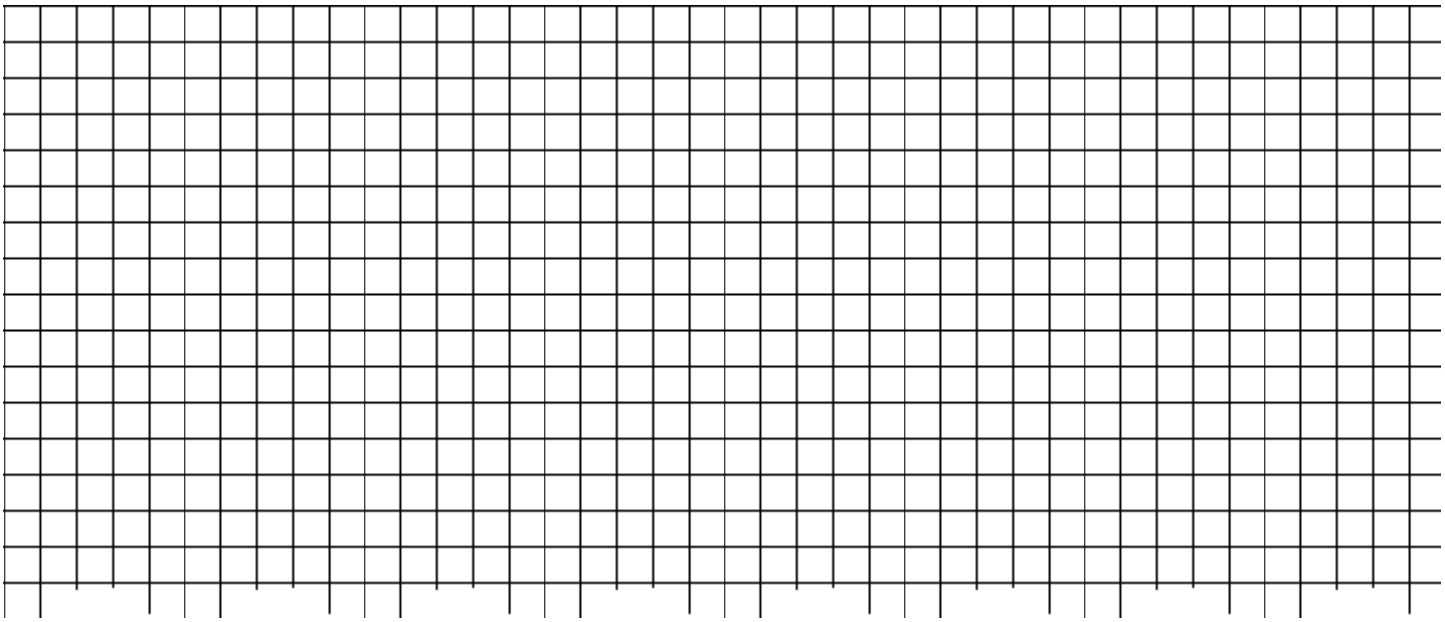


Stanag On Demand Server

On-Demand / Live FMV

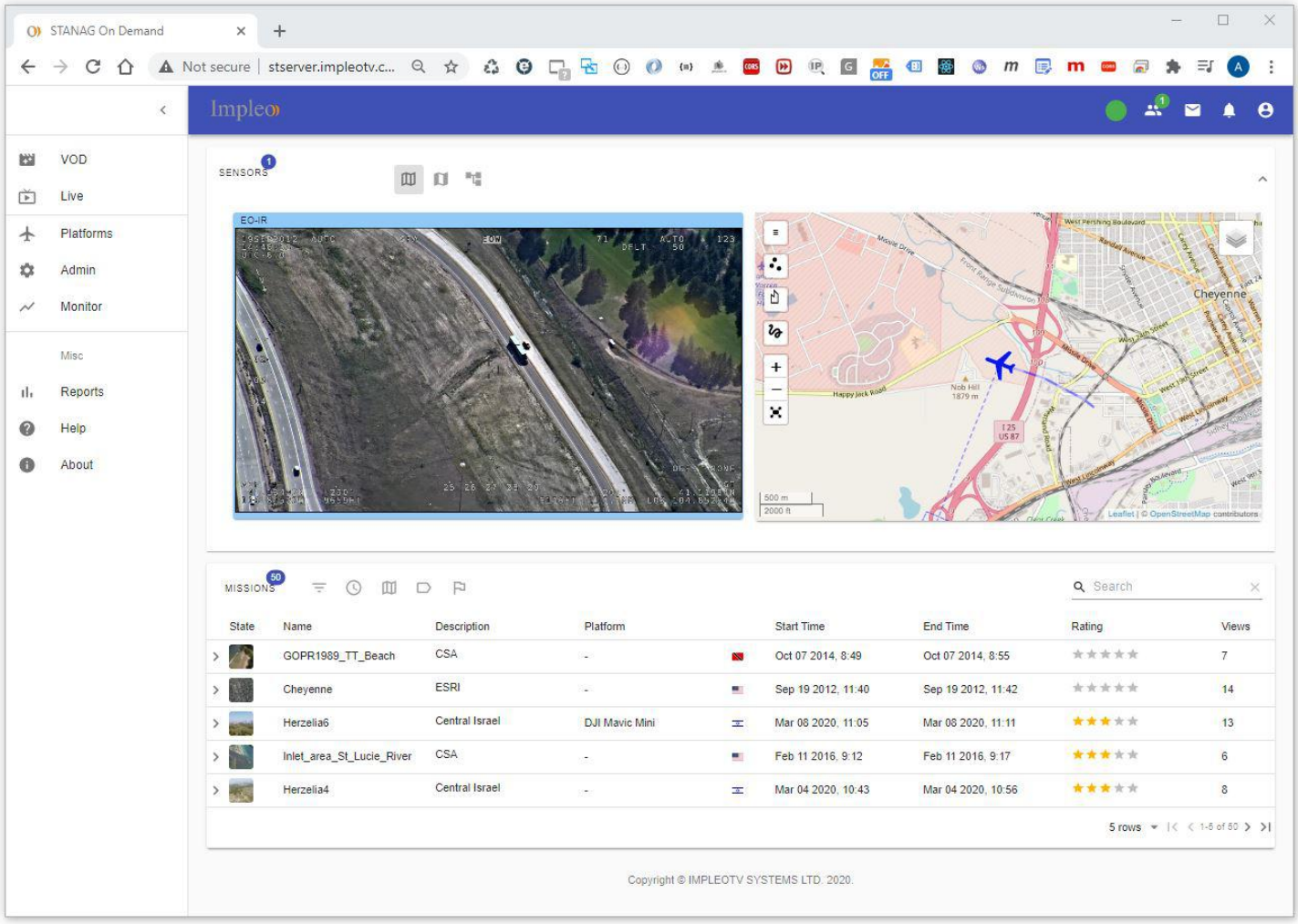
Impleotv Systems Ltd

Table of contents

1. Stanag On Demand Server

STANAG On Demand Server delivers On-Demand / Live FMV and geo-spatial metadata to video analysts and field operators in order to help them easily access, analyze, and present mission information. The server allows efficient review of vast amounts of ISR content using mission related / general spatial queries.

- On Demand / Live and Live Low latency video playback.
- Situational Awareness (video and online/offline maps).
- Geo queries.
- Video bookmarks.
- Detailed STANAG MISB metadata presentation.
- VMTI (Video Moving Target Indicator) support.
- Multi-channel STANAG stream recording (UDP unicast/multicast).
- SRT (Secure Reliable Transport) support.
- Geojson annotations (over the video and maps)
- Powerful administrative tools, user/groups authentication and authorization.
- Video/metadata transcoding services.
- Scheduled operation.
- Activity logger.
- REST API.
- Front-end JS components for custom web client development.
- ITAR free.



STANAG On Demand Server works with **Platforms**, **Missions** and **Sensors**.
More info

For more info please visit ImpleoTV website.

2. Getting started

2.1 Server installation

STANAG On Demand Server is installed as a set of microservices. Both, Linux and Windows can be used as a host.

Though it is possible to deploy the solution by directly installing the components, this manual will describe a preferable method that uses containerization (with Docker) and service orchestration tools. For local server deployment (single host) we'll use **docker-compose** whereas in case of distributed deployment **Kubernetes** is probably more suitable.

2.1.1 Basic setup

If used directly, without reverse proxy, **STANAG On Demand Server** requires the following ports to be opened:

Port	Description
8080	Server Backend and frontend proxy.
8065	Stream Monitor
9001	MQTT over Websocket (for Stream Monitor)
9010 - ...	Video Stream Start Port. Open more ports in the range for every additional channel

Deploying using Docker Compose

Docker compose can be used to quickly start up the **STANAG On Demand Server**.
This deployment method is mostly recommended for stand alone (single host) configuration.

To deploy **STANAG On Demand Server** using **docker-compose**, download and install Docker and Docker Compose, then reboot (important!!!) and follow the steps below.

1. Download **stserver-install.tar.gz** file that contains **docker-compose.yml** and accompanying configuration files and save them to a local folder (for example, **~/stserver/**).

```
wget https://impleotv.com/content/stserver2/setup/stserver-install.tar.gz
```

Extract files:

```
tar -xvf stserver-install.tar.gz
```

2. In a terminal, change directory to the location of **docker-compose.yml** file.
3. Edit **.env** file, if needed.

```
SERVER_NAME=StServer
LICENSE_DIR=~/licenses/stserver/
SERVER_PORT=8080
SUPERVISOR_PORT=8065
MONGO_PORT=27016
MONGO_CONNECTION_STR=mongodb://localhost:27016/dbS2
MQTT_BROKER_HOST=tcp://localhost
USING_REVERSE_PROXY=false
HOST_VIDEO_DIR=~/videos/
HOST_ANNOTATIONS_DIR=~/annotations/
HOST_DOCUMENTS_DIR=~/documents/
HOST_UPLOAD=~/Movies/
USE_WEBRTC_VIDEO=false
```

```
MQTT_BROKER= "your mqtt broker url"
```

If you're using the server without reverse proxy, set **SERVER_PORT=80**
LICENSE_DIR entry specifies a path to the host folder where license files will be saved.
HOST_VIDEO_DIR, **HOST_ANNOTATIONS_DIR** and **HOST_DOCUMENTS_DIR** specify a path to local folders that hold video, annotations and documents.
HOST_UPLOAD specify a path to local folder that will be used for video content that is uploaded via ftp or any file copy method.

The following syntax rules apply to the .env file:

- Each line in an env file to be in VAR=VAL format.
- Lines beginning with # are processed as comments and ignored.
- Blank lines are ignored.

4. Start all the services by running **docker-compose**:

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

When you run this for the first time, docker will download the required images and start containers.

That's it! the server should be running now!

To run as non root, you can set UID environmental variable or run export UID before calling docker-compose

5. Open your browser (http://localhost or at the server IP, port 80) and check that everything starts up correctly

Note, we use **network_mode: host** so the server could receive **udp multicast**. In this case container's network stack is not isolated from the Docker host (the container shares the host's networking namespace). In production, you should put a reverse proxy in front of the server.

To stop the services use the following command:

```
docker-compose down
```

2.1.2 Advanced setup

Running server behind reverse proxy

The preferable way to use the **STANAG On Demand Server** is by running it behind reverse proxy. In this case the server will require only port 80 (or 443, in case of https) to be opened (Stream monitor would still need some other open ports). The reverse proxy will pass requests from port 80 to microservices, handle **https** ssl encryption, etc.

In order to use the server with reverse proxy, set **USING_REVERSE_PROXY** to **true**

```
USING_REVERSE_PROXY=true
```

When server is set to use reverse proxy, its default behavior will be to proxy websocket requests to **/ws** or **/wss**, in case of https (unless you explicitly set another mqtt broker) and **supervisor** microservice to **/supervisor** (for stream monitor).

More on reverse proxy configuration.

2.1.3 Serving video

Though **STANAG On Demand Server** can (and will) handle video content without additional components, for production, you should delegate video serving task to a dedicated http server.
For example, if you're using **Nginx** as reverse proxy you can reroute all requests for video content to another server. Or just serve the video with the **Nginx** itself.

Open another http server on port 8084 and serve the video from there.

```
    location /videos/  {
        proxy_pass         http://localhost:8084/;
        proxy_set_header   Host $http_host;
    }
```

2.1.4 Deploying to remote host

Above we just described a common usage of Docker Compose where we copied the docker-compose.yml file along with some environmental arguments to the local machine. To install the server to additional hosts using this method we can login into that additional machine and repeat the procedure. The disadvantages in this case is that for any change in the application version or Compose file, we have to copy, connect to the remote host and re-run. There are 2 (better) methods to deploy the application to the remote host.

Using DOCKER_HOST environment variable to set up the remote host engine

```
DOCKER_HOST=“ssh://user@remotehost” docker-compose up -d
```

Using Docker contexts

Docker Contexts are an efficient way to automatically switch between different deployment targets.
To access the remote host in an easier way with the Docker client, we can create a context that will hold the connection path to it.

```
docker context create remote --docker "host=ssh://user@remotemachine"
```

Now, we can control everything from the local machine:

```
docker-compose --context remote up -d
```

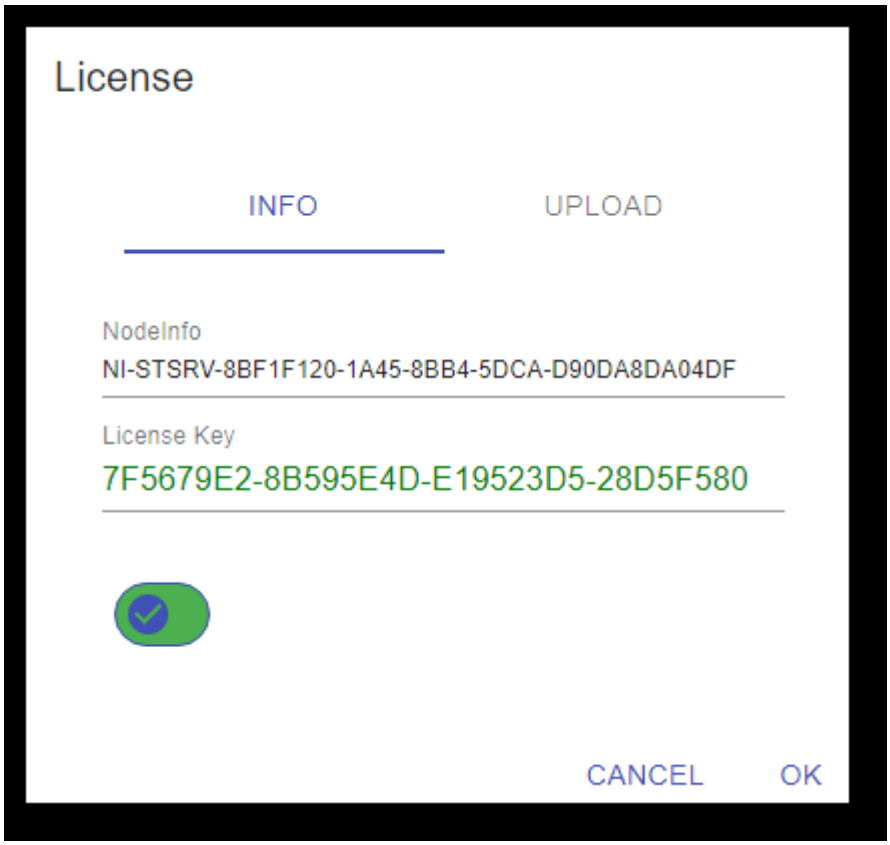
2.1.5 License

STANAG On Demand Server without license will work in demo mode (with restrictions). In order to lift demo restrictions you should provide the license using one of 3 options:

- Using GUI frontend to upload the license
- Passing license info as the arguments (with --licenseFile and --licenseKey)
- Copying license file (.lic) and a key (.txt) file into current working directory **LICENSE_DIR** defined in **.env**

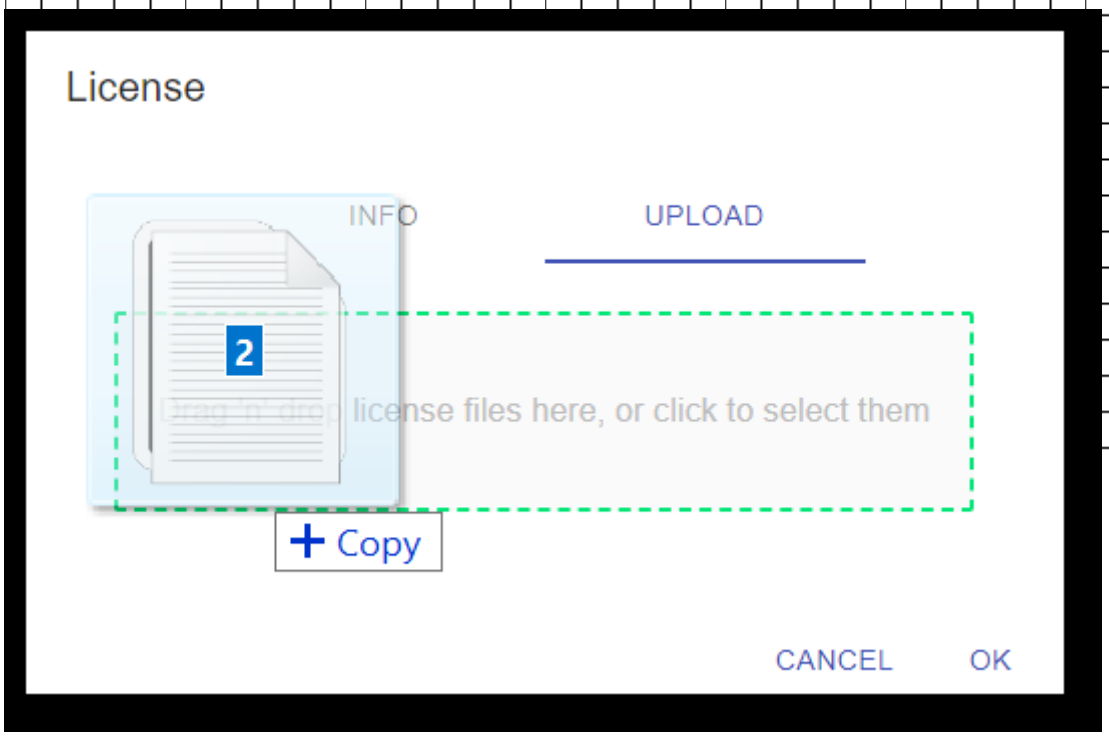
Getting license

In order to get the license, please contact ImpleoTV support, providing the **Node Info** string (can be found at **License** dialog).



Uploading license

You will receive 2 files - license file and key file. Simply drag and drop them into the **Upload** dropzone.



It is also possible to simply copy the licenses to the docker volume mounted directory

2.1.6 Kubernetes

TBD

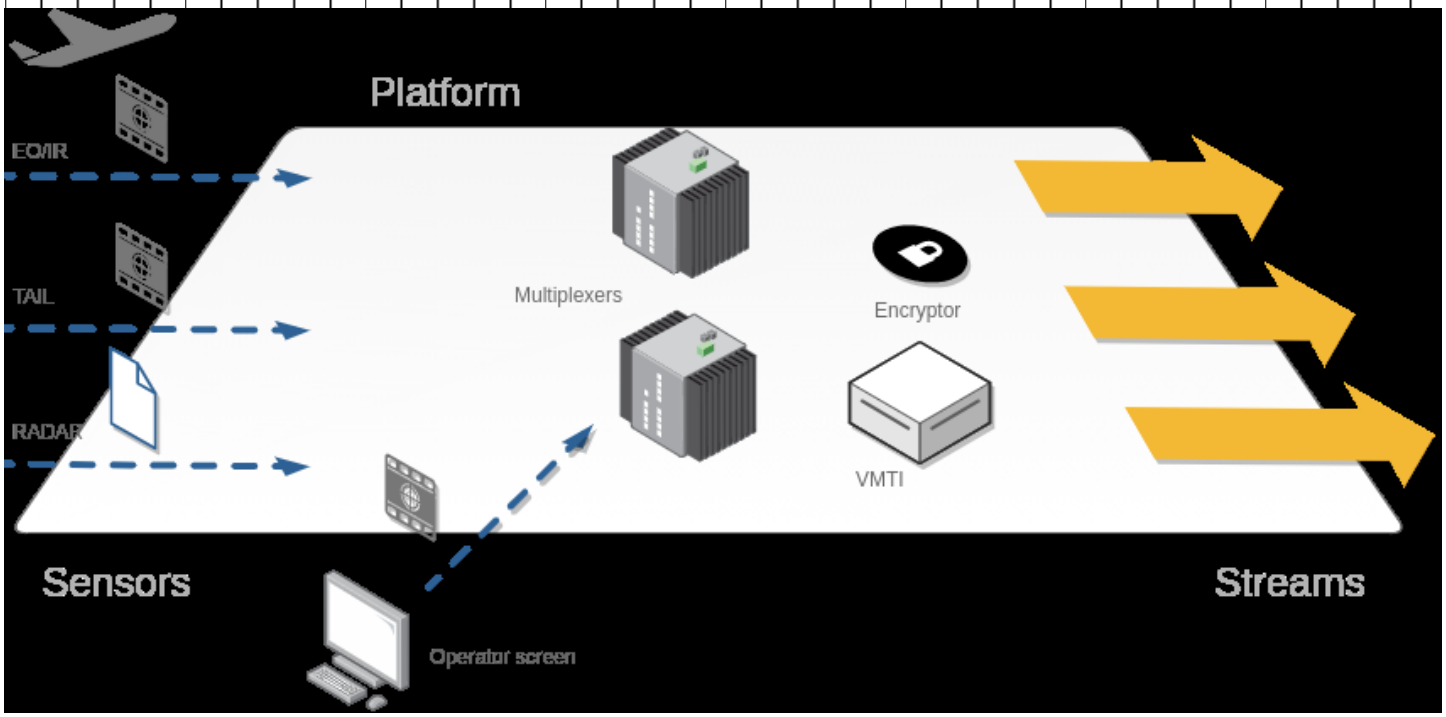
2.1.7 Getting help

To get help with **STANAG On Demand Server**, please contact us at support@impleotv.com.

2.2 Configuration

2.2.1 Platforms and sensors.

In order to be able to work with live video streams you must configure them first. In generally, server works with logical entity **Platform** - plane, drone, car, etc that can carry multiple **Sensors** (video cameras) or other data sources.
Operator configures the **platforms.yml** file according to the existing platforms and network topology.



Basic platform configuration

Let's use an example to show the configuration file that defines two UAV platforms (with names **UAV_1** and **UAV_2**)

First UAV has 4 video sensors:

- EO/IR sensor with the stream coming on udp://227.1.1.1:30120
- Tail camera sensor with the stream coming on udp://227.1.1.2:30122
- Operator screen capture sensor with the stream coming on udp://227.1.1.3:30123
- Radar capture sensor with the stream coming on udp://227.1.1.4:30124

Note, **Radar** sensor has **active** setting set to false (active: false), so this sensor is not monitored (set it to true, to enable monitoring).

Second UAV has only one sensor:

- EO/IR sensor with the stream coming on udp://228.1.1.1:1234

If you only have one UAV (platform) to monitor, there should be one platform in the list.

```
# Platforms
platforms:
# First platform
  - platform: UAV_1
    active: true
    name: UAV_1
    description: First platform
    type: UAV
    sensors:
      - sensor: EO
        name: EO/IR
        description: EO/IR sensor
        active: true
        type: video
        url: udp://227.1.1.1:30120

      - sensor: Tail
        name: Tail
        description: Tail camera
        active: true
        type: video
        url: udp://227.1.1.2:30122

      - sensor: Operator
        name: Operator screen
        active: true
        type: video
        url: udp://227.1.1.3:30123

      - sensor: Radar
        name: Radar
        active: false
        type: video
        url: udp://227.1.1.4:30124

# Second platform
  - platform: UAV_2
    active: true
    name: UAV_2
    description: Second platform
    type: UAV
    sensors:
      - sensor: EO
        active: true
        name: EO/IR
        type: video
        url: udp://228.1.1.1:1234
```

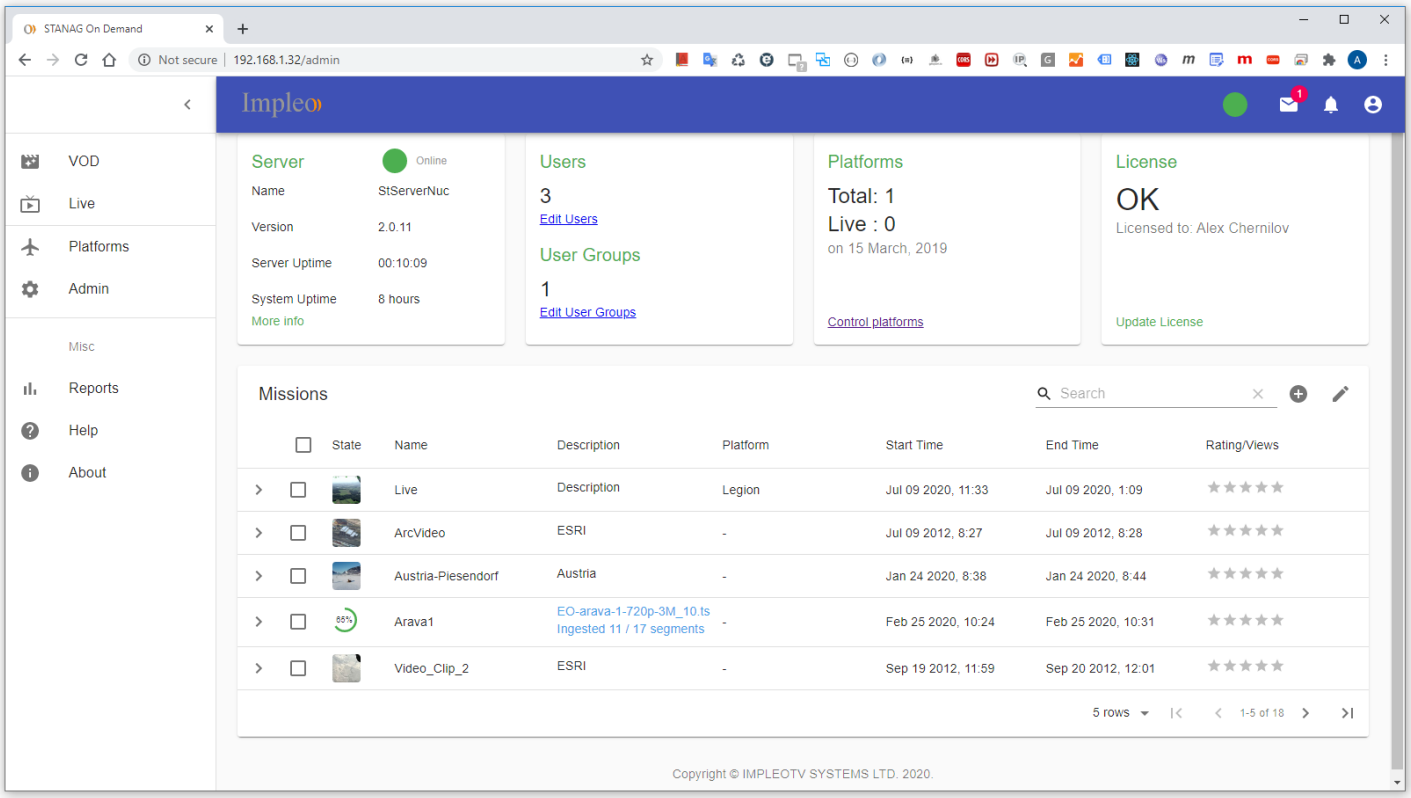
Note. If you have to select the specific network interface, use add ?localaddr= to the url:

```
udp://227.1.1.1:30120?localaddr=192.168.1.28
```

2.3 Admin

Admin window provides a quick access to the basic server operation.

