Open Visualisation Framework

May 02, 2019

Overview

Open Visualisation Environment (OVE) is an open-source software stack, designed to be used in large scale visualisation environments. OVE was developed to meet the requirements of controlling the Data Observatory at the Data Science Institute of Imperial College London, but it is not specialized for that purpose.

OVE can be used for visual analytics on Large High Resolution Displays, for presentations, or for collaborative group work. It allows a user to control the display of content in web browsers distributed across multiple computers by implementing a microservices architecture that allows the distributed execution of applications using web technologies.

The main components of OVE are **OVE Core**, which controls sections and the applications running within them and **OVE Apps**, which provide a set of useful applications for common tasks such as displaying webpages, images or videos and drawing graphs.

OVE Services provides core functionality microservices within OVE such as the *Persistence Service*, which is used to compute absolute positions in pixels from positions expressed relative to a grid or as a percentage of the total space size.

OVE Asset Manager provides an Asset Manager backend that manages files in an object store, a user interface for this manager, a collection of workers to asynchronously process uploaded files (e.g., to create tiles from an image, or expand a zipped archived), and a Read Proxy that can efficiently serve files from the object store.

OVE also provides user interfaces and software development kits that can be used to design and develop projects.

CHAPTER 1

Basic Concepts

An OVE `server` installation supports multiple `spaces`. A `space` is a collection of monitors, which may be attached to different computers, that together form a single display. OVE is designed to be used in Large High Resolution Display environments; but, it is also suitable for use on much smaller displays with a single or a few monitors.

In each space, OVE runs within a number of `clients`. An OVE `client` is a browser window and typically runs full-screen on a single `screen`. A `screen` can span a single monitor or can span multiple monitors that are attached to the same computer. The arrangement of `clients` is described in the `Spaces.json` file. Each `client` has its own `geometry` which is a combination of its `height`, `width`, and 2D coordinates (`x`, `y`). The values for `x` and `y` are equal to (0, 0) at the top-left corner of the `space` and increase on a per pixel basis from left to right and top to bottom, respectively.

An OVE `application` (or `app`) includes server-side and client-side components. Each of the *OVE Apps* confirm to a common structure and provide a way to display a distinct form of commonly used content (subject to the limitations described in the list of pitfalls to avoid).

Each individual `instance` of an OVE `app` is designed to run within its own `section`. `Sections` may span multiple `clients` and can overlap. `Sections` are rectangular regions within an `oveCanvas`. Each `section` has its own `geometry` which is a combination of its `height`, `width`, and 2D coordinates (`x`, `y`). The `geometry` describes the region on which a `section` is deployed on an `oveCanvas`. An `oveCanvas` can also have `groups` of one or more `sections` and `groups`.

Each individual OVE `app` can have its own `configuration` (or `config`) defined within a config.json. The `config` is used to define named `states` other app-specific configuration, which are common to all app `instances`. Each individual `instance` of an OVE `app` can have its own `state` (which can be accessed using APIs exposed by each `app`).

Each OVE `section` does not have a background colour, and a wide majority of the apps have transparent backgrounds making it possible to overlay content from one app above another. This for example, makes it possible to use the *HTML App* to display a legend for a chart or a network. All OVE apps also accept an `opacity` property (at creation time or when updated), making it possible to control the opacity of overlapping content.

1.1 Runtime environment

Each OVE `client` displays the *view* page of OVE core. When a `section` is created, a corresponding `iframe` is created within each `client` that it is associated with. The *view* of the corresponding `app` is loaded into this `iframe`.

In addition to these *views*, `apps` may present a *control* page that can be accessed directly through a web browser; this controller can communicate with the *views* running in `clients` using WebSockets.

When the `iframe` representing a `section` is created, its `margin` CSS property is used to position it correctly, and the `window.ove.geometry` object is set so that each `instance` of an `app` running within each `iframe` can determine what to display.

1.2 High availability of server-side application components

OVE provides a *Persistence Service* which can be used to replicate server-side state among peers. This service can be registered with OVE core or any OVE application as explained in the *documentation*. OVE core also accepts registration of `peer` nodes using the `http://OVE_CORE_HOST:PORT/peers` API method. Once registered OVE peers will cross-post messages that are broadcasted using WebSockets.

CHAPTER 2

Installing OVE

OVE needs to be installed before it can be used to control a display. OVE can be installed either by downloading and compiling the source code of the corresponding components or by running a specific installer available on the **OVE Install** repository.

All contributors to OVE are encouraged to download and compile the source code. All users of OVE are encouraged to use the OVE installers.

Please refer the *OVE Asset Manager installation guide* to install the OVE Asset Manager.

2.1 Installation by running OVE installers

OVE Install scripts are designed to install OVE into a Docker environment.

2.1.1 Prerequisites

- Docker
- Docker Compose

Docker is available in two versions: a free Community Edition (CE), and an Enterprise Edition (EE) that includes commercial support. The Community Edition should be sufficient for most users of OVE, and can be installed by following the instructions for Docker Desktop for Windows, Docker Desktop for Mac or Docker CE for Linux. If you are using a version of Windows or Mac OS that does not meet the requirements listed for the Docker Desktop installer (either because it is too old, or because it is the Home edition of Windows, rather than Pro, Enterprise or Education), you should instead install the legacy Docker Toolbox.

Building installers for non-supported platforms also requires:

- git
- Python

2.1.2 Downloading the OVE installers

The **OVE Install** scripts are available for Linux, Mac (OS X) and Windows operating systems either as a Python 3 or a Python 2 executable application:

- linux-python3-v0.4.0-setup
- linux-python2-v0.4.0-setup
- osx-python3-v0.4.0-setup
- osx-python2-v0.4.0-setup
- windows-python3-v0.4.0-setup.exe
- windows-python2-v0.4.0-setup.exe

2.1.3 Building installers for non-supported platforms

OVE Install provides tools for building the `setup` script for non-supported platforms. The `master` branch of **OVE Install** needs to be cloned in order to proceed:

```
git clone https://github.com/ove/ove-install
cd ove-install
```

Refer the guidelines on developing/building a single setup file for detailed setup instructions.

2.1.4 Running the installers

Once downloaded, the installation script may not be executable on Linux and Mac operating systems. As a resolution, run the following command:

```
chmod u+x *-setup
```

Running the executable will start the step-by-step installation process. This will configure the details of the deployment environment such as hostname, port numbers and environment variables.

The ports are pre-configured to a list of common defaults, but can be changed based on end-user requirements. Each port or port-range is defined as a mapping `HOST_PORT:CONTAINER_PORT`. Only the host ports can be changed, and it is important to note that **container ports must not be changed**.

Each installer is capable of installing the current stable, latest unstable or a previous stable version.

2.1.4.1 Resolving port conflicts

Once the `docker-compose.setup.ove.yml` file is generated, it is important to ensure all `HOST_PORT` values defined on the `docker-compose.setup.ove.yml` file are not currently in use. If this is not the case, corresponding `HOST_PORT` values need to be changed. For example, if another Tuoris instance exists on the host machine, it is most likely that the port `7080` could be in use. In such a situation, the Tuoris `HOST_PORT` needs to be changed on the `docker-compose.setup.ove.yml` file.

2.1.4.2 Environment variables

Please note that the references to `Hostname (or IP address)` noted below should not be replaced with `localhost`, or the Docker hostname because these services need to be accessible from the client/browser. Please

replace it with the `public hostname` or `IP address` of the `host machine`. For a local installation, the `host machine` refers to your own computer. For a server installation the `host machine` refers to the server on which the Docker environment has been setup. The default `PORT` numbers for OVE core, Tuoris, OpenVidu, and other services are provided in the *Running OVE* section.

Before starting up OVE you must configure the environment variables either by providing them during the installation process or by editing the generated `docker-compose.setup.ove.yml` file. The environment variables that can be configured are:

- `OVE_HOST` - Hostname (or IP address) + port of OVE core
- `TUORIS_HOST` - Hostname (or IP address) + port of the Tuoris service (dependency of *SVG App*).
- `OPENVIDU_HOST` - Hostname (or IP address) + port of the OpenVidu service (dependency of *WebRTC App*).
- `openvidu.publicurl` - `https://` + Hostname (or IP address) + port of the OpenVidu service (dependency of *WebRTC App*).
- `OPENVIDU_SECRET` - The OpenVidu secret. Must match `openvidu.secret` configured below.
- `openvidu.secret` - The OpenVidu secret. Must match `OPENVIDU_SECRET` configured above.
- `OVE_SPACES_JSON` - This variable is optional and not defined in the `docker-compose.setup.ove.yml` by default. This accepts a URL for the `Spaces.json` file to be used as a replacement to the default (embedded) `Spaces.json` file available with OVE.
- `LOG_LEVEL` - This variable is optional and not defined in the `docker-compose.setup.ove.yml` by default. This can have values from `0` to `6` and defaults to `5`. The values correspond to:
  - `0` - FATAL
  - `1` - ERROR
  - `2` - WARN (The recommended `LOG_LEVEL` for production deployments)
  - `3` - INFO
  - `4` - DEBUG
  - `5` - TRACE
  - `6` - TRACE_SERVER (Generates additional server-side `TRACE` logs)
- `OVE_PERSISTENCE_SYNC_INTERVAL` - This variable is optional and not defined in the `docker-compose.setup.ove.yml` by default. This accepts an interval (in milliseconds) for synchronising an instance of OVE or of an OVE application with a registered persistence service. This optional variable can be set individually for OVE core and for all OVE applications.
- `OVE_<APP_NAME_IN_UPPERCASE>_CONFIG_JSON` - This variable is optional and not defined in the `docker-compose.setup.ove.yml` by default. This accepts a path to an application-specific `config.json` file. This optional variable is useful when application-specific configuration files are provided at alternative locations on a filesystem (such as when using Docker secrets). `<APP_NAME_IN_UPPERCASE>` must be replaced with the name of the application in upper-case. For example, the corresponding environment variable for the *Networks App* would be `OVE_NETWORKS_CONFIG_JSON`.
- `OVE_MAPS_LAYERS` - This variable is optional and not defined in the `docker-compose.setup.ove.yml` by default. This accepts a URL of a file containing the *Map layers Configuration* in a JSON format and overrides the *default Map layers Configuration* of the *Maps App*.

The OpenVidu server also accepts several other optional environment variables that are not defined in the `docker-compose.setup.ove.yml` by default. These are explained in the documentation on OpenVidu server configuration parameters.

2.1.4.3 Other configuration files

Few other configuration files can be found inside the `config` directory that is auto generated along with the `docker-compose.setup.ove.yml` file:

- `Spaces.json` - The default (embedded) `spaces` configuration of OVE can be modified prior to the initial start-up of OVE. To learn more, please refer the documentation on `Spaces.json`.

2.1.4.4 Using your own certificates for OpenVidu

OpenVidu is a prerequisite for using the WebRTC App. OpenVidu uses secure WebSockets and uses certificates. And, unless you provide your own certificate, it will use a self-signed certificate which will become inconvenient when loading the WebRTC App on multiple web browsers.

You can run OpenVidu with your own certificate by first creating new Java Key Store following the OpenVidu guide on using your own certificate. This will subsequently require the following changes in the auto generated `docker-compose.setup.ove.yml` file:

```
version: '3.1'
services:
  ...

  openvidu-openvidu-call:
    image: openvidu/openvidu-call:latest
    ports:
    - "4443:4443"
    environment:
      openvidu.secret: "MY_SECRET"
      openvidu.publicurl: "https://<Hostname (or IP address)>:4443"
      server.ssl.key-store: /run/secrets/openvidu.jks
      server.ssl.key-store-password: "openvidu"
      server.ssl.key-alias: "openvidu"
    secrets:
      - openvidu.jks

  ...

secrets:
  openvidu.jks:
    file: openvidu.jks
```

To add a trusted CA certificate (`trusted_ca.cer`) to your Java Key Store, run:

```
keytool -import -v -trustcacerts -alias root -file trusted_ca.cer -keystore openvidu.
↪jks -keypass openvidu
```

2.1.5 Starting and stopping the OVE Docker applications

OVE provides separate installation scripts to help users install the necessary components. To install and start OVE on Docker run:

```
docker-compose -f docker-compose.setup.ove.yml up -d
```

Please note that the OVE UI components are re-built when the respective docker container is started for the first time, which may result in them take a bit longer than expected to start. nginx will display a `502 Bad Gateway` status while the OVE UI components are built and started up for the first time. If you see this status message, please give

the system 5 - 30 minutes to complete the installation. A working internet connection is also a must for this initial installation process.

If you wish to install OVE without it automatically starting, use the command:

```
docker-compose -f docker-compose.setup.ove.yml up --no-start
```

Once the installation procedure has completed and OVE has been started, the successful installation of OVE can be verified by accessing the OVE home page (located at: `http://OVE_CORE_HOST:PORT` as noted in the *Running OVE* section) using a web browser.

Once the services have started, you can check their status by running:

```
docker ps
```

The `ps` command will list containers along with their `CONTAINER_ID`. Then, to check logs of an individual container, run:

```
docker logs <CONTAINER_ID>
```

To stop the Docker application run:

```
docker-compose -f docker-compose.setup.ove.yml down
```

To clean-up the Docker runtime first stop any active instances and then run:

```
docker system prune
docker volume prune
```

2.2 Installation from source code

All OVE projects use a build system based on Lerna. Most OVE projects are based on Node.js, compiled with Babel, and deployed on a PM2 runtime. Some OVE projects are based on Python.

2.2.1 Prerequisites

- git
- Node.js (v8.0+) (NPM is also required, but is included with the Node.js installer)
- NPX (install with the command: `npm install -global npx`)
- PM2 (install with the command: `npm install -global pm2`)
- Lerna (install with the command: `npm install -global lerna`)
- nginx

Compiling source code for the Docker environment also requires:

- Docker
- Docker Compose

The *SVG App* requires:

- Tuoris (installation instructions available on GitHub repository)

The *WebRTC App* requires:

- OpenVidu

2.2.2 Downloading source code

All OVE projects can be downloaded from their GitHub repositories:

- OVE Core: master | releases
- OVE Apps: master | releases
- OVE Services: master | releases
- OVE UIs: master | releases
- OVE Asset Manager: master | releases

The `master` branch of each repository contains the latest code, and can also be cloned if you intend to contribute code or fix issues:

```
git clone https://github.com/ove/ove
```

Once the source code has been downloaded OVE can be installed either on a local Node.js environment (such as PM2's Node.js environment) or within a Docker environment. The two approaches are explained below.

2.2.3 Setting up local nginx installation

The OVE core, applications, services and UIs are made up of multiple microservices running on their own ports. The OVE Docker applications use nginx to overcome the complexity of end-users having to expose multiple ports on their systems. Therefore, for a local OVE installation, having nginx installed with the OVE default configuration is recommended. This will ensure all URLs found in the documentation would work without any modification.

Replace `/etc/nginx/conf.d/default.conf` (which might be found in a different path depending on the OS) with the contents of the OVE default configuration. Restart nginx if it is already running.

With this configuration, by default, OVE will run on port `80`. To run OVE on its standard port `8080`, change the server `listen` port in the nginx configuration to `8080`. This will run into a port conflict with the OVE core microservice which also runs on port `8080` by default. To change the port allocated to OVE core microservice to `9080`, modify the `pm2.json` file found inside the git clone of OVE core by replacing `8080` with `9080`. Then change the `proxy_pass http://localhost:8080` line to `proxy_pass http://localhost:9080` in the nginx configuration.

2.2.4 Compiling source code for a local Node.js environment

Once you have cloned or downloaded the code, OVE can be compiled using the Lerna build system:

```
cd ove
lerna bootstrap --hoist
lerna run clean
lerna run build
lerna run test
```

Instructions above are only provided for the **OVE Core** repository. The steps to follow are similar for other repositories.

2.2.4.1 Starting and stopping OVE using the PM2 process manager

The *SVG App* requires an instance of Tuoris to be available before starting it. To start Tuoris run:

```
pm2 start index.js -f -n "tuoris" -- -p PORT -i 1
```

The *WebRTC App* requires an instance of OpenVidu to be available before starting it. To start OpenVidu run:

```
docker run -p 4443:4443 --rm -e openvidu.secret=MY_SECRET openvidu/openvidu-
↪call:latest
```

OVE can then be started using the PM2 process manager. To start OVE on a Linux or MacOS environment run:

```
OVE_HOST="OVE_CORE_HOST:PORT" TUORIS_HOST="TUORIS_HOST:PORT" OPENVIDU_HOST="OPENVIDU_
↪HOST:PORT" pm2 start pm2.json
```

To start OVE on a Windows environment run:

```
OVE_HOST="OVE_CORE_HOST:PORT" TUORIS_HOST="TUORIS_HOST:PORT" OPENVIDU_HOST="OPENVIDU_
↪HOST:PORT" pm2 start pm2-windows.json
```

By default, OVE core and all services run on `localhost`, which should be used in place of `OVE_CORE_HOST` and `TUORIS_HOST` names above. The default `PORT` numbers for OVE core, Tuoris and OpenVidu are provided in the *Running OVE* section.

Once the services have started, you can check their status by running:

```
pm2 status
```

Then, to check logs of all services, run:

```
pm2 logs
```

To stop OVE processes managed by PM2 on a Linux or MacOS environment run:

```
pm2 stop pm2.json
```

To stop OVE processes managed by PM2 on a Windows environment run:

```
pm2 stop pm2-windows.json
```

To clean-up processes managed by PM2 on a Linux or MacOS environment run:

```
pm2 delete pm2.json
```

To clean-up processes managed by PM2 on a Windows environment run:

```
pm2 delete pm2-windows.json
```

2.2.4.2 Starting and stopping OVE UIs in Development

The OVE UI components are developed as React web applications. These components can therefore be launched in development mode, by running:

```
npm run start:dev
```

Once launched, they can be stopped by killing the application using the `Ctrl+C` keyboard shortcut.

Unless OVE Core and the OVE Apps are running on localhost on their default ports, you will also need to modify the configuration file `.env` appropriately.

2.2.5 Compiling source code for a Docker environment

This approach currently works only for Linux and MacOS environments. The `build.sh` script corresponding to each repository can be found under the top most directory of the cloned or downloaded repository or within a `packages/PACKAGE_NAME` directory corresponding to each package.

The `build.sh` script can be executed as:

```
cd ove
./build.sh
```

Instructions above are only provided for the **OVE Core** repository. The steps to follow are similar for other repositories.

2.2.5.1 Starting and stopping the OVE Docker containers

Similar to the `build.sh` script, the `docker-compose.yml` file corresponding to each repository can also be found under the top most directory of the cloned or downloaded repository or within a `packages/PACKAGE_NAME` directory corresponding to each package.

The deployment environment needs to be *pre-configured* before running these scripts.

To start each individual docker container run:

```
SERVICE_VERSION="latest" docker-compose -f docker-compose.yml up -d
```

Once the services have started, you can check their status by running:

```
docker ps
```

The `ps` command will list containers along with their `CONTAINER_ID`. Then, to check logs of an individual container, run:

```
docker logs <CONTAINER_ID>
```

To stop each individual Docker container run:

```
SERVICE_VERSION="latest" docker-compose -f docker-compose.yml down
```

To clean-up the Docker runtime first stop any active instances and then run:

```
docker system prune
docker volume prune
```

2.3 Running OVE

It is recommended to use OVE with Google Chrome, as this is the web browser used for development and in production at the Data Science Institute. However, it should also be compatible with other modern web browsers: if you encounter any browser-specific bugs please report them as an Issue.

For details of how to use OVE, see the Usage page.

After installation, OVE will expose several resources that can be accessed through a web browser:

- OVE home page `http://OVE_CORE_HOST:PORT`
- App control page `http://OVE_CORE_HOST:PORT/app/OVE_APP_NAME/control.html?oveSectionId=0`
- OVE client pages `http://OVE_CORE_HOST:PORT/view.html?oveViewId=LocalNine-0`
  - check `Spaces.json` for more information
- OVE JS library `http://OVE_CORE_HOST:PORT/ove.js`
- OVE API docs `http://OVE_CORE_HOST:PORT/api-docs/`

By default, OVE core, all apps, and all services run on `localhost`, which should be used in place of `OVE_CORE_HOST` above. Note that the docker container might be given a different IP address to the machine on which it is running; in this case, the hostname `localhost` will not work, and you should instead use the IP address printed by the command `docker-machine ip`.

The default `PORT` numbers are:

- 8080 - OVE Core
- 6060 - OVE Asset Manager UI
- 6080 - OVE Asset Manager API
- 6081 - OVE Asset Manager Read proxy
- 7080 - Tuoris
- 4443 - OpenVidu

CHAPTER 3

Using OVE

There are several steps that must be performed in order to use OVE to control a display.

Before using OVE, you will need to install **OVE Core**, any **OVE Apps**, and any **OVE Services** that you intend to use. **OVE UIs** and **OVE SDKs** may or may not be installed based on your requirements, as all core components provide REST APIs that cover all functionalities provided by these components. Installation guidelines can be found in the OVE Installation Guide. As a part of the installation, you must ensure that the OVE core server, and all OVE apps, are accessible from the computers connected to the monitors that will be used in the display.

3.1 Setting up OVE

By default, OVE provides two `spaces`, that are useful for trying out some of the **OVE Apps** with sample content. These are:

- `LocalNine` - This `space` contains nine `clients` arranged in `3 x 3` configuration. Each `client` has a dimension of `1440 x 808` pixels and the total `space` has a dimension of `4320 x 2424` pixels.
- `LocalFour` - This `space` contains four `clients` arranged in `2 x 2` configuration. Each `client` has a dimension of `1440 x 808` pixels and the total `space` has a dimension of `2880 x 1616` pixels.

You will need to modify the `Spaces.json` file and introduce a new `space` in OVE, before it can be used.

3.2 Launching browsers

In order to use OVE, you will need to open the URL of each `client` in a separate web browser window (or tab). Each URL has the form: `http://OVE_CORE_HOST:PORT/view.html?oveViewId={SPACE}-{ID}`. These URLs can also be found on the `http://OVE_CORE_HOST:PORT` page. The values for `OVE_CORE_HOST` and `PORT` must be set as explained in the OVE Installation Guide. The value for `SPACE` would be the name of the `space` used, such as `LocalNine`, `LocalFour`, or a name of a new `space` that has been defined in the `Spaces.json` file. The value for `ID` is the index of the `client` associated with the browser window, in the definition of the `space` in `Spaces.json` file. Not providing the `oveViewId` or providing the `oveViewId` in an invalid format would result

in OVE printing either the `Name of space not provided` or the `Client id not provided` warning on the browser console.

It is recommended to run each OVE `client` in a separate Google Chrome browser launched in full-screen mode. Running Google Chrome in Incognito mode will ensure a smooth operation of OVE. It is also important to set the Google Chrome auto-play policy to `No user gesture is required`, which can be accessed by opening the `chrome://flags/#autoplay-policy` URL. The `Default` auto-play policy prevents the playback of audio and video on OVE. The `Console` in Google Chrome DevTools can be used to debug OVE applications when in development.

We recommend using OVE with Google Chrome as this is the web browser used for development and in production at the Data Science Institute. However, OVE should also be compatible with other modern web browsers: if you encounter any browser-specific bugs please report them as an Issue.

