Kelona Documentation

Release 1.0.0

AO

Feb 23, 2019

Contents:

CHAPTER 1

Overview

Kelona is a Version Control service for large files, binary or text based. It is primarily targeted at storing files for graphics applications, such as:

- .obj files
- .glsl files
- .fbx files
- Proprietary data formats (ie. for Blender or Maya)

Kelona does not have access to file contents, instead tracking metadata and dropping files into a large-scale datastore. The only currently supported backend is Mongo GridFS.

Kelona is a part of the AO Aesel Project, along with CLyman, Crazy Ivan, and Adrestia.

CHAPTER 2

Features

- Storage of large-scale Assets (files), and associated metadata.
- Storage of relationships between assets and other data elements.
- Provide a History of updates on each Asset.

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.

2.1 Getting Started with Kelona

Go Home

2.1.1 Docker

An official Docker Image of Kelona is provided, and to get you up and running quickly, a Docker Compose file is provided as well. To start up a Mongo instance, a Consul instance, and a Kelona instance, simply run the following from the 'compose/min' folder:

```
docker-compose up
```

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

```
curl http://localhost:5635/health
```

Keep in mind that this is not a secure deployment, but is suitable for exploring the *Kelona API*.

2.1.2 Building from Source

Once you've got the required backend services started, build and execute the tests for the repository. Please note that integration tests will fail unless you have instances of the required backend services running:

```
./gradlew check
```

