API Documentation

Instructions to use RigiCloud API

Date: July 16, 2026

Rigi Technologies SA
Route des Flumeaux 39
1008 Prilly, Vaud, Switzerland
UID number: CHE-423.156.595

Main contact:
operations@rigi.tech

Table of contents:

Introduction to Rigitech's APIs

Welcome to Rigitech's API documentation! Our APIs provide a powerful and versatile platform to manage and control various aspects of your drone delivery operations. Whether you're developing a drone monitoring application, automating operations, or integrating our capabilities into your business, our APIs are designed to meet your diverse needs.

What Can You Do with Our APIs?

REST API

Rigitech's REST API serves as the foundation for managing your drone fleet. It offers a wide range of functionalities beyond merely listing drones and batteries. With this API, you can create, modify, and interact with key entities. From scheduling operations to managing resources and even controlling drones, the REST API is your gateway to a world of possibilities. Additionally, it allows you to control drones via a web interface, introducing human validation into critical command execution.

Live WebSocket API

Our Live WebSocket API provides real-time access to vital drone data. You can subscribe to various topics to receive live telemetry, status messages, the current flight path, and other crucial events. This API empowers you to send instant commands to control your drones directly from your application, ensuring swift responsiveness.

AirTraffic API

The AirTraffic API enables you to connect to an external Socket.io server and receive essential information about aerial traffic. You can tailor the data you receive by configuring filters and settings. This functionality helps you monitor and make informed decisions about the presence of other elements in the airspace.

Next Steps

You're on the brink of unlocking a world of possibilities with Rigitech's APIs. Begin by exploring the relevant documentation and start building amazing applications that make the most of your drone fleet and airspace management. If you have any questions or need assistance, feel free to reach out to our support team.

Happy coding!

Getting Started

This tutorial shows you how to connect to RigiCloud API, obtain access tokens, and invoke the API methods using HTTP requests or subscribing to socket streams.

To use the RigiCloud API, it is important that you understand the basics of RESTful web services, websocket and JSON representations.

Step 1: Get a RigiCloud API User Account

To use the RigiCloud API, you need a RigiCloud API User Account. If you already have an account, then you're all set. If not you will need to contact us, and we will provide you an account.

Step 2: Request a secure token

Before to proceed to any Rest call or connect to our live stream you need to login to our system. If the login is correct the system will provide you a secure token needed for the subscriptions or convenient calls. You can invoke the login with this endpoint:

```
POST `https://cloud.rigi.tech/auth/login`;

body: {
    username:<username>,
    password:<password>
}
```

Example:

```
const fetchData=async ()=>{
    let j,response;
    const serverResponse = await fetch(`https://cloud.rigi.tech/auth/login`, {
        mode: "cors",
        method: "POST",
        referrer: "no-referrer",
        body: JSON.stringify({ username: username, password: pw }),
    })
    if(!serverResponse.ok){
        try {
            switch (serverResponse.status){
                case 400:
                     j = await serverResponse.json();
                     break;
                case 403:
                    response =  { message: "Forbidden access to data" };
                case 404:
                    response =  { message: "Address not found" };
                default:
```

```
                    j = await serverResponse.json()
            }
        } catch (e) {
            response =  { message: "Error fetching data" };
        }
    }else{
        j = await serverResponse.json()
    }

    response = { data:j };
    const {data, message} = response
    if(message) console.debug(message)
    if(data) console.debug(data)
}
fetchData()
```

Answer:

```
{
    "success": true,
    "user": {
        "createdAt": <timestamp>,
        "updatedAt": <timestamp>,
        "id": <user_d>,
        "email": <user_email>
        "username": <username>,
        "password": "",
        "initialRoute": <initial_path>,
        "role": <role>
    },
    "initialRoute": <initial_path>,
    "capabilities": [...],
    "settings": [... ],
    "token": <the_token>
}
```

We only need the token from this answer, this token wil be used in all the stream subscriptions calls.

Live WebSocket API

The method above will provide the topic needed to connect to the websocket stream. Once you have the token you can use it to connect with your socket client as described in the corresponding section.

AirTraffic API

The Air traffic Api need a special token. This token needs to be provided by Rigitech. Please contact us to get one.