- Server info
- Users control
- Usergroups control
- License upload
- VOD mission control



Admin window is only visible to users with administrative privileges.

2.4 Users

STANAG On Demand Server requires user's authentication and authorization. All users, except **guest** (default user) must be registered in order to be able to work with the server. So, the usual workflow here:

- User registers on a server, using registration form.
- **Admin** (user with admin rights) assigns the new user to the user groups, allowing **admin** or **groupAdmin** rights, if required.

STANAG On Demand Server comes with 2 predefined users:

2.4.1 Guest user

Guest user is a default user that is pre-configured on the server. This user cannot be deleted. **STANAG On Demand Server** visitors that have not registered will be treated as **guest** users.

```
User: guest
Password: guest
```

2.4.2 superAdmin user

superAdmin user is a default **admin** user that is pre-configured on the server. This user should be used for initial configuration only.

```
User: superAdmin
Password: 123456
```

For security reasons, always delete **superAdmin** user or change the password once you finished with the initial server setup.

2.4.3 groupAdmin user

groupAdmin user is a special type of **Admin** user, with reduced privileges:

- **groupAdmin** can create and manage missions that belong to the particular group.

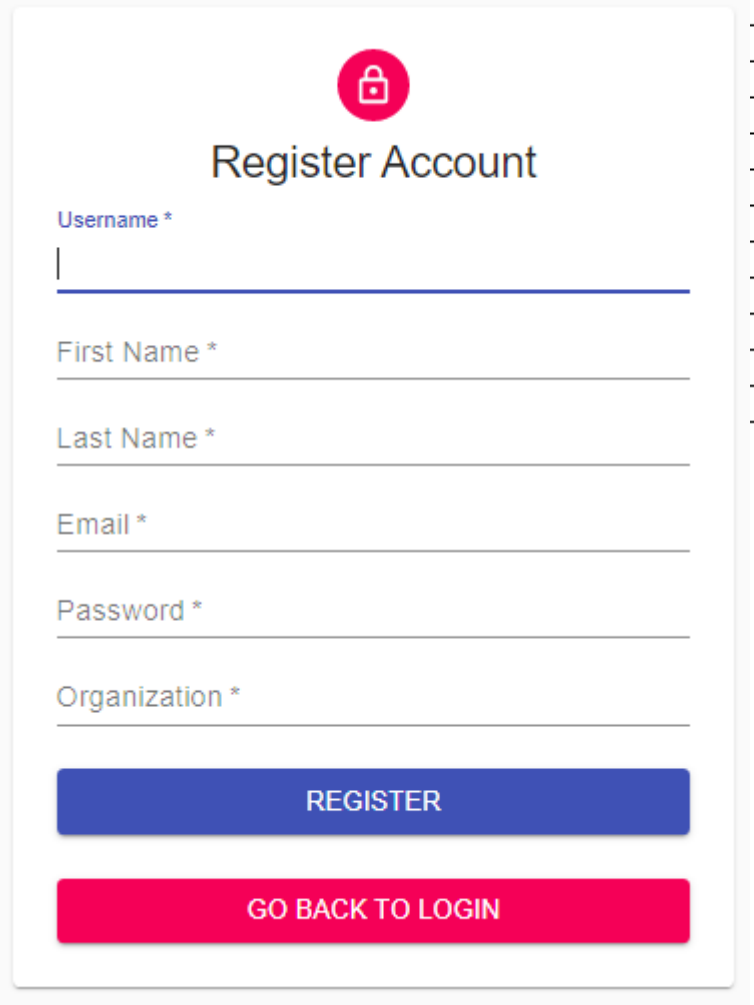
groupAdmin has access to the Admin page, but cannot perform all the available actions, like grant user access rights, etc. To grant a user **groupAdmin** privileges, you must be **Admin** or **superAdmin**.

2.4.4 User login access

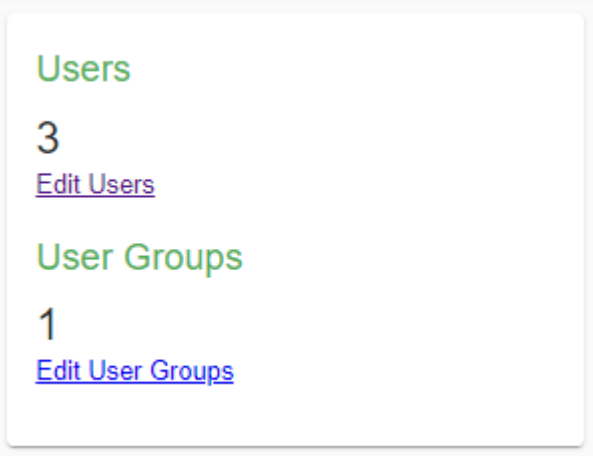
User registration window can be accessed by pressing **Login** icon at the app bar.

2.4.5 User registration

Users should register using the **Register** window.

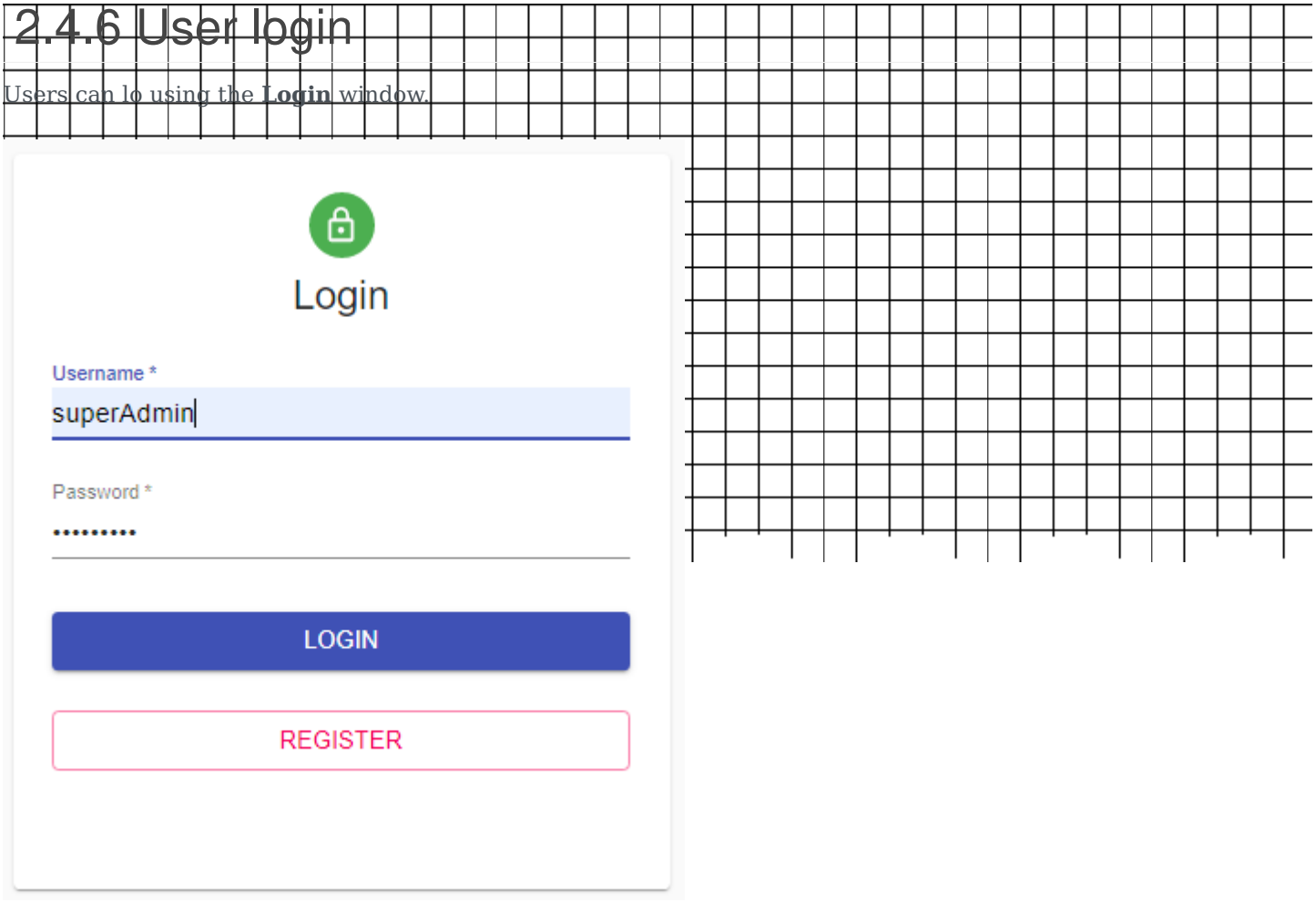


New user will not automatically receive **Admin** rights. Another **Admin** user can grant **Admin** rights using **Edit Users** section of **Admin** window.



2.4.6 User login

Users can lo using the **Login** window.



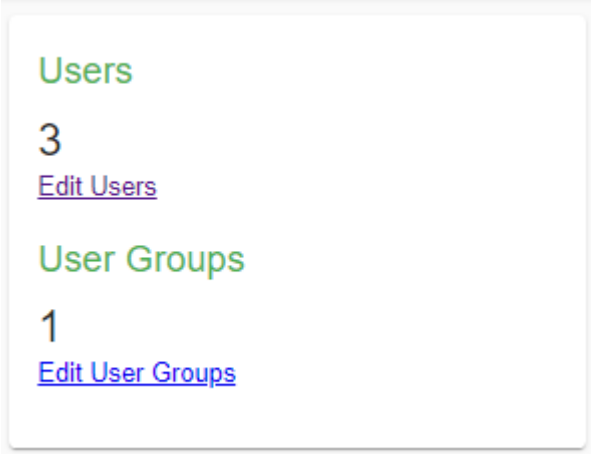
Once user is successfully authenticated, the **Login** icon will turn green.

2.5 Usergroups

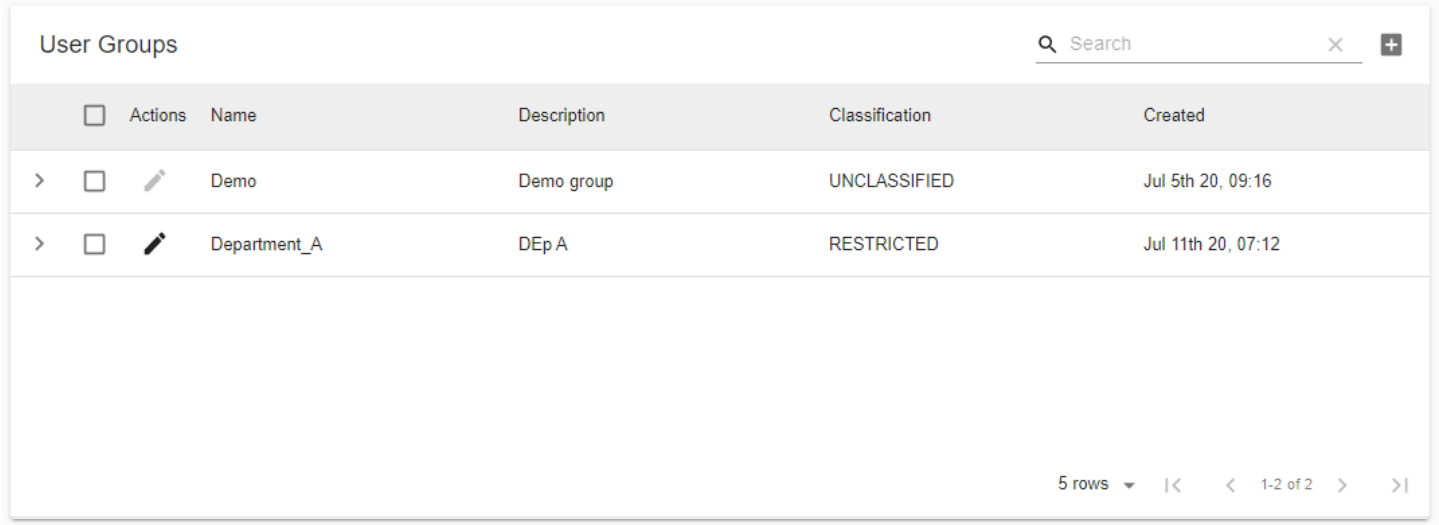
Usergroups allow administrator to limit content access to the specific users that belong to the group.

2.5.1 Demo group

Demo group provides public access to the content. Any user (registered or not) will be able to see the video assinged to this group. **Demo group** is pre-configured and cannot be deleted. Users with Administrator privileges can create and edit **usergroups** via **Edit Usergroups** section of **Admin** window.



2.5.2 Edit usergroups



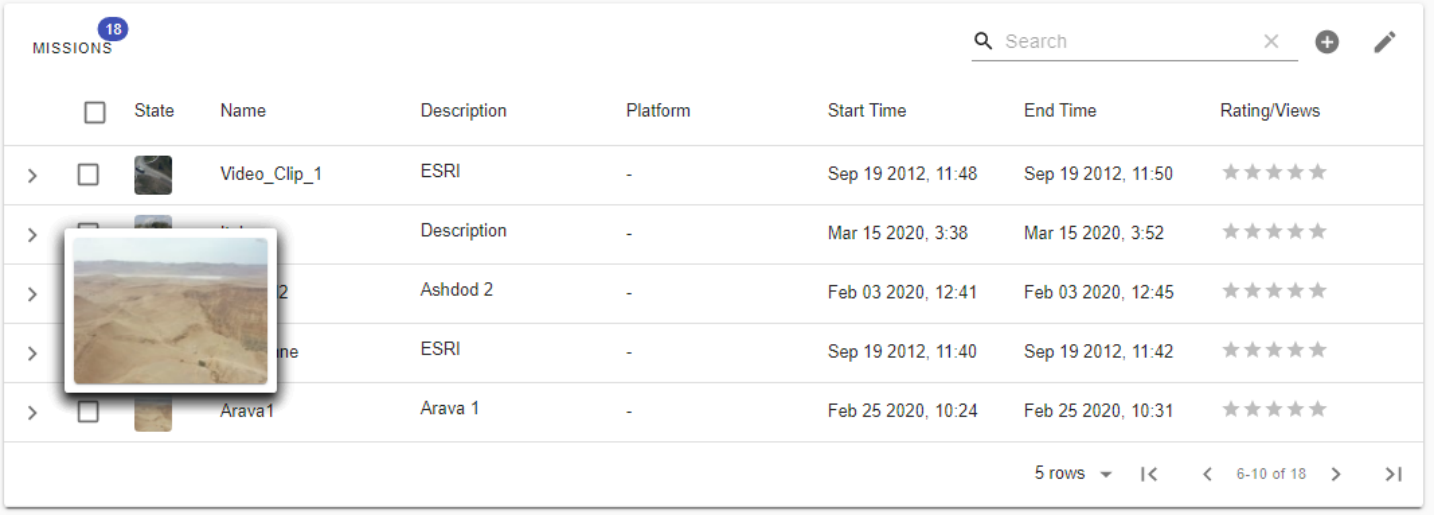
2.6 VOD missions

VOD missions can be created using one of the following methods:

- Manual creation
- Automatic upload with StServer Uploader
- Live mission recording
- Mission import
- Mission creation using server's **REST Api**

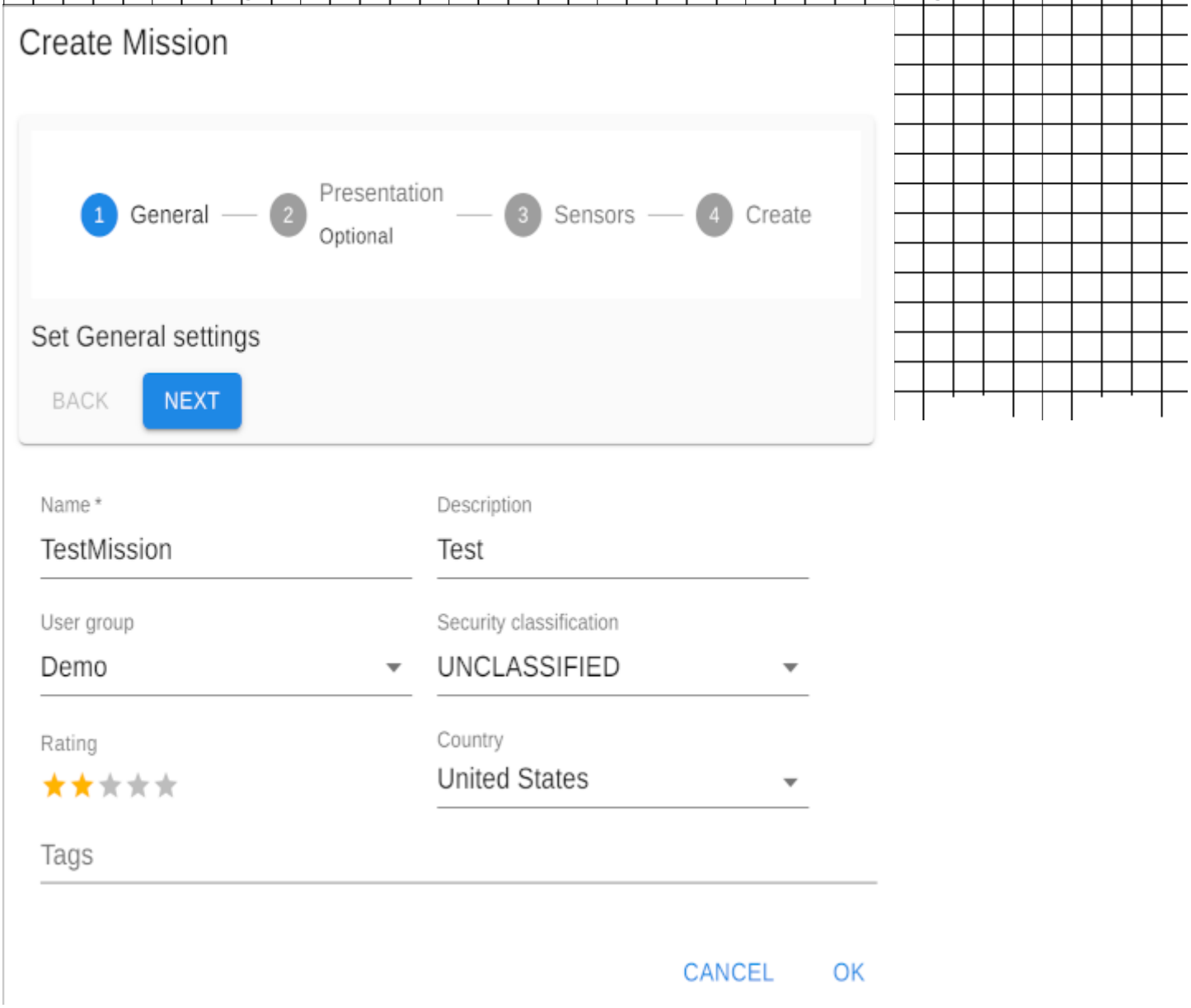
Mission table

Users with admin rights can access **VOD mission control table** section of **Admin** window. This table presents some basic information on available VOD missions. Mission's **sensors** can be accessed by expanding the **mission** row. Hovering over Missions thumbnail will show Mission's key frame.

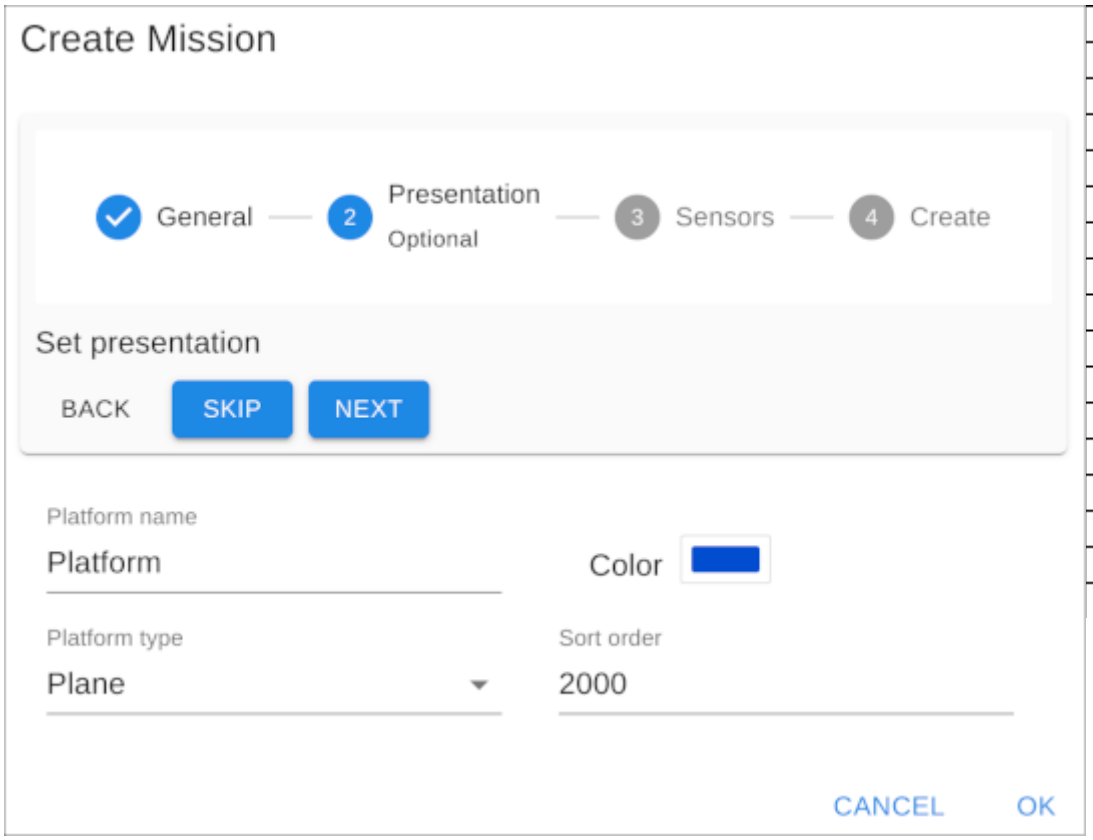


Manual Mission creation

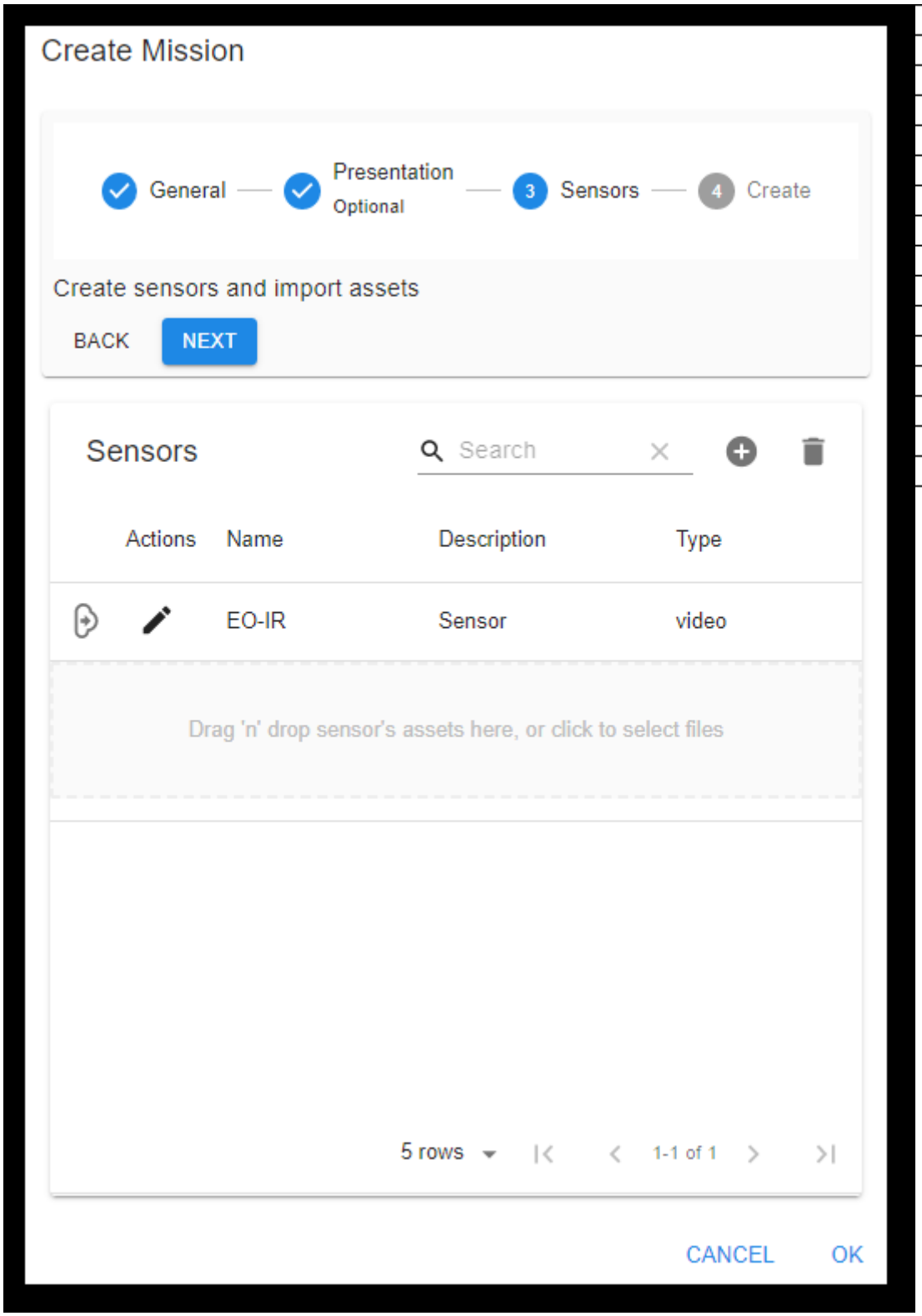
Press **+** button at the right top corner of the **Mission control** table. Mission creation dialog will appear.