To test OVE functionality, you can simply open four web browser windows (or tabs) with the following URLs corresponding to the `LocalFour` space:

```
http://OVE_CORE_HOST:PORT/view.html?oveViewId=LocalFour-0
http://OVE_CORE_HOST:PORT/view.html?oveViewId=LocalFour-1
http://OVE_CORE_HOST:PORT/view.html?oveViewId=LocalFour-2
http://OVE_CORE_HOST:PORT/view.html?oveViewId=LocalFour-3
```

However, in a much larger OVE installation with many monitors, opening browser windows may be sufficiently time-consuming for automation to be worthwhile.

On Windows, a PowerShell script can be used to launch full-screen browsers with the correct URLs. On Linux, a Bash script can be used. On MacOS, either Bash or AppleScript could be used.

3.3 Launching OVE Apps

OVE provides three different ways of launching apps into an OVE environment:

- Using **OVE UI**
- Using **OVE SDKs**
- Using **OVE Core** APIs

The **OVE UI** is designed to be used by the most basic users of OVE. The **OVE Core** APIs are intended to be used by the most advanced users of OVE.

Given below are instructions on how to load sample content using the *Images App*. A complete list of all apps and similar instructions on how to use them can be found in the **OVE Apps** repository.

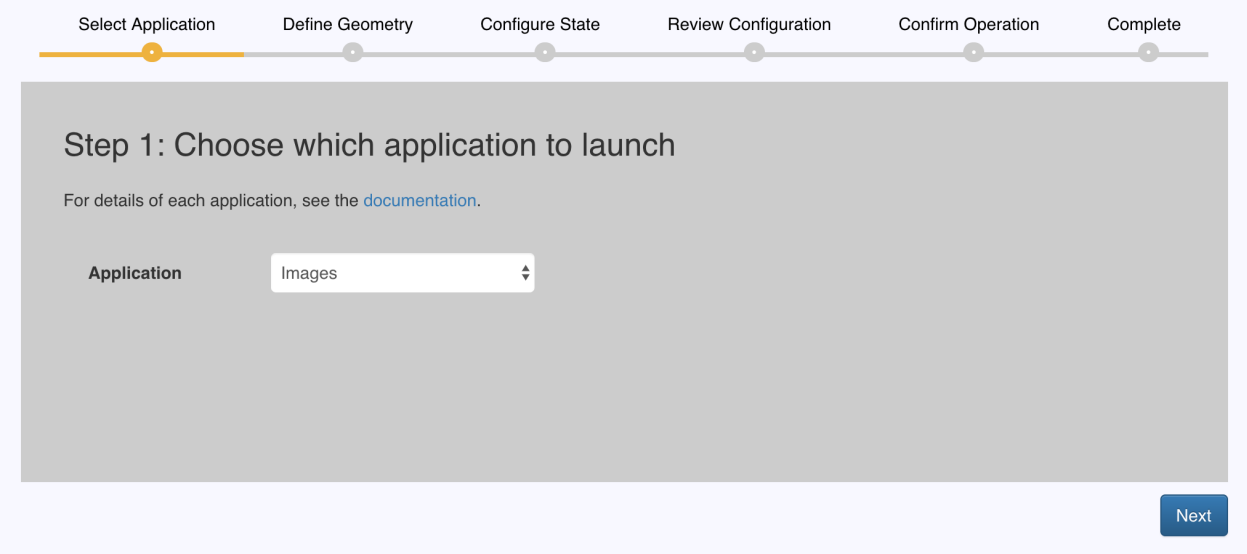
3.3.1 Launching OVE Apps using OVE UI

The *Launcher UI* can be used to launch applications into an OVE environment. The step-by-step process involves:

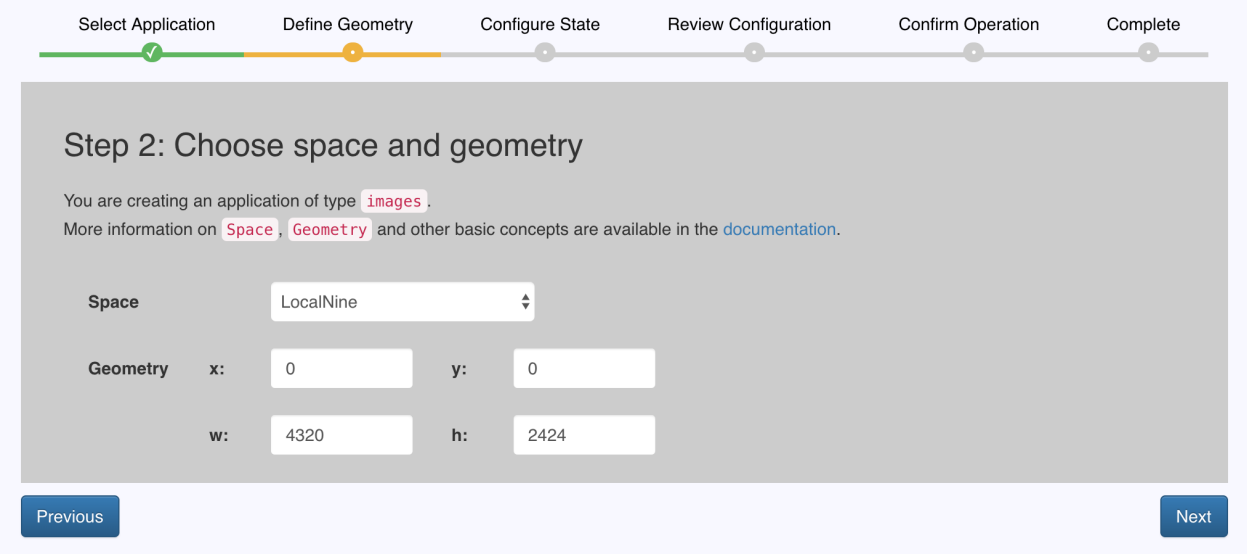
1. Choosing the type of application to launch
2. Selecting space and providing geometry details
3. Configuring initial application state
4. Reviewing state configuration and operation details (where you can choose to delete existing sections or show the app-specific controller)
5. Launching a new application instance

The confirmation step provides a link to *Preview* the space as well as a link to the app-specific usage documentation. The *Launcher UI* can be loaded into a browser window by accessing the URL `http://OVE_CORE_HOST:PORT/ui/launcher`. The user interface provides easy-to-follow instructions and validates user input. For those who wish to upload many apps at once, the *Launcher UI* also provides operating system specific curl commands.

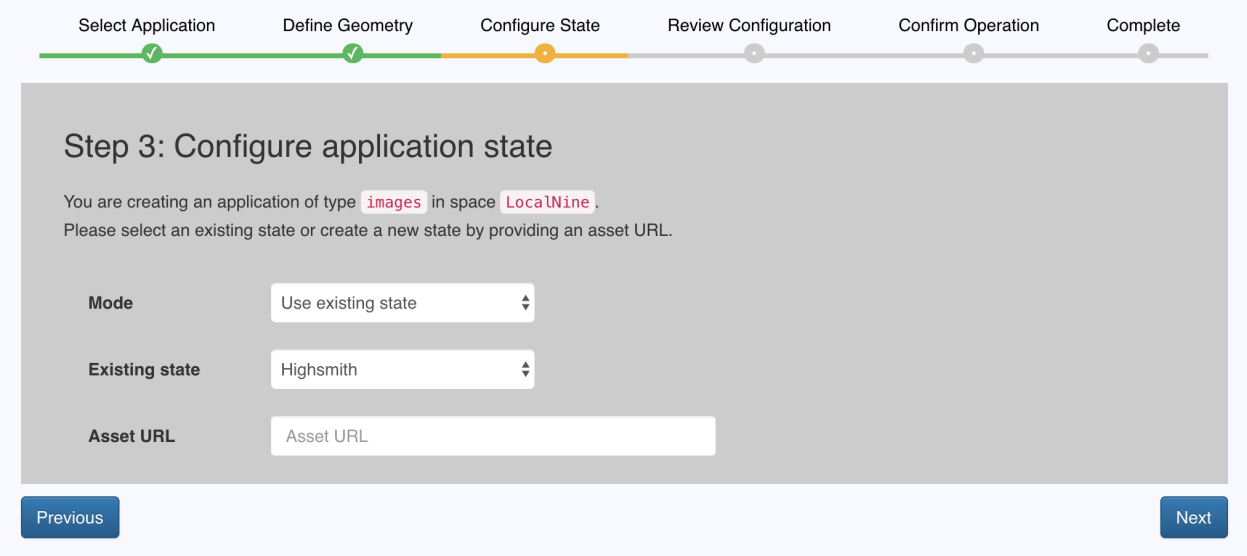
To launch the *Images App* with a sample image, in the `LocalNine` space, select `Images` as the application to launch and press `Next` to proceed.



In the second step, select `LocalNine` as the space and provide `4320` as the width and `2424` as the height and press `Next` to proceed.



In the third step, select `Use existing state` as the mode and `Highsmith` as the existing state and press `Next` to proceed.



Press `Next` in the fourth step and `Launch` in the fifth step.

3.3.2 Launching OVE Apps using the Python Client Library

The *Python Client Library* is one of the SDKs provided by OVE, which can be installed separately by following the *installation instructions*.

To launch the *Images App* with a sample image, in the `LocalNine` space, run:

```
from ove.config import local_space as space

space.enable_online_mode()

space.delete_sections()

image = space.add_section(w=4320, h=2424, x=0, y=0, app_type='images')
image.set_url("https://farm4.staticflickr.com/3107/2431422903_632ce51b56_o_d.jpg",
↪"shelley")
```

3.3.3 Launching OVE Apps using OVE Core APIs

OVE provides a number of useful APIs that can be used to launch applications and load content. The complete list of APIs provided by **OVE Core** is documented at: `http://OVE_CORE_HOST:PORT/api-docs/`.

The APIs for OVE core, all apps, and all services are documented using Swagger, and it is possible to directly invoke them from within the API documentation page (`http://OVE_CORE_HOST:PORT/api-docs/`) using a web browser. Alternatively, a standalone tool such as curl can be used.

An image can be loaded into the `LocalFour` space using the OVE APIs, by running the following commands using curl.

Linux/Mac:

```
curl --header "Content-Type: application/json" --request DELETE http://OVE_CORE_
↪HOST:PORT/sections
curl --header "Content-Type: application/json" --request POST --data '{"app": {"url":
↪"http://OVE_CORE_HOST:PORT/app/images","states": {"load": {"tileSources": "https://
↪openseadragon.github.io/example-images/highsmith/highsmith.dzi"}}}, "space":
↪"LocalFour", "h": 500, "w": 500, "y": 0, "x": 0}' http://OVE_CORE_HOST:PORT/section
```

(continues on next page)

(continued from previous page)

Windows:

```
curl --header "Content-Type: application/json" --request DELETE http://OVE_CORE_
↪HOST:PORT/sections
curl --header "Content-Type: application/json" --request POST --data "{\"app\": {\
↪"url\": \"http://OVE_CORE_HOST:PORT/app/images\", \"states\": {\"load\": {\
↪"tileSources\": \"https://openseadragon.github.io/example-images/highsmith/
↪highsmith.dzi\"}}}, \"space\": \"LocalFour\", \"h\": 500, \"w\": 500, \"y\": 0, \"x\
↪": 0}" http://OVE_CORE_HOST:PORT/section
```

These commands clear all sections on OVE, and create a new section containing an instance of the *Images App*. In this example we are loading a Deep Zoom image.

The OVE core APIs can be used for various other purposes such as grouping sections together, transforming section dimensions by translating and scaling, or for moving sections from one space to another. APIs exposed by OVE apps can be used to set, update or flush application state and to obtain differences between states or to transform from one state to another using pan and zoom operations. A catalogue of all available APIs for OVE core and all active apps is available at `http://OVE_CORE_HOST:PORT`.

3.4 Hosting content

Before using most OVE applications, you may need to host the content that you plan to load into them. OVE expects content to be accessible via URLs both locally and also in a distributed deployment. There are multiple options to host your content such that they are accessible via URLs.

1. Externally hosted static content: this is a straightforward option, but may expose your content to 3rd parties. Choices include cloud hosting providers such as Amazon AWS and Google Cloud or specialised cloud application platform providers such as Heroku and Netlify.
2. *Locally hosted static content*: this is a convenient choice for local testing and also does not expose your content to third parties. Choices include web servers such as Apache HTTP Server, nginx, and Microsoft IIS, or simpler alternatives such as Node.js HTTP Server or Python Simple HTTP Server.
3. Using an object storage: an object storage can be setup either locally or externally and can have open or restricted access. Choices include cloud storage services such as Amazon S3 and locally deployed alternatives (private cloud storage) such as Minio.
4. *Using the OVE Asset Manager*: this internally uses an object storage but is optimised for OVE projects and provides a much better user experience.

3.4.1 Locally hosted static content

Once you have stored the desired content in a directory structure, all you need is to start a web-server. This can be done using one of the approaches shown below:

Node.js:

```
npm install http-server -g
http-server
```

Python 2:

```
python -m SimpleHTTPServer 9999
```

Python 3:

```
python3 -m http.server 9999
```

Please note that you may need to specify a port number if you have chosen to use Python and the default of port 8000 is already in use (the examples above specify 9999 as the port to use).

Once the server has started, the content will be available at the any of the URLs printed on the console, if you have chosen to use Node.js or at `http://localhost:8000` (or corresponding port number), if you have chosen to use Python.

3.4.2 Using the OVE Asset Manager

Please read this short *tutorial* on the basic functionality of the OVE Asset Manager.

3.5 Controlling OVE Apps and designing interactive visualisations

Once an App has been launched in an OVE `space` it can be controlled using the corresponding controller, which provides app-specific functionality. For example, the controller of the *Images App* supports panning and zooming of images that have been loaded. The controller can be loaded into a separate browser window by accessing the URL `http://OVE_CORE_HOST:PORT/app/images/control.html?oveSectionId=0`. Not providing the `oveSectionId` would result in OVE printing the `Section id not provided` warning on the browser console.

A common practice when designing projects with interactive visualisations is to create a custom launcher application that is capable of making API calls. Such applications are usually designed to run on web browsers and invoke the **OVE Core** API using JavaScript code. These applications provide a single-click (or single-touch) experience for launching and controlling OVE apps.

When designing content to be displayed in an OVE environment, please also be aware of the potential pitfalls.

CHAPTER 4

Potential Pitfalls

There are several potential pitfalls that you should be aware of when developing visualisation applications and content that will be displayed using OVE. Larger displays (in either physical dimensions or resolution) present more challenges, and some of these pitfalls are less of a concern for an OVE installation of a smaller scale.

4.1 Technical considerations

Deterministic geometry and rendering. If a section spans multiple clients, you should ensure that each client renders its part of the section in a way that is consistent with the other clients. For example, if a word-cloud is drawn using a random algorithm that is performed independently on each client, a word may appear on more than one client, and words that should span the boundary between two clients may only appear on one client. This applies not only to visualizations that you create, but also to existing websites thay you may want to display: for example, the order of items in a news-feed may change each time it is loaded, and some sites display a list of randomly selected links to related pages.

Browser versions. You should ensure that your content displays correctly in the specific browser version used in the OVE environment where you will be displaying your content. This may well be an older version than the version that is installed on your development machine, as software auto-updates may have been disabled in production environments.

Custom libraries. The core OVE apps use stable libraries tested on all DSI visualization environments; if certain features are not working or not tested, these will be stated in the app description. If you use the *HTML App* to embed content that depends on third-party libraries, you should ensure that they support sufficiently high screen resolutions: for example, chart.js stops rendering area charts at 17000 pixels (less than the horizontal width of the Data Observatory at Imperial, which has a resolution of 30720x4320 pixels). Device emulation within Google Chrome may assist with this process.

Time synchronisation. Be aware that the clocks on each machine may not be precisely synchronised and cannot be used to time animations without taking clock drift into account.

Server scalability. If a section spans multiple OVE clients, then each may make near-simultaneous requests for the same content (e.g., webpages in the case of the *HTML App*, or image tiles in the case of the *Images App*). You should ensure that whatever server(s) that is deployed to serve these requests to be configured to handle the expected load.

Additionally, if you are loading content from a third-party service using an API, you should ensure that you will not exceed limits on the peak request frequency (which could result in your requests being throttled or API keys revoked).

Frame options. A website can set the X-Frame-Options header to indicate that it should not be embedded in an iframe; this will prevent it from being loaded by the *HTML App*. Some websites use this mechanism to disallow embedding except for specific alternative URLs.

4.2 Ergonomic considerations

Background colours. White backgrounds may be blinding when displayed on multiple monitors. Dark or black backgrounds will help to hide screen bezels, which can otherwise be distracting.

Content density. When designing your visualisation and content for a high resolution environment, be aware that content that looks dense on a laptop may look very sparse and spread-out when displayed upon a large screen.

Content sizing. Ensure that your content is readable from the distance at which you expect it to be read by your audience.

Content positioning. If presenting to a standing audience, do not rely on viewers being able to read the lower third of the display area, and position titles high up. For large charts, it is a good practice to duplicate any chart legends so that no part of the chart is far away from the legend. Position content to minimise the interruption of important features by bezels.

Content flow. Think carefully about how you will arrange content. This is particularly important when using a large immersive display, as viewers may have to turn their heads or walk across a room in order to look at a different part of the display, rather than simply moving their eyes. A common practice is to arrange things in order from left to right, and from top to bottom.

Animation. Animation which look reasonable on your laptop screen may become nauseating when viewed on a large screen. To avoid this, animated motion should be slow. Sequentially zooming out, panning, and then zooming back in can also be less nauseating than directly animating a pan in a zoomed-in view.

4.3 User interaction considerations

Controller mobility. If you intend to present to a larger audience, consider creating a control panel that can be operated using a phone or tablet so that you can move around without being tethered to a particular location.

Multiple control panels. Be aware that it is common practice to run multiple control panels for your visualisations; you should design your content to support this. Specifically, this means that control panels should not hold unique state: if they hold state, this should be shared with other control panels (either by using the OVE framework's API endpoints to save and load state, or by communicating directly using a WebSocket). When a new control panel loads, it should not reset the display.

CHAPTER 5

Developing OVE Applications

OVE provides a number of applications to display commonly-used types of content, such as HTML, tiled images, audio and video files, maps, networks, and charts. The *HTML App* is particularly flexible, and allows the hosting of general HTML/JavaScript web applications. However, if existing applications do not meet your needs then you can write a new OVE app.

`@ove-lib/appbase` provides a base library on which OVE applications can be built.

5.1 Application structure

An application typically has the structure:

```
├── README.md
├── package.json
└── src
    ├── client
    │   ├── common
    │   │   ├── <app-name>.css
    │   │   └── <app-name>.js
    │   ├── constants
    │   │   └── <app-name>.js
    │   ├── control
    │   │   ├── <app-name>.css
    │   │   └── <app-name>.js
    │   ├── index.html
    │   └── view
    │       ├── <app-name>.css
    │       └── <app-name>.js
    ├── config.json
    ├── data
    ├── swagger-extensions.yaml
    └── index.js
```

At the top level, the `README.md` file explains the purpose and usage of the application in human-readable form, whereas `package.json` provides machine-readable information about the package (name, version, author, license), its dependencies, and build commands.

Within `src/`, `config.json` describes any pre-configured `states` that are provided as examples (which may depend on data files in `src/data`); `src/index.js` is the file that is actually run by node.js to create a server instance. The `@ove-lib/appbase` base library exposes an API to interact with the app. If the application exposes additional API methods the `src/swagger-extensions.yaml` is used to describe them, so that Swagger can automatically generate documentation.

Applications are partitioned into separate a `control` and `view`, with shared parts placed in `src/client/common/` (for CSS or JavaScript functions) or `src/client/constants` (for JavaScript constants). A single `index.html` file is shared between the `control` and `view`, but it renders different content in both cases due to the inclusion of different JavaScript files.

The JavaScript and CSS files in `control/` are used to render the application's control page; the JavaScript and CSS files in `view/` are used to render the page that is displayed when the application is loaded into a section.

5.2 The index.html file

Before the `index.html` file is served, the placeholders `__OVEHOST__` and `__OVETYPE__` are replaced (this replacement is specified by the `registerRoutesForContent` function defined in `ove-lib-utils/src/index.js`). `__OVEHOST__` is replaced by the host-name that was specified by the `OVE_HOST` environment variable set in `pm2.json` (or `docker-compose.yml`). `__OVETYPE__` is replaced by `view` or `control`.

This mechanism allows the construction of paths for the inclusion of JavaScript or CSS files.

5.3 Logging

Messages can be logged by creating a `OVE.Utils.Logger` object, and then calling the method with the appropriate level (`fatal`, `error`, `warn`, `debug`, `info` or `trace`):

```
const log = OVE.Utils.Logger(Constants.APP_NAME, Constants.LOG_LEVEL);
log.debug('Starting application');
```

5.4 The index.js file and server-side application initialization

For most applications the `index.js` would look similar to:

```
const { Constants } = require('./client/constants/<app-nam>');
const { app, log } = require('@ove-lib/appbase')(__dirname, Constants.APP_NAME);
const server = require('http').createServer(app);

const port = process.env.PORT || 8080;
server.listen(port);
log.info(Constants.APP_NAME, 'application started, port:', port);
```

The `@ove-lib/appbase` base library also exposes `express`, `nodeModules` and `config`. These can be used to register express routes, to expose nodule modules or to access application-specific configuration found in the `config.json` file. To expose a node module from your application:

```
const { express, app, log, nodeModules } = require('@ove-lib/appbase')(__dirname,␣
↪Constants.APP_NAME);
const path = require('path');

log.debug('Using module:', '<module-name>');
app.use('/', express.static(path.join(nodeModules, '<module-name>', '<module-
↪directory>')));
```

5.4.1 The swagger-extensions.yaml file

The `swagger-extensions.yaml` is optional and only found in applications exposing REST API methods. Contents of this file include definitions of Paths and Tags objects according to the Swagger 2.0 Specification.