Get role list

Retrieves all available roles

```
GET /api/1.0/roles
```

Response

Model: application/json

```
ARRAY
[
    {
        "id": <Integer> the unique id of the role
        "name": <String> The role name,
    }
]
```

Example

```
const fetchData=async ()=>{
    let j,response;
    const serverResponse = await fetch(`https://cloud.rigi.tech/api/1.0/roles`, {
        mode: "cors",
        method: "GET",
        referrer: "no-referrer",
        headers: {"authorization": "Bearer " + token}
    })
    if(!serverResponse.ok){
        try {
            switch (serverResponse.status){
                case 400:
                     j = await serverResponse.json();
                     break;
                case 403:
                    response =  { message: "Forbidden access to data" };
                case 404:
                    response =  { message: "Address not found" };
                default:
                     j = await serverResponse.json()
            }
        } catch (e) {
            response =  { message: "Error fetching data" };
        }
```

```
    }else{
        j = await serverResponse.json()
    }

    response = { data:j };
    const {data, message} = response
    if(message) console.debug(message)
    if(data) console.debug(data)
}
fetchData()
```

Answer:


```
[
    {
        "id": 1,
        "name": "admin"
    },
    {
        "id": 6,
        "name": "API Full Control"
    },
    {
        "id": 17,
        "name": "API Read-only"
    },
    {
        "id": 11,
        "name": "Head of Operations"
    },
    {
        "id": 8,
        "name": "Observer"
    },
    {
        "id": 4,
        "name": "Operator"
    },
    {
        "id": 3,
        "name": "R-Engineer"
    },
    {
        "id": 18,
        "name": "R-Manufacturing"
    },
    {
        "id": 14,
        "name": "R-Support"
    },
    {
        "id": 13,
        "name": "R-Viewer (Sales)"
    },
```

```
    {
        "id": 9,
        "name": "UTM viewer"
    }
]
```

Get user list

Retrieves all available users into the specified project

```
GET /api/1.0/users?project={projectId}
```

Response

Model: application/json

```
ARRAY
[
    {
        "id": <Integer> the unique id of the user
        "email": <String> Temail for the user,
        "username": <String> Username for the user,
        "status": <Integer> Status id of the user,
        "role": <Integer> Role id ,
    }
]
```

Query parameters

Some parameters are allowed for a filtered response

> **CAUTION**
> parameters with * are mandatory

```
* project: <Integer> - Get users for that project id
username: <String> - It will return users containing this username in username
email: <String> - It will return users containing this email in email
role: <Integer> - It will return users for that role id
status: <Integer> - It will return users with that status id //(-2: Disabled, -1: Pending, 0: Bloked, 1:
Active)
```

Example

```
const fetchData=async ()=>{
    let j,response;
```

```
    const serverResponse = await fetch(`https://cloud.rigi.tech/api/1.0/users?project=22&status=1`, {
        mode: "cors",
        method: "GET",
        referrer: "no-referrer",
        headers: {"authorization": "Bearer " + token}
    })
    if(!serverResponse.ok){
        try {
            switch (serverResponse.status){
                case 400:
                     j = await serverResponse.json();
                     break;
                case 403:
                    response =  { message: "Forbidden access to data" };
                case 404:
                    response =  { message: "Address not found" };
                default:
                     j = await serverResponse.json()
            }
        } catch (e) {
            response =  { message: "Error fetching data" };
        }
    }else{
        j = await serverResponse.json()
    }

    response = { data:j };
    const {data, message} = response
    if(message) console.debug(message)
    if(data) console.debug(data)
}
fetchData()
```

Answer:

```
[
    {
        "id": 1,
        "email": "cnp1984@gmail.com",
        "username": "carlos",
        "status": 1,
        "role": 1
    },
    {
        "id": 11,
        "email": "beto@gmail.com",
        "username": "beto",
        "status": 1,
        "role": 3
    }
]
```

Get battery types list

Retrieves all available battery types

```
GET /api/1.0/batterytypes
```

Response

Model: application/json

```
ARRAY
[
    {
        "id": <Integer> the battery type id
        "name": <String> The battery type name,
        "manufacture": <String> MAnufacture name,
        "model": <string> Battery typoe model name,
        "voltage": <Integer> Battery type voltage (volts)
        "cells": <Integer> Battery type cells number,
        "chemistry": <string> Battery type chemistry
        "capacity": <Integer> Battery type capacity (Ah)
        "maxDischargeCurrent": <Integer> Battery type discharge current (A)
        "weight": <Integer> Battery type weight (g)
        "length":  <Float> Battery type length (mm)
        "width":  <Float> Battery type width (mm)
        "height":  <Float> Battery type height (mm)
    }
]
```