Fill **Mission name** (must be unique) and description, assign the **usergroup** and **Security classification**, etc.
Press **Next**



Fill **Platform** related info. Assign platform name (if available). Select presentation color, and **Sort order** (the order missions will appear in the **Missions** table) etc.
Press **Next**

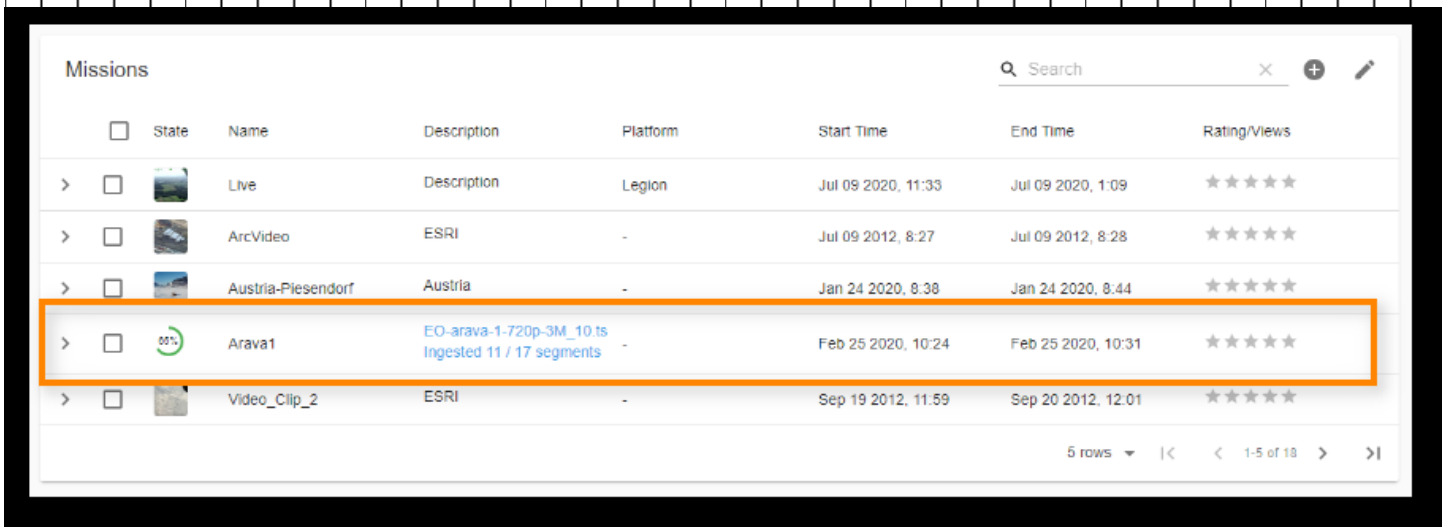


Add video **sensor** (or sensors, if there are more than one), edit name and description.
Expand sensor's **Row** and drag and drop (or click and select) video file.

Supported formats:
- STANAG files (.ts)
- Zip archived HLS - video segments along with a valid video manifest (.m3u8)

Press **Ok** to close window and start upload and processing.

You can find some advanced configuration options in the **sensorHls** section of **./data/config.json** file. More on advanced config.



Now server will create the **mission**, upload the assets, ingest them into internal DB, calculate some geo information (like area) if there is enough required data available, and perform some additional tasks. Upon completion, the mission will be available for playback.

Bulk Mission upload with StServer Uploader

It is possible to automatically upload multiple missions using **StServerUploader**. You can also upload your videos to the server using ftp (or any other method) and use **StServerUploader** to ingest them.

Please see StServer Uploader

2.7 Live missions

STANAG On Demand Server provides two **Live FMV** modes.
- LiveLowLatency mode
- Live HLS mode

These modes address two contradicting requirements for **low latency** vs. **high quality**.

2.7.1 Live low latency

This mode allows video preview with the lowest latency possible. In order to achieve this goal, the server must transcode video on the fly into format that can be fed into the browser, without need for plugins. User can configure bitrate and resolution. Higher bitrate and resolution mean higher quality, but the tradeoff will be the bandwidth.
Additionally, as stream is sent to each and every client, this mode requires more processing power and network resources than the **Live HLS** mode, hence limiting the number of concurrent users. Depending on the video source profile you may expect 150 - 400 ms latency. **Live low latency** mode is always active, so if there is a video stream available it can be seen without any additional actions.

2.7.2 Live HLS

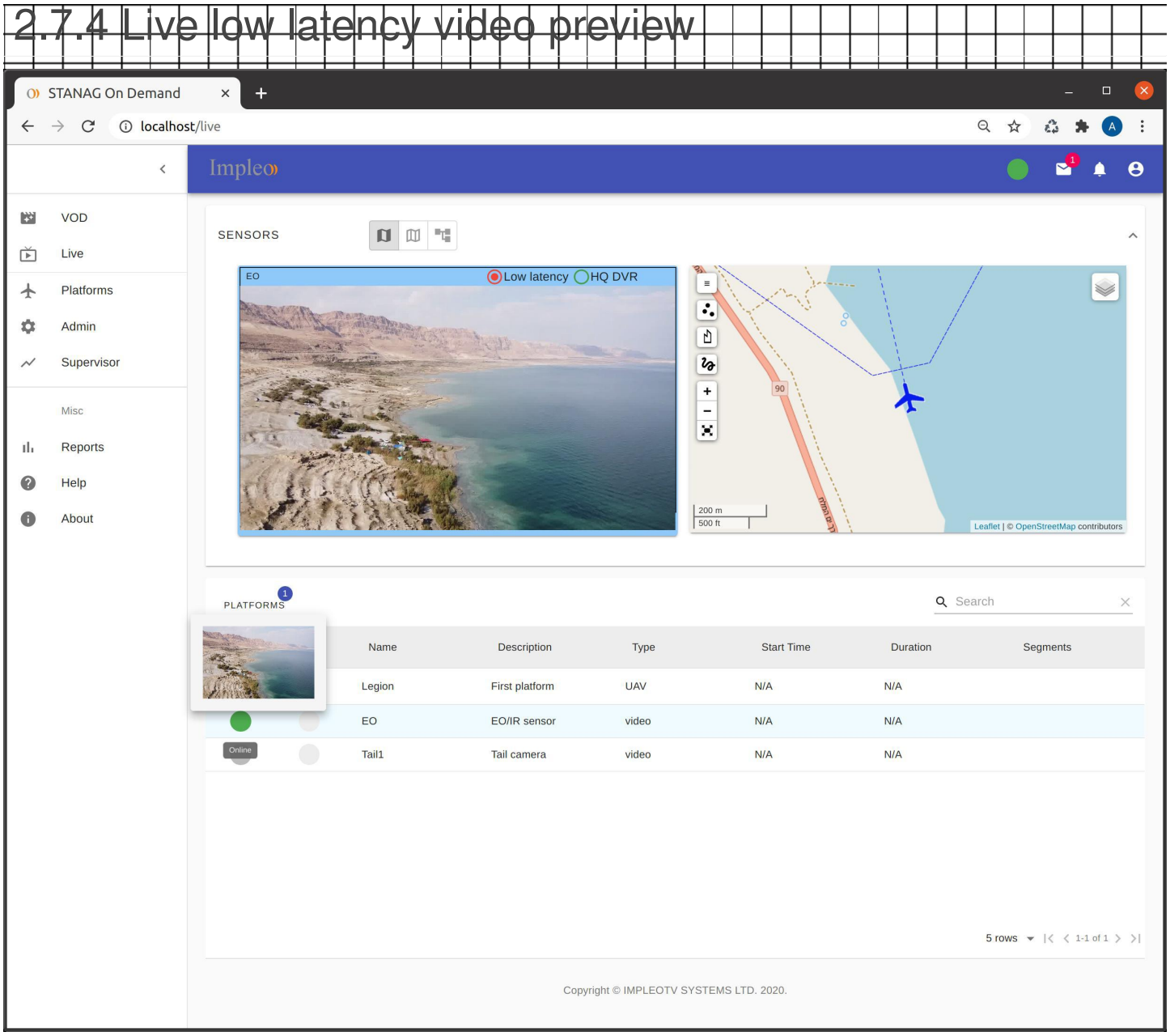
This mode allows video consumption with original quality as well as **DVR** functionality.
The tradeoff will be a significant delay (depends on the HLS segment duration) In this mode you may expect 30 - 40 sec latency.
Unlike **Live low latency** mode, **Live HLS** scales much better. Using **CDN**, you may serve millions of concurrent clients without (almost) any additional load on the server.
Live HLS mode is only available if the recorder is activated.

2.7.3 Video Stream configuration

STANAG On Demand Server works with **Platforms** (such as UAV, vehicle, etc). Every **Platforms** can have one or more video **Sensors**. Network configuration of video sources provided by the **Platforms.yml** file that should be placed into **data** directory.

- Platform configuration Platform configuration
- Live Preview configuration Live stream configuration

2.7.4 Live low latency video preview



You can see the low latency video preview by selecting the desired **Sensor** in the **PLATFORMS** table. When the entire Platform is selected all available sensors will be shown.
In addition to live video, the platform position with sensor's footprint will be also shown (if relevant metadata exists, of course).

Every sensor has the status icon, that represents it's current state.

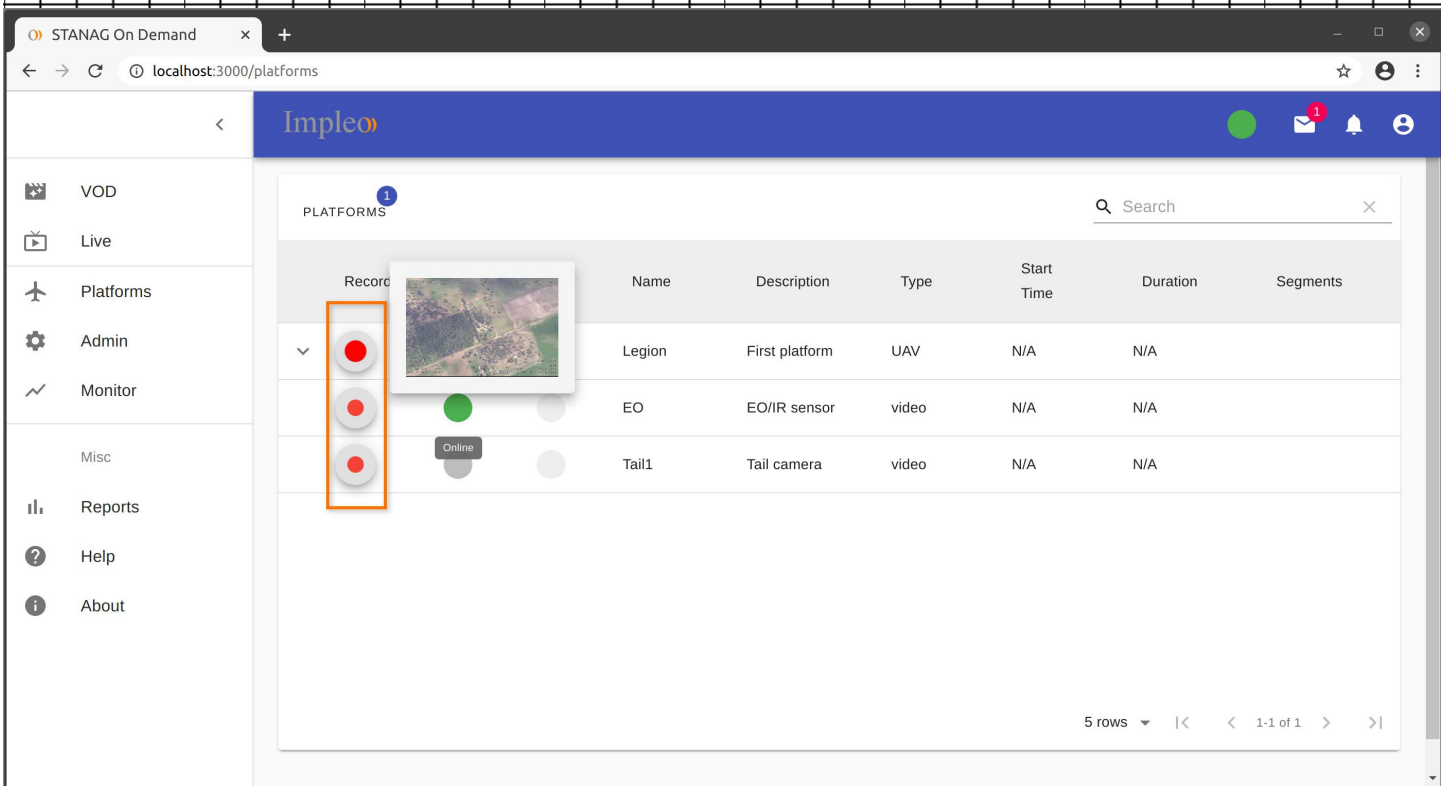
Live stream can be in one of the following states:

- **Unknown**. This state will be reported when Stream Monitor goes offline (or not started yet) and therefore there is no info on the stream available.
- **Offline**. Stream not present.
- **Online**. Stream available.
- **Lost**. Stream lost (not detected for the period of time specified by --timeout. After that, the stream is considered as **Offline**.

Built in stream monitor (Supervisor) provides some additional information on live streams.
For more info, see Supervisor

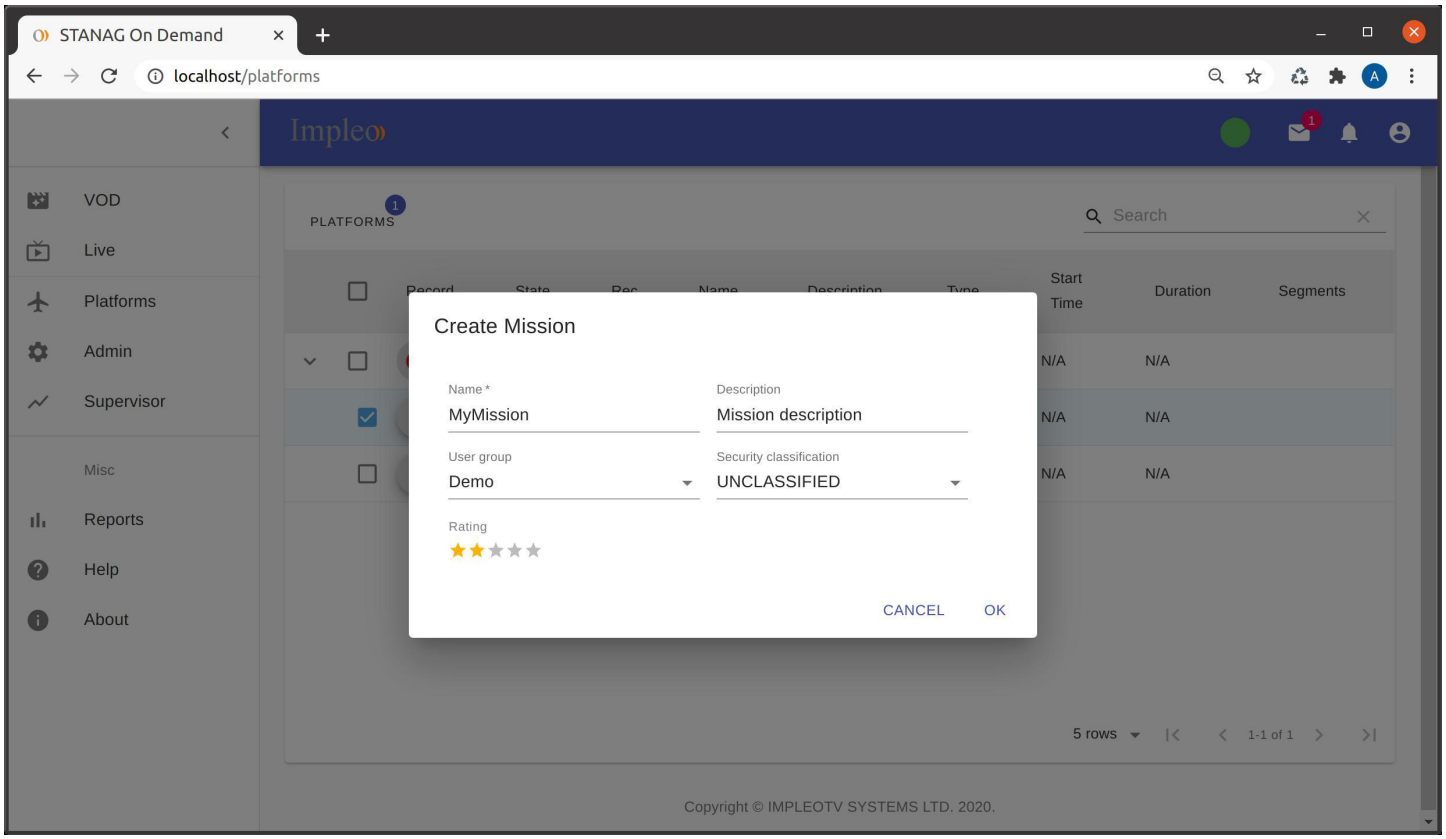
2.7.5 Recorder / Live HLS control

You can control the recorder in the **Platforms** window.



You can see a thumbnail size video preview by hovering over the sensor state icon.

As the logical output of recording will be **Mission**, we must provide some mission level information before starting an actual recording. Select the desired platform's sensors (or entire platform) and click **Record** button.



Fill mission descriptive information and press **Ok**. Recording will start.

If you clicked a **Mission** level record button, all active sensors will be started.
If the **Sensor** level record button was clicked, only that particular sensor will be activated.

When you want to stop recording, there are some options available:
- Stopping **all sensors** and closing (finalizing) the mission.
- Stopping **specific sensor** and keep mission running (even with no active sensors). This mode allows later sensor recording restarting, in a way that the sensor clips will belong to the same mission.

Use **Mission** or **Sensor** level record stop button depending on the required functionality.

You can find some advanced configuration options in the **sensorHls** section of **./data/config.json** file. More on advanced config.

2.7.6 Switching between modes

You can switch between **Live low latency** and **Live HLS** modes using selection buttons at the top of the video window. When in **Live HLS** mode, you can use **DVR functionality** - there will be a slider shown at the bottom of the video window, so you can go back into the recording, use **time shifted** recording, **live pause**, etc.

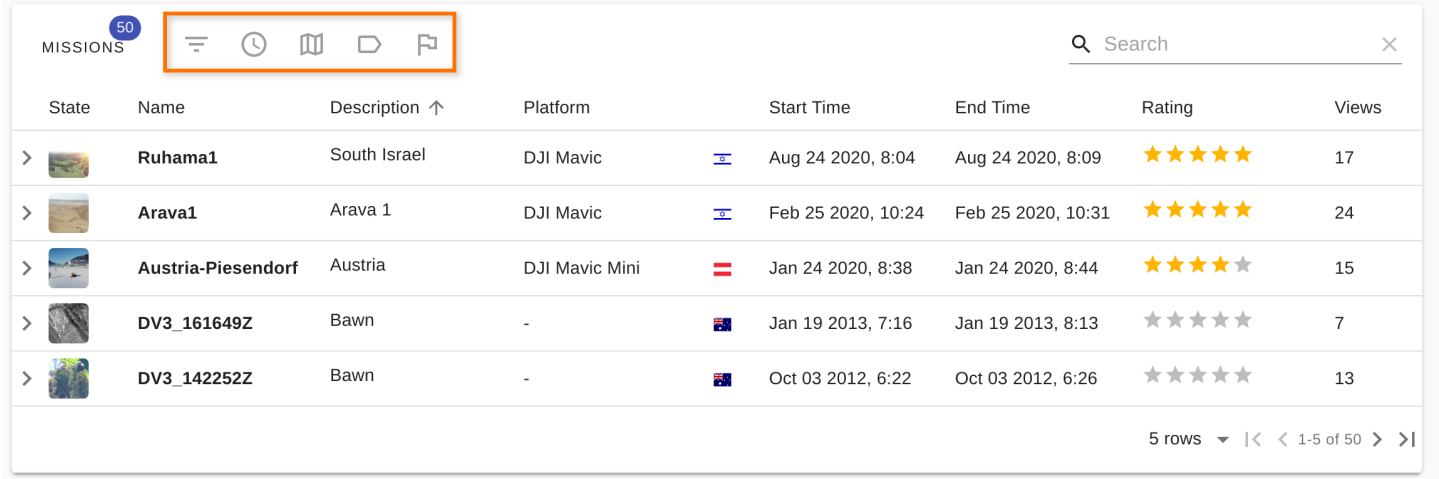
Please note, as recording is done by splitting the stream into a series of short media segments, which are placed on a web server. So, the client won't be able to start HLS playback immediately after recording start - a stream manifest must be first created and at least few segments should be accumulated. So, it will take some time (depending on the selected segment duration) until the Live HLS playback will become available.
Server will ingest metadata when segment recording is complete, so the area, track, etc will gradually become available for analysis and geo queries.

2.8 Filtering missions

At the top of the Missions table you can find some filters that will help to narrow your search to the relevant content only. You can create a custom filter combining the following filters:

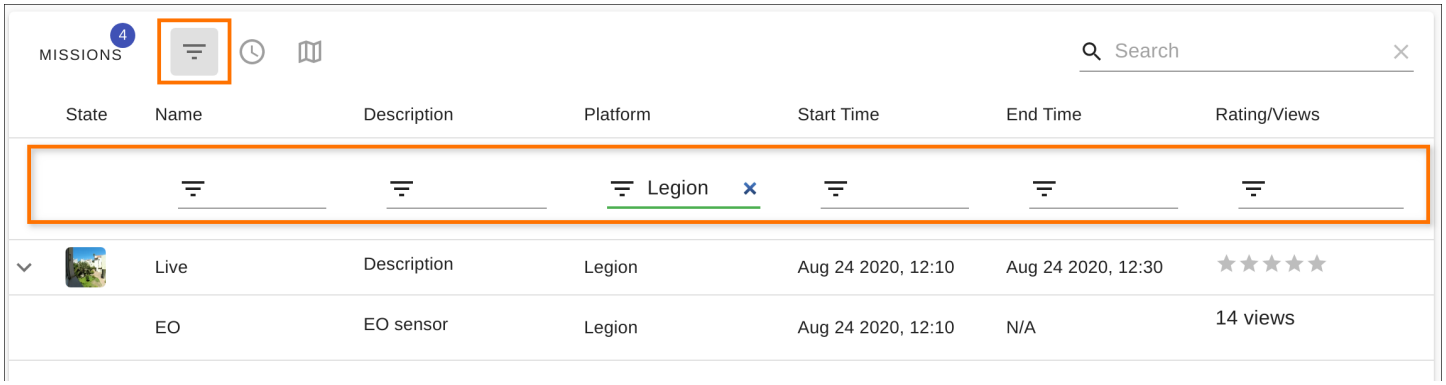
- Columns inline filtering - filter every column separately.
- Mission's start and end time
- Geo queries
- Tag filter
- Country filter

Filters are stored in a session storage, so they are deleted when the browser tab is closed.