5.4.2 Server-side Helper methods

OVE `Utils` provides a number of useful methods, such as `Utils.getOVEHost()`, `Utils.sendMessage(res, status, message)`, `Utils.sendEmptySuccess(res)`, `Utils.getSafeSocket(socket)`, and `Utils.isNullOrEmpty(value)`. To make use of OVE `Utils` from your application to communicate with other OVE components using WebSockets:

```
const { log, Utils } = require('@ove-lib/appbase')(__dirname, Constants.APP_NAME);

let ws;
const getSocket = function () {
    const socketURL = 'ws://' + Utils.getOVEHost();
    log.debug('Establishing WebSocket connection with:', socketURL);
    let socket = new (require('ws'))(socketURL);
    socket.on('error', log.error);
    socket.on('close', function (code) {
        log.warn('Lost websocket connection: closed with code:', code);
        log.warn('Attempting to reconnect in ' + Constants.SOCKET_REFRESH_DELAY + 'ms
↪');
        // If the socket is closed, we try to refresh it.
        setTimeout(getSocket, Constants.SOCKET_REFRESH_DELAY);
    });
    ws = Utils.getSafeSocket(socket);
};
getSocket();

ws.safeSend(JSON.stringify({ appId: Constants.APP_NAME, message: message }));
```

5.5 Clint-side application initialization

On document load, you should create a new OVE object attached to the `window` object of the web browser:

```
window.ove = new OVE(Constants.APP_NAME);
// perform initialization
window.ove.context.isInitialized = true;
```

As part of initialization, you should call `OVE.Utils.initControl` or `OVE.Utils.initView`:

```
const initControl = function (data) {
    // perform further initialization.
}
OVE.Utils.initControl(Constants.DEFAULT_STATE_NAME, initControl);
```

`OVE.Utils.initControl` accepts a function to perform any further initialization that will be called once OVE initializes the controller of the app. It also accepts the name of the default `state` to be loaded as a part of the initialization process.

```
const initView = function () {
    // perform initialization before OVE loads the viewer.
}
const onStateLoaded = function () {
    // called immediately after the state is loaded.
}
OVE.Utils.initView(initView, onStateLoaded);
```

`OVE.Utils.initView` accepts a function to perform initialization before OVE initializes the viewer of the app. Unlike the controller, the viewer needs to be pre-initialized. This is to ensure JavaScript libraries are appropriately initialized before the application state is loaded. `OVE.Utils.initView` also accepts a function that gets called immediately after the state is loaded to the viewer.

If there are any further initialization that needs to be done after OVE initializes the viewer of the app, `OVE.Utils.initView` also accepts an optional function as its third parameter:

```
const postInitView = function () {
    // perform further initialization.
}
OVE.Utils.initView(initView, onStateLoaded, postInitView);
```

You should also ensure that page elements have been resized appropriately.

5.5.1 Resizing

`OVE.Utils` provides two methods for automatically resizing a `<div>` element. `OVE.Utils.resizeController(contentDivName)` scales the element with `id contentDivName` to fit inside both the client and window, whilst maintaining the aspect ratio of the section/content; `OVE.Utils.resizeViewer(contentDivName)` resizes the element with `id contentDivName` to the size of the corresponding section (which may span multiple clients), and then translated based on the client's coordinates.

5.5.2 Client-side Helper methods

`OVE.Utils` provides a number of useful methods, such as `OVE.Utils.getQueryParam(name, defaultValue)`, `OVE.Utils.getURLQueryParam()`, `OVE.Utils.getSpace()`, `OVE.Utils.getClient()`, `OVE.Utils.getViewId()` and `OVE.Utils.getSectionId()`.

`OVE.Utils.Coordinates.transform(vector, inputType, outputType)` provides a mechanism to convert coordinates of one format to another. The input and output types can be one of `OVE.Utils.Coordinates.SCREEN`, `OVE.Utils.Coordinates.SECTION` or `OVE.Utils.Coordinates.SPACE`:

```
// Get location of mouse pointer within the space.
function onMouseEvent(event) {
    const spaceCoordinates = OVE.Utils.Coordinates.transform(
```

(continues on next page)

(continued from previous page)

```
        [event.screenX, event.screenY], OVE.Utils.Coordinates.SCREEN, OVE.Utils.
↪Coordinates.SPACE);
}
```

5.6 The OVE object

The OVE object (`window.ove`) provides a number of useful functions and data structures to handle state, to interpret geometry and to communicate via WebSockets. It also provides a context (`window.ove.context`) to hold the application's local variables. The `window.ove.context.uuid` property provides a unique identifier for each instance of OVE. This can be used to uniquely identify each viewer and controller in the system. The `window.ove.context.hostname` property provides the hostname of the ove.js library.

5.6.1 Handling state

The `window.ove.state` object provides a `window.ove.state.current` data structure to hold the current application state. You can decide what this should contain, given the particular needs of your application.

The current application state (contents of `window.ove.state.current`) can be sent as a WebSocket broadcast by calling `OVE.Utils.broadcastState()` (with no arguments). This will update the current state on all clients that receive the message; you can register a callback function to be called when this change occurs using `OVE.Utils.setOnStateUpdate(callbackFunctionName)`.

The `window.ove.state` object also provides two other methods `cache` and `load`, which can be used to cache the application state on the server and load it sometime later. These methods are internally called by the utility methods provided by `OVE.Utils` and therefore their use is limited to a few advanced use-cases.

5.6.2 Interpreting geometry

The `window.ove.geometry` provides information useful to interpret the geometry of the clients:

- `window.ove.geometry.x` - Displacement along the x-axis relative to the top-left of the section in pixels
- `window.ove.geometry.y` - Displacement along the y-axis relative to the top-left of the section in pixels
- `window.ove.geometry.w` - Width of the client
- `window.ove.geometry.h` - Height of the client
- `window.ove.geometry.section.w` - Width of the section (or the total width of the application)
- `window.ove.geometry.section.h` - Height of the section (or the total height of the application)
- `window.ove.geometry.space.w` - Width of the space (or the total width of all clients)
- `window.ove.geometry.space.h` - Height of the space (or the total height of all clients)
- `window.ove.geometry.offset.x` - Displacement of top-left of the client along the x-axis relative to the top-left of the browser window in pixels
- `window.ove.geometry.offset.y` - Displacement of top-left of the client along the y-axis relative to the top-left of the browser window in pixels

While all of this information is available on a fully initialized viewer, the controller only has:

- `window.ove.geometry.section.w` - Width of the section (or the total width of the application)
- `window.ove.geometry.section.h` - Height of the section (or the total height of the application)

5.6.3 Communicating via WebSockets

The `window.ove.socket` provides two functions `send` and `on` to send and receive messages using WebSockets:

```
window.ove.socket.on(function (message) {
    // logic to interpret the message
});
window.ove.socket.send(message);
```

The `message` argument represents a JSON serializable object in both methods. These methods are particularly useful to trigger remote operations or to expose JavaScript functionality using REST APIs. They can also be used to develop controllers that support interactive operations such as **linking and brushing**, across a number of different application instances or types. The tool used for *Debugging Communication via WebSockets* is a good example on how to use these methods to develop an external controller.

5.6.4 Debugging Communication via WebSockets

OVE provides a tool to debug communications via WebSockets. This tool can be obtained either by downloading it (right-click this link and select **Save as**) or by cloning the source code.

```
git clone https://github.com/ove/ove
cd ove/packages/ove-core/tools/debug-socket
```

To access the browser-based tool you will also need to start a web-server. This can be done using one of the approaches shown below.

Node.js:

```
npm install http-server -g
http-server
```

Python 2:

```
python -m SimpleHTTPServer 9999
```

Python 3:

```
python3 -m http.server 9999
```

Please note that you may need to specify a port number if you have chosen to use Python and the default of port 8000 is already in use (the examples above specify 9999 as the port to use).

Once the application is launched it will be available at the any of the URLs printed on the console, if you have chosen to use Node.js or at `http://localhost:8000` (or corresponding port number), if you have chosen to use Python.

If the tool prompts you to provide `oveHost`, `oveAppName` and `oveSectionId` as query parameters, please modify the URL and provide these parameters.

The `oveHost` parameter takes the form of `OVE_CORE_HOST:PORT`. The `oveAppName` parameter is the name of the application you are interested in debugging, such as `maps`, `images` or `html` (which by convention is always in lower case). The name of the application can also be obtained via the `http://OVE_CORE_HOST:PORT/app/OVE_APP_NAME/name` API. The `oveSectionId` is the identifier of the section in which the application is currently deployed in. This identifier is used when accessing the application's control page or when working with OVE APIs to manage sections.

If the tool has been accessed with the correct parameters, you should see a text box along with a `Send` button. The contents of the text box should automatically change when you perform any operation on the application that you are

currently debugging. You can modify the contents of the text-box and press the `Send` button to control the application from within the tool.

5.6.5 Communicating within a web browser

The `window.ove.frame` provides two functions `send` and `on` to send and receive messages within a web browser:

```
window.ove.frame.on(function (message) {
    // logic to interpret the message
});
window.ove.frame.send(target, message, appId);
```

The `message` argument represents a JSON serializable object in both methods. The `target` argument can be one of `Constants.Frame.PEER` or `Constants.Frame.CHILD`, and the optional `appId` argument identifies the target application. If `window.ove.frame.on` has not been set, all messages would be received by `window.ove.socket.on`. These methods can be used to develop controllers that support interactive operations such as **linking and brushing**, across a number of different application instances or types.

5.7 Embedding OVE within an existing web application

Each OVE `client` can be deployed in its own iFrame and embedded into an existing web application. This approach has been used in the *Whiteboard App* and the *Replicator App*. OVE also supports a number of useful properties that can be passed into the iFrame of each `client`. The `filters` property accepts an `includeOnly` or `exclude` child-property that can be used to specifically include or exclude sections from being displayed within a `client`. Each OVE `client` has a dark grey background, which can be set to `none` using the `transparentBackground` property. The `load` property can be set to forcefully reload the contents of a `client`.

These properties can be passed into all `client` iFrames as a message sent to the `core` application, as noted below:

```
window.ove.frame.send(
    Constants.Frame.CHILD,
    {
        load: true,
        transparentBackground: true,
        filters: { includeOnly: [0, 1] }
    },
    'core');
```

CHAPTER 6

Open Visualisation Environment - Apps

There are several applications designed to run within Open Visualisation Environment (OVE):

- Alignment - helps align the monitors in an OVE installation.
- Audio - supports the playing of audio files within the OVE Framework.
- Charts - supports visualisation of charts using the OVE framework.
- Controller - a unified controller for all OVE apps
- HTML - supports displaying HTML web pages using the OVE framework.
- Images - supports the display of images using the OVE framework.
- Maps - supports visualisation of dynamic maps using the OVE framework.
- Networks - supports visualisation of networks with node-link diagrams using the OVE framework.
- PDF - supports displaying PDF documents using the OVE framework.
- Replicator - replicating content of an OVE installation.
- SVG - supports rendering SVG using the OVE framework.
- Videos - supports playing videos using the OVE framework.
- WebRTC - supports videoconferencing and screen sharing using the OVE framework.
- Whiteboard - creates a whiteboard that can be used within the OVE framework.

OVE needs to be installed before using OVE apps. The OVE Documentation provides installation instructions and a user guide.

CHAPTER 7

Alignment App

This app exists to help align the monitors in an OVE installation.

When installing monitors to create a tiled display, the aim is typically to physically align screens with as little gap between them as possible. However, in practice it is difficult to achieve a perfect alignment and monitors have bezels with non-zero widths. To compensate for this, adjustments can be made to the coordinates recorded for the geometry in the `Spaces.json` file.

7.1 Launching the App

All OVE applications can be launched using the Launcher UI, the Python Client Library, and the OVE APIs. The API used to launch an application is the same for all applications, but the data that is passed into it is application-specific.

To launch the alignment app using the OVE APIs:

Linux/Mac:

```
curl --header "Content-Type: application/json" --request POST --data '{"app": {"url":
↪"http://OVE_CORE_HOST:PORT/app/alignment"}, "space": "OVE_SPACE", "h": 500, "w":␣
↪500, "y": 0, "x": 0}' http://OVE_CORE_HOST:PORT/section
```

Windows:

```
curl --header "Content-Type: application/json" --request POST --data "{\"app\": {\
↪"url\": \"http://OVE_CORE_HOST:PORT/app/alignment\"}, \"space\": \"OVE_SPACE\", \"h\
↪": 500, \"w\": 500, \"y\": 0, \"x\": 0}" http://OVE_CORE_HOST:PORT/section
```

7.2 Controlling the App

The controller of the app can be loaded by accessing the URL `http://OVE_CORE_HOST:PORT/app/alignment/control.html?oveSectionId=SECTION_ID`.

The app enables the display of one of two patterns (a grid of vertical and horizontal lines, or a series of parallel diagonal lines) that span an entire `oveCanvas`. From the controller page, a user can select one or more OVE clients, use the arrow keys to move the pattern on these clients until it aligns with the others, and then export a modified `Spaces.json` file.

The grid pattern allows creation of an alignment such that bezels are ignored: are pixels of the content are displayed, and the bottom pixel of one monitor and the top pixel of the monitor below have adjacent image coordinates, but are physically separated by the bezel width.

The diagonal pattern allows the creation of an alignment that compensates for bezels: the bottom pixel of one monitor and the top pixel of the monitor below do *not* have adjacent image coordinates; instead, content will be displayed as if it was on a continuous surface behind the bezels, with bezels occluding some content.

Users may want to perform both alignment procedures, and save the results as separate configurations in the `Spaces.json` file.

CHAPTER 8

Audio App

This app supports the playing of audio files within the OVE Framework. It is powered by the Howler.js audio library, so supports the Web Audio API by default and falls back to HTML5 Audio if required. This app is intended to provide a distributed layer on top of the Howler Library.

The Audio App does not display any visible content and will, by default, play audio on all browsers a section covers.

The app currently supports `webm`, `mp3` and `wav` files.

8.1 Application State

The state of this app has a format similar to:

```
{
    "url": "https://upload.wikimedia.org/wikipedia/commons/7/74/%22Goin'_Home%22%2C_
↪performed_by_the_United_States_Air_Force_Band.oga"
}
```

8.2 Launching the App

All OVE applications can be launched using the Launcher UI, the Python Client Library, and the OVE APIs. The API used to launch an application is the same for all applications, but the data that is passed into it is application-specific.

To launch the audio app and load an audio file using the OVE APIs:

Linux/Mac:

```
curl --header "Content-Type: application/json" --request POST --data "{\"app\": {\
↪"url\": \"http://OVE_CORE_HOST:PORT/app/audio\",\"states\": {\"load\": {\"url\": \
↪"https://upload.wikimedia.org/wikipedia/commons/7/74/%22Goin'_Home%22%2C_performed_
↪by_the_United_States_Air_Force_Band.oga\"}}}, \"space\": \"OVE_SPACE\", \"h\": 500,␣
↪\"w\": 500, \"y\": 0, \"x\": 0}" http://OVE_CORE_HOST:PORT/section
```

Windows:

```
curl --header "Content-Type: application/json" --request POST --data "{\"app\": {\
↪"url\": \"http://OVE_CORE_HOST:PORT/app/audio\",\"states\": {\"load\": {\"url\": \
↪"https://upload.wikimedia.org/wikipedia/commons/7/74/%22Goin'_Home%22%2C_performed_
↪by_the_United_States_Air_Force_Band.oga\"}}}, \"space\": \"OVE_SPACE\", \"h\": 500,␣
↪\"w\": 500, \"y\": 0, \"x\": 0}" http://OVE_CORE_HOST:PORT/section
```

8.3 Controlling the App

The controller of the app can be loaded by accessing the URL `http://OVE_CORE_HOST:PORT/app/audio/control.html?oveSectionId=SECTION_ID`.

The app's API also exposes operations such as `play`, `pause`, `stop`, `seekTo` and `bufferStatus` related to playback. Volume may be controlled by operations such as `mute`, `unmute`, `volUp`, `volDown`. These operations can be executed on a per-section basis or across all sections. The position of the audio source can be set in the 3D space using the `setPosition` operation. This accepts `x`, `y` and `z` values, which should be set to be within `0.0` and `1.0` for the panning to happen within a given space. If the values are outside of this range the player will reduce the volume to provide an approximate effect.

To play audio using OVE APIs:

```
curl --request POST http://OVE_CORE_HOST:PORT/app/audio/operation/play
```

Instructions on invoking other operations are available on the API Documentation, `http://OVE_CORE_HOST:PORT/app/audio/api-docs/#operation`.

CHAPTER 9

Charts App

This app supports visualisation of charts using the OVE framework. It is based on Vega-Embed, which provides support for both Vega and Vega-Lite: these are both declarative languages for interactive visualizations, but Vega-Lite is a higher level language. Vega-Embed supports Canvas and SVG rendering.

Please note that this app supports visualisation of networks, but OVE also provides a separate Networks App that is specialised for drawing node-link diagrams.

9.1 Application State

The state of this app has a format similar to:

```
{
    "url": "https://raw.githubusercontent.com/vega/vega/master/docs/examples/bar-
↪chart.vg.json",
    "options": {
        "width": 800,
        "height": 800
    }
}
```

The `url` property points to a URL of a Vega-Embed specification. Alternatively, the specification be embedded in the state using a `spec` property. The optional `options` property can be used to specify the `width` and `height` of a chart.

9.2 Launching the App

All OVE applications can be launched using the Launcher UI, the Python Client Library, and the OVE APIs. The API used to launch an application is the same for all applications, but the data that is passed into it is application-specific.

To launch the chart app and display a chart using the OVE APIs:

Linux/Mac:

```
curl --header "Content-Type: application/json" --request POST --data '{"app": {"url":
↪"http://OVE_CORE_HOST:PORT/app/charts","states": {"load": {"url": "https://raw.
↪githubusercontent.com/vega/vega/master/docs/examples/bar-chart.vg.json", "options":
↪{"width": 800, "height": 800}}}}, "space": "OVE_SPACE", "h": 500, "w": 500, "y": 0,
↪"x": 0}' http://OVE_CORE_HOST:PORT/section
```

Windows:

```
curl --header "Content-Type: application/json" --request POST --data "{\"app\": {\
↪"url\": \"http://OVE_CORE_HOST:PORT/app/charts\", \"states\": {\"load\": {\"url\": \
↪"https://raw.githubusercontent.com/vega/vega/master/docs/examples/bar-chart.vg.json\
↪", \"options\": {\"width\": 800, \"height\": 800}}}}, \"space\": \"OVE_SPACE\", \"h\
↪": 500, \"w\": 500, \"y\": 0, \"x\": 0}" http://OVE_CORE_HOST:PORT/section
```

The charts app has a transparent background. If required, a background colour of choice can be set using the Background Utility provided by OVE.

If the charts app is used to display static charts no further controlling would be required after the chart has been loaded. The app provides a controller that can used to control interactive charts.

9.3 Controlling the App

The controller of the app can be loaded by accessing the URL `http://OVE_CORE_HOST:PORT/app/charts/control.html?oveSectionId=SECTION_ID`.

CHAPTER 10

Controller App

This app is a unified controller for all OVE apps. It makes use of transformation APIs exposed by OVE apps and supports common pan and zoom operations.

10.1 Application State

The state of this app has a format similar to:

```
{
    "mode": "group",
    "groupId": "1",
    "showTouch": true
}
```

The `mode` property is mandatory and should have a value of `space`, `group` or `geometry`. The `groupId` property is optional, and must only be provided if `mode` is `group`. The `showTouch` property can be used to specify whether the Touch overlay is enabled for the viewers.

The app will assume control of the entire `space` if the `mode` is set to `space`. But, if the mode is `geometry` it will however be limited to the geometry defined by the `x`, `y`, `w` and `h` properties set when creating the app. In both cases, the visible area of the controllers and the touch surfaces, would be limited to the `x`, `y`, `w` and `h` properties set when creating the app.

10.2 Launching the App

All OVE applications can be launched using the Launcher UI, the Python Client Library, and the OVE APIs. The API used to launch an application is the same for all applications, but the data that is passed into it is application-specific.

To launch the controller app using the OVE APIs:

```
curl --header "Content-Type: application/json" --request POST --data '{"app": {"url":
↪"http://OVE_CORE_HOST:PORT/app/controller","states": {"load": {"mode": "space"}}},
↪"space": "OVE_SPACE", "h": 500, "w": 500, "y": 0, "x": 0}' http://OVE_CORE_
↪HOST:PORT/section
```

(continues on next page)

(continued from previous page)

Windows:

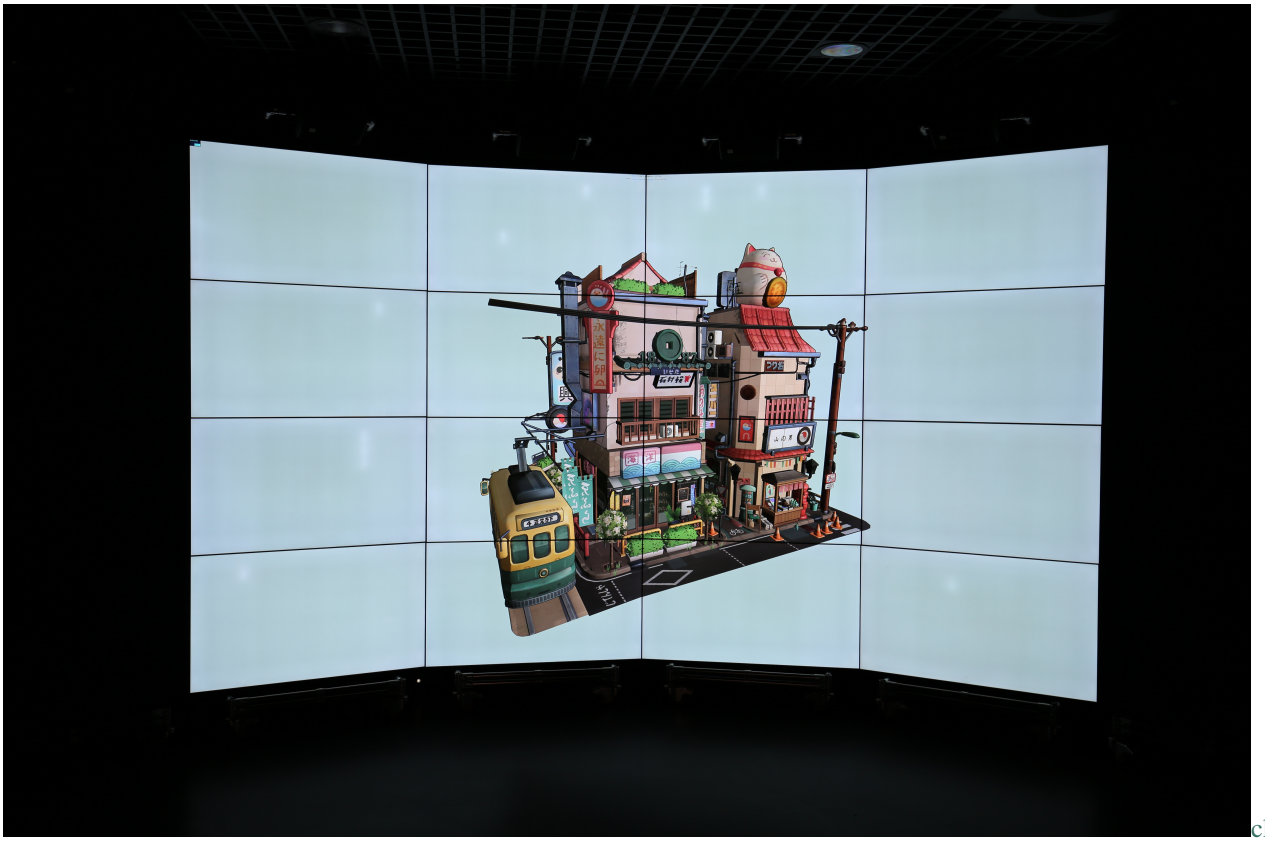
```
curl --header "Content-Type: application/json" --request POST --data "{\"app\": {\
↪"url\": \"http://OVE_CORE_HOST:PORT/app/controller\", \"states\": {\"load\": {\
↪"mode\": \"space\"}}}, \"space\": \"OVE_SPACE\", \"h\": 500, \"w\": 500, \"y\": 0, \
↪"x\": 0}" http://OVE_CORE_HOST:PORT/section
```

10.3 Controlling the App

The controller of the app can be loaded by accessing the URL `http://OVE_CORE_HOST:PORT/app/controller/control.html?oveSectionId=SECTION_ID`. The controller supports panning and zooming operations.

