with a minimum of:

- 8GB RAM Available
- 8GB Hard Disk Space Available

Note that production systems will likely require significantly more resources.

In order to run Aesel on Docker, you should have at least:

- Docker CE >17.03 or Docker EE >17.06
- Docker Compose >1.12.0 for using Docker Compose scripts

Running Aesel natively is supported on the following platforms:

- Ubuntu >16.04
- Redhat/Centos >7

Note that the above recommendations are minimum requirements, and production deployments will likely require significantly more resources, especially when handling very high-volumes of traffic.

Deployment Guides

- *[Development Environment](#)*
- *[Secure, Single-Server Environment](#)*

Scalability

Aesel is built to be horizontally scalable, so it can also run across multiple servers.

Crazy Ivan, CLyman, and Mongo form separate clusters which each manage disparate scenes. Adrestia forms a Service Mesh on top of these clusters, allowing transparent, transactional access. Neo4j, Adrestia, and Kelona are all completely horizontally scalable, with many instances able to run simultaneously.

Consul Agents should run on each server as well. Mongo and Neo4j can all be clustered to increased scalability for the underlying data stores.

Docker

Docker images are available for all system components, so Aesel can be deployed on any architecture that supports Docker (ie. Docker Swarm or Kubernetes).

Load Balancing

Adrestia serves as the gateway for both HTTP, and will automatically load balance between instances of CLyman/Crazy Ivan. An HTTP reverse proxy (such as NGINX) can be used to balance between instances of Adrestia for HTTP requests, if desired.

When a device registers to a scene, it is provided an instance of CLyman to send updates to. This means that the UDP streaming is inherently load balanced within each cluster.

Security

Aesel has several layers of security:

Transactions

Transactional Security utilizes TLS (HTTPS) for encryption.

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
```

Aesel utilizes [Auth0](#) for it's front-end authentication flows. This provides a number of integrations, and I suggest referring to their latest documentation to obtain a better understanding of how this all functions.

Adrestia provides JWT authentication with Spring Security, so is capable of authenticating from User Databases, social log-ins, etc. To test out authentication, you can visit <http://adrestia-address:8080/login>, making sure to replace 'adrestia-address' with the address of your instance.

You can follow [this guide](#) to setup your Auth0 account to integrate with Adrestia correctly.

Events

UDP Events utilize AES-256-cbc encryption, with the key, password, salt, and IV set in the application configuration. AES-256 bit keys can be generated with the below command:

```
openssl enc -aes-256-cbc -k secret -P -md sha1
```

Where 'secret' is a password for generating the key.

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 Crazy Ivan. The key and salt are delivered to end user devices after a registration transaction, which is both authenticated and encrypted.

Configuration

Secure configuration values should be 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.

2.4.4 Deploying a Secure, Single-Server Aesel Environment

Go Home

System Requirements

In order to run Aesel, you should have at least one server with a minimum of:

- 8GB RAM Available
- 8GB Hard Disk Space Available

Note that production systems will likely require significantly more resources.

In order to run Aesel on Docker, you should have at least:

- Docker CE >17.03 or Docker EE >17.06
- Docker Compose >1.12.0 for using Docker Compose scripts

Running Aesel natively is supported on the following platforms:

- Ubuntu >16.04
- Redhat/Centos >7

Note that the above recommendations are minimum requirements, and production deployments will likely require significantly more resources, especially when handling very high-volumes of traffic.

This is a great option for Demo environments, or small LAN-based networks which only need to handle scenes for a few users.

While this is secure, it is not intended for deployment in a cloud production environment. The security of this deployment depends on the security of the underlying server running Aesel.

Download

First, Download the Aesel setup files from <https://github.com/AO-StreetArt/Aesel/archive/master.zip>.

Unzip the files, and open a terminal/command prompt from the main folder.

Open the `aesel.sh` file. This contains the central definitions for the variables you'll fill out. First, pay attention to the `'SSL_BASE_DIR'` variable. This sets the base directory where your SSL Certificates are contained, with a default of `'/var/ssl'`.

Make sure to update the `'NETWORK_INTERFACE_ADDRESS'` environment variable to your server's public IP address before continuing.

Finally, set the Mongo init credentials, that can be used to connect only from the same server that Mongo is running on, in order to administer it.

- MONGO_INIT_USER
- MONGO_INIT_PW

SSL Setup

Before we go any further, let's go ahead and obtain valid SSL Certificates. The best way to do this is through Let's Encrypt, you can follow the tutorials at <https://certbot.eff.org/>. Self-Signed Certificates are not supported. Make sure that your certificates are registered to the same domain that you enter into the AESEL_DOMAIN variable. A basic example certbot command is shown below:

```
certbot certonly --standalone --preferred-challenges http -d AESEL_DOMAIN
```

You may need to copy/convert some of the certs around, below is a tree of the basic file structure needed in /var/ssl (or whatever you enter for the SSL_BASE_DIR):

```
~var
~~ssl
~~~trusted
~~~~neo4j
~~~~~ca.crt
~~~neo4j
~~~~server.crt
~~~~server.key
~~~mongo
~~~~mongodb.pem
~~~clyman
~~~~server.crt
~~~~server.key
~~~ivan
~~~~server.crt
~~~~server.key
~~~adrestia
~~~~certificate.p12
~~~kelona
~~~~certificate.p12
~~~projects
~~~~certificate.p12
```

The pem files generated by certbot can be simply copied anywhere a .crt or .pem file is required. To get a .p12 file, an example is shown below:

```
openssl pkcs12 -export -in fullchain.pem -inkey privkey.pem -out /var/ssl/adrestia/certificate.p12 -name tomcat -CAfile chain.pem -caname root
```

Then, you'll need to open up the aesel.sh file, and enter your export password in the 'SSL_KEYSTORE_PW'

AES Configuration

AES Information is set in the aesel.sh script, and can be generated with:

```
openssl enc -aes-256-cbc -k secret -P -md sha1
```

Where ‘secret’ is a password for generating the key.

Start Database Layer

To start the Aesel DB Layer, run the below command:

```
./aesel.sh db
```

The easiest way to update the neo4j login information is to use [Neo4j Client](#). Simply connect to your instance with username and password neo4j/neo4j, and you will be prompted to change the password.

Finally, you’ll need to setup a Mongo admin user, and separate users for Adrestia and CLyman. You can connect from the same server running Mongo by using the mongo shell with:

```
mongo admin -u <mongo-init-un> -p <mongo-init-pw>
```

Create an administrator user:

```
use admin
```

```
‘db.createUser(
    { user: “myUserAdmin”, pwd: “abc123”, roles: [ { role: “userAdminAnyDatabase”, db: “admin” }, “read-
      WriteAnyDatabase” ]
    }
);’
```

Then, an example user creation for Adrestia is shown below:

```
use _adrestia
```

```
‘db.createUser({ user:”test1”, pwd:”test1”, roles:[
    { role:”readWrite”, db:”_adrestia”
    }
], mechanisms:[
    “SCRAM-SHA-1”
]
});’
```

A similar user should be created for CLyman in the database ‘clyman’, Projects in the database ‘_projects’, and Kelona in the database ‘_avc’.

Authentication Configuration

Open back up the aesel.sh file, and start by entering the Neo4j login into the NEO4J_UN and NEO4J_PW variables. Then, you can update the variables for:

- KELONA_UN
- KELONA_PW
- ADRESTIA_INIT_UN
- ADRESTIA_INIT_PW

- PROJECTS_UN
- PROJECTS_PW
- IVAN_UN
- IVAN_PW
- CLYMAN_UN
- CLYMAN_PW
- MONGO_CLYMAN_USER
- MONGO_CLYMAN_PW
- MONGO_ADRESTIA_USER
- MONGO_ADRESTIA_PW
- MONGO_KELONA_USER
- MONGO_KELONA_PW
- MONGO_PROJECTS_USER
- MONGO_PROJECTS_PW

Start Scene Cluster

Starting a Scene Cluster (Crazy Ivan and CLyman), can be done with the below command:

```
./aesel.sh cluster
```

Start Stateless Services

Core, Stateless Services (Adrestia, Kelona, Projects), can be started with:

```
./aesel.sh core
```

Login

Now you can open your web browser and navigate to <https://localhost:8080/portal/home>. Login with an account you setup on your Auth0 dashboard.

2.4.5 Transactions in Aesel

An Aesel transaction (ie. the set of actions triggered by the receipt of an external message) is atomic, with the exception of Asset Updates. The general assumption is this:

- Scenes & Objects have active atomic enforcement guaranteed, and are safe for multiple devices to interact with simultaneously
- Assets are viewed as setup, and multiple devices are not expected to be updating assets at the same time. Asset reads are safe for multiple devices to execute simultaneously.

Object Locking

when utilizing Object Locking, the first device that requests the lock will be awarded it, with any other competing requests denied. Any lock request or update against the object will fail unless it is executed by the owner (ie. the 'owner' field of the object matches that stored in Aesel), until an unlock request is received from the owner.

It is still possible for other devices to query or get the object, and retrieve the owner ID.

Object Change Streams

In the Object Change Stream API, a significant change occurs in the API. We still guarantee atomic transactions, but the results of any event may fail to reach any given device, or an event may fail to transfer to the server altogether. For this reason, users are generally expected to send a large volume of these messages in most cases, continually informing Aesel of the current values of the object or property in question.

/v1

GET /v1/asset, 28
GET /v1/asset/(asset_key), 29
GET /v1/asset/count, 27
GET /v1/collection, 34
GET /v1/collection/{key}, 33
GET /v1/project, 47
GET /v1/project/(key), 44
GET /v1/relationship, 31
GET /v1/scene/(key), 37
GET /v1/scene/(key)/property/(property_key),
??
GET /v1/scene/(key)/property/query, ??
GET /v1/scene/(scene_key)/object/(object_key),
??
GET /v1/scene/(scene_key)/object/(object_key)/lock,
??
GET /v1/users/, 24
GET /v1/users/(key), 20
POST /v1/asset/, 25
POST /v1/asset/{asset_key}, 26
POST /v1/bulk/asset, 28
POST /v1/bulk/collection, 33
POST /v1/collection, 32
POST /v1/collection/{key}, 33
POST /v1/login, 18
POST /v1/project, 43
POST /v1/project/(key), 44
POST /v1/project/(key)/groups, 45
POST /v1/project/(key)/groups/{groupName},
46
POST /v1/scene/(key), 36
POST /v1/scene/(key)/deregister, 41
POST /v1/scene/(key)/property/, ??
POST /v1/scene/(key)/property/{property_key},
??
POST /v1/scene/(key)/register, 39
POST /v1/scene/(scene_key)/object/, ??

POST /v1/scene/(scene_key)/object/(object_key),
??
POST /v1/scene/(scene_key)/object/query,
??
POST /v1/scene/query, 38
POST /v1/users/sign-up, 19
PUT /v1/project/(key)/groups/{groupName}/scenes/{sceneKey},
46
PUT /v1/relationship, 30
PUT /v1/scene/(key), 35
PUT /v1/scene/(key)/align, 42
PUT /v1/users/(key), 20
PUT /v1/users/(key)/active, 23
PUT /v1/users/(key)/admin, 23
PUT /v1/users/(key)/projects/(projectKey),
21
PUT /v1/users/(key)/scenes/(sceneKey),
22
DELETE /v1/asset/(asset_key), 28
DELETE /v1/collection/{key}, 35
DELETE /v1/project/(key), 48
DELETE /v1/project/(key)/groups/{groupName},
47
DELETE /v1/project/(key)/groups/{groupName}/scenes/{sceneKey},
47
DELETE /v1/relationship, 31
DELETE /v1/scene/(key), 38
DELETE /v1/scene/(key)/property/(property_key),
??
DELETE /v1/scene/(scene_key)/object/(object_key),
??
DELETE /v1/scene/(scene_key)/object/(object_key)/lock,
??
DELETE /v1/users/(key), 25
DELETE /v1/users/(key)/active, 24
DELETE /v1/users/(key)/admin, 23
DELETE /v1/users/(key)/projects/(projectKey),
21
DELETE /v1/users/(key)/scenes/(sceneKey),
22