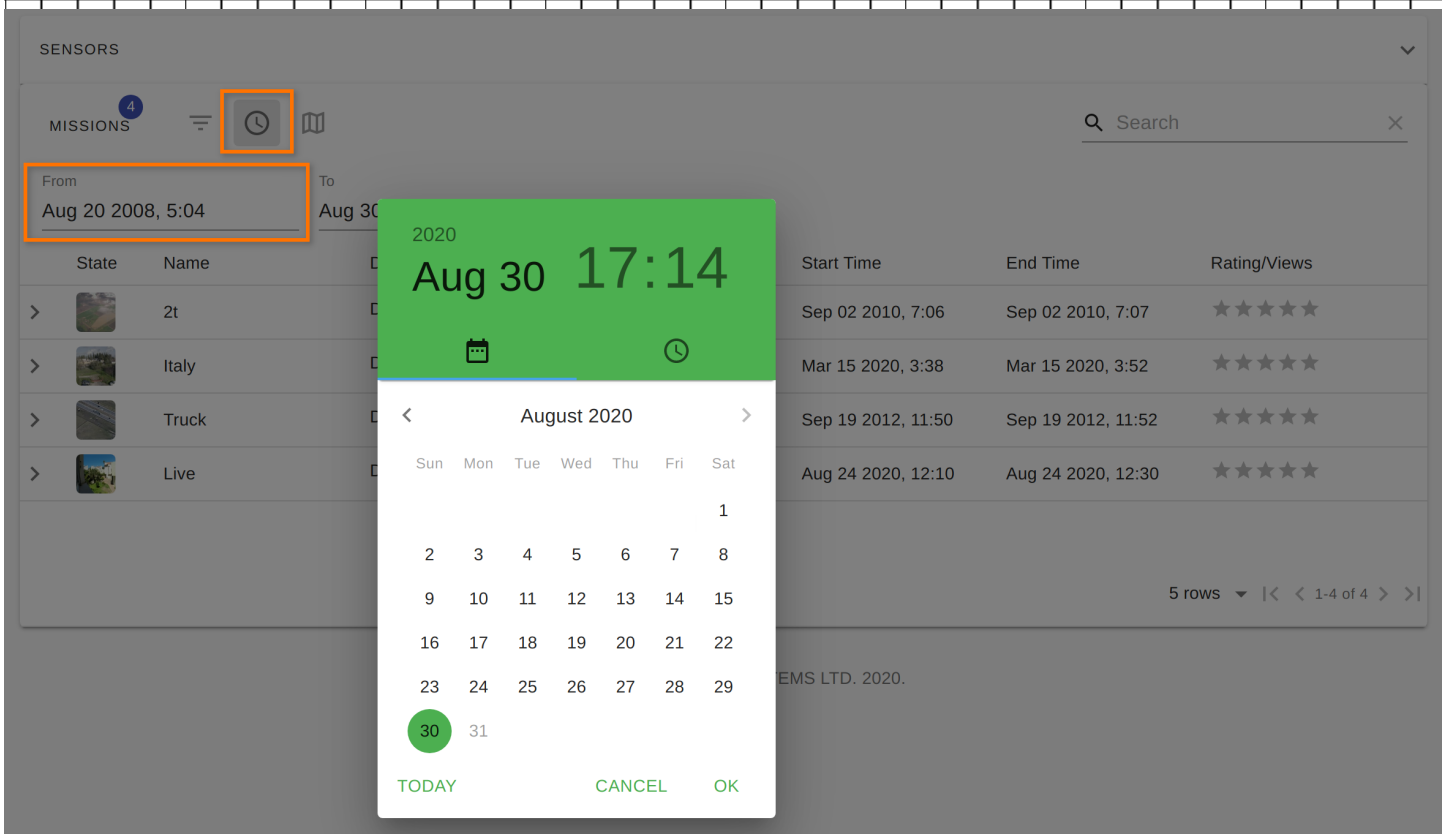
2.8.1 Inline filtering

Type directly into the columns filters:



2.8.2 Time filtering

Select Missions start / end time

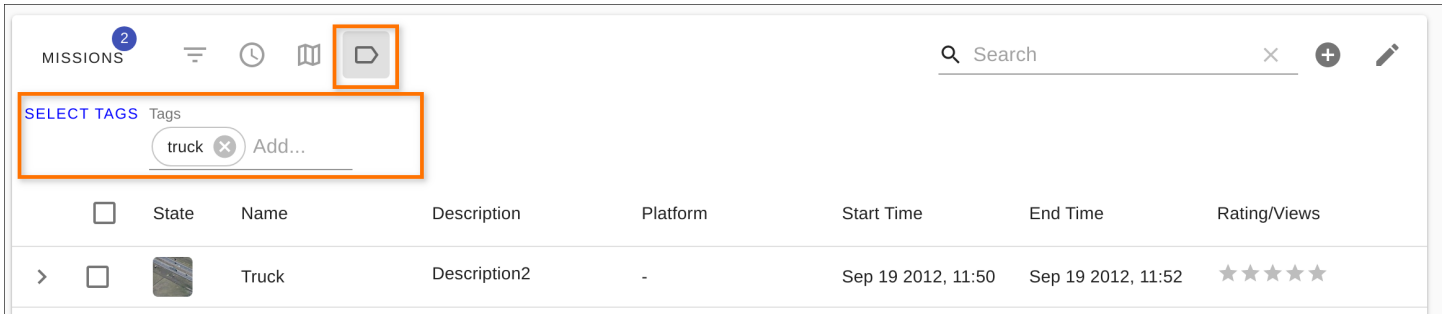


2.8.3 Geo filtering

See Geo Queries / Filter missions for more info on geo queries

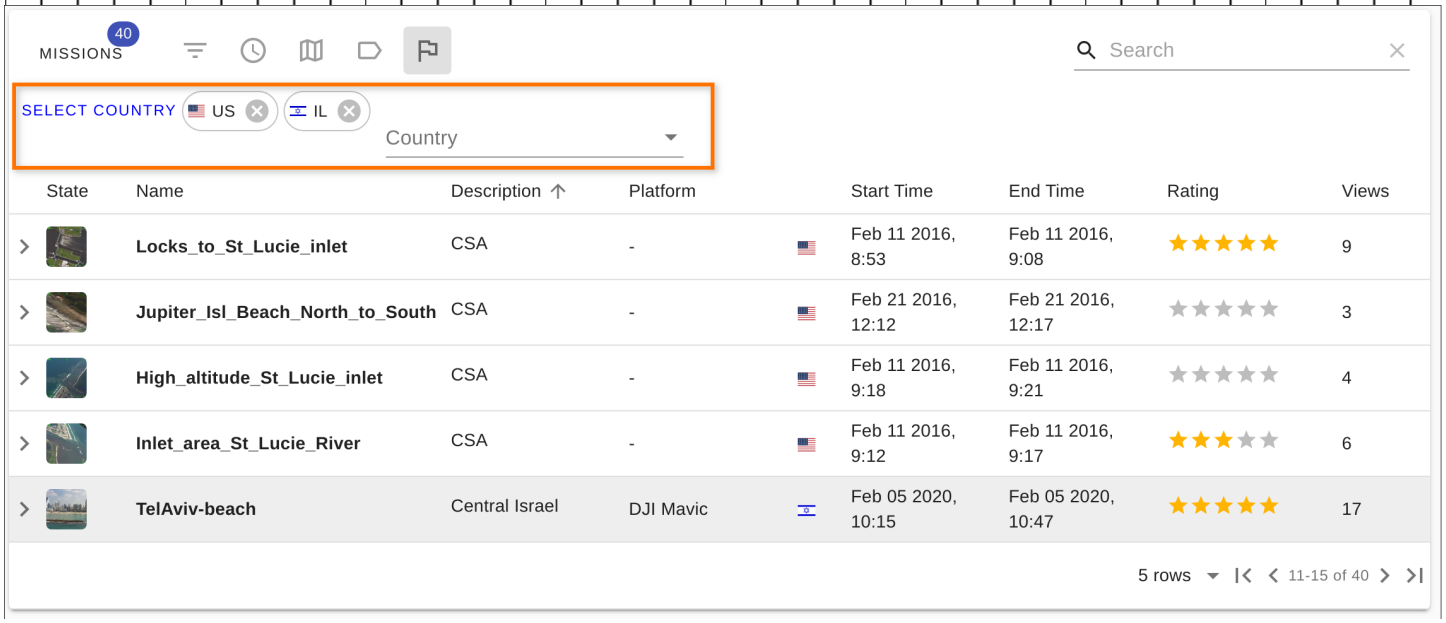
2.8.4 Tag filtering

Select from available tags to filter the missions by tag info



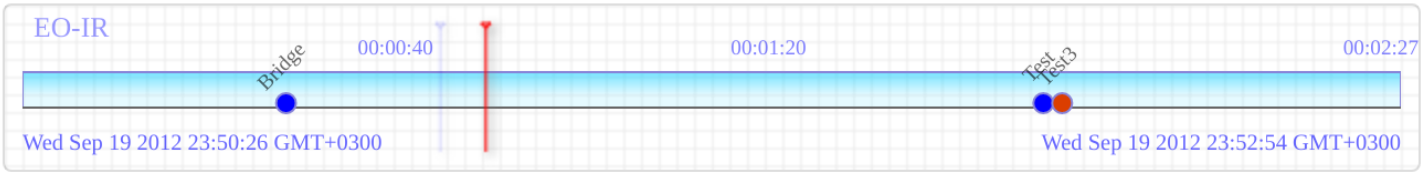
2.8.5 Country filtering

Select from the country list to filter the missions by country code



2.9 Timeline

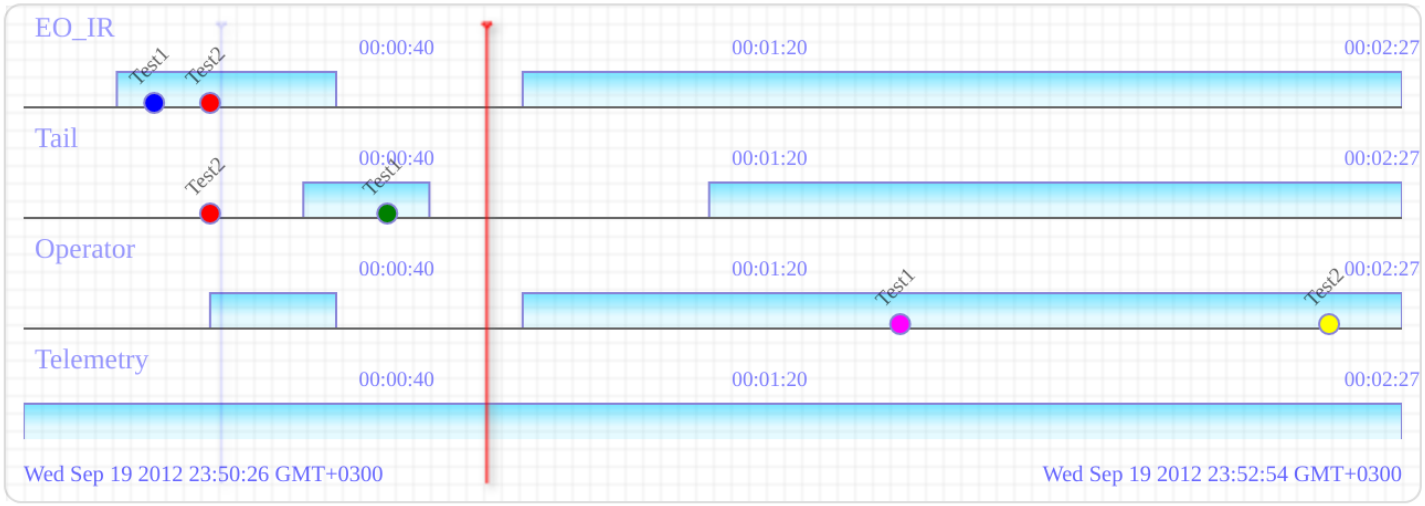
STANAG On Demand Server allows you to present mission / sensor(s) info on a **Mission timeline**.
Sensor's video footage representation is lined up horizontally, along with bookmarks (if exist). The position marker (crosshair) depicts where in the video you are previewing. It also allows you to control the current playback position.



If your recording has gaps, they will be presented on a timeline.

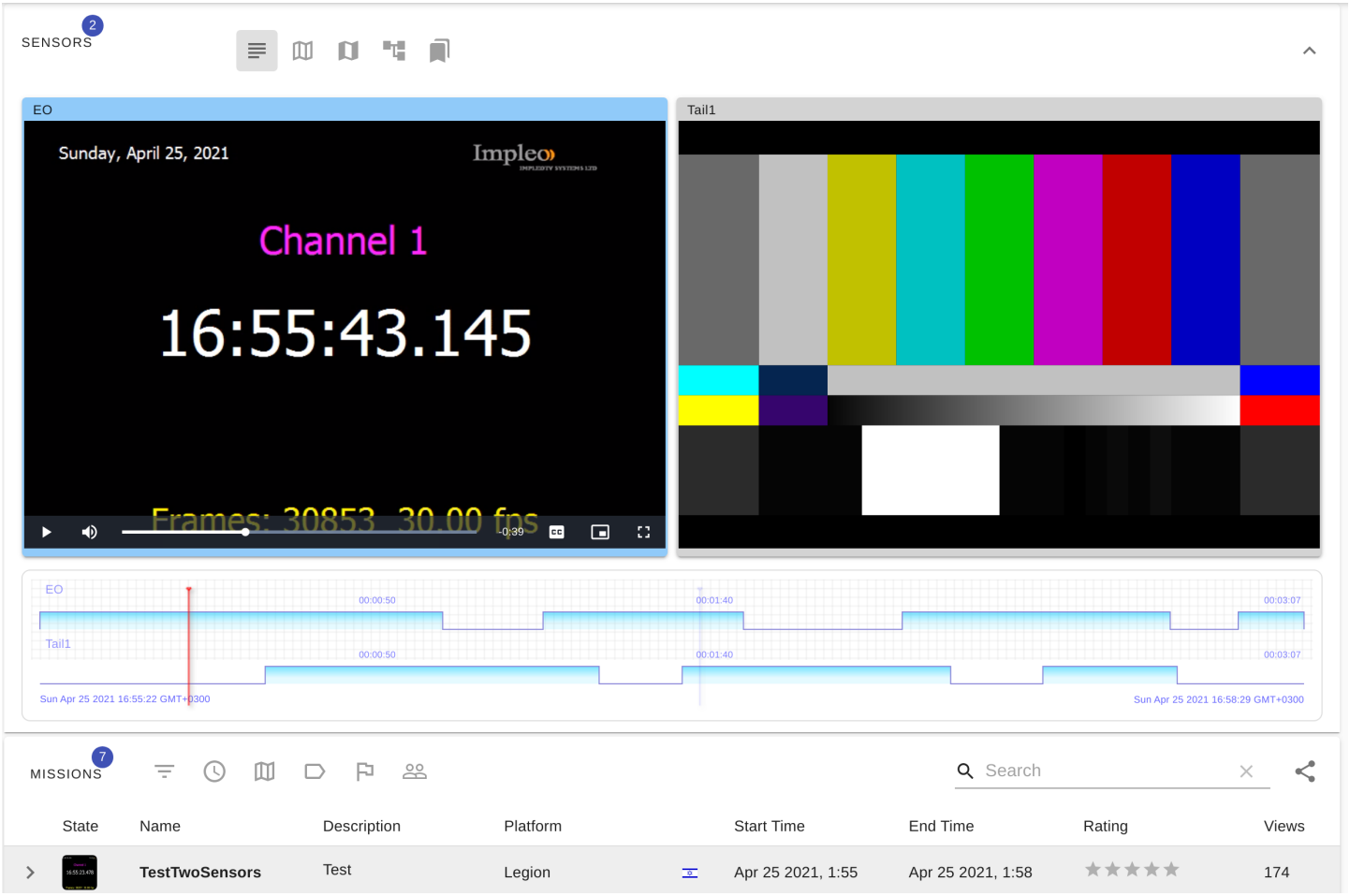
2.9.1 Multi-sensor

Multi-sensor option allows control of all the sensors that belong to the mission.



Synchronized playback

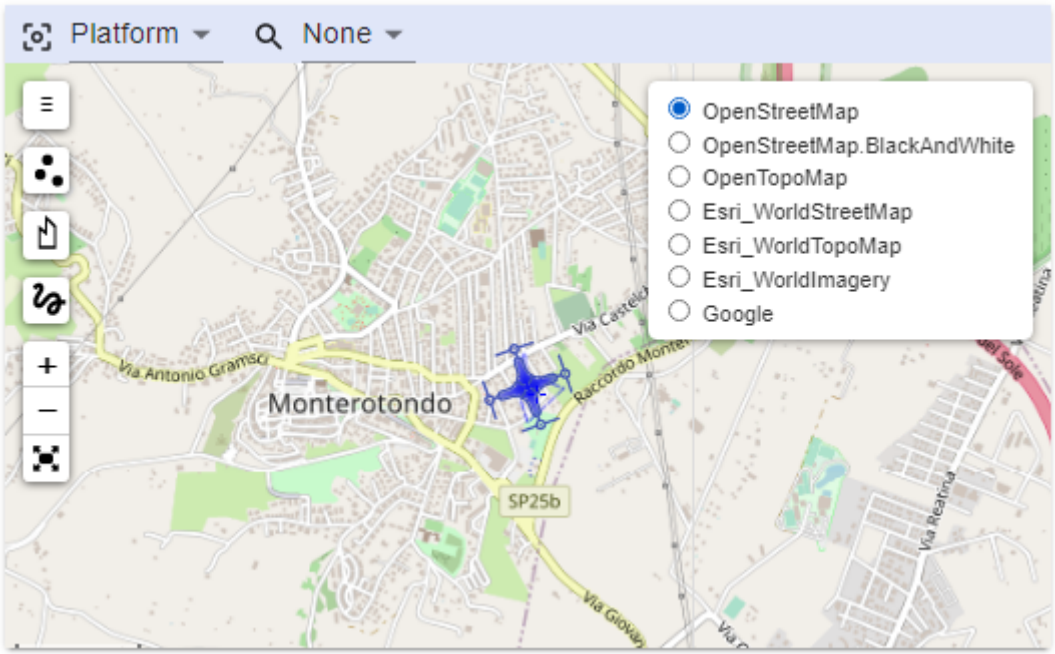
When your mission has multiple sensors, **STANAG On Demand Server** will sync the playback between the recorded video channels.
The active (selected) player will be considered as a **Master** player. All other (slave) players will try to sync to its position (time).
If there is a gap in the recording of the master sensor, it will skip to the next available video. The same will occur if you select a timeline position with no video.
If there is a gap in the recording of one of the slave sensors, their playback will stop until the master player reaches the time where those sensors have video.



Note. The synchronization will only work if all channels contain STANAG video (a video with calendar timestamps).

2.10 Map

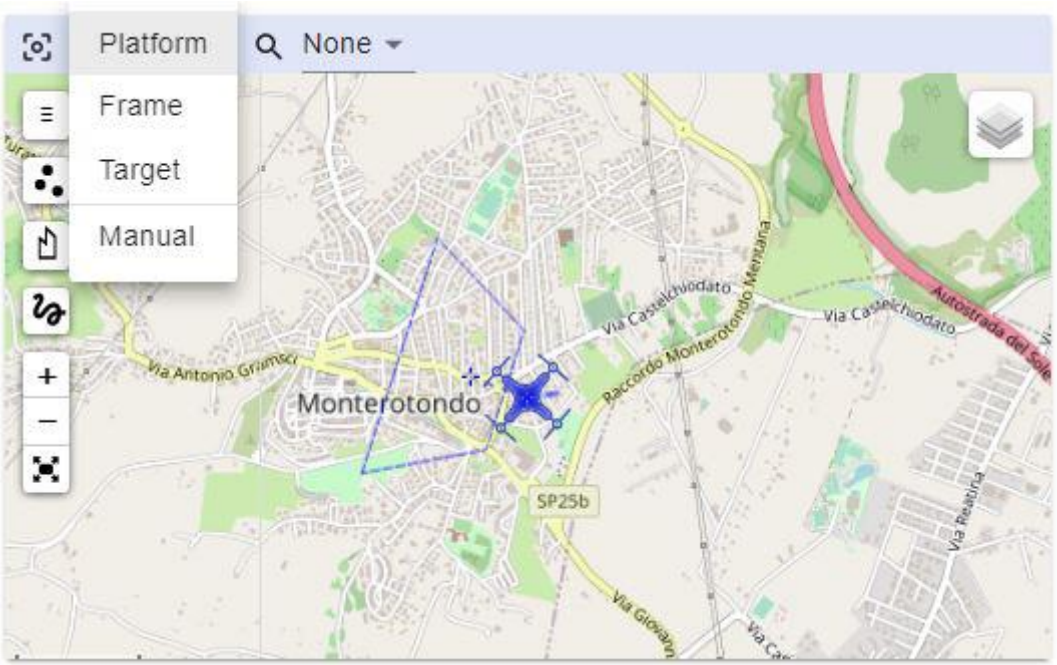
2.10.1 Online Maps



STANAG On Demand Server can use online maps from different providers.

You can select a pre-configured base map in the right top corner of the map.

2.10.2 Center Map control



Map will center itself automatically, depending on the selected mode:

- **Platform** - center map on Platform position
- **Frame** - center map on Frame center
- **Target** - center map on Target (if available)
- **Manual** - do not move map. Manual control

Pre-configured Maps

Server uses some pre-configured map sources. You may change the configuration in the **config.json** file (/data folder)

```
"maps": {
    "layers": [
      {
        "title": "OpenStreet Maps",
        "tileLayer": "https://a.tile.openstreetmap.org/{z}/{x}/{y}.png",
        "attribution": "&copy; <a href=\"http://osm.org/copyright\">OpenStreetMap</a>"
      },
      {
        "title": "OpenStreetMap.BlackAndWhite",
        "tileLayer": "http://{s}.tiles.wmflabs.org/bw-mapnik/{z}/{x}/{y}.png",
        "attribution": "&amp;copy <a href=&quot;http://osm.org/copyright&quot;>OpenStreetMap</a> contributors"
      },
      {
        "title": "OpenTopoMap",
        "tileLayer": "https://{s}.tile.opentopomap.org/{z}/{x}/{y}.png",
        "attribution": "Map data: &copy; <a href=\"http://www.openstreetmap.org/copyright\">OpenStreetMap</a>, <a href=\"h
      },
      {
        "title": "Esri_WorldStreetMap",
        "tileLayer": "https://server.arcgisonline.com/ArcGIS/rest/services/World_Street_Map/MapServer/tile/{z}/{y}/{x}",
        "attribution": "Tiles &copy; Esri &mdash; Source: Esri, DeLorme, NAVTEQ, USGS, Intermap, iPC, NRCAN, Esri Japan, M
      },
      {
        "title": "Esri_WorldTopoMap",
        "tileLayer": "https://server.arcgisonline.com/ArcGIS/rest/services/World_Topo_Map/MapServer/tile/{z}/{y}/{x}",
        "attribution": "Tiles &copy; Esri &mdash; Esri, DeLorme, NAVTEQ, TomTom, Intermap, iPC, USGS, FAO, NPS, NRCAN, Geo
      },
      {
        "title": "Esri WorldImagery",
        "tileLayer": "https://server.arcgisonline.com/ArcGIS/rest/services/World_Imagery/MapServer/tile/{z}/{y}/{x}",
        "attribution": "Tiles &copy; Esri &mdash; Source: Esri, i-cubed, USDA, USGS, AEX, GeoEye, Getmapping, Aerogrid, IG
      }
    ]
  },
```

2.10.3 Offline Maps

TBD

2.11 Geo queries

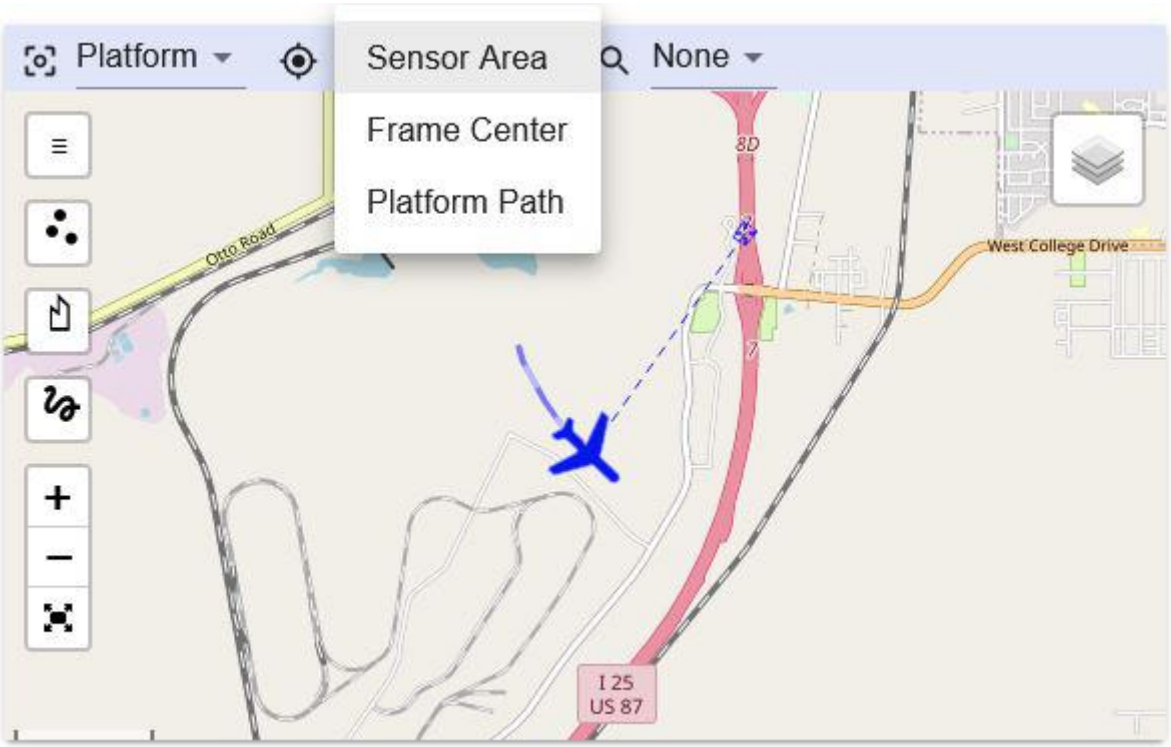
Stanag On Demand server allows you to query a mission database for different types of geographical information. You can efficiently process a vast amounts of ISR data using various query criteria.
For example :

- Find all missions that were recorded during the selected time period at the specific location.
- Find all flights over specific area.
- Find a video taken at the specific location.
- Find out when the UAV / plane was in the area of interest.
- Etc.