And, finally, start Kelona:

```
./gradlew bootRun
```

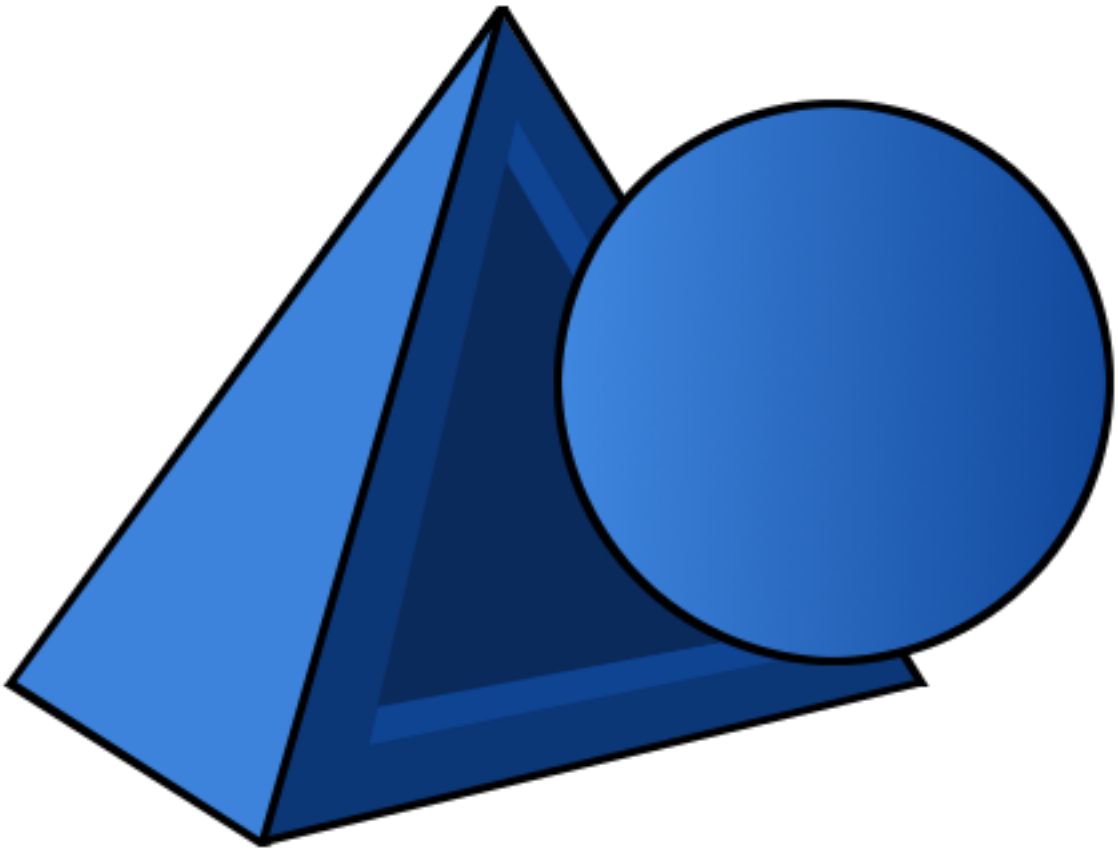
2.1.3 Using the Latest Release

Kelona can also be downloaded as a runnable JAR for the latest release from here.

When using a JAR, unzip the downloaded package, move to the main directory from a terminal, and run:

```
java -jar build/libs/kelona-0.0.1.jar
```

2.2 API Documentation



2.2.1 Asset API

An Asset, at it's core, is simply a file. The general expectation, however, is that this file is quite large, and may be in a binary format. An Asset has some associated metadata.

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 the create a relationship of the specified type.
- **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 HTTP/1.1
Host: localhost:5635
Content-Type: multipart/form-data
```

curl

```
curl -i -X POST http://localhost:5635/v1/asset -H 'Content-Type: multipart/form-data'
```

wget

```
wget -S -O- http://localhost:5635/v1/asset --header='Content-Type: multipart/form-data
↪'
```

httpie

```
http POST http://localhost:5635/v1/asset Content-Type:multipart/form-data
```

python-requests

```
requests.post('http://localhost:5635/v1/asset', headers={'Content-Type': 'multipart/
↪form-data'})
```

response

```
HTTP/1.1 200 OK
Location: http://localhost:5635/v1/asset

new-asset-key
```

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:5635
Content-Type: multipart/form-data
```

curl

```
curl -i -X POST 'http://localhost:5635/v1/asset/{key}' -H 'Content-Type: multipart/
↪form-data'
```

wget

```
wget -S -O- 'http://localhost:5635/v1/asset/{key}' --header='Content-Type: multipart/
↪form-data'
```

httpie

```
http POST 'http://localhost:5635/v1/asset/{key}' Content-Type:multipart/form-data
```

python-requests

```
requests.post('http://localhost:5635/v1/asset/{key}', headers={'Content-Type':
↪'multipart/form-data'})
```

response

```
HTTP/1.1 200 OK
Location: http://localhost:5635/v1/asset/{key}

new-asset-key
```

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:5635
```

curl

```
curl -i 'http://localhost:5635/v1/asset/{key}'
```

wget

```
wget -S -O- 'http://localhost:5635/v1/asset/{key}'
```

httpie

```
http 'http://localhost:5635/v1/asset/{key}'
```

python-requests

```
requests.get('http://localhost:5635/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:5635
```

curl

```
curl -i 'http://localhost:5635/v1/asset/count?file-type=obj'
```

wget

```
wget -S -O- 'http://localhost:5635/v1/asset/count?file-type=obj'
```

httpie

```
http 'http://localhost:5635/v1/asset/count?file-type=obj'
```

python-requests

```
requests.get('http://localhost:5635/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:5635
```

curl

```
curl -i 'http://localhost:5635/v1/asset?file-type=obj'
```

wget

```
wget -S -O- 'http://localhost:5635/v1/asset?file-type=obj'
```

httpie

```
http 'http://localhost:5635/v1/asset?file-type=obj'
```

python-requests

```
requests.get('http://localhost:5635/v1/asset?file-type=obj')
```

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:5635
```

curl

```
curl -i -X DELETE 'http://localhost:5635/v1/asset/{key}'
```

wget

```
wget -S -O- --method=DELETE 'http://localhost:5635/v1/asset/{key}'
```

httpie

```
http DELETE 'http://localhost:5635/v1/asset/{key}'
```

python-requests

```
requests.delete('http://localhost:5635/v1/asset/{key}')
```

2.2.2 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.