CHAPTER 11

HTML App



click to play the video of the HTML app

This app supports displaying HTML web pages using the OVE framework.

Each OVE Client which overlaps the section in which the HTML App is loaded creates an `iframe` that containing the whole web page: the `width` and `height` properties of this frame are set to the dimensions of the whole section, and a CSS `transform` of type `translate` is applied to shift the content so that the correct portion is displayed.

Loading the whole page within each OVE Client is inevitably somewhat inefficient. If you are loading a map or tiled image, it will be more efficient to use the corresponding OVE App, for which each client will load and render only what it needs to display.

Seen above is a video of the HTML app rendering the webgl - animation - keyframes example from three.js displaying the model Littlest Tokyo by Glen Fox recorded in HD resolution at the Imperial College Data Science Institute's Data Observatory.

11.1 Utilities

The HTML app hosts within it a number of utilities useful for another OVE app or a generic web page:

1. Background Utility
2. Distributed.js Library

11.2 Application State

The state of this app has a format similar to:

```
{
    "url": "http://my.domain"
}
```

The `url`, property is mandatory. Optionally, `launchDelay` and `changeAt` properties can be provided to control the initial delay to pre-load the contents of the web page and the precise time at which all clients will change the page they display.

11.3 Launching the App

All OVE applications can be launched using the Launcher UI, the Python Client Library, and the OVE APIs. The API used to launch an application is the same for all applications, but the data that is passed into it is application-specific.

To launch the HTML app and load a web page using the OVE APIs:

Linux/Mac:

```
curl --header "Content-Type: application/json" --request POST --data '{"app": {"url":
↪"http://OVE_CORE_HOST:PORT/app/html","states": {"load": {"url": "http://my.domain"}}
↪}, "space": "OVE_SPACE", "h": 500, "w": 500, "y": 0, "x": 0}' http://OVE_CORE_
↪HOST:PORT/section
```

Windows:

```
curl --header "Content-Type: application/json" --request POST --data "{\"app\": {\
↪"url\": \"http://OVE_CORE_HOST:PORT/app/html\", \"states\": {\"load\": {\"url\": \
↪"http://my.domain\"}}}, \"space\": \"OVE_SPACE\", \"h\": 500, \"w\": 500, \"y\": 0,␣
↪\"x\": 0}" http://OVE_CORE_HOST:PORT/section
```

If the HTML app is used to display static web pages no further controlling would be required after the web page has been loaded. The app provides a controller and exposes API that can used to control interactive web pages.

11.4 Controlling the App

The controller of the app can be loaded by accessing the URL `http://OVE_CORE_HOST:PORT/app/html/control.html?oveSectionId=SECTION_ID`.

The app's API also exposes a `refresh` operation. This operation can be executed on an individual web page or across all web pages.

To refresh a web page that is already loaded, using OVE APIs:

```
curl  --request POST http://OVE_CORE_HOST:PORT/app/html/operation/refresh
```

CHAPTER 12

Images App


photograph of the images app

This app supports the display of images using the OVE framework. It is based on OpenSeadragon and supports high resolution zoomable images. These images can be of any format OpenSeadragon supports such as Deep Zoom Image (DZI) or Simple Image.

Seen above is an image of the images app displaying the In2White Mont Blanc HD panorama DZI photographed at

the Imperial College Data Science Institute's Data Observatory.

12.1 Application State

The state of this app has a format similar to:

```
{
    "tileSources": "https://openseadragon.github.io/example-images/highsmith/
↪highsmith.dzi"
}
```

The `tileSources` property points to a DZI file and could be of an XML or JSON format as explained in the OpenSeadragon documentation.

Optionally, a `url` property can be set instead of the `tileSources` property. If it had an extension of `.dzi`, `.xml` or `.json` the `tileSources` property will be set to point to this URL. If the URL had any other extension or had no extension at all, it will be assumed to be a simple image.

The app also supports alternative types of content using other types of tile sources supported by OpenSeadragon.

12.2 Launching the App

All OVE applications can be launched using the Launcher UI, the Python Client Library, and the OVE APIs. The API used to launch an application is the same for all applications, but the data that is passed into it is application-specific.

To launch the images app and display an image using the OVE APIs:

Linux/Mac:

```
curl --header "Content-Type: application/json" --request POST --data '{"app": {"url":
↪"http://OVE_CORE_HOST:PORT/app/images","states": {"load": {"tileSources": "https://
↪openseadragon.github.io/example-images/highsmith/highsmith.dzi"}}}, "space": "OVE_
↪SPACE", "h": 500, "w": 500, "y": 0, "x": 0}' http://OVE_CORE_HOST:PORT/section
```

Windows:

```
curl --header "Content-Type: application/json" --request POST --data "{\"app\": {\
↪"url\": \"http://OVE_CORE_HOST:PORT/app/images\", \"states\": {\"load\": {\
↪"tileSources\": \"https://openseadragon.github.io/example-images/highsmith/
↪highsmith.dzi\"}}}, \"space\": \"OVE_SPACE\", \"h\": 500, \"w\": 500, \"y\": 0, \"x\
↪": 0}" http://OVE_CORE_HOST:PORT/section
```

The images app has a transparent background. If required, a background colour of choice can be set using the Background Utility provided by OVE.

12.3 Controlling the App

The controller of the app can be loaded by accessing the URL `http://OVE_CORE_HOST:PORT/app/images/control.html?oveSectionId=0`. The controller supports panning and zooming of images.

CHAPTER 13

Maps App

photograph of the maps app

This app supports visualisation of dynamic maps using the OVE framework. It gives users the option of using either OpenLayers or Leaflet as the underlying mapping library and supports tiled map layers (from Bing, OSM, etc), the CARTO platform, vector data described in formats such as GeoJSON and custom overlays built using JavaScript libraries such as D3.js.

The maps app depends on a Map Layers configuration that can be provided within the `config.json` file (either embedded or as a URL) or as an environment variable named `OVE_MAPS_LAYERS`, that points to a URL.

Seen above is an image of the maps app displaying the ArcGIS world topographic map photographed at the Imperial College Data Science Institute's Data Observatory.

13.1 Application State

The state of this app has a format similar to:

```
{
    "center": ["-11137.70850550061", "6710544.04980525"],
    "resolution": "77",
    "zoom": "12",
    "enabledLayers":["23"],
    "scripts":["http://my.domain/customLayer.js"]
}
```

The `center`, `resolution` and `zoom` properties are mandatory. It is also possible to store these properties in a file with a `.json` extension and provide its location using the optional `url` property. If the `url` property is provided, `center`, `resolution` and `zoom` can be omitted from the state configuration.

Optionally, there can be one or more enabled layers and one or more scripts to load custom overlays as seen above. The `enabledLayers` property accepts a list of integer values. These integer values correspond to the order (starting from 0) in which the layers were defined on the Map Layers configuration.

The `scripts` property accepts a list of URLs of JavaScript files.

13.2 Designing Custom Overlays

Custom overlays for the maps app are loaded using a single script as explained above in the *Application State* section. However, there may be the requirement to load multiple dependent JavaScript libraries, and also other resources such as datasets and CSS files. The example below shows what a typical `customLayer.js` should look like.

```
let transformURL;

(function () {
    if (!window.customLayer) {
        let js_files = ["https://d3js.org/d3.v5.min.js", "./script.js"];
        let css_files = ["./customLayer.css"];

        transformURL = function(url) {
            if (url.startsWith('./')) {
                return getHostName(true, 'customLayer.js') + '/' + url.slice(2);
            } else {
                return url;
            }
        };

        js_files = js_files.map(transformURL);
        css_files = css_files.map(transformURL);

        const first = $('script:first');
```

(continues on next page)

(continued from previous page)

```
        js_files.forEach(function (e) {
            $('<script>', {src: e}).insertBefore(first);
        });

        css_files.forEach(function (e) {
            $('<link>', {href: e, rel: "stylesheet", type: "text/css"}).
↪insertBefore(first);
        });

        setTimeout(function () {
            window.map = window.ove.context.map;
            init();
        }, 2000);

        window.customLayer = true;

        function getHostName (withScheme, scriptName) {
            let scripts = document.getElementsByTagName('script');
            for (let i = 0; i < scripts.length; i++) {
                if (scripts[i].src.indexOf(scriptName) > 0) {
                    return scripts[i].src.substring(
                        withScheme ? 0 : scripts[i].src.indexOf('//') + 2,
                        scripts[i].src.lastIndexOf('/'));
                }
            }
            return undefined;
        };
    }
})();
```

13.3 Launching the App

All OVE applications can be launched using the Launcher UI, the Python Client Library, and the OVE APIs. The API used to launch an application is the same for all applications, but the data that is passed into it is application-specific.

To launch the maps app and display a map using the OVE APIs:

Linux/Mac:

```
curl --header "Content-Type: application/json" --request POST --data '{"app": {"url":
↪"http://OVE_CORE_HOST:PORT/app/maps","states": {"load": {"center": ["-11137.
↪70850550061", "6710544.04980525"], "resolution": "77", "zoom": "12", "enabledLayers
↪":["23"], "scripts":["http://my.domain/customLayer.js"]}}}, "space": "OVE_SPACE", "h
↪": 500, "w": 500, "y": 0, "x": 0}' http://OVE_CORE_HOST:PORT/section
```

Windows:

```
curl --header "Content-Type: application/json" --request POST --data "{\"app\": {\
↪"url\": \"http://OVE_CORE_HOST:PORT/app/maps\", \"states\": {\"load\": {\"center\":␣
↪[\"-11137.70850550061\", \"6710544.04980525\"], \"resolution\": \"77\", \"zoom\": \
↪"12\", \"enabledLayers\":[\"23\"], \"scripts\":[\"http://my.domain/customLayer.js\
↪"]}}}, \"space\": \"OVE_SPACE\", \"h\": 500, \"w\": 500, \"y\": 0, \"x\": 0}" http:/
↪/OVE_CORE_HOST:PORT/section
```

13.4 Controlling the App

The controller of the app can be loaded by accessing the URL `http://OVE_CORE_HOST:PORT/app/maps/control.html?oveSectionId=SECTION_ID&layers=23`. The `layers` parameter in the URL is optional and can have more than one value at a time separated by commas. The controller supports panning and zooming of maps.

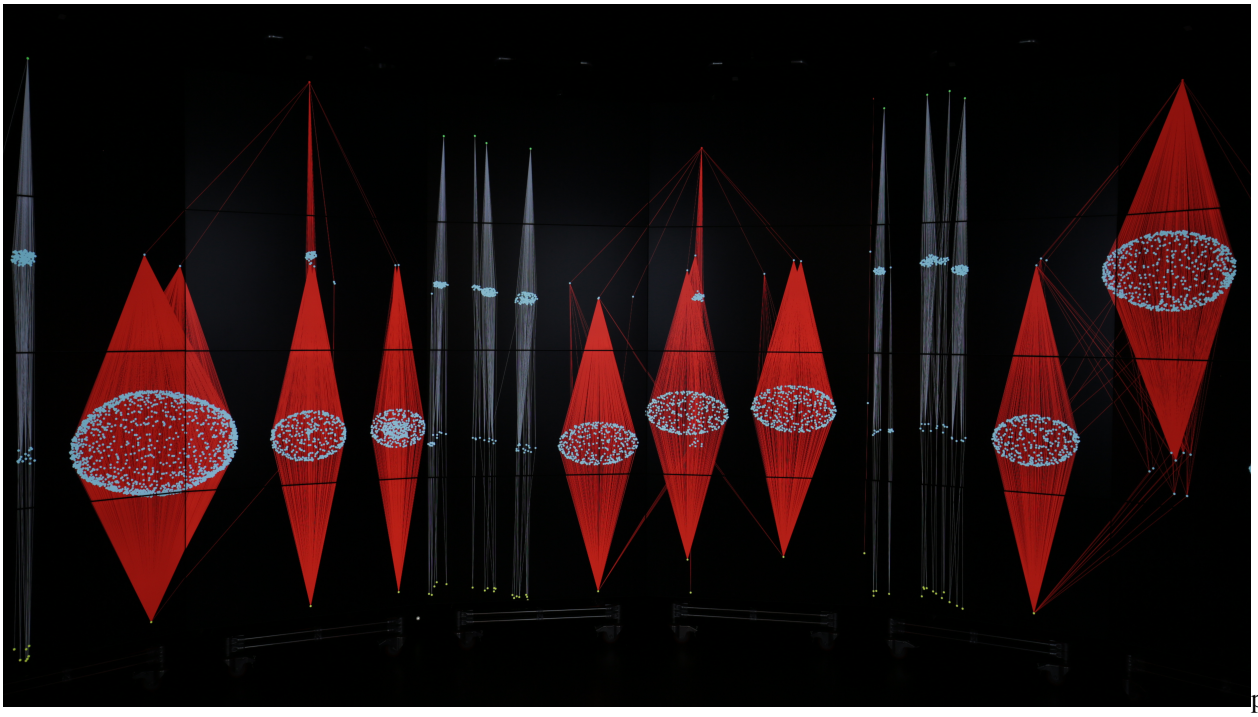
13.5 Key considerations when using the App

All considerations when using this app are directly related to its reliability and performance:

1. The maps app tends to load many tiles on screens with higher resolutions and tile servers that are slow and remote tend to perform very poorly compared to servers that are much faster and locally hosted.
2. JavaScript executed to load custom overlays must not introduce any performance bottlenecks or execute code that has a negative impact on the reliability of OVE.
3. If the Map Layers configuration is specified as a URL (rather than being directly embedded in the `config.json` file), then this must be available before the server-side of this app is launched using PM2 or Docker.

CHAPTER 14

Networks App



photograph of the networks app

This app supports visualisation of networks with node-link diagrams using the OVE framework. It is based on Sigma, a JavaScript library dedicated to drawing graphs. This supports datasets specified in Sigma's JSON format or GEXF. Sigma supports Canvas, SVG and WebGL rendering.

Seen above is an image of the networks app displaying a network topology graph photographed at the Imperial College Data Science Institute's Data Observatory.

14.1 Application State

The state of this app has a format similar to:

```
{
    "jsonURL": "data/sample.json",
    "settings": {
        "autoRescale": true,
        "clone": false,
        "defaultNodeColor": "#ec5148"
    },
    "renderer": "canvas"
}
```

The `jsonURL` property should be replaced with with `gexfURL` property depending on the format in which the dataset is specified. Optionally, a `url` property can be set instead of the `jsonURL` and `gexfURL` properties. Based on its extension, it will be used to set either of the `jsonURL` and `gexfURL` properties. If it had no extension at all, it will be ignored.

Information on all available `settings` can be found in Sigma documentation. The `renderer` property is optional and defaults to `webgl`.

If the content is available on a Neo4j database the state of this app needs to have a format similar to:

```
{
    "neo4j": {
        "x": { "min": 0, "max": 100 },
        "y": { "min": 0, "max": 100 },
        "db": { "url": "http://localhost:7474", "user": "neo4j", "password": "admin" }
↪,
        "query": "MATCH (n) WHERE n.y >= Y_MIN AND n.y < Y_MAX AND n.x >= X_MIN AND n.
↪x < X_MAX RETURN n LIMIT 100"
      },
    "settings": {
        "autoRescale": true,
        "clone": false,
        "rescaleIgnoreSize": true,
        "skipErrors": true
    },
    "renderer": "canvas"
}
```

A Cypher query should be provided as the `query` along with the database connection details as the value of the `db` property. The `min` and `max` values along the `x` and `y` axes also needs to be provided if the graph coordinates does not map to pixel coordinates on the screens.

The optional `boundingBoxColor` property specifies the colour of the bounding box. The optional `boundingBoxNodeSize` property specifies the size of the nodes used to draw the bounding box. The optional `disableTiling` disables tiling of the graph on a per-client basis. All these properties must be defined within the `neo4j` property.

Information on all available `settings` can be found in Sigma documentation. The `renderer` property is optional and defaults to `webgl`.

14.2 Launching the App

All OVE applications can be launched using the Launcher UI, the Python Client Library, and the OVE APIs. The API used to launch an application is the same for all applications, but the data that is passed into it is application-specific.

To launch the networks app and display a node-link diagram using the OVE APIs:

Linux/Mac:

```
curl --header "Content-Type: application/json" --request POST --data '{"app": {"url":
↪"http://OVE_CORE_HOST:PORT/app/networks","states": {"load": {"jsonURL": "data/
↪sample.json", "settings": { "autoRescale": true, "clone": false, "defaultNodeColor
↪": "#ec5148"}, "renderer": "canvas"}}}, "space": "OVE_SPACE", "h": 500, "w": 500, "y
↪": 0, "x": 0}' http://OVE_CORE_HOST:PORT/section
```

Windows:

```
curl --header "Content-Type: application/json" --request POST --data "{\"app\": {\
↪"url\": \"http://OVE_CORE_HOST:PORT/app/networks\", \"states\": {\"load\": {\
↪"jsonURL\": \"data/sample.json\", \"settings\": { \"autoRescale\": true, \"clone\":␣
↪false, \"defaultNodeColor\": \"#ec5148\"}, \"renderer\": \"canvas\"}}}, \"space\": \
↪"OVE_SPACE\", \"h\": 500, \"w\": 500, \"y\": 0, \"x\": 0}" http://OVE_CORE_
↪HOST:PORT/section
```

The networks app has a transparent background. If required, a background colour of choice can be set using the Background Utility provided by OVE.

If the networks app is used to display static networks no further controlling would be required after the node-link diagram has been loaded. The app provides a controller and exposes API that can used to control interactive graphs.

14.3 Controlling the App

The controller of the app can be loaded by accessing the URL `http://OVE_CORE_HOST:PORT/app/networks/control.html?oveSectionId=SECTION_ID`.

The app's API also exposes operations such as `showOnly`, `color`, `labelNodes`, `neighborsOf` and `reset`. These operations can be executed on a per-network basis or across all networks.

The implementation makes it possible to query on all properties of nodes and edges. The filters used must confirm to the OData v3.0 specification and the colors used must have the format `rgb(x, y, z)` (where x, y, z are integers in the range 0-255).

Sigma supports chaining of multiple filters and operations such as `showOnly` and `color` operations combine node and edge filters if provided together: which has the effect of a logical AND.

To show only the nodes having a size greater than or equal to 2, using OVE APIs:

```
curl  --request POST http://OVE_CORE_HOST:PORT/app/networks/operation/showOnly?
↪filter=size%20ge%202
```

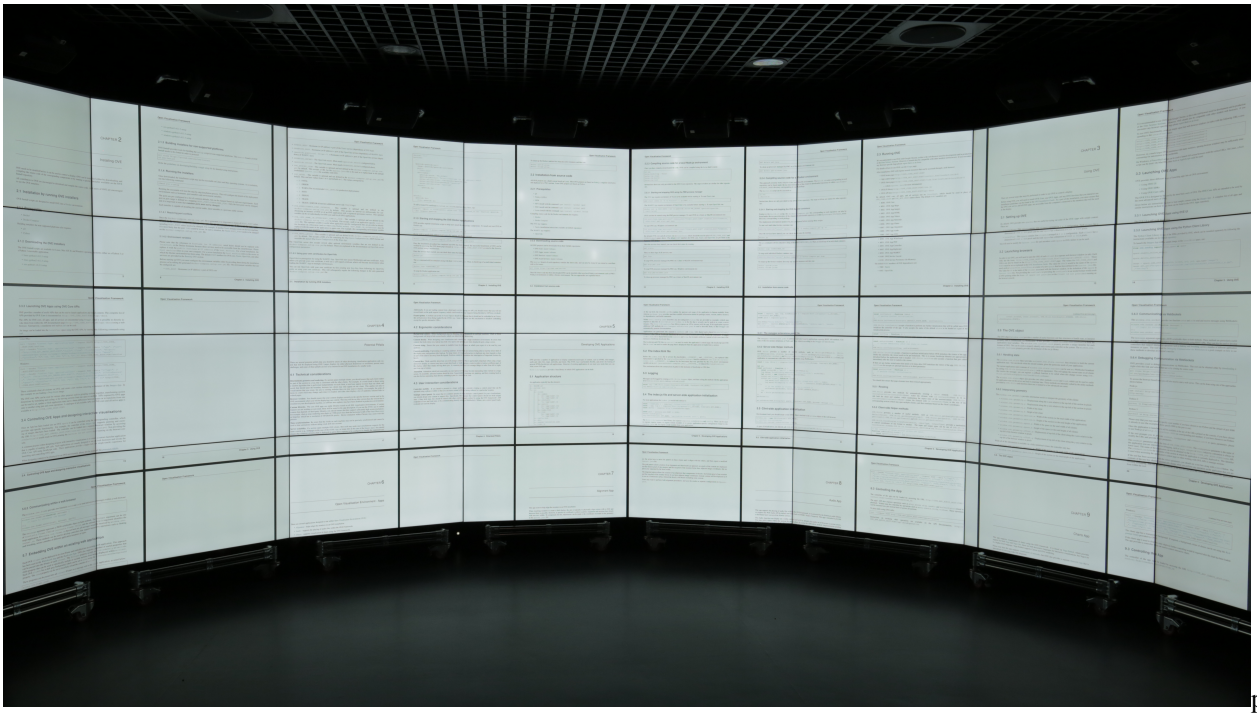
To reset the network to its original state:

```
curl  --request POST http://OVE_CORE_HOST:PORT/app/networks/operation/reset
```

Instructions on invoking all operations are available on the API Documentation, `http://OVE_CORE_HOST:PORT/app/networks/api-docs/#operation`.

CHAPTER 15

PDF App


photograph of the PDF app

This app supports displaying PDF documents using the OVE framework. It is based on PDF.js, a Portable Document Format (PDF) viewer that is built with HTML5.

Seen above is an image of the PDF app displaying the OVE documentation in PDF format photographed at the Imperial College Data Science Institute's Data Observatory.

15.1 Application State

The state of this app has a format similar to:

```
{
    "url": "https://raw.githubusercontent.com/mozilla/pdf.js/master/test/pdfs/
↪TAMReview.pdf",
    "settings": {
        "scale": 2,
        "offset": {
            "x": 0,
            "y": 0
        },
        "pageGap": 50,
        "startPage": 1,
        "endPage": 10,
        "scrolling": "horizontal"
    }
}
```

The app accepts a `url` of a PDF document. All `settings` are optional. The `scale` property defines the scale at which each page is rendered. It can be used to automatically zoom contents of a PDF when it loads. Similarly, `offset` can be used to automatically pan contents of a PDF when it loads. The `pageGap` is the number of pixels between each adjacent page. The app also accepts a `startPage` and `endPage` which can be used to limit the number of pages rendered. It will automatically compute the number of pages and decide on whether it is best to scroll horizontally or vertically depending on the dimensions of the section. This behaviour can be overridden by defining the `scrolling` property to be either `vertical` or `horizontal`.

15.2 Launching the App

All OVE applications can be launched using the Launcher UI, the Python Client Library, and the OVE APIs. The API used to launch an application is the same for all applications, but the data that is passed into it is application-specific.