Example

```
const fetchData=async ()=>{
    let j,response;
    const serverResponse = await fetch(`https://cloud.rigi.tech/api/1.0/batterytypes`, {
        mode: "cors",
        method: "GET",
        referrer: "no-referrer",
        headers: {"authorization": "Bearer " + token}
    })
    if(!serverResponse.ok){
        try {
            switch (serverResponse.status){
                case 400:
                    j = await serverResponse.json();
```

```
                    break;
                case 403:
                    response =  { message: "Forbidden access to data" };
                case 404:
                    response =  { message: "Address not found" };
                default:
                    j = await serverResponse.json()
            }
        } catch (e) {
            response =  { message: "Error fetching data" };
        }
    }else{
        j = await serverResponse.json()
    }

    response = { data:j };
    const {data, message} = response
    if(message) console.debug(message)
    if(data) console.debug(data)
}
fetchData()
```

Answer:


```
[
    {
        "id": 1,
        "name": "Eiger Kamikaze Test Battery (12S)",
        "manufacture": "Vant",
        "model": "Vant 22Ah 6S 1190190HP AS150 - pack of 2",
        "voltage": 22,
        "cells": 6,
        "chemistry": "LiPo",
        "capacity": 22,
        "maxDischargeCurrent": 660,
        "weight": 2700,
        "length": 203,
        "width": 69,
        "height": 94
    },
    {
        "id": 2,
        "name": "Minto Flight Battery (6S)",
        "manufacture": "Tattu",
        "model": "Tattu 7Ah 6S 25C XT90",
        "voltage": 22.2,
        "cells": 6,
        "chemistry": "LiPo",
        "capacity": 7000,
        "maxDischargeCurrent": 175,
        "weight": 907.5,
        "length": 136.48,
        "width": 68.28,
        "height": 42.07
```

```
    }
]
```

Get batteries list

Retrieves all available batteries into the specified project

```
GET /api/1.0/batteries?project={projectId}
```

Response

Model: application/json

```
ARRAY
[
    {
        "id": <Integer> Battery id
        "serialNumber": <String> The battery Serial number,
        "generalComments": <String> Battery general comments,
        "manufacturerId": <String> Battery tmanufacturer id
        "status": <Integer> Battery status, //(-1: DELETED, 1: ACTIVE, 2: ARCHIVED)
        "base": <Integer> Base id
        "batteryType": <Integer> Battery type id
        "firstUse": <String> Battery first usage registered
        "totalFlightTime": <String> Total flight time for this battery (HH:mm:ss)
        "totalNumberOfOperations":  <Integer> Battery number of operations
        "registrationDate":  <String> Battery registration date
        "kmFlown":  <Float> Battery km flown (km)
    }
]
```

Query parameters

Some parameters are allowed for a filtered response

> **CAUTION**
> parameters with * are mandatory

```
* project: <Integer> - Get batteries for that project id
status: <String> - It will return batteries with the specified status id (-1 = Deleted, 0 = Active, 2 =
Archived)
```

Example

```
const fetchData=async ()=>{
    let j,response;
    const serverResponse = await fetch(`https://cloud.rigi.tech/api/1.0/batteries?project=22&status=1`,
{
        mode: "cors",
        method: "GET",
        referrer: "no-referrer",
        headers: {"authorization": "Bearer " + token}
    })
    if(!serverResponse.ok){
        try {
            switch (serverResponse.status){
                case 400:
                     j = await serverResponse.json();
                     break;
                case 403:
                    response =  { message: "Forbidden access to data" };
                case 404:
                    response =  { message: "Address not found" };
                default:
                     j = await serverResponse.json()
            }
        } catch (e) {
            response =  { message: "Error fetching data" };
        }
    }else{
        j = await serverResponse.json()
    }

    response = { data:j };
    const {data, message} = response
    if(message) console.debug(message)
    if(data) console.debug(data)
}
fetchData()
```

Answer:


```
[
    {
        "id": 11,
        "serialNumber": "BAT-RT-0002",
        "generalComments": "Testing battery.",
        "manufacturerId": "",
        "status": 1,
        "base": 55,
        "batteryType": 2,
        "firstUse": "2021-11-24T08:50:50.000Z",
        "totalFlightTime": "15:20:16",
        "totalNumberOfOperations": 55,
        "registrationDate": "2021-03-01T00:00:00.000Z",
        "kmFlown": 2074.01
    },
```

```
    {
        "id": 12,
        "serialNumber": "BAT-RT-0001",
        "generalComments": "Testing battery 2.",
        "manufacturerId": "",
        "status": 1,
        "base": 55,
        "batteryType": 2,
        "firstUse": "2021-11-29T09:52:55.000Z",
        "totalFlightTime": "01:47:14",
        "totalNumberOfOperations": 4,
        "registrationDate": "2021-03-01T00:00:00.000Z",
        "kmFlown": 182.84
    }
]
```