2.11.1 Area Of Interest (AOI)

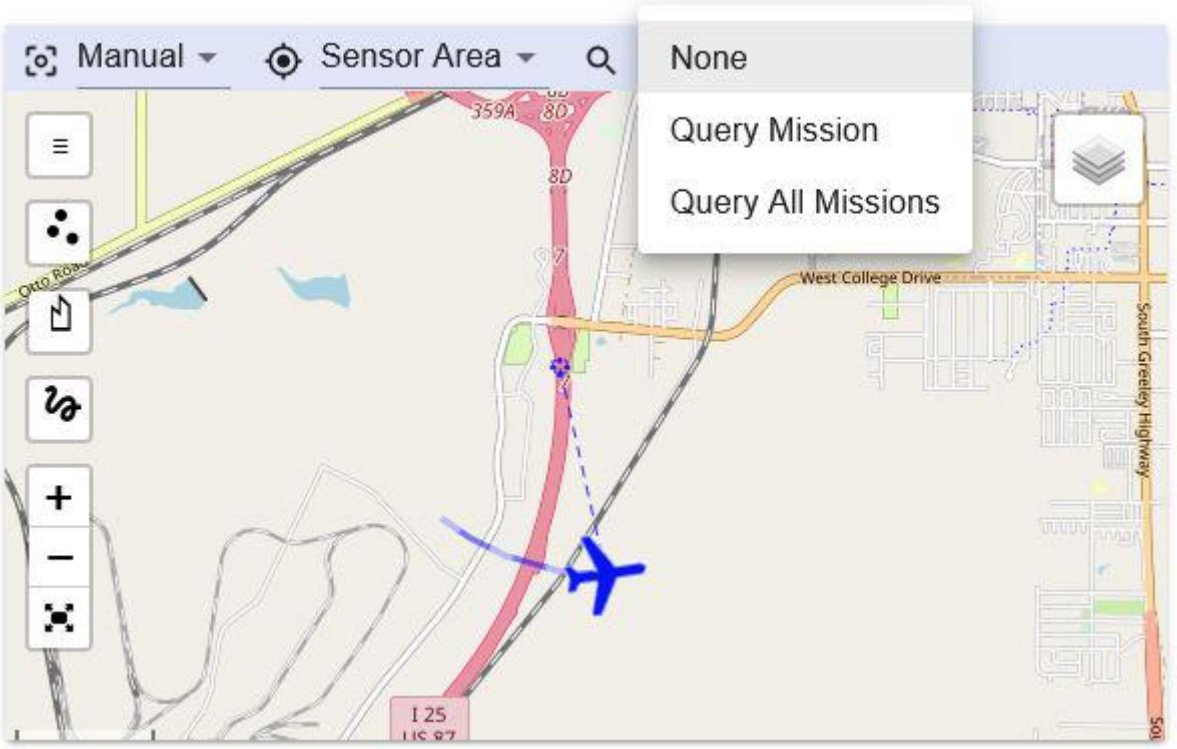
Before we perform any query we have to select the desired area of interest:

- **Filmed area** - query filmed area information (based on corner points)
- **Frame center trail** - query center of FOV trail
- **Platform path** - query platform's swath path



Of course you can only get results when the relevant metadata is present in a mission database. For example, if there are no corner points (offsets or full) it is impossible (without calculating them from FOV and other data) to get the filmed area information.

2.11.2 Query types

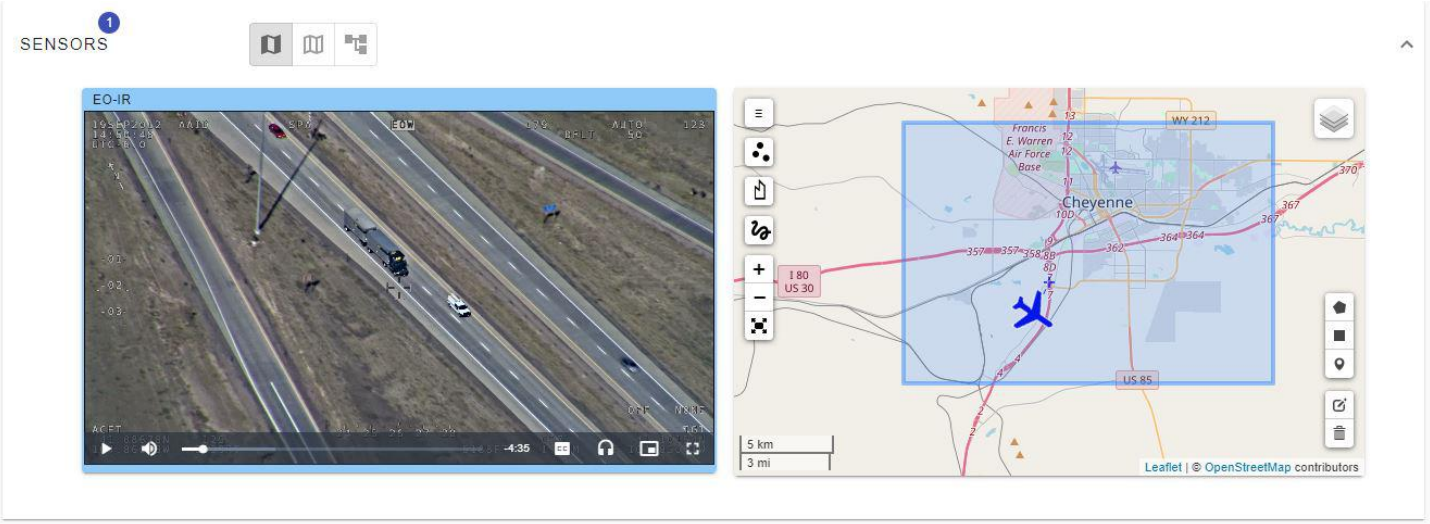


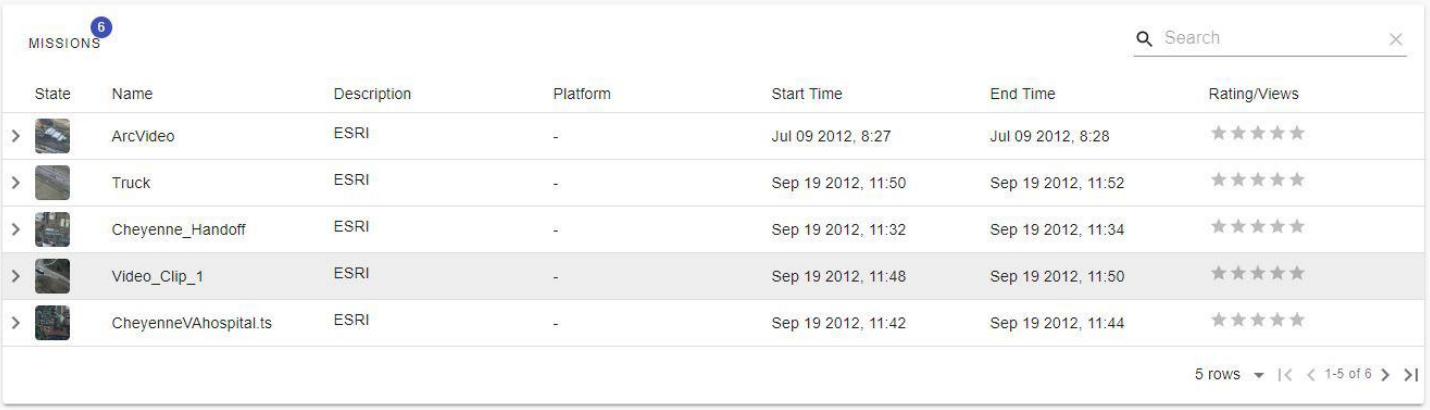
Filter missions

This query type can be used for finding missions that have video content geographically related to the specific area of interest:
Perform query by drawing one of the following shapes on a map:

- **Polygon** (intersect)
- **Rectangle** (intersect)
- **Point** (include)

To find missions that contain a video taken at the particular region of interest, select **Filmed area** query type in the map menu.
Use Drawing tools on the map to select the desired area. You can draw a rectangle, polygon or just set a marker point.
Your web client will now query the server and return a list of missions that covered the selected area. Mission panel should now only contain the relevant missions.
You can add an additional criteria to your search, for example, restrict the search to the specific time range.



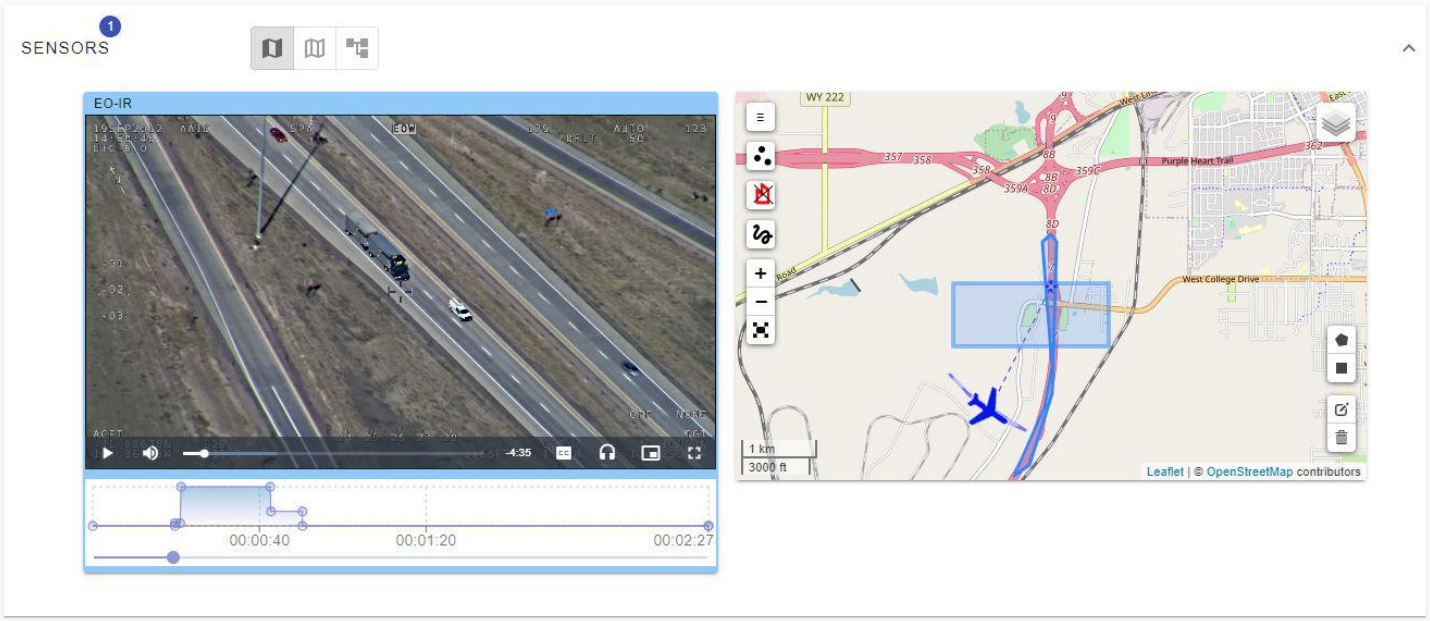


Query one particular mission

This query type can be used for a relevant part of video using geo coordinates.

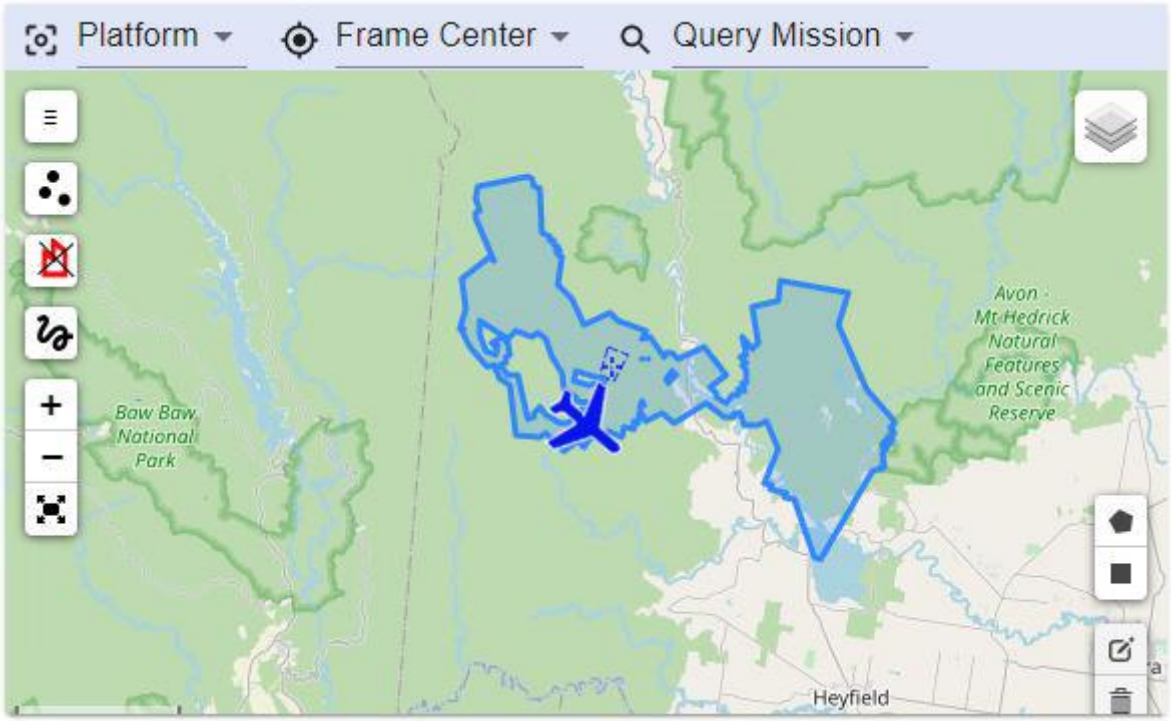
- **Polygon** (intersect)
- **Rectangle** (intersect)

It is effectively a **video coverage** query - extremely useful tool is a map query that graphically shows where exactly in the video clip you can find the places selected on the map. As shown below, the place selected on a map had been only filmed at the beginning of the mission, so if this is the only location we're interested in, there is no need to go over entire video.



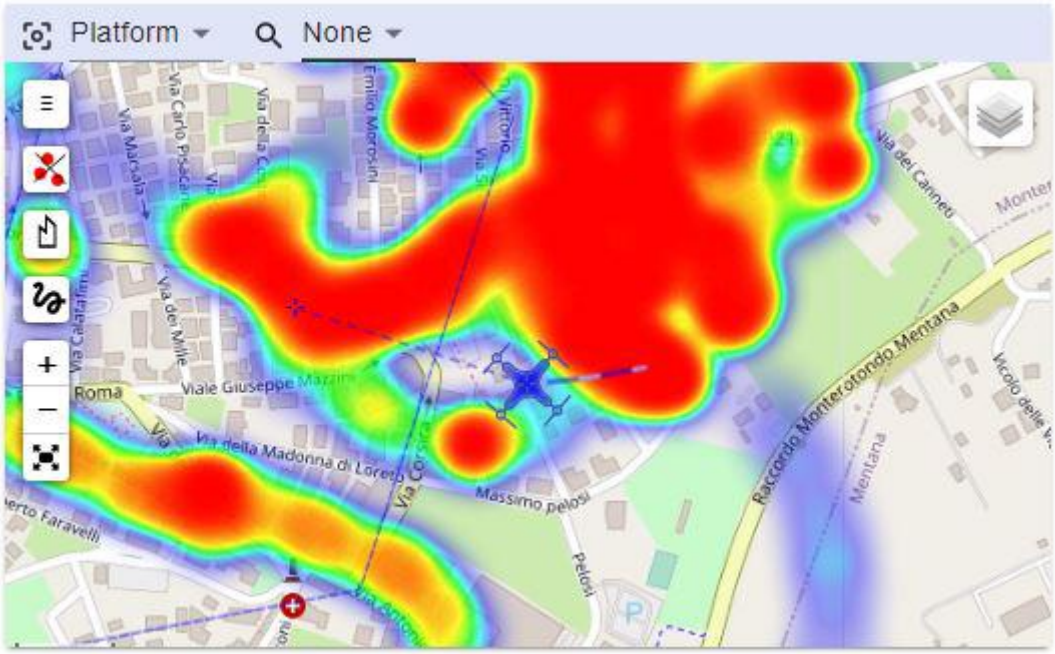
2.11.3 Show filmed area

The area polygon is calculated from corner points (offsets or full).



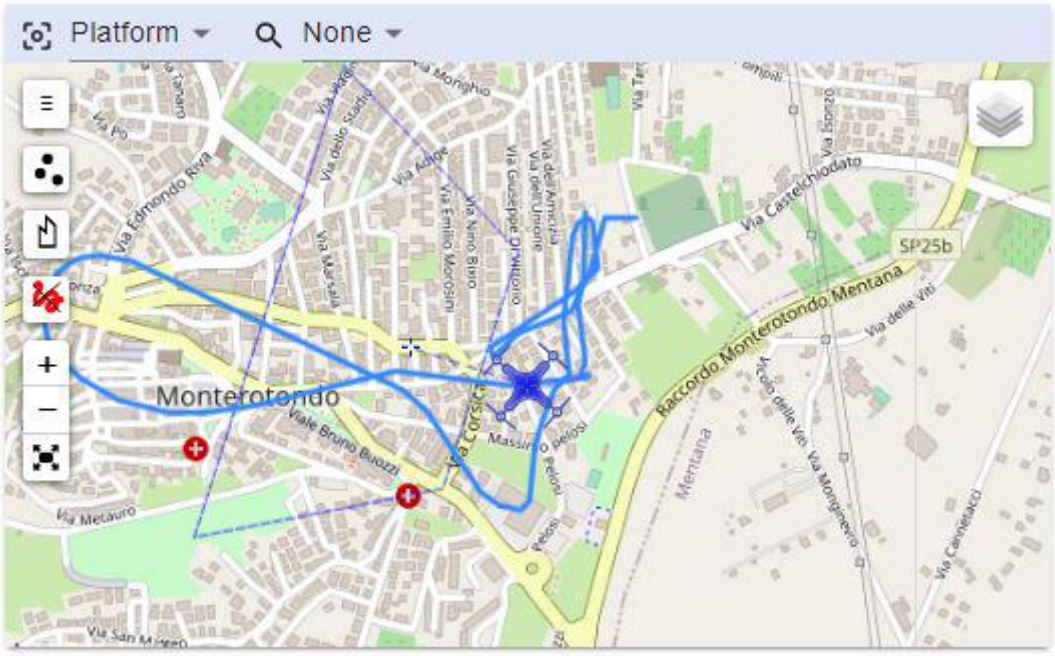
2.11.4 Show area of interest (heat map)

The heatmap emphasizes the area the operator focused the camerat most of the time.



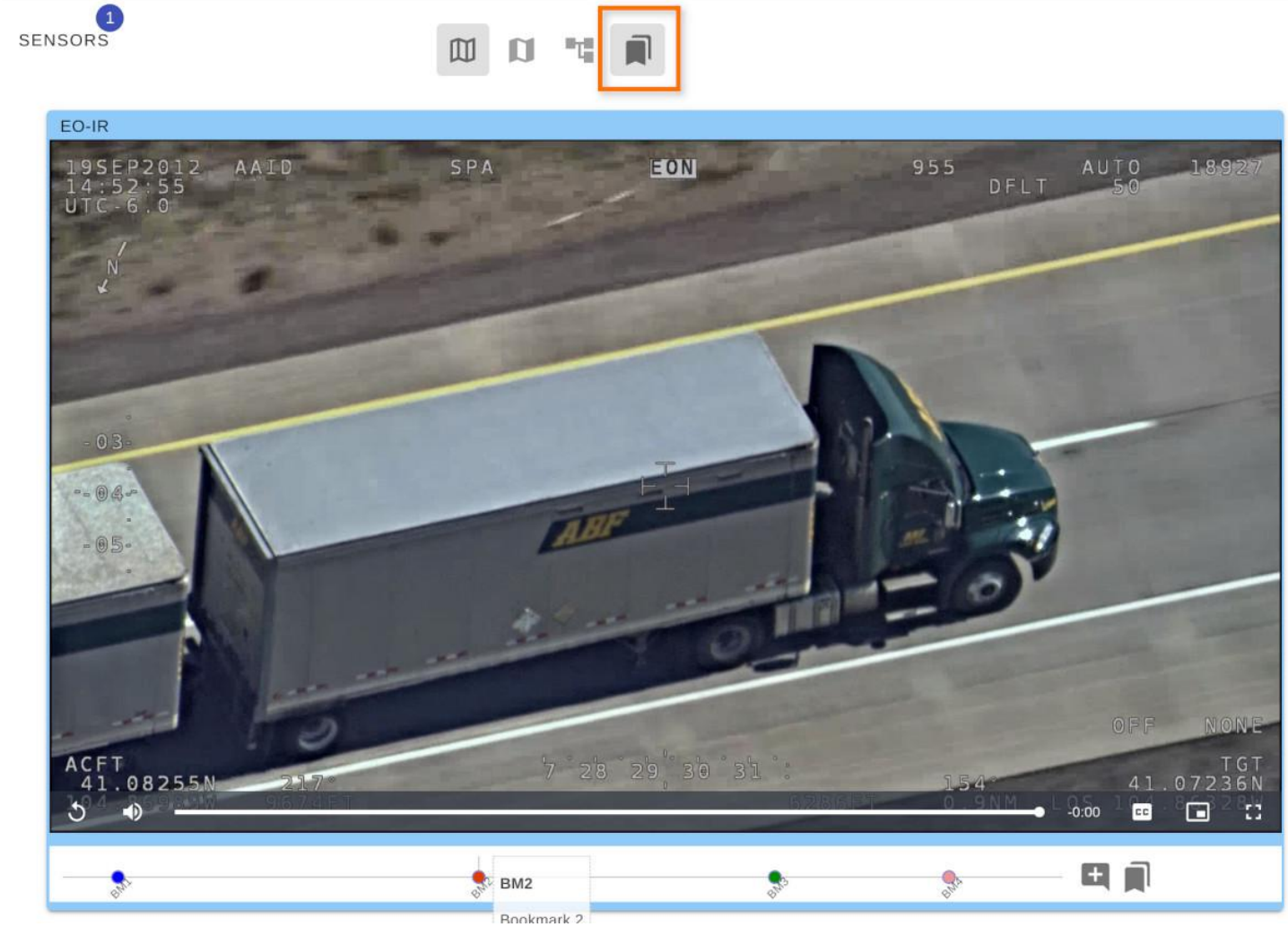
2.11.5 Show path

This query shoes a swath path built from the platform's coordinates.



2.12 Bookmarks

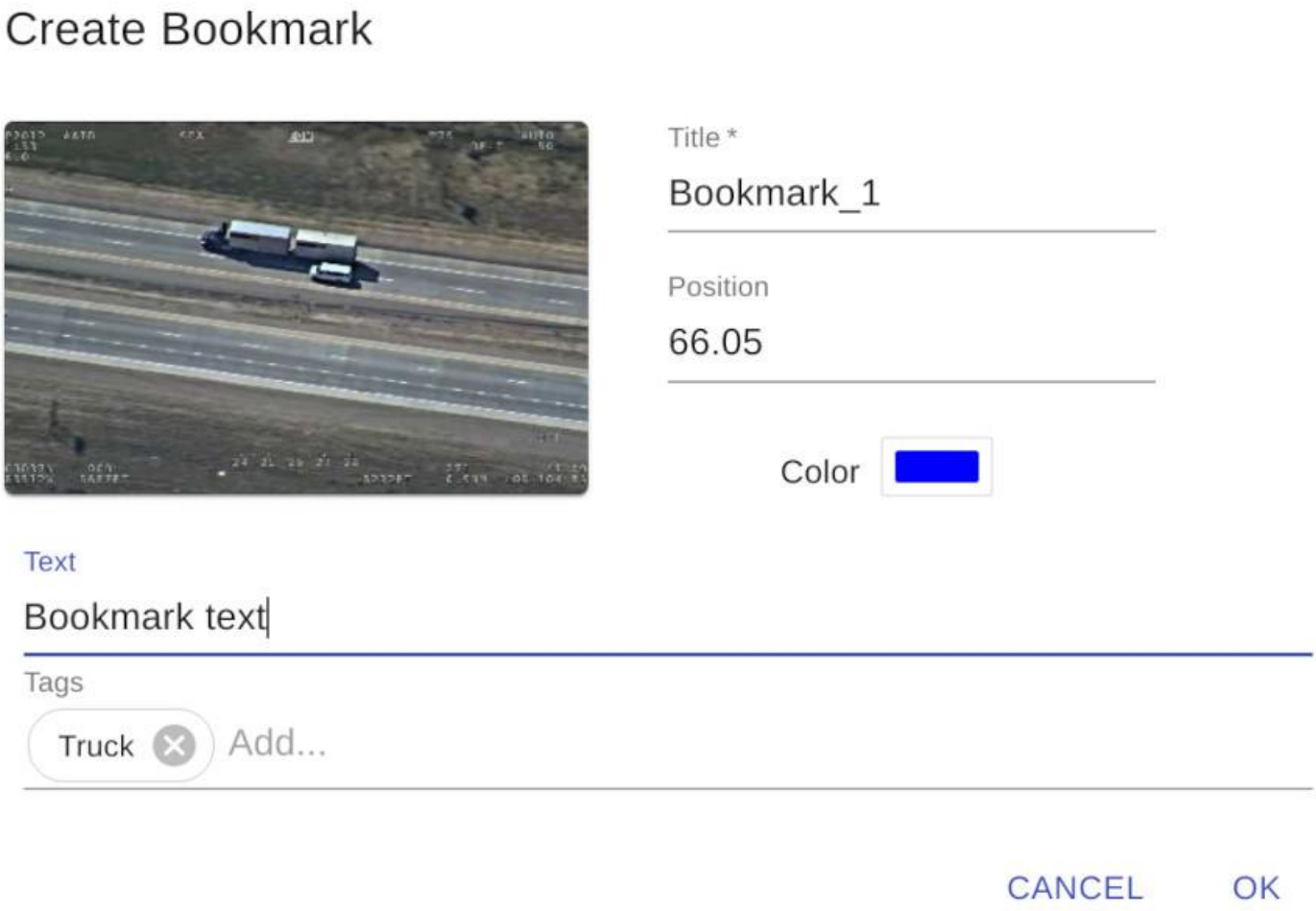
Bookmarking is the ability to save and later return to a specific point in the video. Bookmarks save the viewer's position along with some additional descriptive information.