To launch the PDF app and display a PDF document using the OVE APIs:

```
curl --header "Content-Type: application/json" --request POST --data '{"app": {"url":
↪"http://OVE_CORE_HOST:PORT/app/pdf","states": {"load": {"url": "https://raw.
↪githubusercontent.com/mozilla/pdf.js/master/test/pdfs/TAMReview.pdf"}}}, "space":
↪"OVE_SPACE", "h": 500, "w": 500, "y": 0, "x": 0}' http://OVE_CORE_HOST:PORT/section
```

Windows:

```
curl --header "Content-Type: application/json" --request POST --data "{\"app\": {\
↪"url\": \"http://OVE_CORE_HOST:PORT/app/pdf\", \"states\": {\"load\": {\"url\": \
↪"https://raw.githubusercontent.com/mozilla/pdf.js/master/test/pdfs/TAMReview.pdf\"}}
↪}, \"space\": \"OVE_SPACE\", \"h\": 500, \"w\": 500, \"y\": 0, \"x\": 0}" http://
↪OVE_CORE_HOST:PORT/section
```

If the PDF app is used to display static PDF documents no further controlling would be required after the PDF has been loaded. The app provides a controller that can used to pan and zoom PDF documents.

15.3 Controlling the App

The controller of the app can be loaded by accessing the URL `http://OVE_CORE_HOST:PORT/app/pdf/control.html?oveSectionId=SECTION_ID`.

CHAPTER 16

Replicator App

This app is capable of replicating content from an OVE `space` (or a portion of a space), a `group` or one or more individual `sections`. The displayed content may be replicated from either from the same OVE instance as the replicator app, or a remote instance. It can render content at different pixel dimensions by scaling whilst preserving retaining the same aspect ratios.

The replicator app is instantiated just like any other app. However, it does not have a controller, and therefore behaves slightly differently to other apps at runtime.

16.1 Application State

The state of this app has a format similar to:

```
{
    "mode": "space",
    "spaceName": "LocalNine",
    "groupIds": [0, 1],
    "sectionIds": [2, 3, 4],
    "crop": {
        "x": 0,
        "y": 0,
        "w": 100,
        "h": 100,
    },
    "oveHost": "localhost:8080",
    "border": "solid gold",
    "background": "#222"
}
```

16.1.1 Selecting what to replicate

The `mode` property is mandatory and should have a value of `space`, `group`, or `section`.

If the `mode` has been set to `space`, the `spaceName` property should generally be provided. It must be set if content is replicated from a remote OVE instance, and will otherwise default to the name of the space in which the replicator section is created (creating a mini-map view of the space).

If the `mode` is `group`, then the `groupIds` property must be provided.

If the `mode` is `section`, then the `sectionIds` property must be provided.

If the `mode` is `group` or `section`, then the `spaceName` property will default to the name of the space that contains the most sections.

The optional `oveHost` property can be set in order to connect to a remote OVE instance; if it is not set, content will be replicated from the same OVE instance that the replicator app is created on.

Regardless of which `mode` is used, the optional `crop` property can be set: this defines a rectangular region, and nothing outside this region is replicated. The contents of the region selected by `spaceName`/`groupIds`/`sectionIds` will be scaled proportionally so that the crop rectangle fits within the replicator app's own dimensions (defined as `w` and `h` of the geometry when creating the app).

The optional `border` property can be used to set the border-style and border-color (but not border-width) of the replicated region. This can be useful when the replication is deployed as an overlay (for example, as a mini-map).

Setting a `border` will also make the replicator app's background opaque. The optional `background` property can be set to change the colour and the opacity of the background.

16.2 Launching the App

All OVE applications can be launched using the Launcher UI, the Python Client Library, and the OVE APIs. The API used to launch an application is the same for all applications, but the data that is passed into it is application-specific.

To replicate content using the OVE APIs:

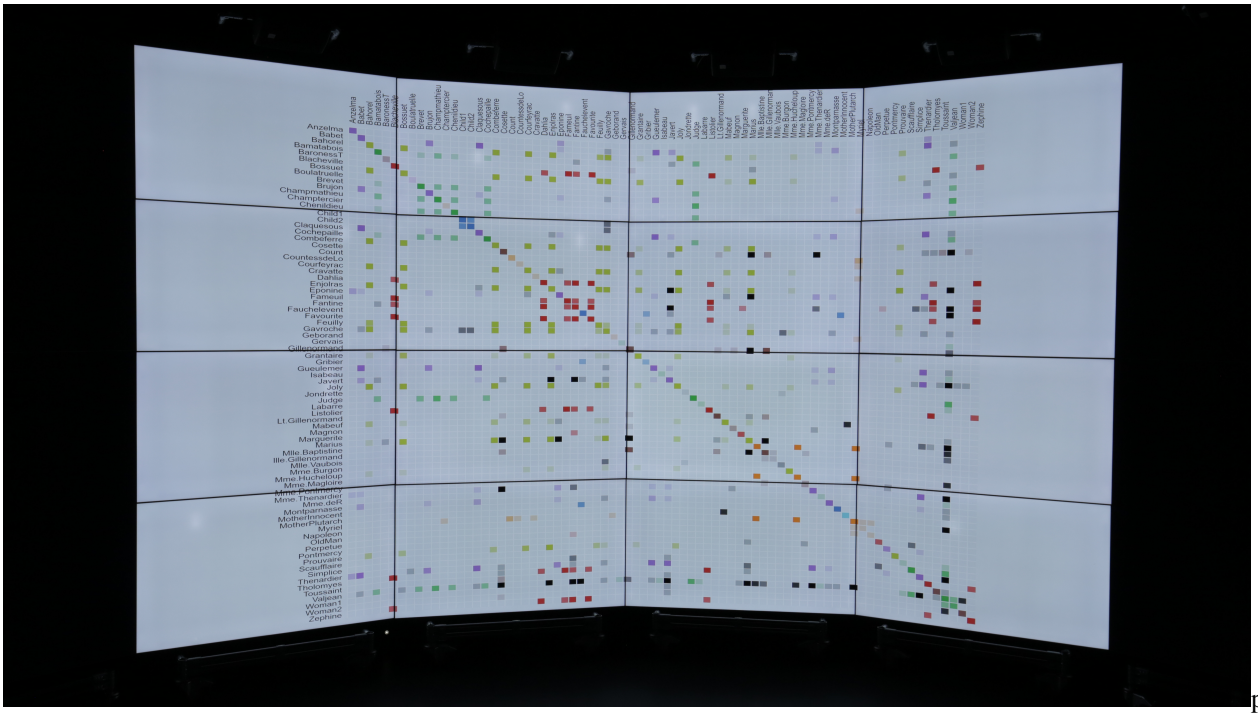
```
curl --header "Content-Type: application/json" --request POST --data '{"app": {"url":
↪"http://OVE_CORE_HOST:PORT/app/replicator","states": {"load": {"mode": "space",
↪"spaceName": "LocalNine"}}}, "space": "OVE_SPACE", "h": 500, "w": 500, "y": 0, "x":␣
↪0}' http://OVE_CORE_HOST:PORT/section
```

Windows:

```
curl --header "Content-Type: application/json" --request POST --data "{\"app\": {\
↪"url\": \"http://OVE_CORE_HOST:PORT/app/replicator\", \"states\": {\"load\": {\
↪"mode\": \"space\", \"spaceName\": \"LocalNine\"}}}, \"space\": \"OVE_SPACE\", \"h\
↪": 500, \"w\": 500, \"y\": 0, \"x\": 0}" http://OVE_CORE_HOST:PORT/section
```

CHAPTER 17

SVG App



photograph of the SVG app

This app supports rendering SVG using the OVE framework. It is based on Tuoris, a middleware for distributed SVG rendering. An installation of Tuoris is required to use the SVG app. More information on installing Tuoris can be found in the OVE installation guide.

The SVG app depends on an environment variable named `TUORIS_HOST`, that points to the URL at which the Tuoris instance runs.

Seen above is an image of the SVG app displaying the Les Misérables co-occurrence matrix diagram by Mike Bostok photographed at the Imperial College Data Science Institute's Data Observatory.

17.1 Application State

The state of this app has a format similar to:

```
{
    "url": "https://upload.wikimedia.org/wikipedia/commons/6/6c/Trajans-Column-lower-
↪animated.svg"
}
```

17.2 Launching the App

All OVE applications can be launched using the Launcher UI, the Python Client Library, and the OVE APIs. The API used to launch an application is the same for all applications, but the data that is passed into it is application-specific.

To launch the SVG app and display an SVG using the OVE APIs:

Linux/Mac:

```
curl --header "Content-Type: application/json" --request POST --data '{"app": {"url":
↪"http://OVE_CORE_HOST:PORT/app/svg","states": {"load": {"url": "https://upload.
↪wikimedia.org/wikipedia/commons/6/6c/Trajans-Column-lower-animated.svg"}}}, "space
↪": "OVE_SPACE", "h": 500, "w": 500, "y": 0, "x": 0}' http://OVE_CORE_HOST:PORT/
↪section
```

Windows:

```
curl --header "Content-Type: application/json" --request POST --data "{\"app\": {\
↪"url\": \"http://OVE_CORE_HOST:PORT/app/svg\", \"states\": {\"load\": {\"url\": \
↪"https://upload.wikimedia.org/wikipedia/commons/6/6c/Trajans-Column-lower-animated.
↪svg\"}}}, \"space\": \"OVE_SPACE\", \"h\": 500, \"w\": 500, \"y\": 0, \"x\": 0}"␣
↪http://OVE_CORE_HOST:PORT/section
```

The SVG app has a transparent background. If required, a background colour of choice can be set using the Background Utility provided by OVE.

If the SVG app is used to display static SVGs no further controlling would be required after the SVG has been loaded. The app provides a controller that can used to control interactive SVGs.

17.3 Controlling the App

The controller of the app can be loaded by accessing the URL `http://OVE_CORE_HOST:PORT/app/svg/control.html?oveSectionId=SECTION_ID`.

CHAPTER 18

Videos App

click to play the video of the videos app

This app supports playing videos using the OVE framework. It provides a generic player for any video (`HTML5`), and a player based on the YouTube IFrame Player API (`YouTube`).

Seen above is a video of the videos app playing the 2030.wikimedia.org YouTube video in 4K resolution published by the Wikimedia Foundation recorded in HD resolution at the Imperial College Data Science Institute's Data Observatory.

18.1 Application State

The state of this app has a format similar to:

```
{
    "url": "http://www.youtube.com/embed/XY3NP4JHXZ4"
}
```

The player is selected automatically based on the URL: Any YouTube URL uses the YouTube player and all other URLs use the HTML5 player. By default, all videos will play muted. An optional property, `unmuted` can be set to `true` to play videos unmuted.

18.2 Launching the App

All OVE applications can be launched using the Launcher UI, the Python Client Library, and the OVE APIs. The API used to launch an application is the same for all applications, but the data that is passed into it is application-specific.

To launch the SVG app and load a video using the OVE APIs:

Linux/Mac:

```
curl --header "Content-Type: application/json" --request POST --data '{"app": {"url":
↪"http://OVE_CORE_HOST:PORT/app/videos","states": {"load": {"url": "http://www.
↪youtube.com/embed/XY3NP4JHXZ4"}}}, "space": "OVE_SPACE", "h": 500, "w": 500, "y": 0,
↪ "x": 0}' http://OVE_CORE_HOST:PORT/section
```

Windows:

```
curl --header "Content-Type: application/json" --request POST --data "{\"app\": {\
↪"url\": \"http://OVE_CORE_HOST:PORT/app/videos\", \"states\": {\"load\": {\"url\": \
↪"http://www.youtube.com/embed/XY3NP4JHXZ4\"}}}, \"space\": \"OVE_SPACE\", \"h\":␣
↪500, \"w\": 500, \"y\": 0, \"x\": 0}" http://OVE_CORE_HOST:PORT/section
```

18.3 Controlling the App

The controller of the app can be loaded by accessing the URL `http://OVE_CORE_HOST:PORT/app/videos/control.html?oveSectionId=SECTION_ID`.

The app's API also exposes operations such as `play`, `pause`, `stop`, `mute`, `seekTo` and `bufferStatus`. These operations can be executed on a per-video basis or across all videos.

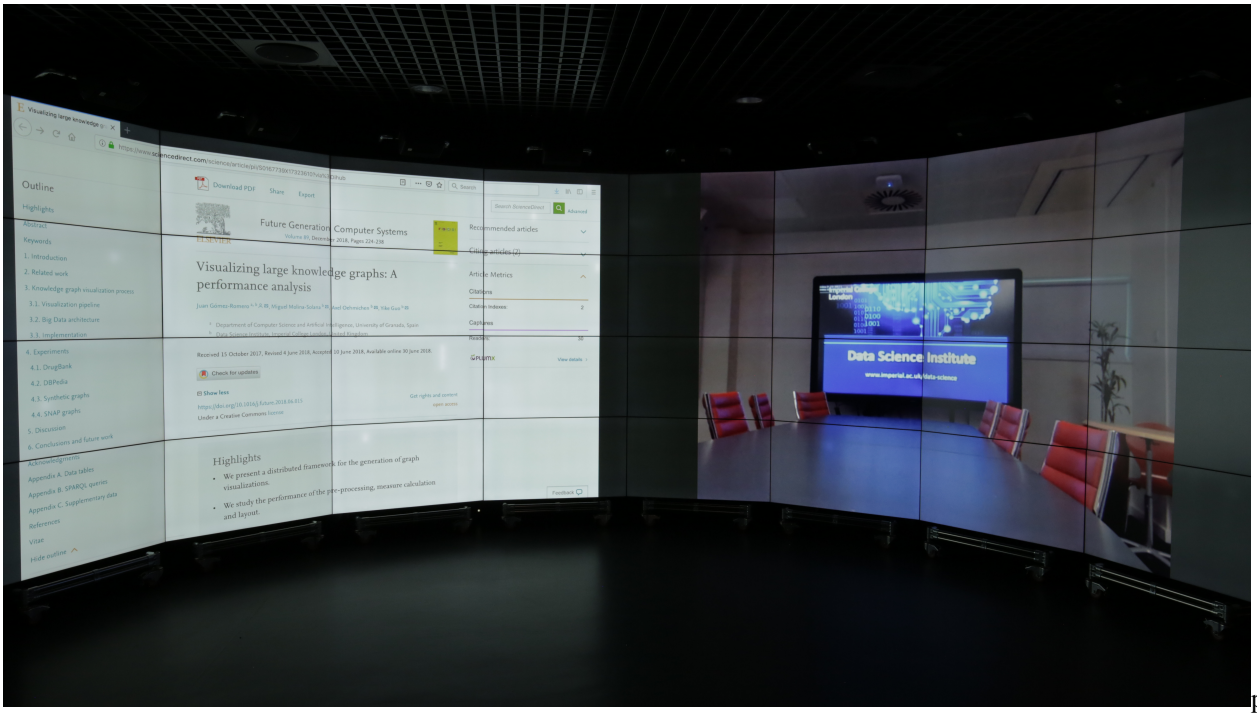
To play videos using OVE APIs:

```
curl  --request POST http://OVE_CORE_HOST:PORT/app/videos/operation/play
```

Instructions on invoking other operations are available on the API Documentation, `http://OVE_CORE_HOST:PORT/app/videos/api-docs/#operation`.

CHAPTER 19

WebRTC App



photograph of the WebRTC app

This app supports one-to-one, one-to-many and many-to-many videoconferencing and screen sharing using the OVE framework. It is based on OpenVidu, a platform designed to facilitate the addition of WebRTC videoconferencing into existing web and mobile applications. An installation of OpenVidu is required to use the WebRTC app. More information on installing OpenVidu can be found in the OVE installation guide.

The WebRTC app depends on an environment variable named `OPENVIDU_HOST`, that provides the hostname and port number at which the OpenVidu instance runs. The OpenVidu Secret must be provided by setting the `OPENVIDU_SECRET` environment variable on the production server or by editing the `config.json` file which must reside only on the production server.

Seen above is an image of the WebRTC app displaying a shared screen with the Visualizing large knowledge graphs: A performance analysis publication by Gómez-Romero et al. and a vidoconferencing call initiated at the board room of the Imperial College Data Science Institute's photographed at the Imperial College Data Science Institute's Data Observatory.

19.1 Application State

The state of this app has a format similar to:

```
{
    "sessionId": "random",
    "maxSessions": 8
}
```

The `sessionId` property can be a new or existing OpenVidu session identifier: if it is set to `random`, then the controller will generate a random `sessionId`. It is also possible to provide a `url` property which contains the `sessionId`: in such situations, the controller will extract the last path segment and use this as the `sessionId`.

The optional `maxSessions` property defines the maximum number of external users that can connect to the session.

19.2 Launching the App

All OVE applications can be launched using the Launcher UI, the Python Client Library, and the OVE APIs. The API used to launch an application is the same for all applications, but the data that is passed into it is application-specific.

To launch the WebRTC app using the OVE APIs:

Linux/Mac:

```
curl --header "Content-Type: application/json" --request POST --data '{"app": {"url":
↪"http://OVE_CORE_HOST:PORT/app/webrtc","states": {"load": {"sessionId": "random"}}},
↪ "space": "OVE_SPACE", "h": 500, "w": 500, "y": 0, "x": 0}' http://OVE_CORE_
↪HOST:PORT/section
```

Windows:

```
curl --header "Content-Type: application/json" --request POST --data "{\"app\": {\
↪"url\": \"http://OVE_CORE_HOST:PORT/app/webrtc\", \"states\": {\"load\": {\
↪"sessionId\": \"random\"}}}, \"space\": \"OVE_SPACE\", \"h\": 500, \"w\": 500, \"y\
↪": 0, \"x\": 0}" http://OVE_CORE_HOST:PORT/section
```

19.3 Controlling the App

A vidoconferencing call must first be initialized using the controller, which is accessible at `http://OVE_CORE_HOST:PORT/app/webrtc/control.html?oveSectionId=SECTION_ID`.

Participants can then join this call by connecting to the OpenVidu client (accessible by opening `OPENVIDU_HOST` with a web browser) and providing the `sessionId` as the name of the call room.

CHAPTER 20

Whiteboard App

This app creates a whiteboard that can be used within the OVE framework. The app creates a collaborative canvas that accepts handwriting and text input.

The app's background is transparent, so it can be used to annotate content displayed using a different OVE App.

The controller UI design of the whiteboard app is based on codoodler.

20.1 Launching the App

All OVE applications can be launched using the Launcher UI, the Python Client Library, and the OVE APIs. The API used to launch an application is the same for all applications, but the data that is passed into it is application-specific.

To launch the whiteboard app and display it using the OVE APIs:

Linux/Mac:

```
curl --header "Content-Type: application/json" --request POST --data '{"app": {"url":
↪"http://OVE_CORE_HOST:PORT/app/whiteboard"}, "space": "OVE_SPACE", "h": 500, "w":␣
↪500, "y": 0, "x": 0}' http://OVE_CORE_HOST:PORT/section
```

Windows:

```
curl --header "Content-Type: application/json" --request POST --data "{\"app\": {\
↪"url\": \"http://OVE_CORE_HOST:PORT/app/whiteboard\"}, \"space\": \"OVE_SPACE\", \
↪"h\": 500, \"w\": 500, \"y\": 0, \"x\": 0}" http://OVE_CORE_HOST:PORT/section
```

20.2 Controlling the App

The controller of the app can be loaded by accessing the URL `http://OVE_CORE_HOST:PORT/app/whiteboard/control.html?oveSectionId=SECTION_ID`.

CHAPTER 21

Open Visualisation Environment - Services

The OVE Services repository contains microservices that provide non-core functionality for Open Visualisation Environment (OVE).

It currently includes:

- An In-Memory Persistence Provider that acts as a key-value store
- A Layout Service that converts a layout expressed in abstract units (whose interpretation may depend on the dimensions of the space in which it is applied) to a static layout expressed in absolute pixel positions.

The OVE Services are intended to be used as part of a complete installation of OVE. The OVE Documentation provides installation instructions and a user guide.

CHAPTER 22

OVE Persistence Service - In-Memory

This service provides persistence for the OVE framework using an in-memory storage implementation. This is also the most straightforward persistence service implementation and is not highly available.

22.1 Registering a Persistence Service

All persistence services for OVE implements a common API and can be registered in the same way. However, the persistence services may have their own configurations, concerning their underlying storage implementations. Such configuration should be made according to the respective persistence service's documentation.

A persistence service can be registered with OVE using the following API:

Linux/Mac:

```
curl --header "Content-Type: application/json" --request POST --data '{"url": "http://
↪OVE_CORE_HOST:PORT/service/persistence/inmemory"}' http://OVE_CORE_HOST:PORT/
↪persistence
```

Windows:

```
curl --header "Content-Type: application/json" --request POST --data "{\"url\": \
↪"http://OVE_CORE_HOST:PORT/service/persistence/inmemory\"}" http://OVE_CORE_
↪HOST:PORT/persistence
```

A persistence service can be unregistered from OVE using the following API:

Linux/Mac/Windows:

```
curl --header "Content-Type: application/json" --request DELETE  http://OVE_CORE_
↪HOST:PORT/persistence
```

A persistence service can be registered with any OVE app using the following API:

Linux/Mac:

```
curl --header "Content-Type: application/json" --request POST --data '{"url": "http://
↪OVE_CORE_HOST:PORT/service/persistence/inmemory"}' http://OVE_CORE_HOST:PORT/app/
↪OVE_APP_NAME/persistence
```

Windows:

```
curl --header "Content-Type: application/json" --request POST --data "{\"url\": \
↪"http://OVE_CORE_HOST:PORT/service/persistence/inmemory\"}" http://OVE_CORE_
↪HOST:PORT/app/OVE_APP_NAME/persistence
```

A persistence service can be unregistered from any OVE App using the following API:

Linux/Mac/Windows:

```
curl --header "Content-Type: application/json" --request DELETE  http://OVE_CORE_
↪HOST:PORT/app/OVE_APP_NAME/persistence
```

CHAPTER 23

Open Visualisation Environment - UI

This repository contains a collection of UI components designed to be used with Open Visualisation Environment (OVE).

These components are all browser based, and provide an alternative to using the OVE APIs directly or through one of the OVE SDKs.

The currently available components are:

- Demo, which launches demo scenarios on OVE
- Launcher, which launches OVE applications
- Preview, which provides a preview of an OVE Space
- Status, which displays status of all OVE components

OVE must be installed before the OVE UIs can be used. The OVE Documentation provides installation instructions and a user guide.

CHAPTER 24

Demo UI

This React web application launches demo scenarios on OVE. The demo UI can be accessed by opening the URL `http://OVE_CORE_HOST:PORT/ui/demo` with a web browser. Simply loading this page and taking no action will leading to the displaying of the OVE splash screen followed by the details of the OVE clients.

CHAPTER 25

Launcher UI

This React web application provides a wizard that will guide you through launching an OVE App step-by-step.

It will also display the curl command that can be used to repeat this action at the command line.

The launcher UI can be accessed by opening the URL `http://OVE_CORE_HOST:PORT/ui/launcher` with a web browser. Before using the launcher UI, you will need to host the content that you plan to load into the respective applications. For more information on hosting content for OVE, see the Usage page.