Get bases list

Retrieves all available bases into the specified project

```
GET /api/1.0/bases?project={projectId}
```

Response

Model: application/json

```
ARRAY
[
    {
        "id": <Integer> Base unique id
        "name": <String> The base name,
        "description": <String> The base description
        "lat": <Float> The base latitude
        "lng": <Float> The base longitude
        "altitude": <Integer> The base altitude (AMSL)
    }
]
```

Query parameters

> **CAUTION**
> parameters with * are mandatory

```
* project: <Integer> - Get drones for that project id
```

Example

```
const fetchData=async ()=>{
    let j,response;
    const serverResponse = await fetch(`https://cloud.rigi.tech/api/1.0/bases?project=22`, {
        mode: "cors",
        method: "GET",
        referrer: "no-referrer",
        headers: {"authorization": "Bearer " + token}
    })
```

```
    if(!serverResponse.ok){
        try {
            switch (serverResponse.status){
                case 400:
                     j = await serverResponse.json();
                     break;
                case 403:
                    response =  { message: "Forbidden access to data" };
                case 404:
                    response =  { message: "Address not found" };
                default:
                     j = await serverResponse.json()
            }
        } catch (e) {
            response =  { message: "Error fetching data" };
        }
    }else{
        j = await serverResponse.json()
    }

    response = { data:j };
    const {data, message} = response
    if(message) console.debug(message)
    if(data) console.debug(data)
}
fetchData()
```

Answer:

```
[
    {
        "id": 8,
        "name": "Lausanne airport",
        "description": "Hospital in Svendborg",
        "lat": 50.0549,
        "lng": 10.6039,
        "altitude": 480
    },
    {
        "id": 9,
        "name": "Geneve airport",
        "description": "Hospital in Odense",
        "lat": 50.3849,
        "lng": 10.3701,
        "altitude": 400
    },
]
```

Get drone list

Retrieves all available drones into the specified project

```
GET /api/1.0/drones?project={projectId}
```

Response

Model: application/json

```
ARRAY
[
    {
        "id": <Integer> the unique id of the drone
        "serialNumber": <String> The serial number for the drone,
        "model": <String> Drone model,
        "revision": <string> Revision for the drone,
        "registrationDate": <String> Drone registration date into the system,
        "simulator": <Boolean> Indicates if is a simulator,
        "emptyWeight": <Float> Empty weight for the drone (in kg),
        "status": <Integer> Status for the drone (1 = active, 0 = inactive)
    }
]
```

Query parameters

Some parameters are allowed for a filtered response

> **CAUTION**
> parameters with * are mandatory

```
* project: <Integer> - Get drones for that project id
serialNumber: <String> - It will return anything containing this string in serialNumber
simulator: <Integer> - It will return simulators or not simulators, (1=is simulator, 0 = is not
simulator)
```

Example

```
const fetchData=async ()=>{
    let j,response;
    const serverResponse = await fetch(`https://cloud.rigi.tech/api/1.0/drones?
project=3&serialNumber=GA`, {
        mode: "cors",
        method: "GET",
        referrer: "no-referrer",
        headers: {"authorization": "Bearer " + token}
    })
    if(!serverResponse.ok){
        try {
            switch (serverResponse.status){
                case 400:
                     j = await serverResponse.json();
                     break;
                case 403:
                    response =  { message: "Forbidden access to data" };
                case 404:
                    response =  { message: "Address not found" };
                default:
                     j = await serverResponse.json()
            }
        } catch (e) {
            response =  { message: "Error fetching data" };
        }
    }else{
        j = await serverResponse.json()
    }

    response = { data:j };
    const {data, message} = response
    if(message) console.debug(message)
    if(data) console.debug(data)
}
fetchData()
```

Answer:

```
[
    {
        "id": 41,
        "serialNumber": "GA_DEV_CARLOS",
        "model": "SimVtol",
        "revision": "2",
        "registrationDate": "1970-01-01T00:00:00.000Z",
        "simulator": true,
        "emptyWeight": 10,
        "status": 1
    },
    {
        "id": 41,
        "serialNumber": "GA_DEV_CARLOS2",
        "model": "SimVtol",
```

```
        "revision": "2",
        "registrationDate": "1970-01-01T00:00:00.000Z",
        "simulator": true,
        "emptyWeight": 10,
        "status": 1
    }
]
```

Get drone information

Retrieves drone information for the specified drone. You can get the drone id listing all the drones first

```
GET /api/1.0/drone/{droneId}
```

TIP

droneId can be obtained from the get drone list endpoint. You can use the id field or the serialNumber field. Both values will work.