Bookmarks belong to **sensor**, so a mission may have independent bookmarks for any of it's videos. Clicking on the existing bookmark will seek to the saved position.
In order to have access to bookmarking functionality, make sure you've enabled **Bookmarks** in the toolbar, as shown above.

2.12.1 Creating bookmarks

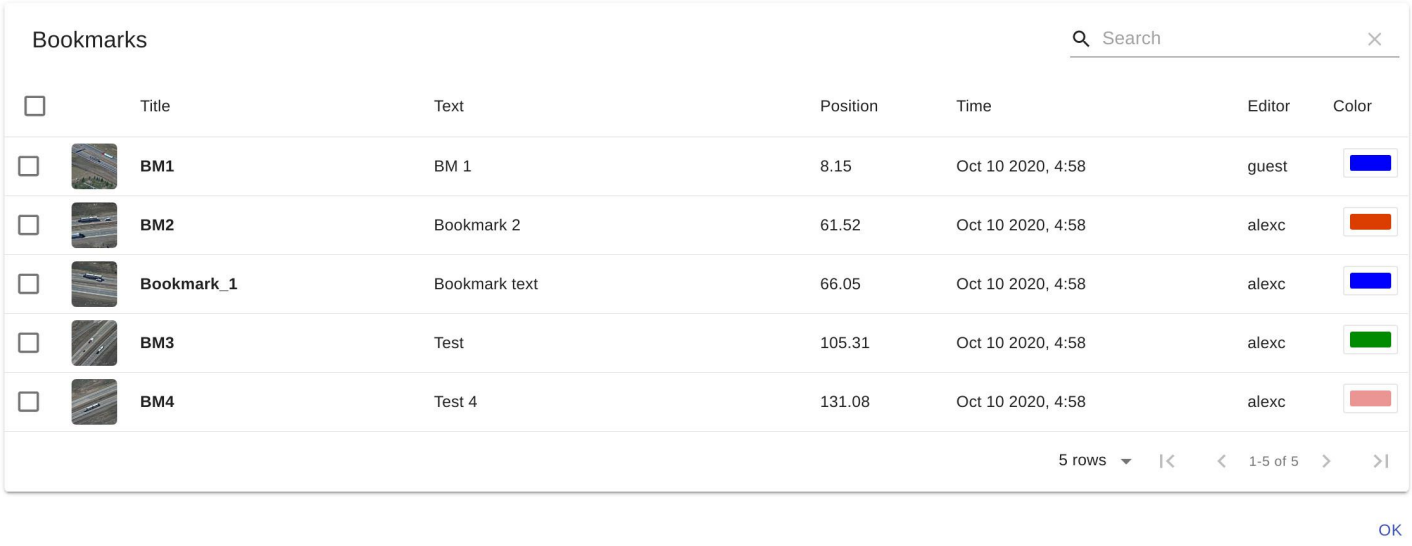
Press **Add bookmark** and the bookmark creation dialog will open.



Fill the required info and press Ok. A new bookmark will be created and added to the timeline.

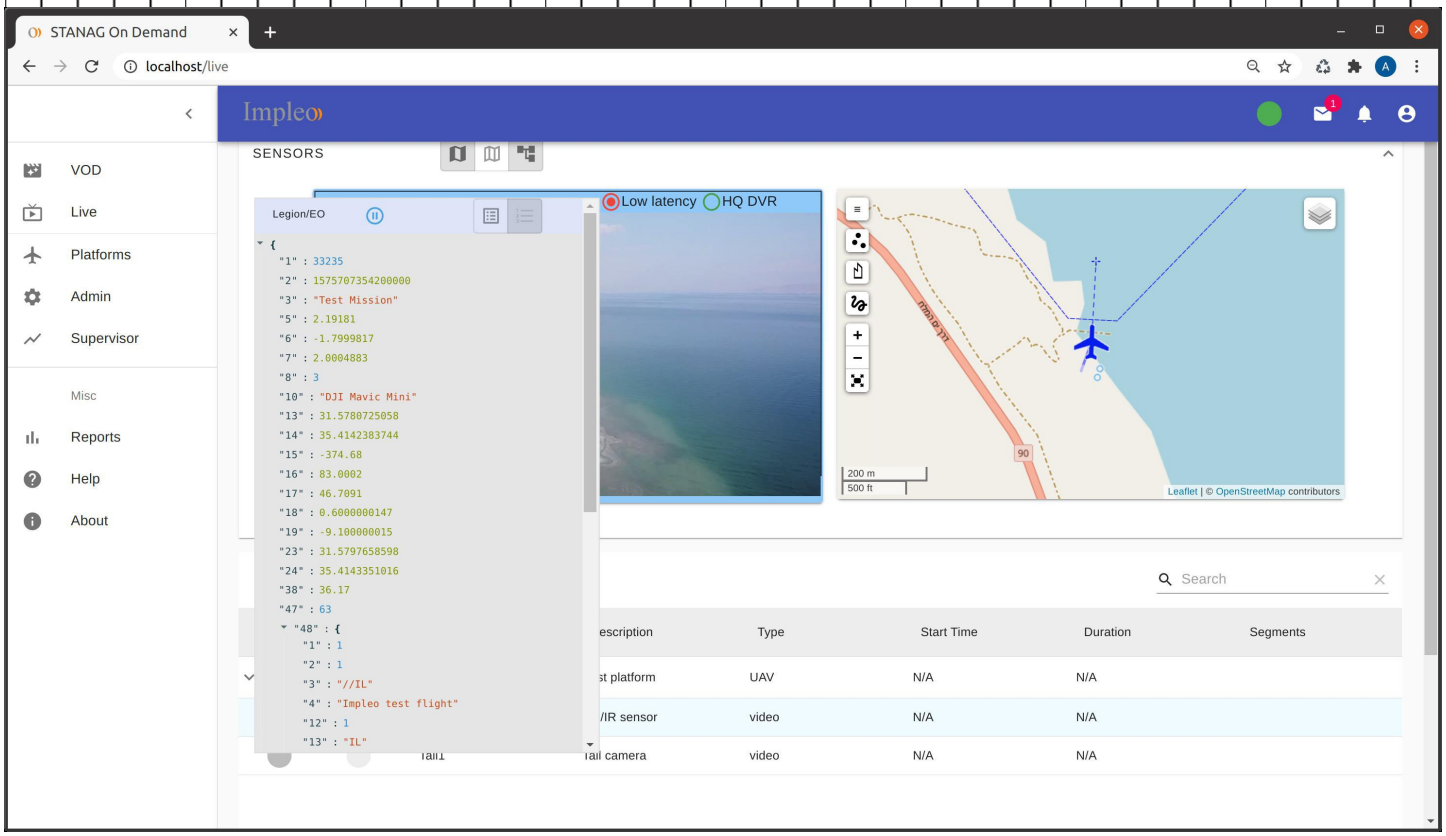
2.12.2 Deleting bookmarks

You can delete bookmarks using **Edit bookmarks** dialog.



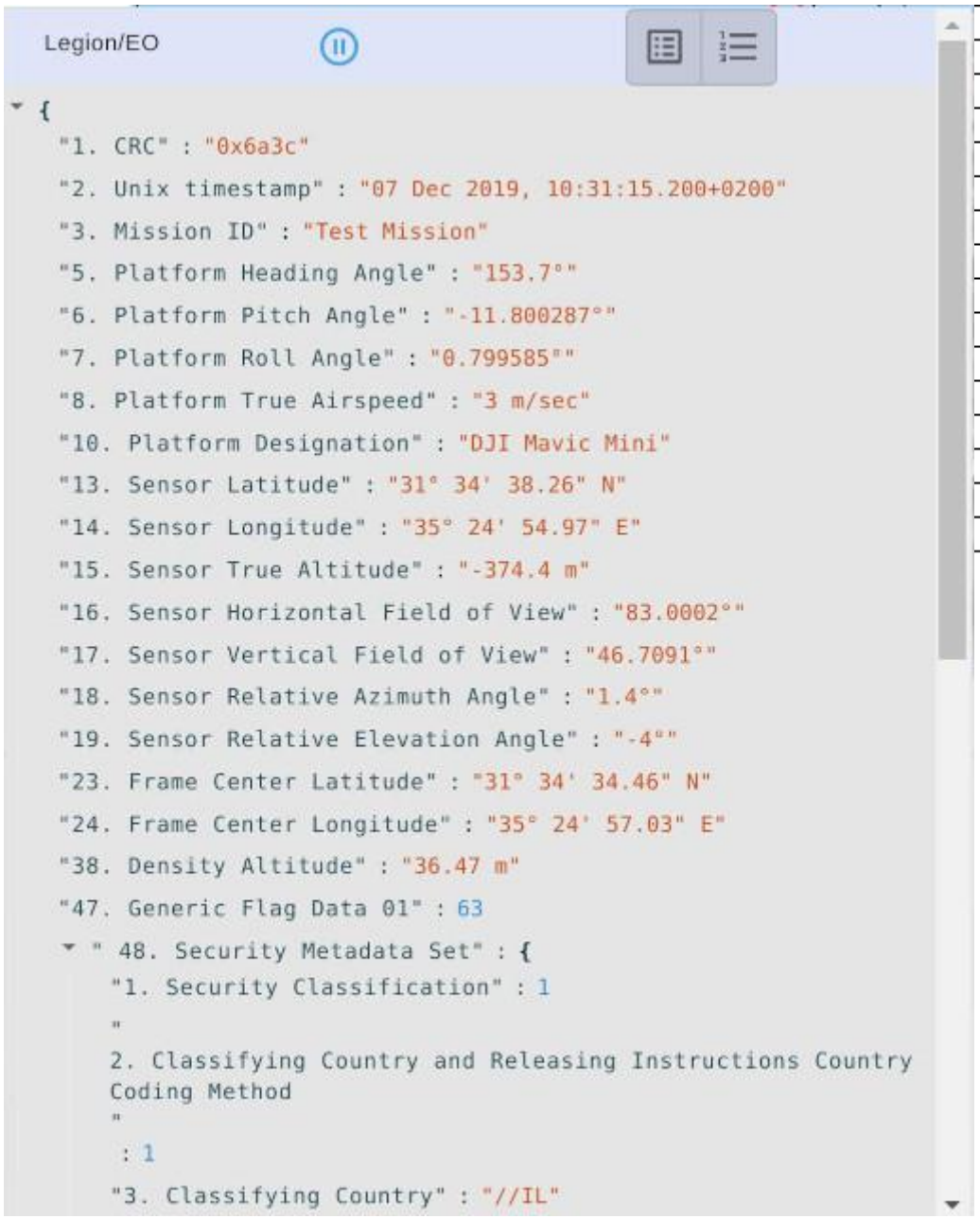
2.13 Klv View

You can inspect Klv packets in JSON VIEW mode, by opening the Klv View window.



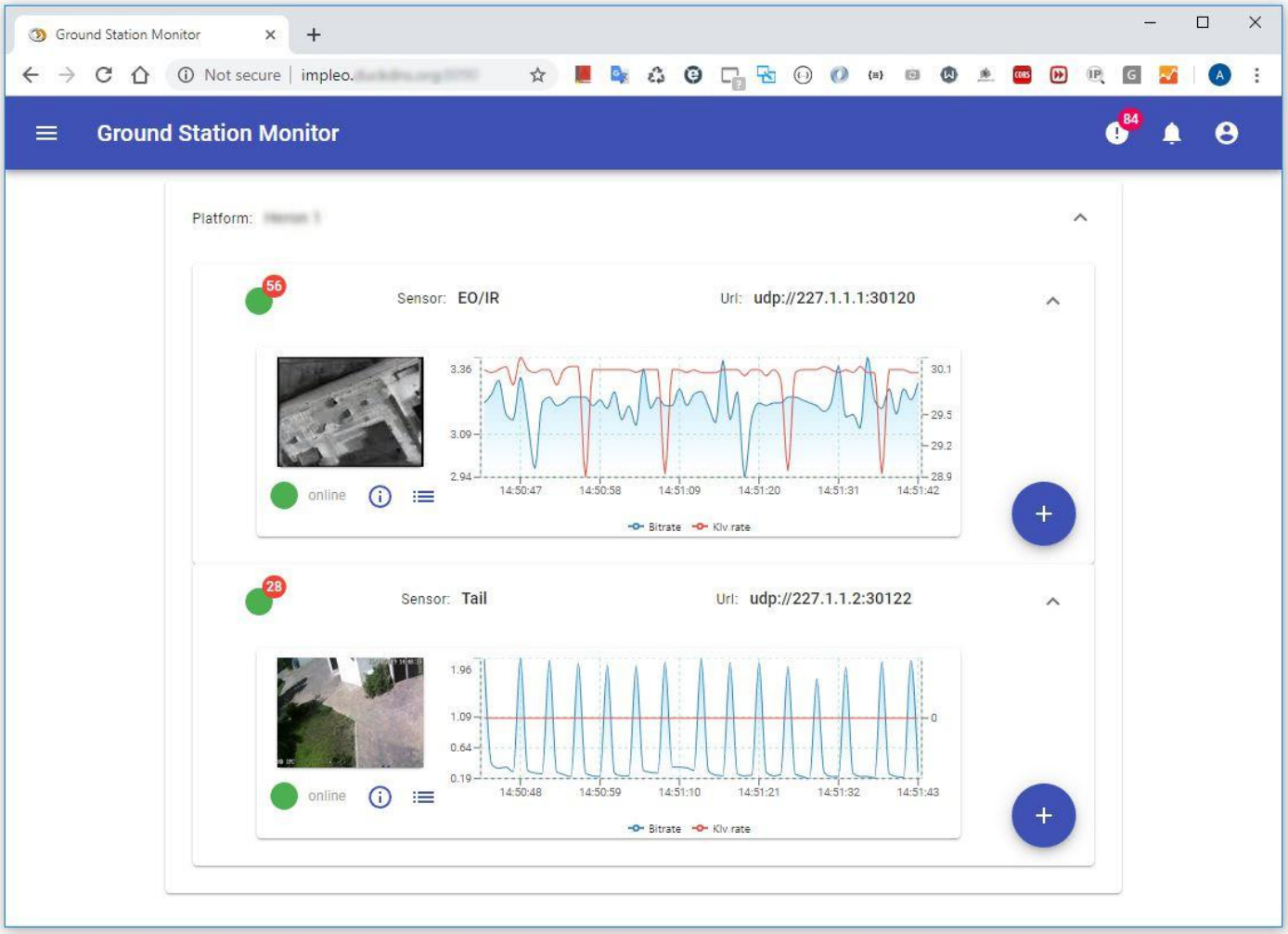
There are two modes available:

- Compact - only tags (as they appear in the standard) are presented. The values are in a native form.
- Detailed - tags and names are presented. Values are in a "more" human readable format.



2.14 Ground station STANAG 4609 stream monitor

StSupervisor is a Ground Station STANAG 4609 stream monitoring service



2.14.1 Main features:

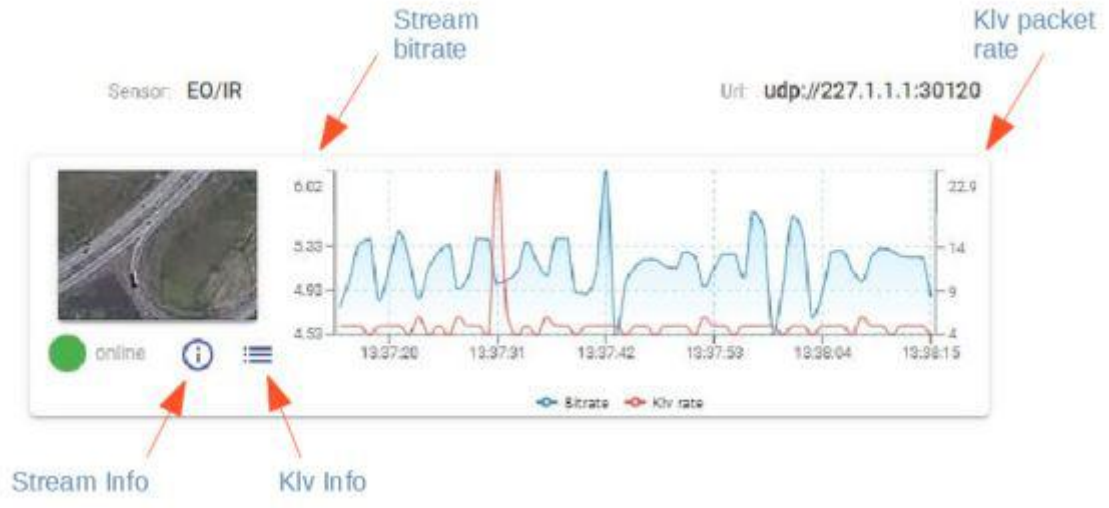
- Monitors a pre-configured list of multicast video streams (yaml configuration file)
- Multiple UAV platforms (with multiple sensors each)
- Timeout, Bitrate and Klv rate reporting
- Low latency video preview
- MISB 0601, 0102, 0903 live decoding
- Stream parameters detection
- User defined **event triggers** based on speed, altitude, location etc
- MQTT messages for easy integration (InfluxDB, Grafana, etc)
- Cross platform Windows/Linux. Easy setup - stand alone app / Docker
- Desktop and mobile UI

2.14.2 Configuration

StSupervisor uses 2 configuration files located in the **data** directory:
- **supervisor.yml** - application configuration file. More info on Supervisor Configuration
- **platforms.yml** - platforms, sensors etc configuration file. More info on Platform Configuration

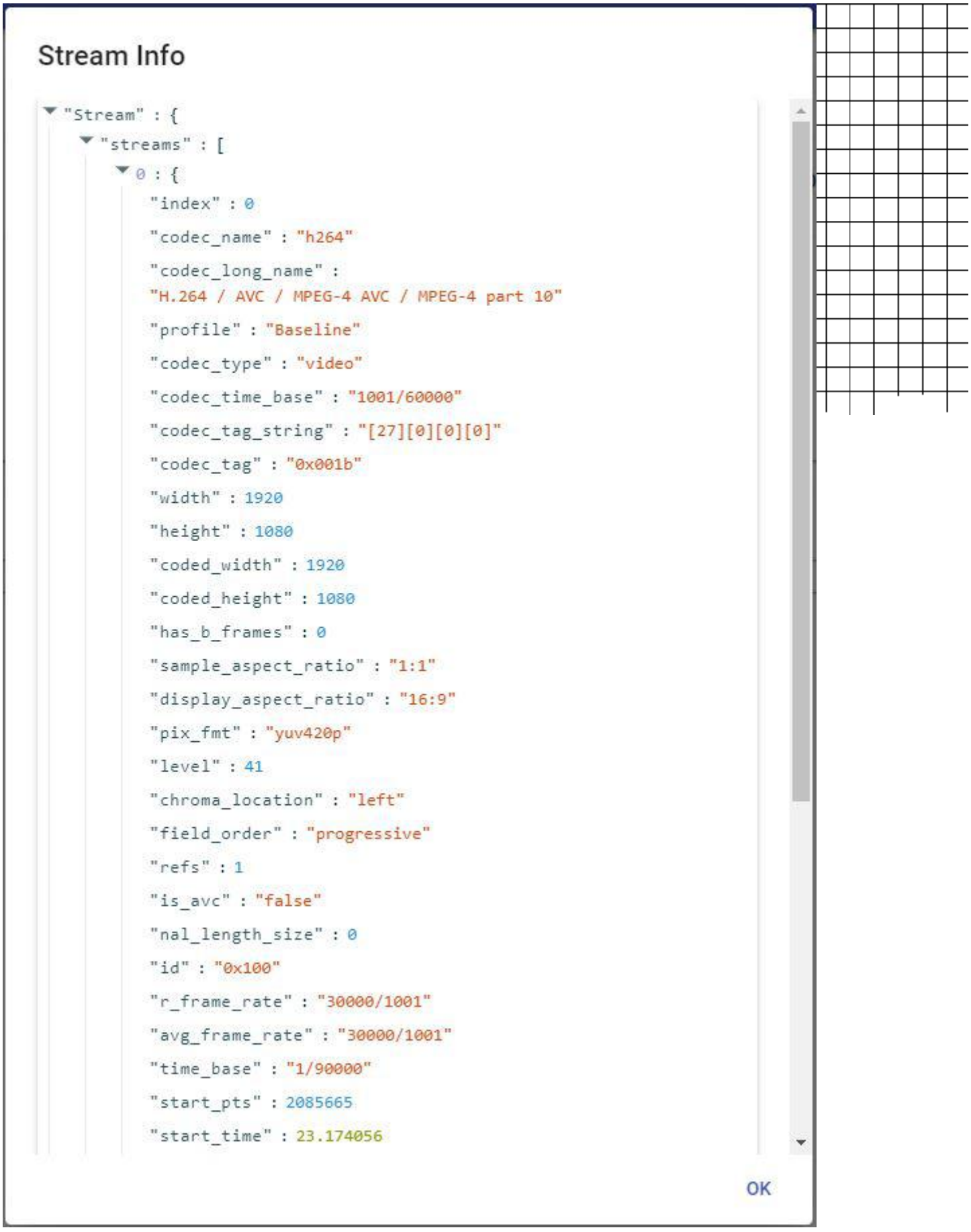
2.14.3 Sensor presentation

StSupervisor presents stream info per sensor (video/data channel). One platform can carry multiple sensors.



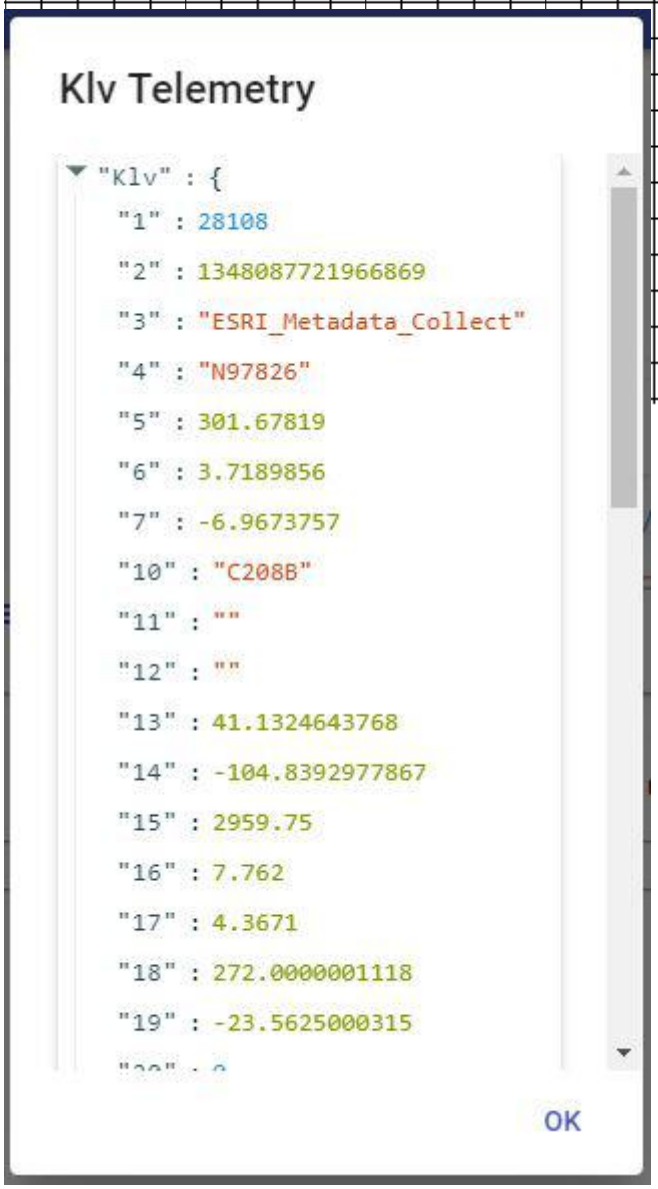
2.14.4 Stream Info

StSupervisor performs stream parameter detection (on a first stream appearance). Press **StreamInfo** button in order to access this information.



2.14.5 MISB KLV Info

If STANAG MISB 0601.X metadata is present in the stream, **StSupervisor** will display it in real time (the sampling frequency can be configured). Press **Klv telemetry** button to access this information.