25.1 Using the Launcher UI

The launcher UI takes users through a step-by-step process to select, configure and deploy an application into an OVE space.

It is mostly self-explanatory, but there are a few things to note:

1. The launcher expects the chosen application to be running before you proceed to the second step. If it is not, the launcher will report a `selected application is not available` error.
2. The launcher lists all spaces available within the pre-configured OVE environment. Not all instances of OVE may have the same spaces, and if you cannot find the space that you were expecting to select, you may be connecting to the wrong OVE instance. In such a situation, please validate the `REACT_APP_OVE_HOST` configuration that you have provided when installing the launcher.
3. The launcher can only be used to create sections that lie entirely within a space. If your chosen goemetry does not lie entirely within a space, you will be shown a form validation error.
4. The third step for configuring the application state for a section varies according to the chosen application. It is not displayed in the case of the Controller App and the Replicator App. It is displayed for the Alignment App and the Whiteboard App, but a form is not available to the user. For the Maps App and the WebRTC App, a choice of mode is not available and only pre-loaded named states can be selected. The complete form is available for all other apps providing options for selecting a pre-loaded named state or providing a URL for loading a new asset.

5. The fourth step for reviewing the state configuration and operation details provides the users with the opportunity to make changes to the JSON configuration that will be loaded when the application is launched. This option is available for all apps except for the Alignment App and the Whiteboard App.
6. The fourth step also lets users decide whether all existing sections in a space must be deleted or retained before the new section is created when launching the chosen app.
7. For all apps except for the Replicator App users can choose whether the application-specific controller is displayed on the browser. This choice is selected by default for the Alignment, Controller, Maps, SVG, WebRTC and Whiteboard apps.
8. The last two steps are displayed for all types of applications. The curl command to delete the existing sections on a space will only be available if the corresponding option was available and selected in the fourth step. The final step is merely a confirmation and the user is not expected to take further action unless if they wanted to launch another app, launch the application-specific controller or preview the space which should include the newly launched app.

CHAPTER 26

Preview UI

This React web application can be used to preview an OVE Space. It also allows users to download corresponding commands and configurations that can be used to recreate the same space using tools such as curl or SDKs such as the Python Client Library.

The preview UI can be accessed by opening the URL `http://OVE_CORE_HOST:PORT/ui/preview?oveSpace=<SPACE_NAME>` with a web browser (replacing `<SPACE_NAME>` with the name of the space that you wish to view).

CHAPTER 27

Status UI

This React web application displays runtime status (and availability information) of all OVE components in a given installation. The status UI can be accessed by opening the URL `http://OVE_CORE_HOST:PORT/ui/status` with a web browser.

CHAPTER 28

Open Visualisation Environment - Asset Manager and Proxy

This repository contains an Asset Manager (AM) to manage data sets for access to an installation of Open Visualisation Environment (OVE), as well as a high speed proxy to provide authenticated access to data from a range of sources.

28.1 Concepts

The Asset Manager stores files in an S3 compatible object **Store**, such as Minio. One instance of the Asset Manager can work with multiple stores.

An **Asset** is a collection of one or more **files** that can be treated as a single unit and versioned, processed and displayed together. Each asset has associated metadata (e.g. processing state, name, description, tags, etc.).

Assets (and non-asset objects in JSON format) can be grouped together into a **Project**. While the AM does not assign any specific meaning to a project, and allows you to group assets by any criteria, other tools may interpret projects as indicating that particular content should be displayed together (e.g., a gallery might display all of the assets in a selected project).

Each time any file in an asset is updated, a new **Version** of the whole asset is recorded. Previous versions of the asset are retained, and can still be accessed.

Workers can be scheduled to asynchronously perform a task performed on a file, such as converting it to a new file format. Workers operate non-destructively and do not modify or delete uploaded files, so do not create new versions of an asset when they run.

28.2 Components

The Asset Manager is split into 3 parts:

- The **Backend** exposes a full RESTful API for operations on projects, assets and files. This service can work independently from all the other services.
- The **Read proxy** exposes an HTTP endpoint to read individual files off the object store. The AM Read Proxy is authenticated by the same rules as the Backend service.

- The **UI** exposes a User Interface for most of the operations performed by the Backend service.

The **Workers** asynchronously perform additional tasks on assets:

- **Unzip**: please see the worker documentation for more details
- **DeepZoomImage**: please see the worker documentation for more details
- **Tulip network layout**: please see the worker documentation for more details

CHAPTER 29

Installing OVE Asset Manager

29.1 Installation by running OVE installers

The easiest option to run all the services is to use the OVE Installer that allows you to configure all the services interactively.

The installer creates a Docker Compose file for the Asset Manager services (the asset manager service, UI, asset workers, and MinIO instance), separate from the docker-compose file for the other OVE services.

29.1.1 S3 Store - MinIO Configuration

This step is **optional**, and can be skipped if you already have a Amazon S3 compatible object store.

The Asset Manager was tested against MinIO, an open object storage server.

The docker-compose configuration to spin up a MinIO instance is:

```
version: '3'
services:
  minio-store:
    image: minio/minio:latest
    ports:
    - "9000:9000"
    volumes:
      - "minio-storage-data:/data"
    environment:
      MINIO_ACCESS_KEY: "MINIO_ACCESS_KEY"
      MINIO_SECRET_KEY: "MINIO_SECRET_KEY"
    command: server /data

volumes:
  minio-storage-data:
```

While this docker setup is perfect for testing, it is recommended to use a bare-metal install in production. Please refer the MinIO documentation for more details.

29.2 Alternative installation for a Docker environment without using OVE installers

It is possible to run the Asset Manager services with Docker without using docker-compose. However, this guide uses docker-compose, as this allows the configuration to be expressed as blocks of YAML, which is clearer than listing long Docker commands with many arguments.

In the example **docker-compose.yml** below, the service is configured for production use. If you wish to enable different options please check the documentation for each service.

Note: Please set the **service version** parameter before running the config.

The Asset Manager service requires a config file to run. A template of the config file can be found in **config/credentials.template.json**; this can be copied to **config/credentials.json** and modified as required (refer to the Asset Manager Backend configuration for more details).

If you do not have an existing S3 compatible object store, please read the *MinIO configuration* section.

```
version: '3'
services:
  ovehub-ove-asset-manager-service:
    image: ovehub/ove-asset-manager-service:stable
    ports:
      - "6080:6080"
    volumes:
      - ./config/:/code/config/:ro
    environment:
      GUNICORN_THREADS: "8"
      SERVICE_LOG_LEVEL: "info"

  ovehub-ove-asset-manager-proxy:
    image: ovehub/ove-asset-manager-proxy:stable
    ports:
      - "6081:6081"
    volumes:
      - ./config/:/code/config/:ro
    environment:
      GUNICORN_THREADS: "8"
      SERVICE_LOG_LEVEL: "info"

  ovehub-ove-asset-manager-worker-zip:
    image: ovehub/ove-asset-manager-worker-zip:stable
    environment:
      SERVICE_LOG_LEVEL: "info"
      SERVICE_AM_HOSTNAME: "ovehub-ove-asset-manager-service"
      SERVICE_AM_PORT: "6080"

  ovehub-ove-asset-manager-worker-gigaimage:
    image: dzi
    environment:
      SERVICE_LOG_LEVEL: "info"
      SERVICE_AM_HOSTNAME: "ovehub-ove-asset-manager-service"
      SERVICE_AM_PORT: "6080"
```

(continues on next page)

(continued from previous page)

```
  ovehub-ove-asset-manager-ui:
    image: ovehub/ove-asset-manager-ui:stable
    ports:
      - "6060:6060"
    environment:
      SERVICE_LOG_LEVEL: "info"
      SERVICE_AM_HOSTNAME: "ovehub-ove-asset-manager-service"
      SERVICE_AM_PORT: "6080"

  ovehub-ove-asset-manager-worker-tulip:
    image: ovehub/ove-asset-manager-worker-tulip:stable
    environment:
      SERVICE_LOG_LEVEL: "info"
      SERVICE_AM_HOSTNAME: "ovehub-ove-asset-manager-service"
      SERVICE_AM_PORT: "6080"
```

You can run this by executing:

```
docker-compose up -d
```

To shutdown all the docker images:

```
docker-compose down
```

29.3 Installation for a non-Docker environment

All the services can can run perfectly on bare-metal Linux or MacOS as well. To start please clone this repository or download a release.

The services have been tested on CPython 3.6 and PyPy3.6 v7.0. For better performance PyPy is recommended, but for convenience CPython (the default Python interpreter) can be used instead.

A virtual environment can be created by executing:

```
virtualenv -p python3 env && source env/bin/activate
```

The terminal should display something like **env** at the beginning of the line, indicating that the virtual environment is active. You can then safely install the dependencies within the same virtual environment:

```
pip -r requirements.txt && pip -r requirements.ui.txt
```

If you wish to install the Deep Zoom Worker, then `pyvips` needs to be installed in the virtual environment as well. Please check the install guide for details on how to install the library and the system bindings.

To start the Asset Manager Backend:

```
./start-am.sh
```

29.3.1 User Interface

To downloaded required JavaScript libraries, change directory to the root folder and execute:

```
npm install
```

If the command is successful all the web assets will be downloaded into **ui/static/vendors/**. If this folder is empty after execution of the `npm` command, then something has failed: please check the execution logs for more details.

After downloading all the web assets, it is safe to delete the **node_modules** folder.

To start the User Interface:

```
./start-ui.sh
```

29.3.2 Workers

Deep Zoom worker:

```
WORKER_CLASS="workers.gigaimage.ImageWorker" ./start-worker.sh
```

Zip worker:

```
WORKER_CLASS="workers.zip.ZipWorker" ./start-worker.sh
```

Tulip graph layout worker:

```
WORKER_CLASS="workers.tulip.NetworkWorker" ./start-worker.sh
```

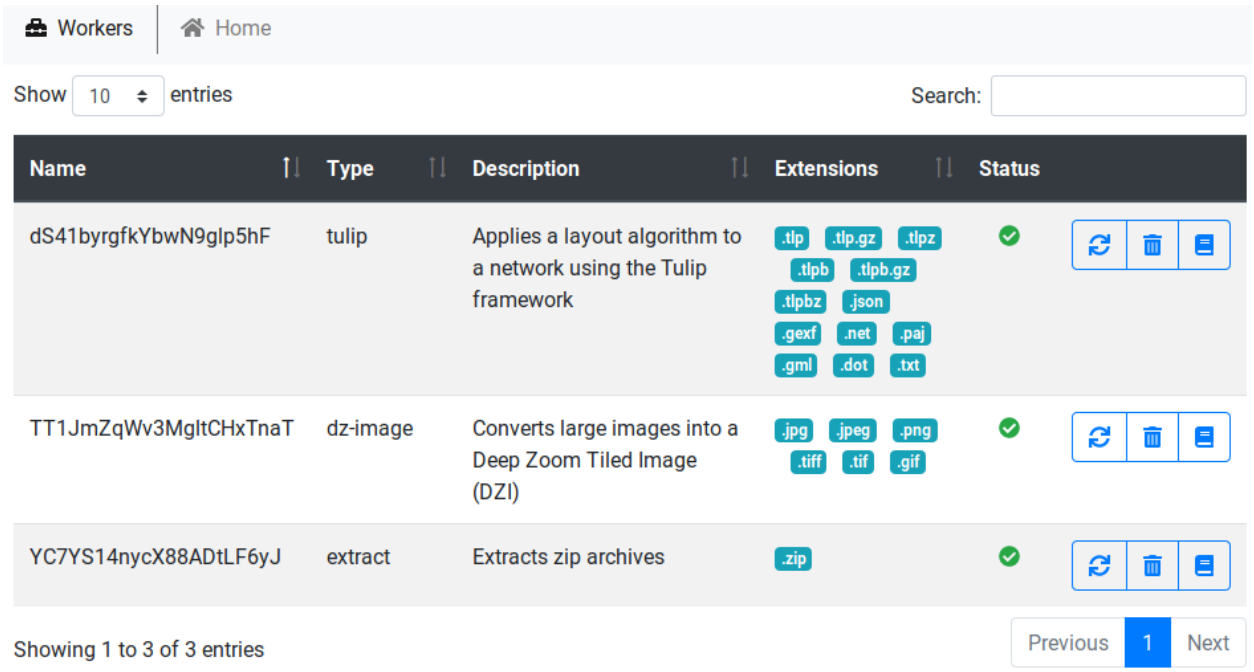
CHAPTER 30

Using OVE Asset Manager

The OVE asset manager interface presents two main views: a table of workers, and a hierarchy of *stores*, *projects* and *assets*, each represented by a table.

30.1 Managing workers

Workers are separate programs that can asynchronously process files (e.g., to convert it to a new file format, expand a zip archive, or apply a layout algorithm to a file describing a network).



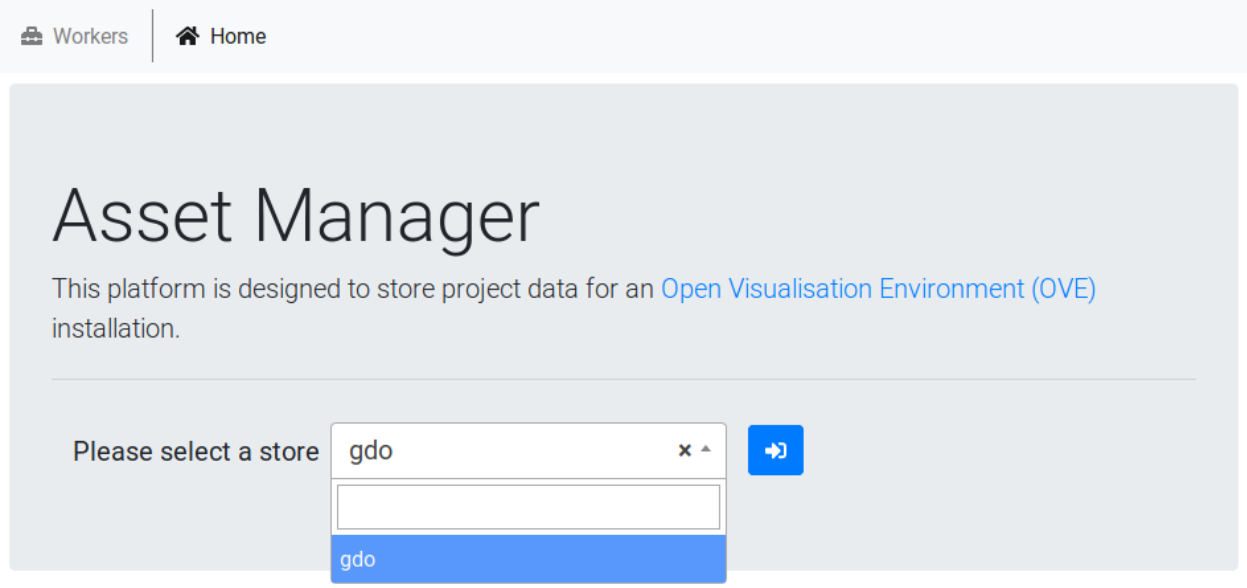
When a worker program starts running, it automatically *registers* itself with the Asset Manager using a REST API. Multiple instances of the same `type` of worker can register with the same instance of the Asset Manager.

The status column indicates whether the worker instance is `ready` (white tick in green circle), currently `processing` a file (spinning circles), or has encountered an `error` (exclamation mark in circle); worker status can be reset to `ready` by clicking on the button with a reset icon.

A worker should automatically de-register itself when it stops running, but it is also possible to manually de-register a worker by clicking on the button with a trash can icon.

The documentation for a worker can be viewed by clicking on the button with a book icon.

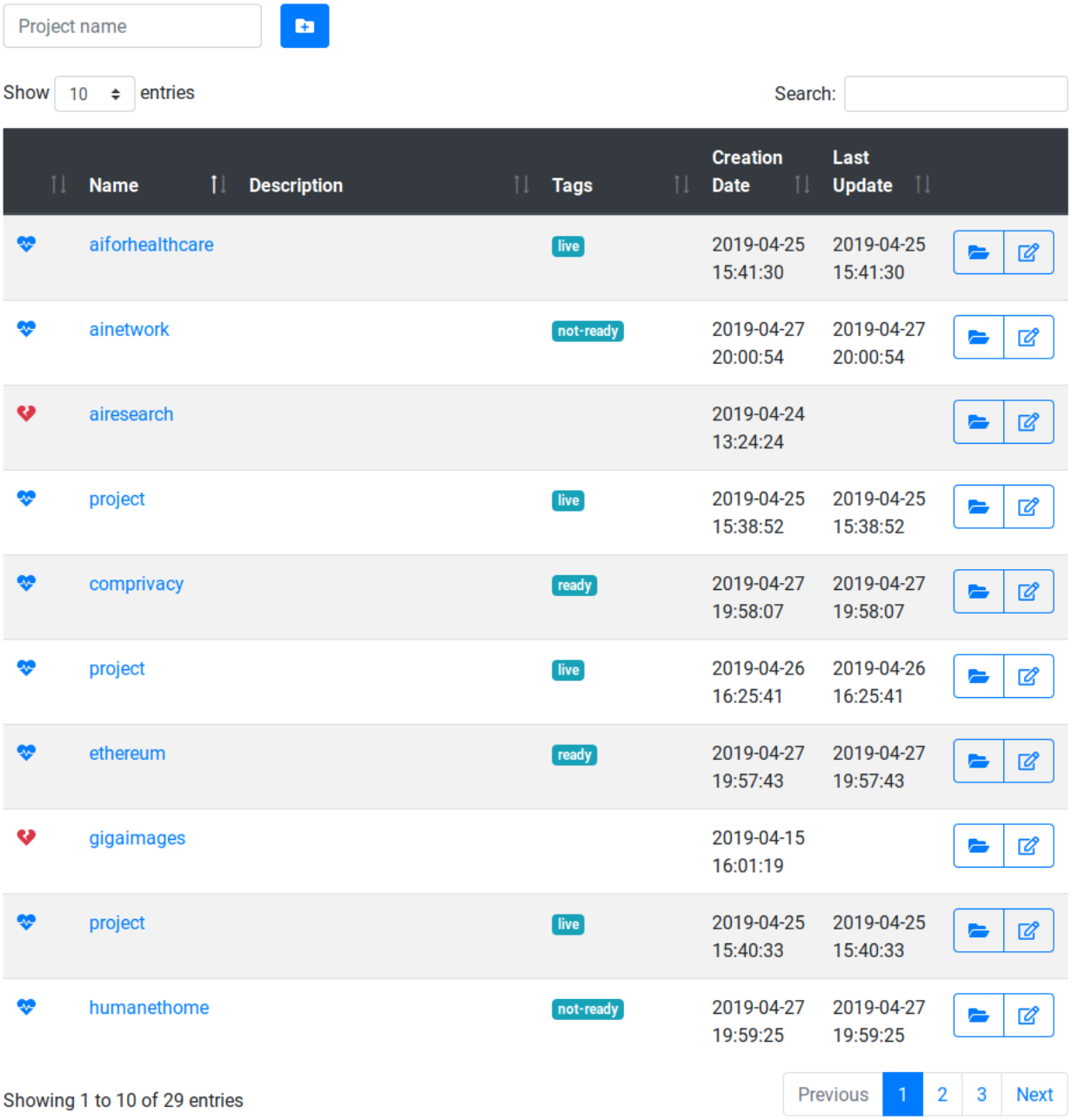
30.2 Managing projects and assets



The **home page** displays a list of S3 compatible object *stores*; selecting a store opens a list of the *projects* it contains.

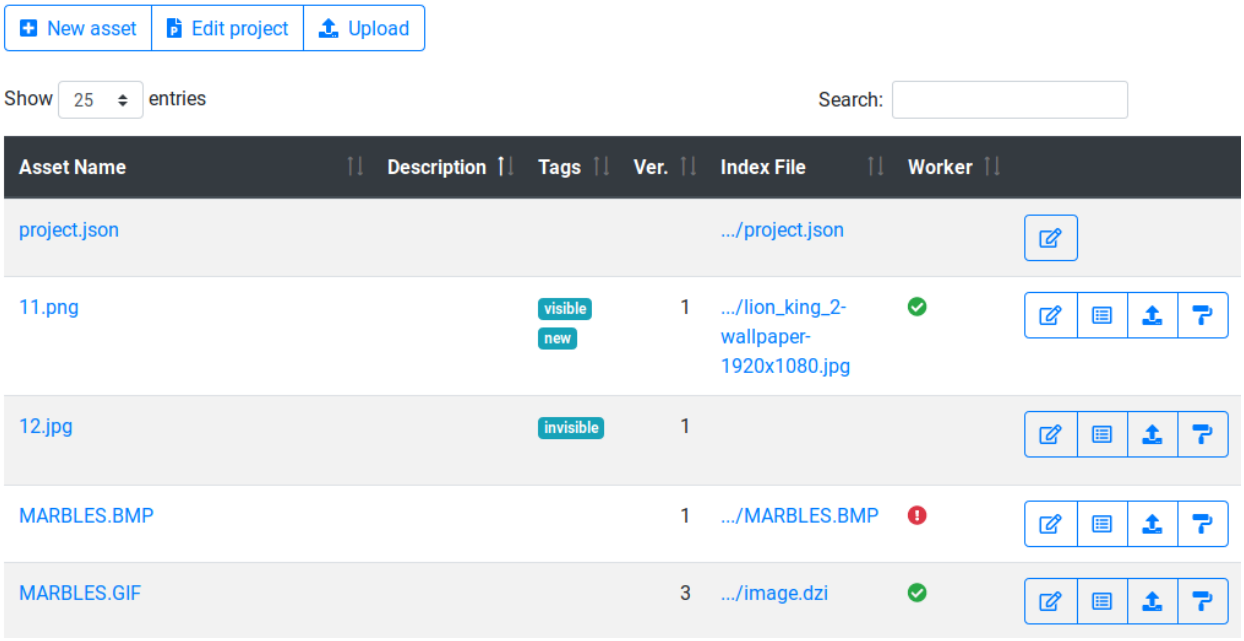
30.2.1 Project list

You can create a new project by typing its name into the input box and pressing the button with an icon that is a plus sign superimposed on a folder.