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:5635
Content-Type: application/json

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

curl

```
curl -i -X POST http://localhost:5635/v1/collection -H 'Content-Type: application/json
↪' --data-raw '{"category": "test", "description": "This is a test", "name":
↪"testCollection", "tags": ["test1"]}'
```

wget

```
wget -S -O- http://localhost:5635/v1/collection --header='Content-Type: application/
↪json' --post-data='{"category": "test", "description": "This is a test", "name":
↪"testCollection", "tags": ["test1"]}'
```

httpie

```
echo '{
  "category": "test",
  "description": "This is a test",
  "name": "testCollection",
  "tags": [
    "test1"
  ]
}' | http POST http://localhost:5635/v1/collection Content-Type:application/json
```

python-requests

```
requests.post('http://localhost:5635/v1/collection', headers={'Content-Type':
↪'application/json'}, json={'category': 'test', 'description': 'This is a test',
↪'name': 'testCollection', 'tags': ['test1']})
```

response

```
HTTP/1.1 200 OK
Location: http://localhost:5635/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:5635
```

curl

```
curl -i 'http://localhost:5635/v1/collection/{key}'
```

wget

```
wget -S -O- 'http://localhost:5635/v1/collection/{key}'
```

httpie

```
http 'http://localhost:5635/v1/collection/{key}'
```

python-requests

```
requests.get('http://localhost:5635/v1/collection/{key}')
```

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:5635
Content-Type: application/json

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

curl

```
curl -i -X POST 'http://localhost:5635/v1/collection/{key}' -H 'Content-Type:␣
↪application/json' --data-raw '{"category": "test", "description": "This is a test",
↪"name": "testCollection", "tags": ["test1"]}'
```

wget

```
wget -S -O- 'http://localhost:5635/v1/collection/{key}' --header='Content-Type:␣
↪application/json' --post-data='{"category": "test", "description": "This is a test",
↪ "name": "testCollection", "tags": ["test1"]}'
```

httpie

```
echo '{
  "category": "test",
  "description": "This is a test",
  "name": "testCollection",
  "tags": [
    "test1"
  ]
}' | http POST 'http://localhost:5635/v1/collection/{key}' Content-Type:application/
↪json
```

python-requests

```
requests.post('http://localhost:5635/v1/collection/{key}', headers={'Content-Type':
↪'application/json'}, json={'category': 'test', 'description': 'This is a test',
↪'name': 'testCollection', 'tags': ['test1']})
```

response

```
HTTP/1.1 200 OK
Location: http://localhost:5635/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:5635
```

curl

```
curl -i 'http://localhost:5635/v1/collection?name=test&num_records=10&page=0'
```

wget

```
wget -S -O- 'http://localhost:5635/v1/collection?name=test&num_records=10&page=0'
```

httpie

```
http 'http://localhost:5635/v1/collection?name=test&num_records=10&page=0'
```

python-requests

```
requests.get('http://localhost:5635/v1/collection?name=test&num_records=10&page=0')
```

response

```
HTTP/1.1 200 OK
Location: http://localhost:5635/v1/collection?name=test&num_records=10&page=0

[
    {
        "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:5635
```

curl

```
curl -i -X DELETE 'http://localhost:5635/v1/collection/{key}'
```

wget

```
wget -S -O- --method=DELETE 'http://localhost:5635/v1/collection/{key}'
```

httpie

```
http DELETE 'http://localhost:5635/v1/collection/{key}'
```

python-requests

```
requests.delete('http://localhost:5635/v1/collection/{key}')
```

2.2.3 Asset History API

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

Asset History Retrieval

GET /v1/history/ (*asset_key*)
Get the Asset History associated to a particular Asset.