The following standards are supported: -MISB601.X -MISB102.X -MISB903.4

2.14.6 Mqtt reporting

Stream Monitor sends events to Mqtt broker in the following format: applicationName/platformId/sensorId/event, where **event** has one of the following values:

event	Description
state	Current state
detection	Stream detection results (JSON)
bitrate	Current stream bitrate
custom events	User defined events
config	General configuration information - httpServer, wsVideoPreview
demoExpired	Demo Expired

For example,

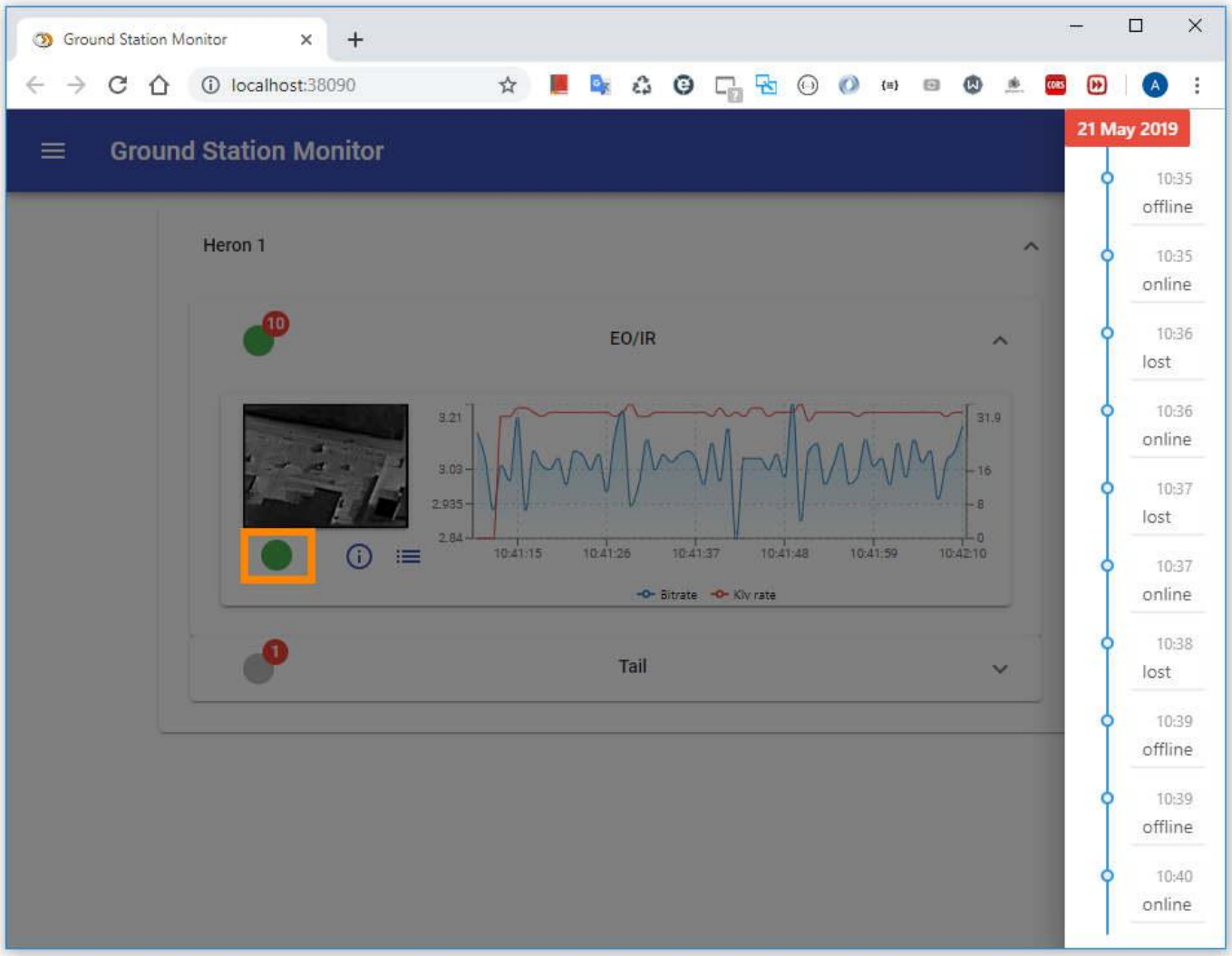
```
StreamMonitor/Heron/EO/state
```

2.14.7 State events

Stream Monitor reports the following stream state events:

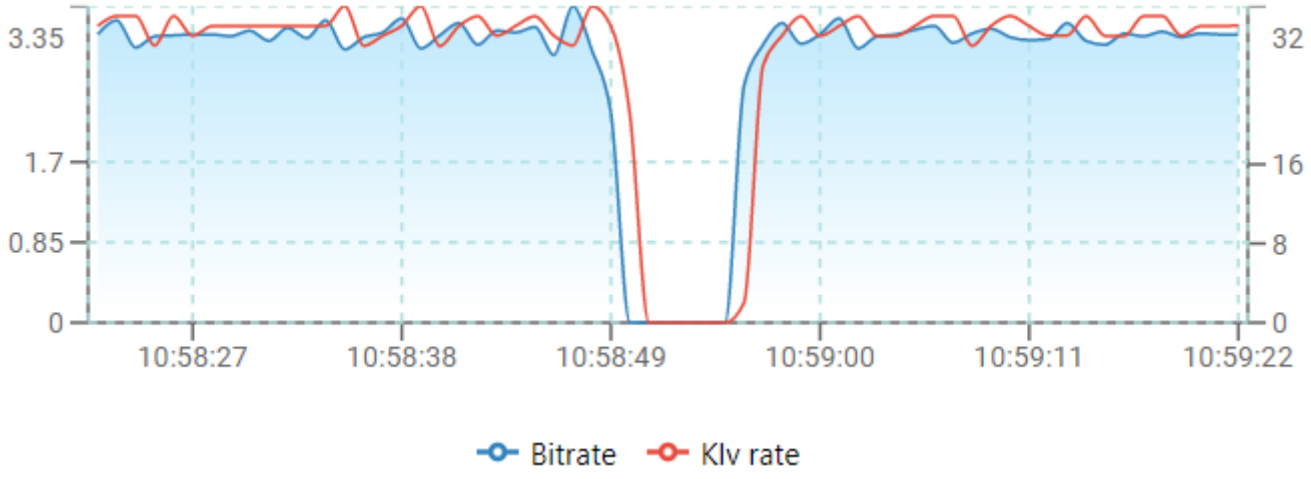
- Unknown. This state will be reported when Stream Monitor goes offline (or not started yet) and therefore there is no info on the stream available.
- Offline. Stream is not present.
- Online. Stream is running.
- Lost. Stream lost (not detected present for the period of time specified by --timeout). After that, the stream is considered as **Offline**.

You can view **Event Timeline** by pressing the **State** icon.

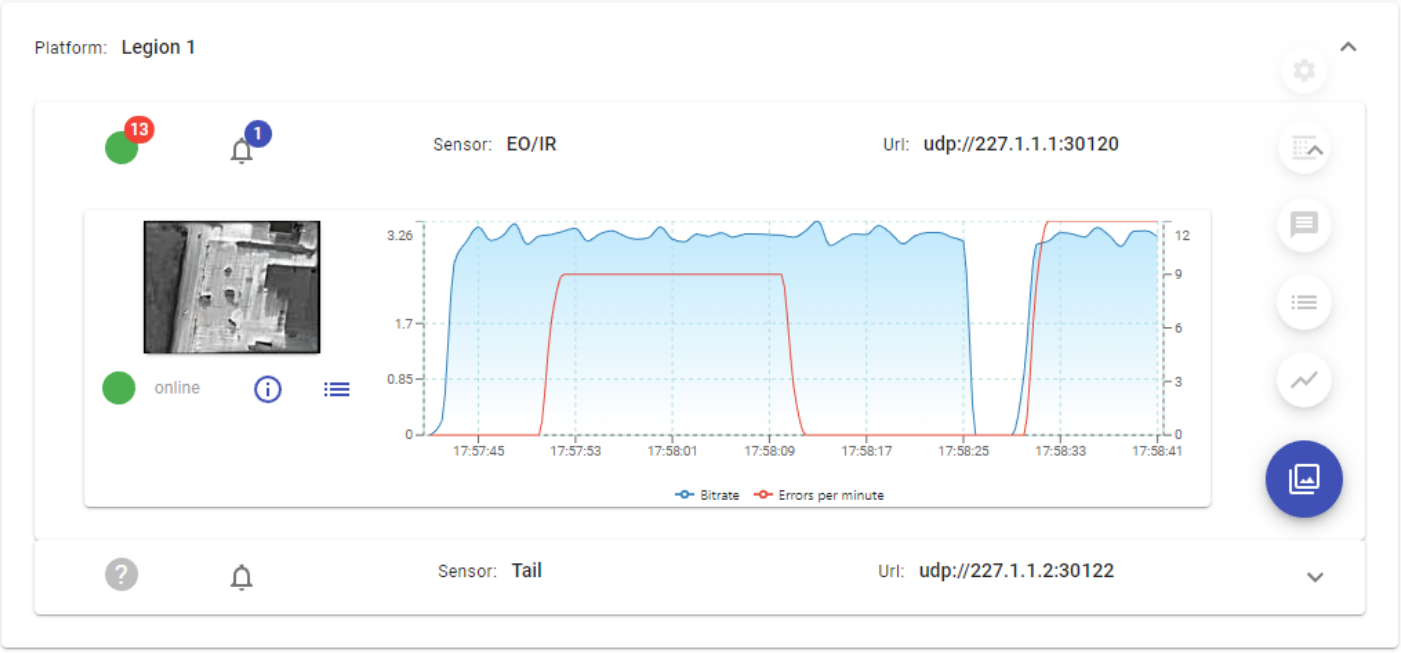


2.14.8 Bitrate and KlvPacket rate reporting

Stream Monitor periodically sends stream bitrate and **STANAG** Klv packet rate notifications that are presented as a real time chart. You can configure an update rate, if needed.



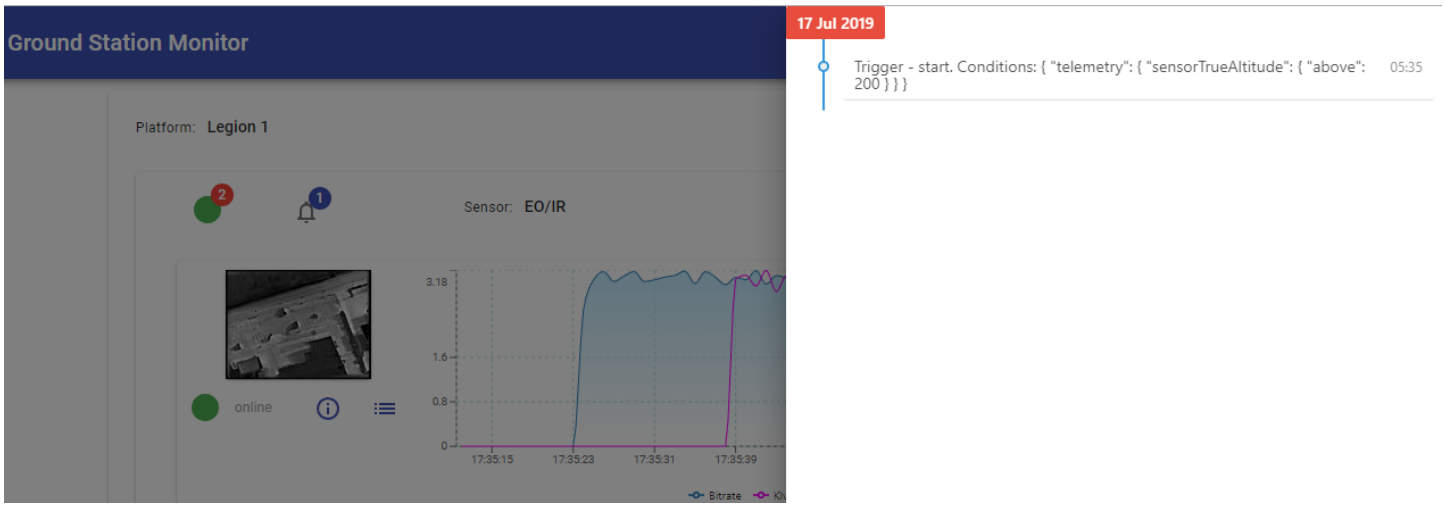
Alternatively, you can select another composed chart, like **Bitrate** and **Stream Error Rate** reporting.



2.14.9 Triggers

Triggers are used to report **events** / issue **commands** when certain **conditions** are met. **Stream Monitor** will send Mqtt message with "trigger" topic and conditions payload when there is a match. Upon reporting, the trigger is removed from an internal queue.

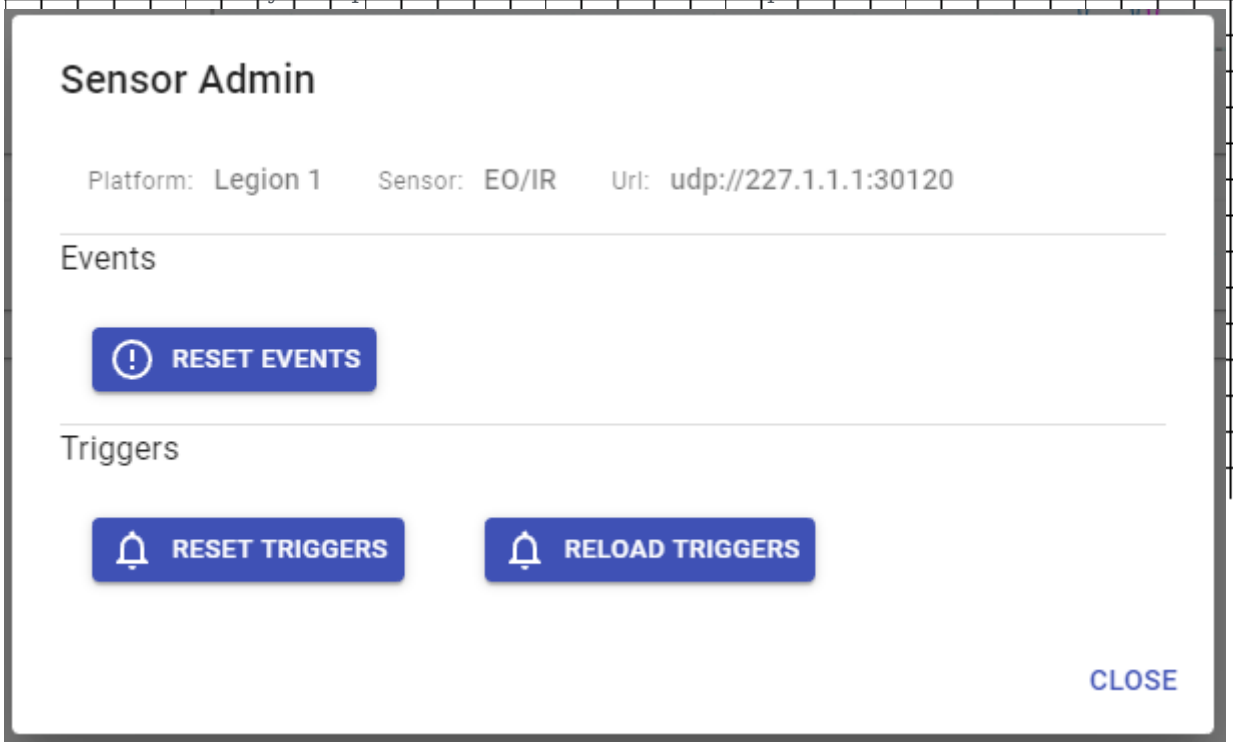
The logic: - Event is fired when the conditions inside the trigger are true (logical AND) - Event is fired only once (if **armed** state is **true**). After that, the **armed** state is set to **false**. You can reload triggers (**Sensor Admin** page)



More on Triggers

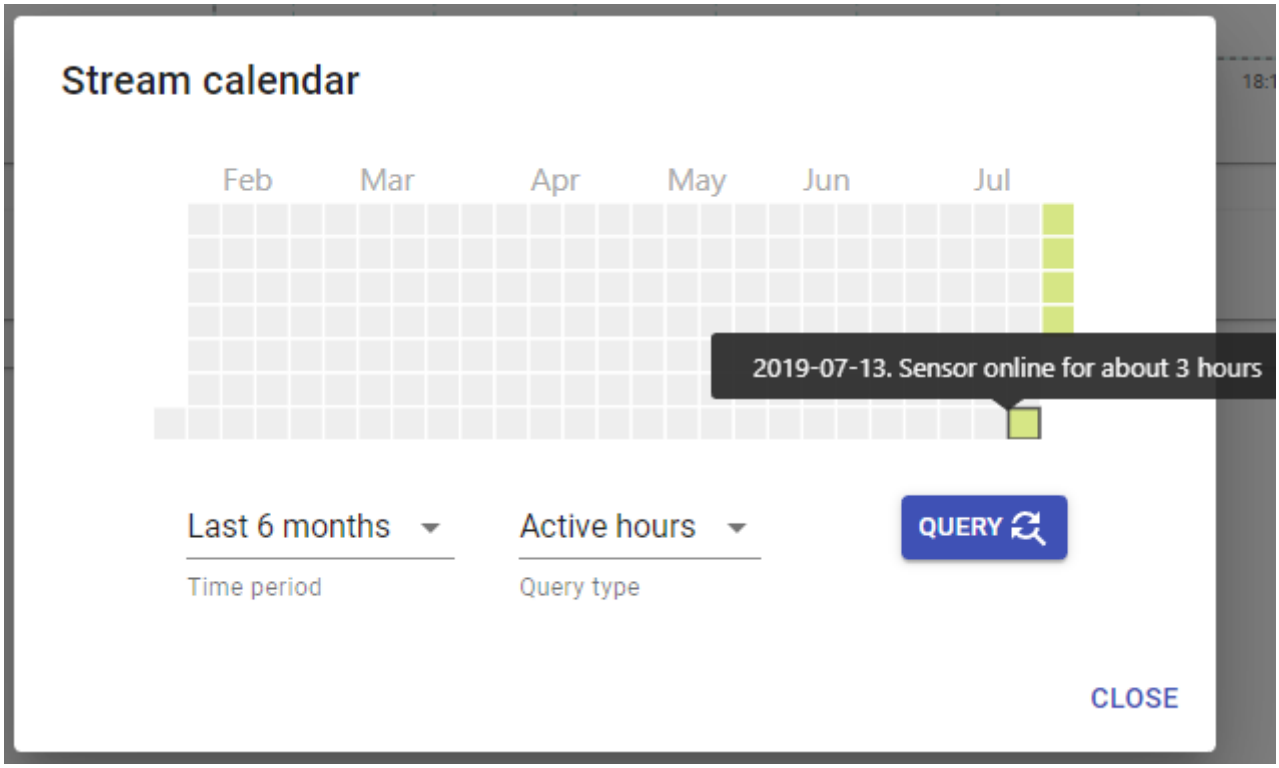
2.14.10 Sensor Admin

Sensor admin allows you to perform administrative tasks on specific sensor.



2.14.11 Stream Calendar

Stream calendar shows the activity history.

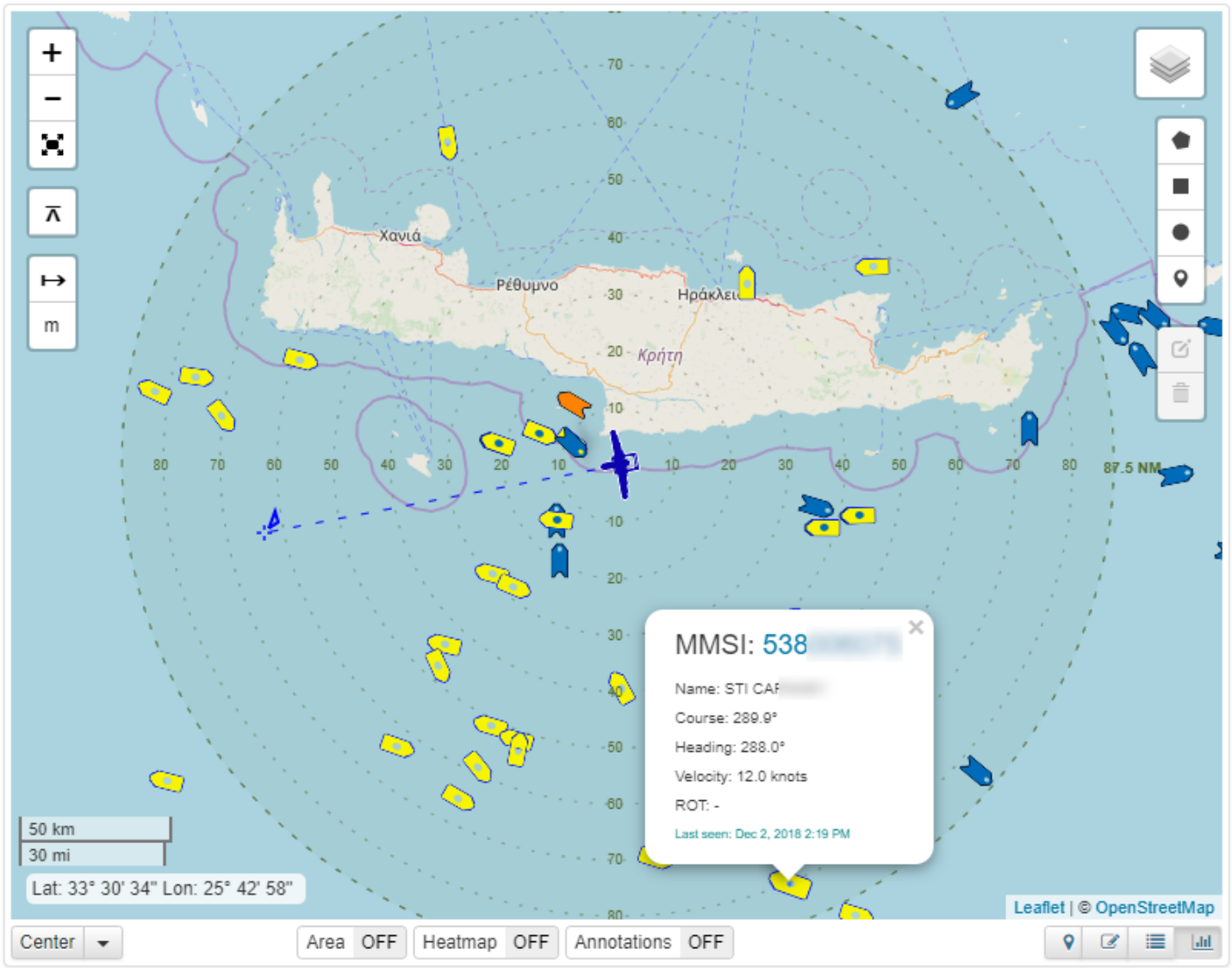


\note The data is not stored locally with the application. You must configure the **InfluxDB** server address in **supervisor.yml** file in order to be able to collect this information.

2.15 Maritime Targets

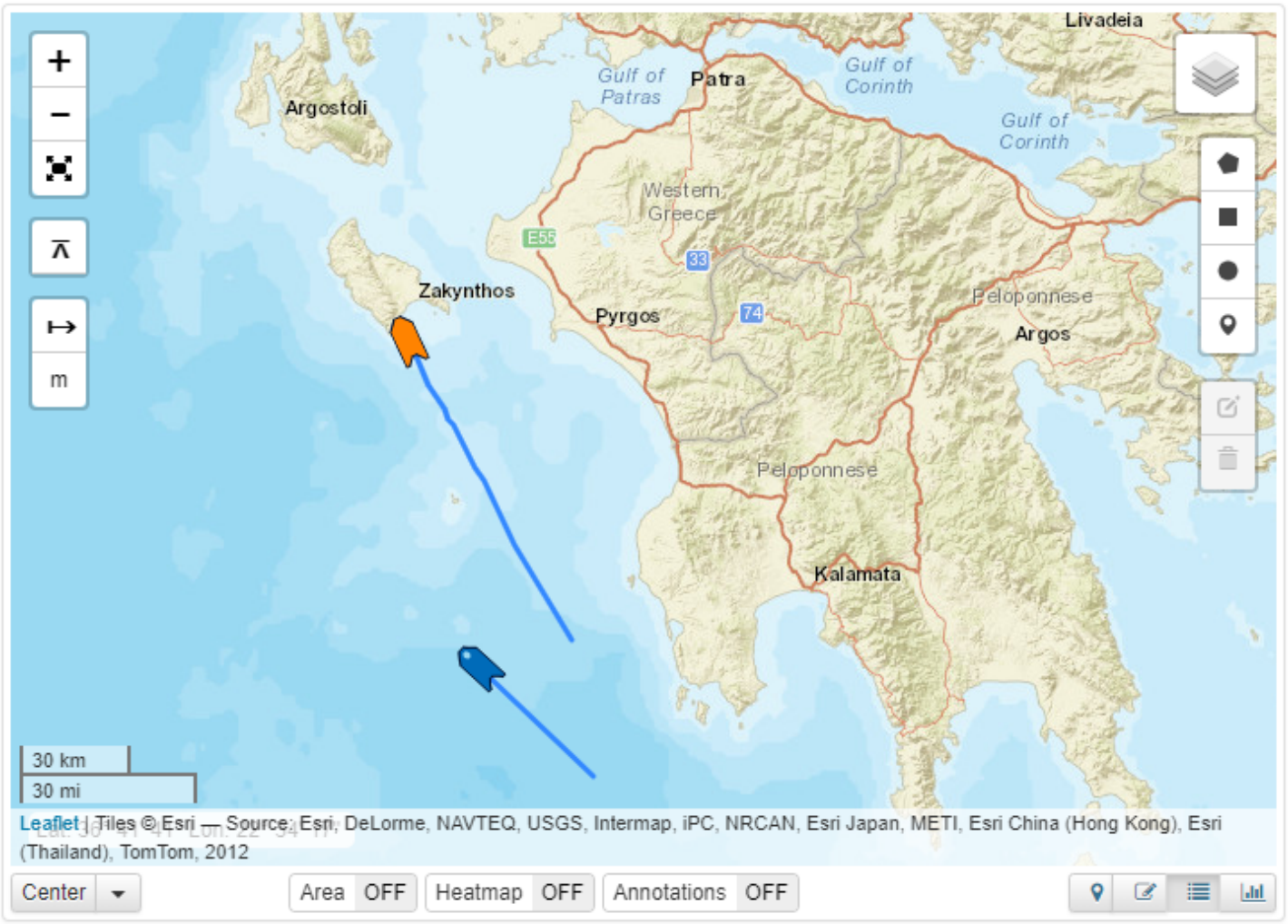
STANAG On Demand Server supports the following maritime targets acquisition and processing:
- AIS
- Radar

Note, **Maritime Targets** is an optional feature that requires radar integration, so it only works out of the box with specific platforms.



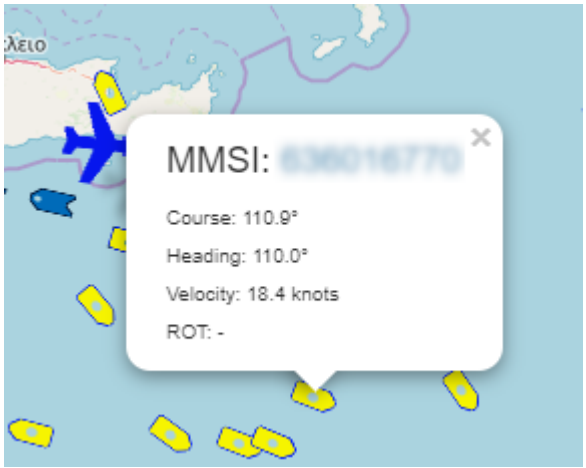
2.15.1 Target Path

When you select the target, its historical path (stored in the internal DB) is automatically presented on the map.