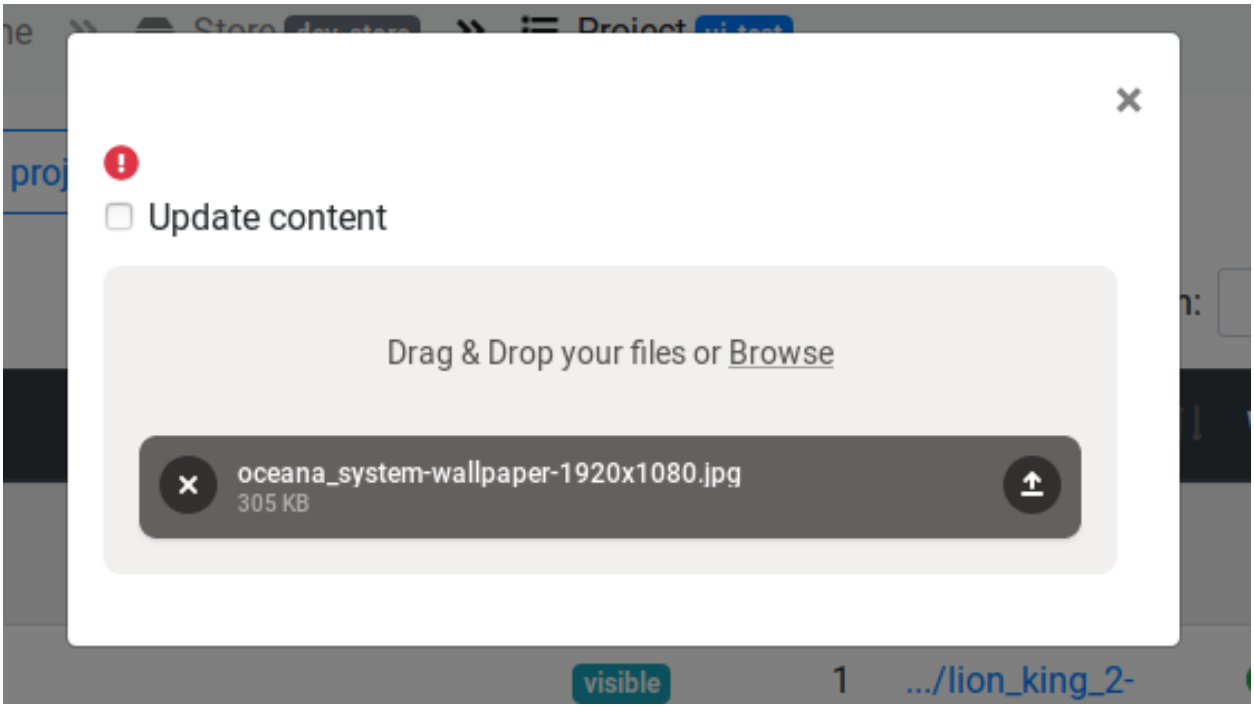
From the **project list**, you can edit the details of a project by clicking on the pencil icon, or list the **assets** in the project by clicking on either the project name or the folder icon.

30.2.2 Asset list

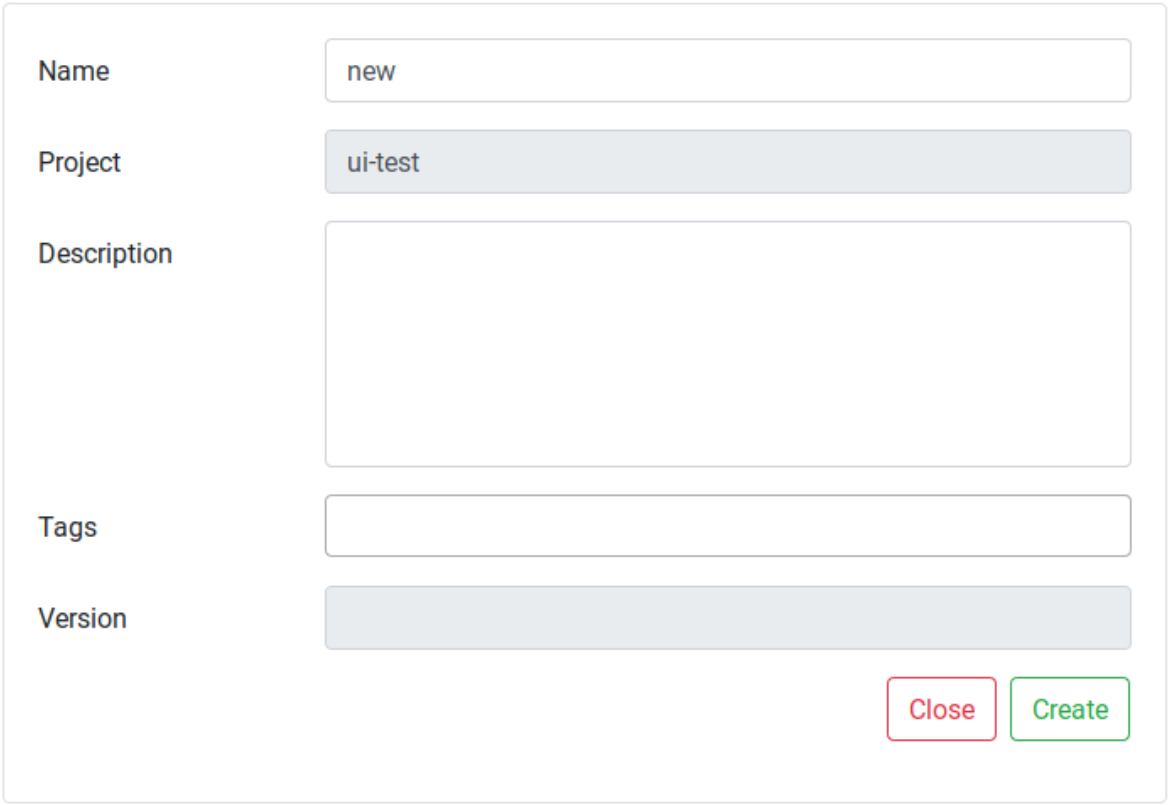


From a project's **asset list**, you can add assets to a project in one of two ways:

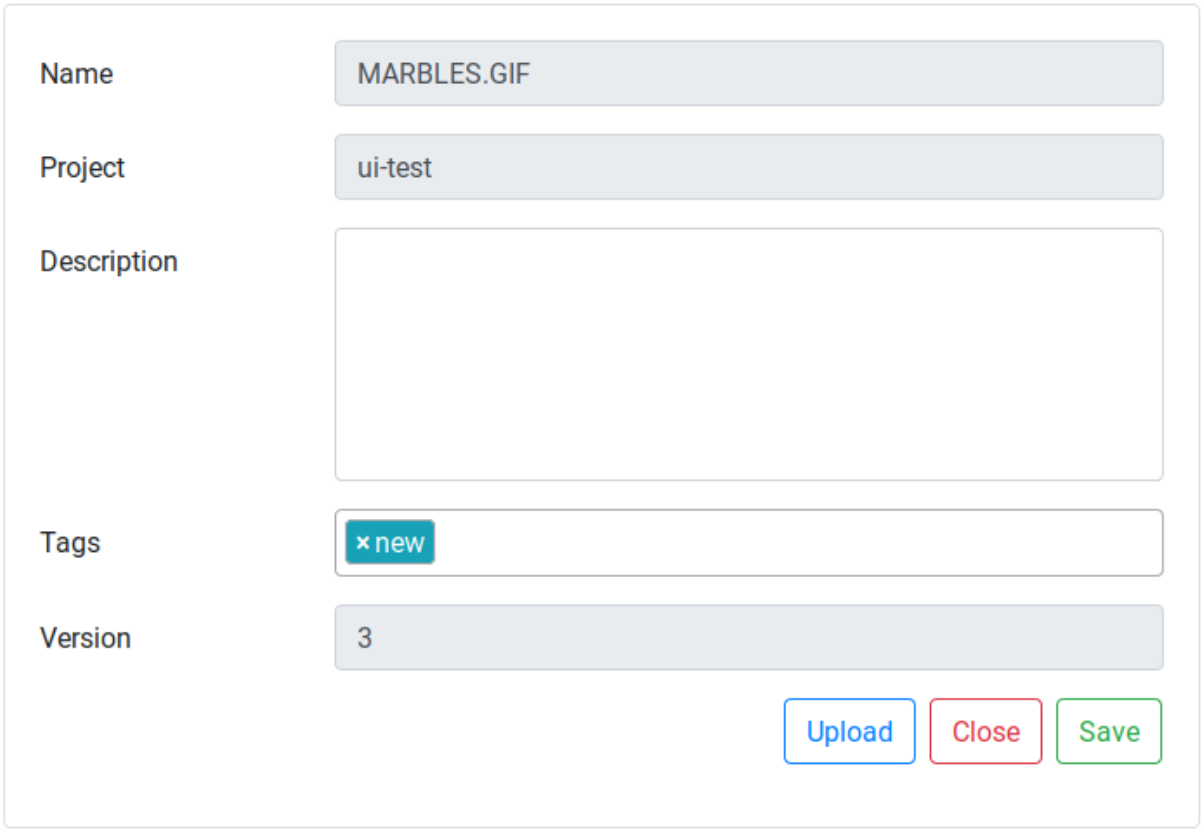
- You can click on the `Upload` button, and select one or more files to upload. An asset will be automatically created for each file, with an asset name determined by the filename, as seen in the image bellow:



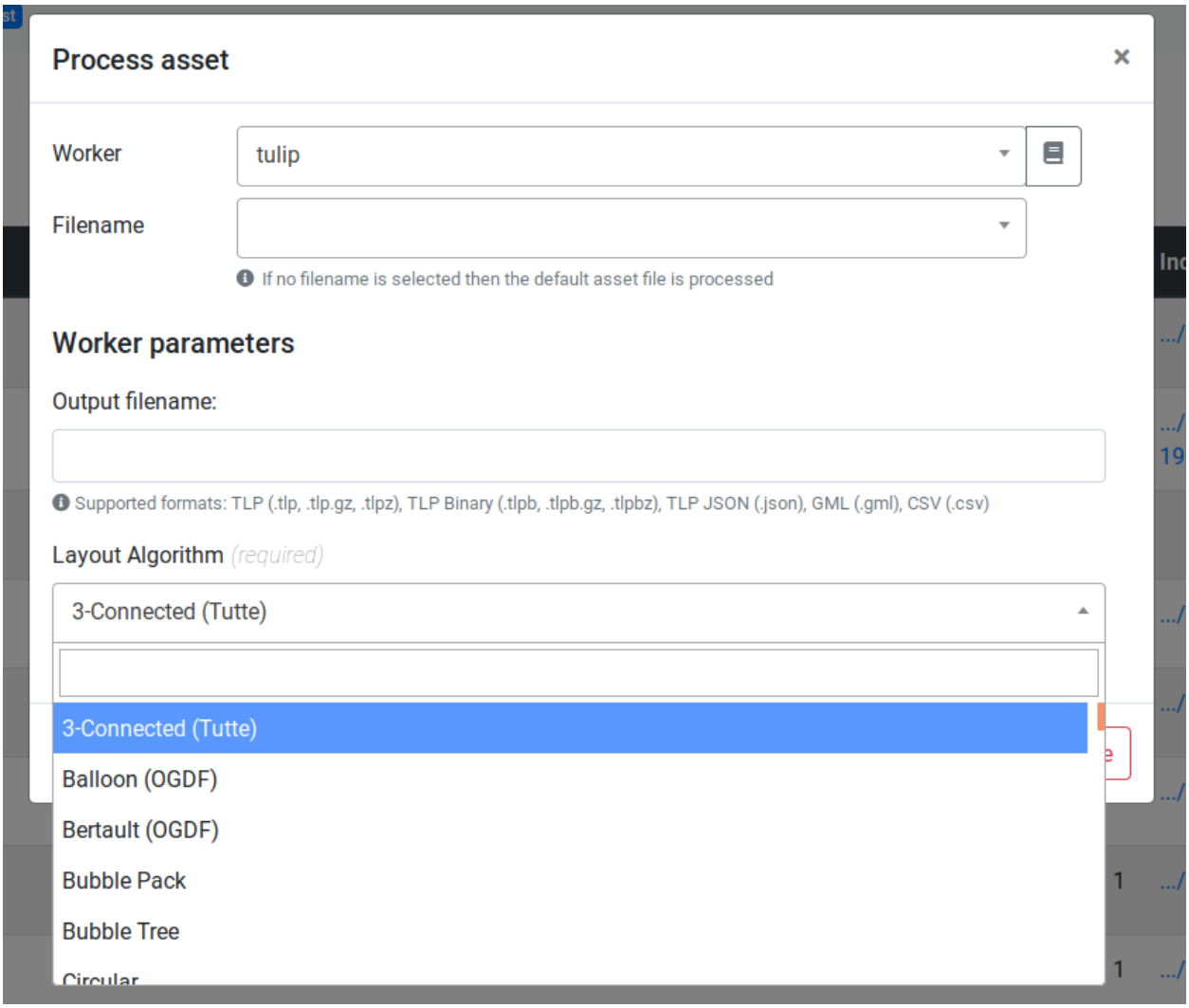
- Alternatively, you can manually create an asset by clicking on the `New Asset` button, enter the asset name, and click `Save`.



You can then add an asset to this file from the Edit Asset page, which the asset creation page will automatically redirect to after it is submitted, or by clicking the upload icon. Regardless of how the asset was originally created, you can use these methods to upload a modified file.



A worker can be instructed to process an asset by clicking on the button with a paint-roller icon.



After being processed by a worker, an asset may contain multiple files (e.g., because a zip file has been expanded into a directory, or because an image has been decomposed into tiles). Clicking the button with a checklist icon lists the contents of an asset, with a link that allows each to be opened. Regardless of the number of a files in an asset, there will be a single file designated as the *Index File*; a link to this is provided in a column of the asset table.

As well as assets, a project can contain a *project file*, which explains how its assets should be displayed by OVE. This file can be edited (or created if it does not yet exist) by clicking on the `Edit project file` button above the table.

CHAPTER 31

OVE Asset Manager REST API

This API is designed to allow you to perform the majority of necessary file operations using a REST API

- **/api/list**:
  - `GET`: *Lists available file stores*
  - **Response:**
    - **Success**: **HTTP Code:** 200 **Content:** `['store1', 'store2', '...']`

- **/api/{store_id}/list**
  - `GET`: *Lists available projects in a file store*
  - **Query params:** `metadata=true | false` - optional, if true the project list include some metadata
  - **Response:**
    - **Success**: **HTTP Code:** 200 **Content:** `[{name: "project1", "creationDate": "date"}, "..."]`
    - **Store not found**: **HTTP Code:** 400 Bad Request **Content:** `{title="Store not found", description="..."}`

- **/api/{store_id}/create**
  - `POST`: *Create project*
  - **Data Params:** `Requires JSON body`

```
{ "name":"project_name" }
```

  - **Response:**
    - **Success**: **HTTP Code:** 200 **Content:** `{'Project': project_name}`

---

    * **Error**: Store not found **HTTP Code:** 400 Bad Request **Content:** `{title="Store not found", description="..."}`
    * **Error**: Project already exists **HTTP Code:** 409 Conflict **Content:** `{title="Project already in use", description="..."}`
  - **Notes:** *With S3 storage, the project name must conform to s3 bucket name conventions*

- **/api/{store_id}/{project_name}/list**
  - `GET`: *Lists available assets in a file store with meta data*
  - **Query params:**
    * `includeEmpty=(True|False)` - optional, if true all folders are included regardless of whether they are ove assets‘
    * `filterByTag=<tag_name>` - optional, filter all assets tagged by tag_name
  - **Response:**
    * **Success**: **HTTP Code:** 200 **Content:** `{ "Assets" : ["Asset1"] }`
    * **Store not found**: **HTTP Code:** 400 Bad Request **Content:** `{title="Store not found", description="..."}`
    * **Project not found**: **HTTP Code:** 400 Bad Request **Content:** `{title="Project not found", description="..."}`

- **/api/{store_id}/{project_name}/create**
  - `POST`: *Create asset*
  - **Data Params:** `Requires JSON body`

```
{ "name":"asset_name" }
```

  - **Response:**
    * **Success**: **HTTP Code:** 200 **Content:** `{'Asset': asset_name}`
    * **Error**: Store not found **HTTP Code:** 400 Bad Request **Content:** `{title="Store not found", description="..."}`
    * **Project not found**: **HTTP Code:** 400 Bad Request **Content:** `{title="Project not found", description="..."}`
    * **Error**: Asset already exists **HTTP Code:** 409 Conflict **Content:** `{title="Asset already in use", description="..."}`
  - **Notes:** _With S3 storage, asset name is not allowed to contain / or any restricted asset names (e.g. new, .ovemeta, list, create)

- **/api/{store_id}/{project_name}/object/{object_id}**
  - `HEAD`: *Check if an object exists*
  - **Response:**
    * **Success**: HTTP Code 200
    * **Not found**: HTTP Code 404

  - `GET`: *Get an object in JSON format*
  - **Response:**
    * **Success**: **HTTP Code:** 200 **Content:** `{ ... }`
    * **Store not found**: **HTTP Code:** 400 Bad Request **Content:** `{title="Store not found", description="..."}`
    * **Project not found**: **HTTP Code:** 400 Bad Request **Content:** `{title="Project not found", description="..."}`
    * **Object not found**: **HTTP Code:** 400 Bad Request **Content:** `{title="Object not found", description="..."}`
  - `POST`: *Create an object in JSON format*
  - **Data Params:** `{ ... }`
  - **Response:**
    * **Success**: **HTTP Code:** 200 **Content:** `{ ... }`
    * **Store not found**: **HTTP Code:** 400 Bad Request **Content:** `{title="Store not found", description="..."}`
    * **Project not found**: **HTTP Code:** 400 Bad Request **Content:** `{title="Project not found", description="..."}`
    * **Object not found**: **HTTP Code:** 400 Bad Request **Content:** `{title="Object not found", description="..."}`
    * **Object already in use**: **HTTP Code:** 400 Bad Request **Content:** `{title="Object already in use", description="..."}`
  - `PUT`: *Update an object in JSON format*
  - **Data Params:** `{ ... }`
  - **Response:**
    * **Success**: **HTTP Code:** 200 **Content:** `{ ... }`
    * **Store not found**: **HTTP Code:** 400 Bad Request **Content:** `{title="Store not found", description="..."}`
    * **Project not found**: **HTTP Code:** 400 Bad Request **Content:** `{title="Project not found", description="..."}`
    * **Object not found**: **HTTP Code:** 400 Bad Request **Content:** `{title="Object not found", description="..."}`

- **/api/{store_id}/{project_name}/object/{object_id}/info**
  - `GET`: *Get the object metadata*
  - **Response:**
    * **Success**: **HTTP Code:** 200 **Content:** `{ name="...", index_file="..."}`
    * **Store not found**: **HTTP Code:** 400 Bad Request **Content:** `{title="Store not found", description="..."}`
    * **Project not found**: **HTTP Code:** 400 Bad Request **Content:** `{title="Project not found", description="..."}`

        * **Object not found**: **HTTP Code:** 400 Bad Request **Content:** `{title="Object not found", description="..."}`

- **/api/{store_id}/{project_name}/meta/{asset_id}**
  - `HEAD`: *Check if an asset exists*
  - **Response:**
    * **Success**: HTTP Code 200
    * **Not found**: HTTP Code 404
  - `GET`: *Get an asset metadata*
  - **Response:**
    * **Success**: **HTTP Code:** 200 **Content:** `{ "name": "...", "project": "...", "description": "...", "index_file": "...", "version": "...", "history": "...", "tags": "...", "worker": "...", "processing_status": "...", "processing_error": "..." }`
    * **Store not found**: **HTTP Code:** 400 Bad Request **Content:** `{title="Store not found", description="..."}`
    * **Project not found**: **HTTP Code:** 400 Bad Request **Content:** `{title="Project not found", description="..."}`
    * **Asset not found**: **HTTP Code:** 400 Bad Request **Content:** `{title="Asset not found", description="..."}`
  - `POST`: *Update a project metadata*
  - **Data Params:** `{"description": "...", "tags": "..."}`
  - **Response:**
    * **Success**: **HTTP Code:** 200 **Content:** `{"Status": "OK"}`
    * **Store not found**: **HTTP Code:** 400 Bad Request **Content:** `{title="Store not found", description="..."}`
    * **Project not found**: **HTTP Code:** 400 Bad Request **Content:** `{title="Project not found", description="..."}`
    * **Asset not found**: **HTTP Code:** 400 Bad Request **Content:** `{title="Asset not found", description="..."}`

- **/api/{store_id}/{project_id}/files/{asset_id}**
  - `GET`: *list of files under the current version of the asset*
  - **Response:**
    * **Success**: **HTTP Code:** 200 **Content:** `[{name: "...", url: "..."}]`
    * **Store not found**: **HTTP Code:** 400 Bad Request **Content:** `{title="Store not found", description="..."}`
    * **Project not found**: **HTTP Code:** 400 Bad Request **Content:** `{title="Project not found", description="..."}`

    * **Asset not found**: **HTTP Code:** 400 Bad Request **Content:** `{title="Asset not found", description="..."}`

- **/api/{store_id}/{project_name}/upload/{asset_id}**
  - `POST`: *Upload an asset*
  - **Headers:** `content-disposition: filename="<file_name>"`
  - **Body:** `the file octet stream`
  - **Response:**
    * **Success**: **HTTP Code:** 200 **Content:** `{"Status": "OK"}`
    * **Store not found**: **HTTP Code:** 400 Bad Request **Content:** `{title="Store not found", description="..."}`
    * **Project not found**: **HTTP Code:** 400 Bad Request **Content:** `{title="Project not found", description="..."}`
    * **Asset not found**: **HTTP Code:** 400 Bad Request **Content:** `{title="Asset not found", description="..."}`
    * **Asset exists**: **HTTP Code:** 409 Conflict **Content:** `{title="Asset exists", description="..."}`

- **/api/{store_id}/{project_name}/update/{asset_id}**
  - `POST`: *Update an asset file*
  - **Headers:** `content-disposition: filename="<file_name>"`
  - **Body:** `the file octet stream`
  - **Response:**
    * **Success**: **HTTP Code:** 200 **Content:** `{"Status": "OK"}`
    * **Store not found**: **HTTP Code:** 400 Bad Request **Content:** `{title="Store not found", description="..."}`
    * **Project not found**: **HTTP Code:** 400 Bad Request **Content:** `{title="Project not found", description="..."}`
    * **Asset not found**: **HTTP Code:** 400 Bad Request **Content:** `{title="Asset not found", description="..."}`

- **/api/{store_id}/{project_name}/createUpload/{asset_id}**
  - `POST`: *Create an asset and upload the file content*
  - **Headers:** `content-disposition: filename="<file_name>"`
  - **Body:** `the file octet stream`
  - **Response:**
    * **Success**: **HTTP Code:** 200 **Content:** `{"Status": "OK"}`
    * **Store not found**: **HTTP Code:** 400 Bad Request **Content:** `{title="Store not found", description="..."}`

---

        * **Project not found**: **HTTP Code:** 400 Bad Request **Content:** `{title="Project not found", description="..."}`

- **/api/{store_id}/{project_name}/process/{asset_id}**
  - `POST`: *Schedule a worker processing task on the selected asset*
  - **Data Params:** `{"worker_type": "...",}`
  - **Response:**
    * **Success**: **HTTP Code:** 200 **Content:** `{"Status": "OK"}`
    * **Store not found**: **HTTP Code:** 400 Bad Request **Content:** `{title="Store not found", description="..."}`
    * **Project not found**: **HTTP Code:** 400 Bad Request **Content:** `{title="Project not found", description="..."}`
    * **Asset not found**: **HTTP Code:** 400 Bad Request **Content:** `{title="Asset not found", description="..."}`
    * **Worker not found**: **HTTP Code:** 400 Bad Request **Content:** `{title="Worker not found", description="..."}`

- **/api/workers**
  - `GET`: *Worker list*
  - **Response:**
    * **Success**: **HTTP Code:** 200 **Content:** `['worker1', ...]`
  - `POST`: *Register worker*
  - **Data Params:** `{ "name": "...", "type": "...", "description": "...", "extensions": "...", "status": "...", "callback": "...", "status_callback": "...", "parameters": "...", "docs": "..." }`
  - **Response:**
    * **Success**: **HTTP Code:** 200 **Content:** `{ "name": "...", "type": "...", "description": "...", "extensions": "...", "status": "...", "callback": "...", "status_callback": "...", "parameters": "...", "docs": "..." }`
    * **Worker already exists**: **HTTP Code:** 400 Bad Request **Content:** `{title="Worker already exists", description="..."}`
  - `PATCH`: *Update the status of a worker*
  - **Data Params:** `{ name: "...", status: "..." }`
  - **Response:**
    * **Success**: **HTTP Code:** 200 **Content:** `{ 'Status': 'OK' }`
    * **Worker not found**: **HTTP Code:** 400 Bad Request **Content:** `{title="Worker not found", description="..."}`
  - `DELETE`: *Unregister a worker*
  - **Data Params:** `{ name: "..." }`

- **Response:**
  - **Success**: **HTTP Code:** 200 **Content:** `{ 'Status': 'OK' }`
  - **Worker not found**: **HTTP Code:** 400 Bad Request **Content:** `{title="Worker not found", description="..."}`

CHAPTER 32

Development Guide

32.1 Starting the services locally

Please refer to the local install guide to start the development.