Status Codes

- 200 OK – Success

http

```
GET /v1/asset-history/{key} HTTP/1.1
Host: localhost:5635
```

curl

```
curl -i 'http://localhost:5635/v1/asset-history/{key}'
```

wget

```
wget -S -O- 'http://localhost:5635/v1/asset-history/{key}'
```

httpie

```
http 'http://localhost:5635/v1/asset-history/{key}'
```

python-requests

```
requests.get('http://localhost:5635/v1/asset-history/{key}')
```

response

```
HTTP/1.1 200 OK
Location: http://localhost:5635/v1/asset-history/{key}

[]
```

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 can be used to model relationships with both external sources (such as a Renderable Object in a video game), or to model relationships between assets (such as having one Asset be the thumbnail of another).

By storing these relationships within Kelona, any updated Asset is immediately associated to any other data sources. Any user loading that related object must query Kelona to retrieve the necessary Assets, which they do by querying this API.

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:5635
Content-Type: application/json

{
  "assetId": "asset123",
  "relationshipType": "scene",
  "relatedId": "scene123"
}
```

curl

```
curl -i -X PUT http://localhost:5635/v1/relationship -H 'Content-Type: application/
↪json' --data-raw '{"assetId": "asset123", "relatedId": "scene123", "relationshipType
↪": "scene"}'
```

wget

```
wget -S -O- --method=PUT http://localhost:5635/v1/relationship --header='Content-
↪Type: application/json' --body-data='{"assetId": "asset123", "relatedId": "scene123
↪", "relationshipType": "scene"}'
```

httpie

```
echo '{
  "assetId": "asset123",
  "relatedId": "scene123",
  "relationshipType": "scene"
}' | http PUT http://localhost:5635/v1/relationship Content-Type:application/json
```

python-requests

```
requests.put('http://localhost:5635/v1/relationship', headers={'Content-Type':
↪'application/json'}, json={'assetId': 'asset123', 'relatedId': 'scene123',
↪'relationshipType': 'scene'})
```

response

```
HTTP/1.1 200 OK
Location: http://localhost:5635/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:5635
```

curl

```
curl -i -X DELETE 'http://localhost:5635/v1/relationship?type=scene&related=123&
↪asset=456'
```

wget

```
wget -S -O- --method=DELETE 'http://localhost:5635/v1/relationship?type=scene&
↪related=123&asset=456'
```

httpie

```
http DELETE 'http://localhost:5635/v1/relationship?type=scene&related=123&asset=456'
```

python-requests

```
requests.delete('http://localhost:5635/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*) – Optional. The Asset ID of the Relationship to find.
- **related** (*string*) – Optional. The Related ID of the Relationship to find.
- **type** (*string*) – Optional. The type of Relationship to find.

Status Codes

- 200 OK – Success

http

```
GET /v1/relationship?type=scene&related=123 HTTP/1.1
Host: localhost:5635
```

curl

```
curl -i 'http://localhost:5635/v1/relationship?type=scene&related=123'
```

wget

```
wget -S -O- 'http://localhost:5635/v1/relationship?type=scene&related=123'
```

httpie

```
http 'http://localhost:5635/v1/relationship?type=scene&related=123'
```

python-requests

```
requests.get('http://localhost:5635/v1/relationship?type=scene&related=123')
```

response

```
HTTP/1.1 200 OK
Location: http://localhost:5635/v1/relationship?type=scene&related=123

[
    {
        "id": "5bbd6ea100bd75575fb32caa",
        "assetId": "5bbd6ea100bd75575fb32ca8",
        "relationshipType": "thumbnail",
        "relatedId": "5bbd6da600bd75575fb32ca5"
    }
]
```

2.3 Developer Notes

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

2.3.1 Continuous Integration

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

Latest CI Runs

2.3.2 Documentation

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

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

Go Home

Kelona

/v1