2.15.2 Marine Traffic site

The **MMSI** number of the **AIS** target popup is clickable, so you can get additional information on the vessel by visiting https://www.marinetraffic.com website, which will be opened in the new browser tab once you click the **MMSI** number.



MarineTraffic Live Map Explore Data Community Solutions

GHENA

Inland, General Cargo maritime

Create notifications for this Vessel | Fleet controls: Add to Fleet | Suggest updated values

IMO: **8303989** ENI: **1500**
MMSI: **352245000**
Call Sign: **3FAV5**
Flag: **Panama [PA]**
AIS Vessel Type: **Cargo**

Gross Tonnage: **40035**
Deadweight: **17090 t**
Length Overall x Breadth Extreme: **190m × 32.2m**
Year Built: **1984**
Status: **Active**

Upload a photo | Ship Photos: 33

Voyage Info

For full access Try Voyage Data

UY MVD
ATD : 2018-11-01 03:10 LT (UTC -3)

TR DRC
ATA : 2018-11-23 00:48 LT (UTC +3)

Past Track | Route Forecast

Distance Travelled	•••••
Draught	9.7m
Load Condition	•••••
Speed recorded (Max / Average)	14.6 / 12.7 knots

Itineraries History | Latest Positions

Reported ETA Received: 2018-11-24 19:57 LT (UTC +3)

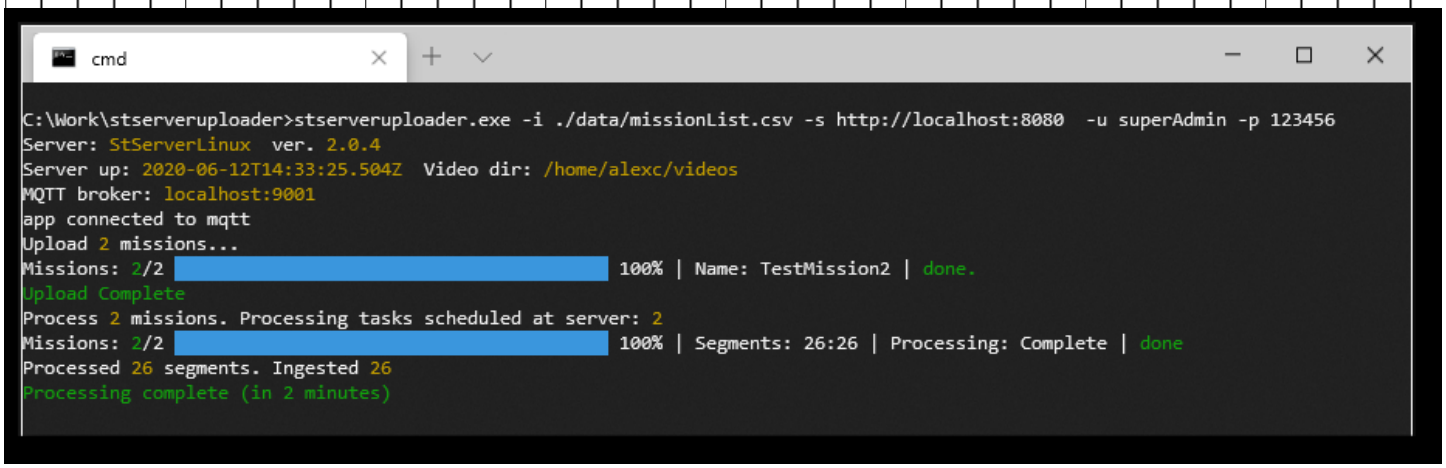
Vessel Particulars

Last update: 2018-11-23 03:25:01

3. Utilities

3.1 Mission uploader

In addition to manual mission creation, it is possible to automate the process and upload large number of content file (available locally or uploaded from remote).



3.1.1 OS

- Windows (64 bit)
- Linux (64 bit)

3.1.2 StServerUploader usage

Windows

```
stserveruploader.exe -i ./data/missionList.csv -s http://yourServerUrl  -u userName -p password
```

Linux

```
./stserveruploader.run -i ./data/missionList.csv -s http://yourServerUrl  -u userName -p password
```

Arguments

Flag	Name	Description
-i	--input	Input configuration file path
-s	--server	Server url
-u	--user	User name
-p	--password	Password
-v	--version	Version info
	--printUsage	Print args description (true/false)

Configuration file (.csv or .json) provides a list of missions to be created and the location of the content to upload.

Upload utility will first check your local machine for specified files and upload them, if found. If not, it will treat the file path as a local path at the server and try to locate the files there. This can be used instead of files upload over http. For example, you can upload the files using ftp and just reference them in the .csv or .json.

Note, if server runs in Docker, you must map your upload directory, so server can find it while running in the container.

3.1.3 Configuring upload directories with Docker

Let's assume that you've configured the ftp server on a host computer to upload your video content to ~/Movies folder.
Set the **HOST_UPLOAD** environmental variable (in the **.env** file) to that location and **STANAG On Demand Server** will map it as a volume.

```
HOST_UPLOAD=~/Movies/
```

By default, the server will set /app/upload/ inside container as a folder for upload.
So, now you can simply reference the video to this directory (including the subdirectories) in the .csv or .json

```
Mission._id,Mission.name,Mission.description,Mission.tags,Mission.usergroups,Mission.platform.name,Mission.platform.type,M
,TestMission,Test Mission,"flight,test","Demo, Group1",Heron,Plane,UNCLASSIFIED,EO/IR,Main EO/IR,video,camera,200, ./uploa
```

3.1.4 Data format

StServerUploader supports 2 text formats:

- json
- csv

JSON configuration file format

Json configuration is an array of missions, where every mission can have an array of sensors.
For example:

```
[
   {
      "Mission":{
         "_id":"",
         "name":"TestMission",
         "description":"Test Mission",
         "tags":"flight,test",
         "usergroups":"Demo, Group1",
         "platform":{
            "name":"Heron",
            "type":"Plane"
         },
         "securityClassification":"UNCLASSIFIED",
         "sensors":[
            {
               "name":"EO-IR",
               "description":"Main EO/IR",
               "type":"video",
               "tags":"camera",
               "sampling":"200",
               "files":"D:\\Movie\\ArcGIS\\Truck.ts"
            },
            {
               "name":"Tail",
               "description":"Tail camera",
               "type":"video",
               "tags":"camera",
               "sampling":"200",
               "files":"D:\\Movie\\Operator.ts"
            }
         ]
      }
   }
]
```

CSV configuration file format

CSV configuration is essentially a flat text representation of missions (like in json format), where every mission can have multiple sensors. In order to express the hierarchical nature of **json**, a "." (1dot) prefix is used. First line represents the fields.
For example:

```
Mission._id,Mission.name,Mission.description,Mission.tags,Mission.usergroups,Mission.platform.name,Mission.platform.type,M
,TestMission,Test Mission,"flight,test","Demo, Group1",Heron,Plane,UNCLASSIFIED,EO/IR,Main EO/IR,video,camera,200,D:/Movie
,TestMission2,Test Mission2,"flight,test","Demo, Group1",Heron,Plane,UNCLASSIFIED,EO/IR,Main EO/IR,video,camera,200,D:\Mov
```

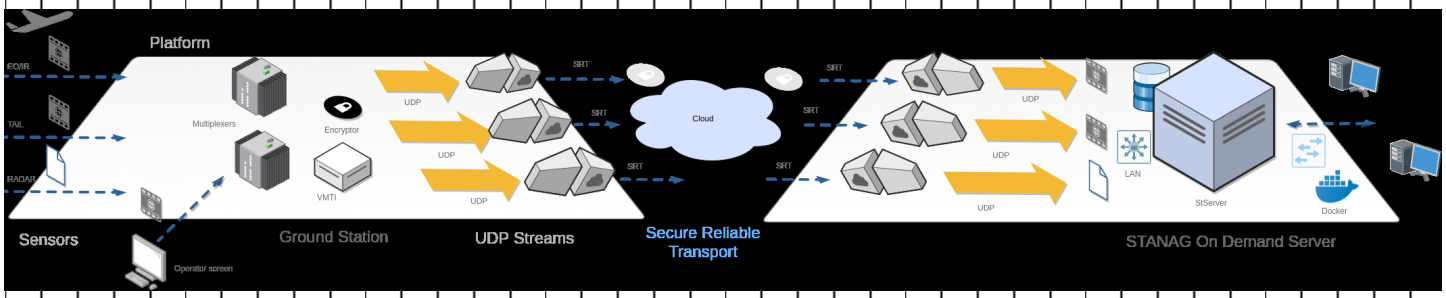
Platform options:

Platform type can be one of the following:
- Plane
- Helicopter

- Quadcopter
- Camera
- ROV
- Diver

default: Plane

3.2 Recaster



TBD

3.3 Scripts

Note, the below is subject to change!

Stanag On Demand Server Scripts is a set of maintenance utilities for the **StanagOnDemand** server.

When no specific instructions are provided, the script obtains configuration information from the **config.json** file located in the sibling directory **./data** The following info is automatically extracted:

- Database connection string
- Video folder

This info can be overridden by the explicit arguments provided to the script.

3.3.1 Modes of operation

- Backup
- Restore
- Export mission(s)
- Import mission(s)

3.3.2 Script arguments

Short Command	Command	Description
-m	--mode	Modes: Backup or Restore DB / Export or Import mission
-u	--uri	DB Uri
-r	--root	Root folder for export / import. There may be one or more missions in that folder.
-c	--collections	Collection array to export. Comma separated. All by default
-d	--drop	Drop every collection from the target database before restoring the collection. Default: true
-v	--video	Video folder location. If specified 'none', video is not backed up / restored
	--missions	When in the Mission export or import mode, provide **missionId**. Multiple (comma or space separated) missions can be provided.
	--exportDbOnly	Export DB only. Skip video. Default - false.
	--version	Script version

3.4 Backup

Backup script will perform the following actions:

- Extract the DB collections and copy them to **db** sub-folder of the supplied root
- Copy **missions** assets found in **videos** to the **video** sub-folder of the supplied root
- Copy **config.json** file found in **data** folder to the supplied root

Usage:

```
node index.js -m backup -r c:/tmp/backup/
```

For simplicity, the default script configuration can be run as a Windows batch file. The Output directory should be supplied with a parameter.

backup.bat

```
backup c:/tmp/backup
```

3.5 Restore

Restore script will perform the following actions:

- Import DB collections into the MongoDB from the supplied **root/db**
- Copy **missions** assets found in the **video** sub-folder of the supplied root to **videos** destination (either specified in **data/config.json** or provided explicitly)
- Copy **config.json** file found in **root** to **data** Server's folder

Note, by default, the document collections that exists in the **DB** will be dropped and the **missions** will be overwritten. Usage:

```
node index.js -m restore -r c:/tmp/backup/
```

For simplicity, the default script configuration can be run as a Windows batch file. The Input (archive) directory should be supplied with a parameter.

restore.bat

```
restore c:/tmp/backup
```

3.6 Export mission

You can export one or more missions at once. Missions are exported as directories, containing assets and relevant db information.

*Export script will perform the following actions:

- Extract Mission's DB collections and copy them to **db** sub-folder of the supplied root
- Copy **missions** assets found in **videos** to the supplied root

Usage:

For single mission (by MissionId)

```
node index.js -m export --missions 5bc6e9de4b75ae3ef0383023 -r c:/tmp/backup/
```

Or using batch file:

```
export 5bc6e9de4b75ae3ef0383023 c:/tmp/backup
```

The resulting archive should look something like this:

```
tmp/
├── /backup/
│   ├── 5bc6e9de4b75ae3ef0383023/
│                               ├── db
│                               ├── mymission.m3u8
│                               ├── mymission0000.ts
│                               ├── mymission0001.ts
│                               ├── mymission0002.ts
│                               ├── mymission0003.ts
```

For multiple missions (by MissionId)

```
node index.js -m export --missions 5bc6e9de4b75ae3ef0383023,5bc70d124b75ae3ef038315a,5bdab6623c4cf92f646d0b93 -r c:/tmp/ba
```

The resulting archive should look something like this:

```
tmp/
├── /backup/
   ├── 5bc6e9de4b75ae3ef0383023/
   │                           ├── db
   │                               ├── mymission.m3u8
   │                               ├── mymission0000.ts
   │                               ├── mymission0001.ts
   │                               ├── mymission0002.ts
   │                               ├── mymission0003.ts
   ├── 5bc70d124b75ae3ef038315a/
   │                           ├── db
   │                               ├── mymission2.m3u8
   │                               ├── mymission20000.ts
   │                               ├── mymission20001.ts
   │                               ├── mymission20002.ts
   │                               ├── mymission20003.ts
   ...                             ...
```

3.7 Import mission

Import script will perform the following actions:

- Import DB collections
- Copy **missions** assets to the video folder (under sub-folder missionId)

The location of video assets is taken from the server's configuration file

Usage:

For single mission import provide the name of the backed up mission folder

```
node index.js -m import -r c:/tmp/backup/5bc6e9de4b75ae3ef0383023
```

Or using batch file:

```
import c:/tmp/backup/5bc6e9de4b75ae3ef0383023
```

For importing multiple missions (from one folder) provide the path to the folder

For example, as depicted below, the folder c:/tmp/backup contains 4 exported missions:

```
tmp/
├── /backup/
|    ├── 5bc70d124b75ae3ef038315a
|    ├── 5bc8586e4b75ae3ef0383651
|    ├── 5bc8586e4b75ae3ef0383856
|    ├── 5bc8586e4b75ae3ef0383968
```

So, to import all missions, just provide a folder location as a **--root** argument:

```
node index.js -m import -r c:/tmp/backup/
```

4. Microservices

4.1 Reverse proxy

In order to use the **STANAG On Demand Server** in production environment you should consider putting it behind reverse proxy server.
Reverse proxy service sits in front of the **StServer** service, accepting requests from clients for resources located on the server. From the client point of view, the reverse proxy appears to be the actual **STANAG On Demand Server** server. Though it is possible to operate all the internal microservices without it (for example, for testing or inside a bigger system) it is highly recommended that you use it.

4.1.1 NGINX reverse proxy

Though for every particular setup there will be a different reverse proxy configuration, here is a simplified example that does the following:
- Forwards **frontend** and **api** requests to the server running on port 8080.
- Intercepts and forwards requests for video content to the video http server running on port 8084.

```
server {

        listen 80;
        listen [::]:80;

        client_max_body_size 0;                 # to enable large video files upload

        server_name stserver.mydomain.com;

        location /  {

                    proxy_pass         http://localhost:8080;
                    proxy_set_header   X-Forwarded-For $remote_addr;
                    proxy_set_header   Host $http_host;
                    proxy_http_version 1.1;
                    proxy_set_header Upgrade $http_upgrade;
                    proxy_set_header Connection "upgrade";
        }


        location /videos/  {

                    proxy_pass         http://localhost:8084;
                    proxy_set_header   X-Forwarded-For $remote_addr;
                    proxy_set_header   Host $http_host;
        }
}
```

4.2 StSupervisor (Ground Station Monitor)

StSupervisor is a Ground Station STANAG 4609 stream monitoring service used for video / telemetry stream monitoring.

4.2.1 Main features

- Monitors a pre-configured list of multicast video streams (simple yaml configuration file)
- Multiple UAV platforms (with multiple sensors each)
- Timeout, Bitrate, Stream errors and Klv rate reporting
- Low latency video preview
- MISB 0601, 0102, 0903 live decoding
- Stream parameters detection
- User defined **event triggers** based on speed, altitude, location etc
- MQTT messages for easy integration (InfluxDB, Grafana, etc)
- Cross platform Windows/Linux. Easy setup.
- Desktop and mobile UI.

Additional information can be found here StSupervisor

4.2.2 Usage

StSupervisor can be used as stand alone application (no external dependencies required) or as a Docker container.

Stand alone operation

Windows

To start the playback run **stsupervisor.exe** with the command line argument that specifies the configuration file.

```
stsupervisor EoSensor.yml
```

Linux

To start the playback run **stsupervisor.run** with the command line argument that specifies the configuration file.

```
./stsupervisor.run EoSensor.yml
```

Docker container

Create application and platform configuration files (as explained below) in a host directory and mount the directory to the /app/config folder in the container:

```
docker run --rm -it --net=host -v ~/stservices/stsupervisor/config:/app/config impleo/stsupervisor:1.0.12
```

Docker compose

```
version: "3.4"

services:

# StSupervisor
  stsupervisor:
    image: impleo/stsupervisor:1.0.12
    container_name: stsupervisor
    restart: always
    network_mode: host
    volumes:
      - ~/Work/stservices/stsupervisor/config:/app/config
```

Licensing

Application without licdnse will work in demo mode (for about 15 min). In ordar to lift demo restrictions you should provide the license using one of two options:

- passing license info as the arguments (with --licenseFile and --licenseKey)
- copying license file (.lic) and a key (.txt) file into current working directory

Please contact info@impleotv.com for the licensing information.

Configuration

StSupervisor uses 2 configuration files:

- supervisor.yml - application configuration file
- platforms.yml - platforms, sensors etc configuration file

When launched without parameters, **StSupervisor** will look for these two files in a current directory. You can use arbitrary files by supplying --appConfig and --platformConfig arguments. Also, some of the file parameters can be overwritten with the options shown in the **Options** table.

.ENV

StSupervisor will also check for presence of the .env file. If found, the parameter defined in it will override the configuration arguments. The following entries can be defined:

- APP_CONFIG - StSupervisor yaml configuration file. Default - supervisor.yml in the config dir
- PLATFORM_CONFIG - Platforms yaml configuration file. Default - platforms.yml in the config dir
- SERVER_NAME - Server name

Options

Flag	Name	Description
	--appConfig	StSupervisor yaml configuration file. Default - supervisor.yml in root dir
	--platformConfig	Platforms yaml configuration file. Default - platforms.yml in root dir
	--mqttBrokerHost	Mqtt broker host url. If not provided, internal mqtt server will be launched
	--mqttPort	Mqtt broker port
	--mqttWsPort	Mqtt websocket broker port. Default 9001
	--httpPort	HTTP server port. Default 8065. Port to access the application.
	--httpMonitorStartPort	Channel HTTP port. Will start from this value and increment for each instance. Default - 4000
	--wsVideoStreamStartPort	Web socket video preview stream port. Will start from this value. Default - 9010
	--videoBitrate	Video preview bitrate. Default - 96K
	--videoResolution	Video preview resolution. Default - '128x96'
	--stServerMode	Work with StServer
	--nodeInfo	Show NodeInfo string
	--licenseFile	License file
	--licenseKey	License Key
	--printUsage	Print args description (true/false)
-v	--version	Version

Note **StSupervisor** works with multicast streaams only. As it is a monitoring service, it makes no sense to directly consume unicast streams. If, for any reason you only have a unicast in your system and want to monitor the stream, consider using ImpleoTV's **unicast to multicast** recaster.

Basic StSupervisor app configuration

Here is an example of the application configuration file:

```
# StSupervisor config
server:
  httpPort: 8065

# Mqtt section
mqtt:
  brokerHost: tcp://localhost
  internal: true  # use (create) internal MQTT broker
  port: 1883
  wsPort: 9001
  username:
  password:

# influxdb section
influxdb:
  host: 35.178.214.216 # influxdb server address
  port: 8086
  username:
  password:
  database: telegraf

# Stream monitor config section
streamMonitor:
  httpStartPort: 4000               # create stream monitor server starting from port 4000
  wsVideoStreamStartPort: 9010      # create websocket starting from port 4000
  bitrateReportingPeriod: 1000      # report bitrate every 1000 ms
  metadataSampling: 1000            # sample metadata every 1000 ms
  timeout: 1000                     # report stream timeout after 1000 ms
  noSignalAfter: 30000              # report no signal, and restart the monitor (this will allow to receive a different st

# licensing section
license:
  file: StSupervisor-Lenovo.lic
  key: 022B9FA1-9500AD67-86CAD66D-2A459B66
```

Basic platform configuration

Here is an example of the configuration file that defines two UAV platforms (with names **UAV_1** and **UAV_2**)

First UAV has 4 video sensors:

- EO/IR sensor with the stream coming on udp://227.1.1.1:30120
- Tail camera sensor with the stream coming on udp://227.1.1.2:30122
- Operator screen capture sensor with the stream coming on udp://227.1.1.3:30123
- Radar capture sensor with the stream coming on udp://227.1.1.4:30124

Note, **Radar** sensor has **active** setting set to false (active: false), so this sensor is not monitored (set it to true, to enable monitoring).

Second UAV has only one sensor:

- EO/IR sensor with the stream coming on udp://228.1.1.1:1234

If you only have one UAV (platform) to monitor, there should be one platform in the list.

```
# Platforms
platforms:
# First platform
  - platform: UAV_1
    active: true
    name: UAV_1
    description: First platform
    type: UAV
    sensors:
      - sensor: EO
        name: EO/IR
        description: EO/IR sensor
        active: true
        type: video
        url: udp://227.1.1.1:30120

      - sensor: Tail
        name: Tail
        description: Tail camera
        active: true
        type: video
        url: udp://227.1.1.2:30122

      - sensor: Operator
        name: Operator screen
        active: true
        type: video
        url: udp://227.1.1.3:30123

      - sensor: Radar
        name: Radar
        active: false
        type: video
        url: udp://227.1.1.4:30124

# Second platform
  - platform: UAV_2
    active: true
    name: UAV_2
    description: Second platform
    type: UAV
    sensors:
      - sensor: EO
        active: true
        name: EO/IR
        type: video
        url: udp://228.1.1.1:1234
```

Note. If you have to select the specific network interface, use add ?localaddr= to the url:

```
udp://227.1.1.1:30120?localaddr=192.168.1.28
```

Operation

To start stream monitoring run **stsupervisor.exe** (with or without additional command line arguments).

```
stsupervisor
```

4.2.3 Mqtt broker

If no broker host parameter provided, the **StSupervisor** will launch an internal broker. You can use any **Mqtt** broker (note, it must support websockets)

For example, to use **Mosca** Mqtt broker one can launch it with the following command:

```
mosca -v --port 1883 --http-port 9001 --http-bundle --http-static ./
```

4.2.4 Mqtt reporting

Stream Monitor sends events to Mqtt broker in the following format: applicationName/platformId/sensorId/event, where **event** has one of the following values:

event	**Description**
state	Current state
detection	Stream detection results (JSON)
bitrate	Current stream bitrate
custom events	User defined events
config	General configuration information - httpServer, wsVideoPreview
demoExpired	Demo Expired

For example,

```
StreamMonitor/Heron/EO/state
```

4.2.5 State events

Stream Monitor reports the following stream state events:

- Unknown. This state will be reported when Stream Monitor goes offline (or not started yet) and therefore there is no info on the stream available.
- Offline. Stream is not present.
- Online. Stream is running.

4.2.6 Bitrate reporting

Stream Monitor periodically sends bitrate notifications with configurable period

Triggers

Triggers are used to issue **events** when certain **conditions** are met. **Stream Monitor** will send Mqtt message with "trigger" topic and conditions payload when there is a match. Upon reporting, the trigger is removed from an internal queue.

The logic: - Event is fired when the conditions inside the trigger resolve to **true** (logical **AND**) - Event is fired only once (if **armed** state is **true**). After that, the **armed** state is set to **false**.

For example, below are YAML defined triggers:

```
triggers:
  # Send start message when stream becomes online
    - trigger: start
      armed: true
    conditions:
    state: online
  # Send start message based on telemetry defined conditions - ground speed > 100 km/h and altitude > 200 m
    - trigger: start
    armed: true
    conditions:
      state: online
        telemetry:
          speed:
            above: 100
          altitude:
                        above: 200
    # Send arrived message based on telemetry defined conditions - platform location is inside of predefined polygon
    - trigger: arrived
      armed: true
    conditions:
      telemetry:
        location:
          coordinates: [[[34.7373,32.0686],[34.8616,32.0686],[34.8616,32.1238],[34.7373,32.12387],[34.7373,32.0686]]]
    # Send stop message based on telemetry defined conditions - ground speed < 20
    - trigger: stop
      armed: true
    conditions:
      telemetry:
        speed:
          below: 20
```

Triggers can also be included as a separate file:

```
   triggers: !!inc/file triggers.yml
```

4.2.7 HTTP interface

GET

version

Command:

```
http://localhost:8065/version
```

Response:

```
1.0.0
```

platforms

Command:

```
http://localhost:8065/platforms
```

Response:

```
[
  {
    "platform": "UAV_1",
    "active": true,
    ...
     "sensors": [
         ...
     ]
  },
  {
    "platform": "UAV_2",
    "active": true,
    ...
    "sensors": [
        ...
     ]
  }
  ...

]
```

config

Command:

```
http://localhost:8065/config
```

Response:

```
{
  "mqtt": {
    "brokerHost": "tcp://localhost",
    "port": 1883,
    "wsPort": 9001,
    "username": null,
    "password": null
  },
  "streamMonitor": {
    "httpStartPort": 4000,
    "wsVideoStreamStartPort": 9010,
    "bitrateReportingPeriod": 1000,
    "metadataSampling": 1000,
    "timeout": 1000,
    "noSignalAfter": 30000
  }
}
```

nodeinfo

Command:

```
http://localhost:8065/nodeinfo
```

Response:

```
a43c1b0a-a53a-0c90-8810-c06ab1ff3967
```

license

Command:

```
http://localhost:8065/license
```

Response:

```
returns license
```

readme

Command:

```
http://localhost:8065/readme
```

Response:

```
README content...
```

4.3 Frontend

By default, **STANAG On Demand Server** backend acts as a standard http sever when it comes to the static frontend. It is possible however to provide the frontend as a microservice, placing it into separate container or external http server.

4.4 MQTT broker

STANAG On Demand Server uses **MQTT broker** for internal services communication. User can subscribe to the events in order to customize the functionality.

4.5 StServer

STANAG On Demand Server is a main services that provides the following functionality:
- Serve api requests
- Serve (optionally) frontend
- Serve (optionally) video content

4.6 DB

STANAG On Demand Server uses **MongoDB** to store mission descriptive metadata and ingested telemetry.