32.2 Dependencies

All AM services use the following libraries:

- falcon (docs) - for the REST API
- gunicorn - as the production web server
- urllib3 - as a REST client

Backend and workers only:

- minio - as an Amazon S3 compatible object store

UI only:

- Jinja2 - for HTML templating
- markdown2 - converting the worker documentation from Markdown to HTML

32.3 Adding new AM Backend functionality

The code for the AM Backend can be found in the **am** module:

- **init.py**: is used to configure the app and setup all the app routes
- **main.py**: a startup script that can be used for development and testing
- **controller.py**: the backend controller which connects the mangers to the API routes

- **managers.py**: the worker manager, keeping the state of all registered workers
- **routes.py**: all the REST routes, in falcon format

32.4 Adding new UI functionality

The code for the AM Backend can be found in the **ui** module:

- **init.py**: is used to configure the app and setup all the app routes
- **main.py**: a startup script that can be used for development and testing
- **backend.py**: the UI controller which connects the mangers to the API routes
- ***_utils.py**: utils classes for alerts or Jinja2 integration
- **backend.py**: the client used to make requests to the backend
- **controller.py**: the UI controller which connects the mangers to the API routes
- **routes.py**: all the REST routes, in falcon format (falcon docs on routing)
- **middleware.py**: a content type validator

The Jinja2 templates written in HTML and are located in the **ui/templates**.

The frontend uses a few JavaScript libraries. All these must be added to **package.json** and the assets (js, css and font files exposed by the library) must be added to **move-assets.js**.

32.5 Adding a new Worker

Creating a worker requires changes in several places:

- a Markdown documentation file in `docs/workers`
- a Dockerfile in a new subdirectory of `docker/`
- the worker itself, in a new subdirectory of `workers/`
- an addition to `docker-compose.yml`

You will also need to modify the `docker-compose.am.yml` template used by the OVE Installer.

All new workers can extend the `BaseWorker` class. The methods that require an implementation are marked as abstract in the base class:

```
from typing import List, Dict
from common.entities import OveMeta

from workers.base import BaseWorker

class NewWorker(BaseWorker):
    def worker_type(self) -> str:
        """
        :return: the worker type as a string. This value can be a valid WorkerType or␣
↪anything else
        """
        return ""

    def extensions(self) -> List:
```

(continues on next page)

(continued from previous page)

```
        """
        :return: the extensions handled by this worker
        """
        return []

    def description(self) -> str:
        """
        :return: description in human-readable format
        """
        return ""

    def docs(self) -> str:
        """
        :return: the worker documentation document url, in markdown format
        see https://github.com/trentm/python-markdown2 for more details
        """
        return ""

    def parameters(self) -> Dict:
        """
        :return: the worker parameter description, in json-form format:
        see http://www.alpacajs.org/ for more details
        """
        return {}

    def process(self, project_name: str, meta: OveMeta, options: Dict):
        """
        Override this to start processing
        :param project_name: name of the project to process
        :param meta: the object to process
        :param options: task options, passed by the asset manager. Can be empty
        :return: None
        :raises: Any exception is treated properly and logged by the safe_process␣
↪method
        """
        pass
        # this is where the worker processing happens
```

To test a worker using Docker, you will need to expose the correct port (e.g, with `-p 6090:6090`), and set the environment variables `SERVICE_AM_HOSTNAME`, `SERVICE_AM_PORT`, and `SERVICE_HOSTNAME`. If testing with an instance of the Asset Manager on another machine, that machine must be able to resolve the `SERVICE_HOSTNAME` to the machine running the worker.

CHAPTER 33

OVE Python Client Library

This library provides a Python interface to the Open Visualization Environment (OVE) developed at the Imperial College Data Science Institute for use with the Data Observatory.

33.1 Installation

The source code can be downloaded from the Releases page or cloned by running:

```
git clone https://github.com/ove/ove-sdks
```

To install with setup.py:

```
cd ove-sdks/python
python setup.py install
```

33.2 Example Usage

Note: The space is automatically put into offline mode which allows you to design your space without pushing changes live into the DO environment. If you wish to interact with the space directly you can **enable online mode** after import.

There is also an *browser_opening* mode: if both this and *online mode* is enabled, then the control page for a section will be automatically opened in your web browser after it is created.

```
from ove import save_file

from ove.ove import Space
space = Space(ove_host="localhost", space_name="LocalNine", control_port=8080)

# enable live mode if you wish to interact with the space directly
# space.enable_online_mode()
```

(continues on next page)

(continued from previous page)

```
# uncomment this if you wish to automatically open control pages for new sections in␣
↪a web browsers
# space.enable_browser_opening()

space.delete_sections()

video = space.add_section(w=2880, h=1616, x=720, y=404, app_type='videos')
video.set_url('https://www.youtube.com/watch?v=QJo-VFs1X5c')

# Wait for the video to buffer before calling the following command:
space.videos.play()

html = space.add_section(w=2880, h=1616, x=720, y=404, app_type='html')
html.set_url("http://metafilter.com")

image = space.add_section(w=5000, h=3000, x=7200, y=1000, app_type='images')
image.set_url("https://farm4.staticflickr.com/3107/2431422903_632ce51b56_o_d.jpg",
↪"shelley")

map1 = space.add_section(w=5000, h=3000, x=10200, y=800, app_type='maps')
map1.set_position(latitude=0, longitude=0, zoom=5)

network = space.add_section(w=5000, h=3000, x=10200, y=800, app_type='networks')
network.set_data(json_url="https://raw.githubusercontent.com/ove/ove-apps/master/
↪packages/ove-app-networks/src/data/sample.json")

chart = space.add_section_by_grid(w=1, h=1, r=0, c=0, app_type="charts")

chart.set_specification(spec_url="https://raw.githubusercontent.com/vega/vega/master/
↪docs/examples/bar-chart.vg.json", options={"width": 900-35, "height": 900-35})

# the state of the space can be saved automatically with the save state file util␣
↪function
save_file(json_state=space.to_json(title="Title of the presentation"), filename="my_
↪state.json")
```

Videos can also be controlled independently:

```
from ove.ove import Space
space = Space(ove_host="localhost", space_name="LocalNine", control_port=8080)

space.delete_sections()

video = space.add_section(w=2880, h=1616, x=720, y=404, app_type='videos')
video.set_url('https://www.youtube.com/watch?v=QJo-VFs1X5c')

video2 = space.add_section_by_grid(r=1, c=1, w=1, h=1, app_type='videos')
video2.set_url("https://www.youtube.com/watch?v=XY3NP4JHXZ4")

video.play()
video2.play()
video.pause()
video2.pause()
```

`ove-python` also includes a local web-server that can be used to host images.

```
import matplotlib.pyplot as plt

from ove.server import Server

from ove.ove import Space
space = Space(ove_host="localhost", space_name="LocalNine", control_port=8080)

s = Server()
s.start_server()

a = plt.figure()
plt.plot([1, 2, 3], [4, 5, 6])

# share_matplotlib() exports a plot object to PNG, and returns the url where can be␣
↪accessed
url = s.share_matplotlib(a)
image = space.add_section_by_grid(w=1, h=1, r=2, c=2, app_type='images')
image.set_url(url)
```

Web content can be displayed in a similar way:

```
from ove.ove import Space
space = Space(ove_host="localhost", space_name="LocalNine", control_port=8080)

space.delete_sections()

space.set_grid(space.geometry["screen_rows"], space.geometry["screen_cols"])

html = space.add_section_by_grid(w=2, h=2, r=0, c=0, app_type='html')
html.set_url("http://metafilter.com")
```

CHAPTER 34

Distributed.js Library

This is a JavaScript library for distributing JavaScript function calls and variables from controllers to viewers. The `setDistributed` function distributes the execution of functions and the `watch` function propagates updates on one or more properties from controller to viewers.

This library also exposes a number of helpers, one of which is THREE.Distributed, which is library for distributing Three.js functionality using the Distributed.js library.

CHAPTER 35

Background Utility

If a web page or an app has a transparent background by design, a background colour of choice can be set using this utility:

Linux/Mac:

```
curl --header "Content-Type: application/json" --request POST --data '{"app": {"url":
↪"http://OVE_CORE_HOST:PORT/app/html","states": {"load": {"url": "http://OVE_CORE_
↪HOST:PORT/app/html/data/background/index.html?background=COLOUR"}}}, "space": "OVE_
↪SPACE", "h": 500, "w": 500, "y": 0, "x": 0}' http://OVE_CORE_HOST:PORT/section
```

Windows:

```
curl --header "Content-Type: application/json" --request POST --data "{\"app\": {\
↪"url\": \"http://OVE_CORE_HOST:PORT/app/html\", \"states\": {\"load\": {\"url\": \
↪"http://OVE_CORE_HOST:PORT/app/html/data/background/index.html?background=COLOUR\"}}
↪}, \"space\": \"OVE_SPACE\", \"h\": 500, \"w\": 500, \"y\": 0, \"x\": 0}" http://
↪OVE_CORE_HOST:PORT/section
```

The background colour can be specified either as a keyword or hexadecimal value such as `Black` or `#f0f0f0`.

CHAPTER 36

Spaces.json File

The `Spaces.json` file is used to define the `spaces` of OVE and describe the arrangement of `clients` within each `space` (these terms are explained in the Basic Concepts page).

Each `client` must have its `geometry` specified by four properties:

- `h`: the height of the client in pixels
- `w`: the width of the client in pixels
- `x`: the x-coordinate of the top-left corner of the client (x-coordinate values begin at 0 and increase from left to right)
- `y`: the y-coordinate of the top-left corner of the client (y-coordinate values begin at 0 increase downards)

The order in which the `clients` are defined determines the `oveViewID` of each, and thus the URL that should be opened in each browser window.

By default, OVE assumes that there is a one-to-one mapping between `client` pixels and `screen` pixels. The optional `scale` property can be used to specify an alternative mapping. It accepts either a single scaling factor, or separate scaling factors for the x and y axes, similar to the CSS `scale()` function.

The default `Spaces.json` file defines two spaces (`LocalFour` and `LocalNine`), corresponding to `2x2` and `3x3` arrangements of `clients`. Each `client` in these spaces has a resolution of `1440x808`, so they have a total resolution of `2880x1616` and `4320x2424` respectively.

OVE **must be relaunched** after any changes are made to `Spaces.json`.

36.1 Examples

Additional spaces can be defined by adding descriptions to the `Spaces.json` file.

A number of example additions are provided below.

36.1.1 Single client

A space having a single `client` with a 4K resolution can be defined by adding:

```
"4K": [
    {"x": 0, "y": 0, "w": 3840, "h": 2160, "scale": 1}
]
```

The `scale` property is optional and can be omitted.

36.1.2 `2x2` grid of screens

A `2x2` grid of `clients`, each with with Full HD (1080p) resolution can be defined by adding:

```
"HDFour": [
    {"x": 0, "y": 1080, "w": 1920, "h": 1080},
    {"x": 1920, "y": 1080, "w": 1920, "h": 1080},
    {"x": 0, "y": 0, "w": 1920, "h": 1080},
    {"x": 1920, "y": 0, "w": 1920, "h": 1080}
]
```

In the above arrangement, the first `client` would be positioned bottom-left, the second `client` would be positioned bottom-right, the third `client` would be positioned top-left, and the fourth `client` would be positioned top-right.

36.1.3 Non-contiguous arrangements and bezel correction

The specification of `clients` on the `Spaces.json` file need not always be contiguous and non-overlapping. The following is a perfectly valid configuration:

```
"HDFour-alternative": [
    {"x": 0, "y": 1090, "w": 1920, "h": 1080},
    {"x": 1930, "y": 1090, "w": 1920, "h": 1080},
    {"x": 0, "y": 0, "w": 1920, "h": 1080},
    {"x": 1930, "y": 0, "w": 1920, "h": 1080}
]
```

The above arrangement leaves room for a `10 pixel` wide bezel between the each `client` in both horizontal and vertical directions and thereby creates a `space` of a `3850x2170` resolution (only 8,294,400 of the 8,354,500 pixels in this space are actually displayed by a `client`).

The *Alignment App* exists to help align the monitors by correcting the `space` configuration to compensate for the thickness of the bezels and any gaps that may exist between adjacent `screens`.

CHAPTER 37

MapLayers.json File

The maps app can be configured by embedding the map layers configuration within the `config.json` file, or by providing a URL that points to a `MapLayers.json` file. The format of the JSON configuration remains the same in either of these two approaches.

The `MapLayers.json` file contains an array of map layers. There are two different configuration formats for OpenLayers and Leaflet. These as described below. The mapping library is selected according to the configuration format of the first layer and assumes all other layers follow the same format. If no layers were defined, OpenLayers will be selected as the mapping library.

By default, the `config.json` exposes layers in the OpenLayers configuration format. To use Leaflet, edit the `config.json` file and replace the `layers` property with `layers_ol` and the `layers_leaflet` property with `layers`.

37.1 OpenLayers configuration format

The configuration format for OpenLayers has a structure similar to:

```
[
    {
        "type": "ol.layer.Tile",
        "visible": false,
        "wms": false,
        "source": {
            "type": "ol.source.OSM",
            "config": {
                "crossOrigin": null,
                "url": "https://{a-c}.tile.thunderforest.com/transport/{z}/{x}/{y}.png
↪"
            }
        }
    },
    {
```

(continues on next page)

(continued from previous page)

```
        "type": "ol.layer.Vector",
        "visible": false,
        "wms": true,
        "source": {
            "type": "ol.source.Vector",
            "config": {
                "url": "data/sampleGeo.json"
            }
        },
        "style": {
            "fill": {
                "color": "#B29255"
            },
            "stroke": {
                "color": "#715E3A",
                "width": 4
            }
        },
        "opacity": 0.7
    }
]
```

The library supports two types of map layers, `ol.layer.Tile` and `ol.layer.Vector`.

Both types of layer have a `visible` property that describes whether the layer is visible by default and a `wms` property which describes whether the layer is a displaying data from a `Web Map Service`. These properties have Boolean values.

The layers also have a common `source` property, but their contents differ. The `source` of a vector layer always must have a `type` equal to `ol.source.Vector`, and the `url` of the `config` property must point to a GeoJSON file. The `source` of a tile layer can be any layer source for tile data supported by OpenLayers such as `ol.source.OSM` or `ol.source.BingMaps`. The `config` object is passed as an argument of the constructor for the specified source, and its properties depend on which source is used; more information can be found in the OpenLayers documentation.

Unlike tile layers, vector layers have two additional properties, `style` and `opacity` which can be used to customise their appearance. The `opacity` property has a numeric value between 0 and 1, and the `style` property accepts a JSON structure with two properties (`fill` and `stroke`) within it. These correspond to configuration provided when creating `ol.style.Fill` and `ol.style.Stroke` objects, respectively.

37.1.1 Using the CARTO platform with OpenLayers

The configuration formats for using CARTO platform with OpenLayers have structures similar to:

```
[
    {
        "type": "ol.layer.Tile",
        "visible": false,
        "wms": false,
        "source": {
            "type": "ol.source.XYZ",
            "config": {
                "url": "http://api.cartocdn.com/base-dark/{z}/{x}/{y}.png"
            }
        }
    },
    {
```

(continues on next page)

(continued from previous page)

```
        "type": "ol.TorqueLayer",
        "visible": false,
        "wms": false,
        "source": {
            "user": "viz2",
            "table": "ow",
            "zIndex": 100,
            "cartocss": "Map { -torque-time-attribute: \"date\"; -torque-aggregation-
↪function: \"count(cartodb_id)\"; -torque-frame-count: 760; -torque-animation-
↪duration: 15; -torque-resolution: 2 } #layer {   marker-width: 3;   marker-fill-
↪opacity: 0.8;   marker-fill: #FEE391;    comp-op: \"lighten\";   [value > 2] {␣
↪marker-fill: #FEC44F; }   [value > 3] { marker-fill: #FE9929; }   [value > 4] {␣
↪marker-fill: #EC7014; }   [value > 5] { marker-fill: #CC4C02; }   [value > 6] {␣
↪marker-fill: #993404; }   [value > 7] { marker-fill: #662506; }   [frame-offset =␣
↪1] { marker-width: 10; marker-fill-opacity: 0.05;}   [frame-offset = 2] { marker-
↪width: 15; marker-fill-opacity: 0.02;} }"
        }
    },
    {
        "type": "ol.layer.Tile",
        "visible": false,
        "wms": false,
        "source": {
            "type": "ol.source.CartoDB",
            "config": {
                "account": "documentation",
                "config": {
                    "layers": [{
                        "type": "cartodb",
                        "options": {
                            "cartocss_version": "2.1.1",
                            "cartocss": "#layer { polygon-fill: #1E90FF; polygon-
↪opacity: 0.6;}",
                            "sql": "select * from european_countries_e where name =
↪'France'"
                        }
                    }]
                }
            }
        }
    }
]
```

The implementation supports the use of CARTO base maps, Torque.js and the CartoDB source of OpenLayers. These can be used to display base map tiles, animations and vector overlays.

The configuration of the `source` property of the `ol.TorqueLayer` type (which is similar to `L.TorqueLayer`) is explained in the Torque.js reference documentation. The configuration of the `ol.source.CartoDB` is explained in the OpenLayers API documentation.

To see CARTO platform examples in OVE, load the controller by accessing the URL `http://OVE_CORE_HOST:PORT/app/maps/control.html?oveSectionId=SECTION_ID&layers=2,3,4&state=World`. To learn more, see examples on using OpenLayers with the CARTO Platform and the example on using Torque.js with OpenLayers.

37.2 Leaflet configuration format

The configuration format for Leaflet has a structure similar to:

```
[
    {
        "type": "L.tileLayer",
        "visible": false,
        "wms": false,
        "url": "http://services.arcgisonline.com/ArcGIS/rest/services/World_Topo_Map/
↪MapServer/tile/{z}/{y}/{x}.jpg"
    },
    {
        "type": "L.geoJSON",
        "visible": false,
        "wms": false,
        "data": [
        {
            "type": "LineString",
            "coordinates": [[-100, 40], [-105, 45], [-110, 55]]
        },
        {
            "type": "LineString",
            "coordinates": [[-105, 40], [-110, 45], [-115, 55]]
        }],
        "options": {
            "style": {
                "fill": true,
                "fillColor": "#B29255",
                "fillOpacity": 0.7,
                "color": "#715E3A",
                "weight": 4,
                "opacity": 0.7
            }
        }
    }
]
```

The library supports raster and vector map layers as well as geoJSON layers. Unlike OpenLayers, they are not grouped into two significant layers. A complete catalogue of all Leaflet map layers are available in their documentation.

Both types of layer have a `visible` property that describes whether the layer is visible by default and a `wms` property which describes whether the layer is a displaying data from a `Web Map Service`. These properties have Boolean values.

Each layer has its own properties. `L.tileLayer` must have a `url` property. All vector layers must have a `bounds` property. `L.imageOverlay` and `L.videoOverlay` must have a `url` as well as a `bounds` property. All layers accept an optional `options` property as well. All of these combinations are explained in the Leaflet documentation.

Unlike in OpenLayers, Leaflet expects GeoJSON to be embedded and defined using a `data` property. Like other layers it also accepts an optional `options` property. The `style` property is defined within this `options` property as seen above. `opacity` is a part of the `style` property.

37.2.1 Using the CARTO platform with Leaflet

The configuration formats for using CARTO platform with Leaflet have structures similar to:

```
[
    {
        "type": "L.tileLayer",
        "visible": false,
        "wms": false,
        "url": "http://{s}.api.cartocdn.com/base-dark/{z}/{x}/{y}.png"
    },
    {
        "type": "L.TorqueLayer",
        "visible": false,
        "wms": false,
        "source": {
            "user": "viz2",
            "table": "ow",
            "zIndex": 100,
            "cartocss": "Map { -torque-time-attribute: \"date\"; -torque-aggregation-
↪function: \"count(cartodb_id)\"; -torque-frame-count: 760; -torque-animation-
↪duration: 15; -torque-resolution: 2 } #layer {   marker-width: 3;   marker-fill-
↪opacity: 0.8;   marker-fill: #FEE391;    comp-op: \"lighten\";   [value > 2] {␣
↪marker-fill: #FEC44F; }   [value > 3] { marker-fill: #FE9929; }   [value > 4] {␣
↪marker-fill: #EC7014; }   [value > 5] { marker-fill: #CC4C02; }   [value > 6] {␣
↪marker-fill: #993404; }   [value > 7] { marker-fill: #662506; }   [frame-offset =␣
↪1] { marker-width: 10; marker-fill-opacity: 0.05;}   [frame-offset = 2] { marker-
↪width: 15; marker-fill-opacity: 0.02;} }"
      }
    },
    {
        "type": "L.cartoDB",
        "visible": false,
        "wms": false,
        "source": {
            "apiKey": "default_public",
            "username": "cartojs-test",
            "layers": [{
                "cartocss": "#layer { marker-width: 7; marker-fill: #EE4D5A; marker-
↪line-color: #FFFFFF; }",
                "sql": "SELECT * FROM ne_10m_populated_places_simple WHERE adm0name =
↪'United Kingdom'"
            }]
        }
    }
]
```

The implementation supports the use of CARTO base maps, Torque.js and CARTO.js. These can be used to display base map tiles, animations and vector overlays.

The configuration of the `source` property of the `L.TorqueLayer` type is explained in the Torque.js reference documentation. The `source` property of the `L.cartoDB` type includes the `apiKey` and `username` required by the CARTO.js client and a `layers` property which defines a list of carto.layer.Layer objects.

To see CARTO platform examples in OVE, load the controller by accessing the URL `http://OVE_CORE_HOST:PORT/app/maps/control.html?oveSectionId=SECTION_ID&layers=2,3,4&state=World`. To learn more, see examples on using CARTO.js and the example on using Torque.js with Leaflet.