Response

Model: application/json

```
{
    "id": <Integer> the unique id of the drone
    "serialNumber": <String> The serial number for the drone,
    "model": <String> Drone model,
    "revision": <string> Revision for the drone,
    "registrationDate": <String> Drone registration date into the system,
    "simulator": <Boolean> Indicates if is a simulator,
    "emptyWeight": <Float> Empty weight for the drone (in kg),
    "status": <Integer> Status for the drone (1 = active, 0 = inactive)
}
```

Example

```
const fetchData=async ()=>{
    let j,response;
    const serverResponse = await fetch(`https://cloud.rigi.tech/api/1.0/drone/41`, {
        mode: "cors",
        method: "GET",
        referrer: "no-referrer",
        headers: {"authorization": "Bearer " + token}
    })
    if(!serverResponse.ok){
        try {
            switch (serverResponse.status){
                case 400:
                    j = await serverResponse.json();
                    break;
```

```
                case 403:
                    response =  { message: "Forbidden access to data" };
                case 404:
                    response =  { message: "Address not found" };
                default:
                     j = await serverResponse.json()
            }
        } catch (e) {
            response =  { message: "Error fetching data" };
        }
    }else{
        j = await serverResponse.json()
    }

    response = { data:j };
    const {data, message} = response
    if(message) console.debug(message)
    if(data) console.debug(data)
}
fetchData()
```

Answer:

```
{
    "id": 41,
    "serialNumber": "SIM_DEV_CARLOS",
    "model": "SimVtol",
    "revision": "2",
    "registrationDate": "1970-01-01T00:00:00.000Z",
    "simulator": true,
    "emptyWeight": 10,
    "status": 1
}
```

Get current drone mission

Retrieves the current mission for the specified drone

```
GET /api/1.0/drone/{droneId}/mission
```

TIP

droneId can be obtained from the get drone list endpoint. You can use the id field or the serialNumber field. Both values will work.

Response

Model: application/json

You can find the route specification at Flightplan format

CAUTION

If the drone is not connected to the system, this call will return an empty mission as value.

Example

```
const fetchData=async ()=>{
    let j,response;
    const serverResponse = await fetch(`https://cloud.rigi.tech/api/1.0/drone/41/misssion`, {
        mode: "cors",
        method: "GET",
        referrer: "no-referrer",
        headers: {"authorization": "Bearer " + token}
    })
    if(!serverResponse.ok){
        try {
            switch (serverResponse.status){
                case 400:
                     j = await serverResponse.json();
                     break;
                case 403:
                    response =  { message: "Forbidden access to data" };
                case 404:
                    response =  { message: "Address not found" };
                default:
                     j = await serverResponse.json()
            }
        } catch (e) {
```

```
            response =  { message: "Error fetching data" };
        }
    }else{
        j = await serverResponse.json()
    }

    response = { data:j };
    const {data, message} = response
    if(message) console.debug(message)
    if(data) console.debug(data)
}
fetchData()
```

Answer:


```
{
    "fileType": "Plan",
    "geoFence": {
        "circles": [],
        "polygons": [],
        "version": 2
    },
    "groundStation": "QGroundControl",
    "mission": {
        "cruiseSpeed": 15,
        "firmwareType": 12,
        "hoverSpeed": 5,
        "items": [
            {
                "AMSLAltAboveTerrain": 1502,
                "WGS84Altitude": 1534.622186577788,
                "AltitudeAboveTerrain": null,
                "Altitude": 80,
                "AltitudeMode": 1,
                "autoContinue": true,
                "command": 22,
                "doJumpId": 1,
                "frame": 3,
                "params": [
                    0,
                    0,
                    0,
                    null,
                    46.528106480035156,
                    7.111464262022529,
                    80
                ],
                "type": "SimpleItem"
            },
            {
                "AMSLAltAboveTerrain": 1702,
                "WGS84Altitude": 1746.7657908230472,
                "AltitudeAboveTerrain": null,
                "Altitude": 80,
```

```
            "AltitudeMode": 1,
            "autoContinue": true,
            "command": 16,
            "doJumpId": 2,
            "frame": 3,
            "params": [
                0,
                0,
                0,
                null,
                46.577215025068966,
                7.130690336241279,
                80
            ],
            "type": "SimpleItem"
        },
        {
            "AMSLAltAboveTerrain": 1702,
            "WGS84Altitude": 1746.7657908230472,
            "AltitudeAboveTerrain": null,
            "Altitude": 80,
            "AltitudeMode": 1,
            "autoContinue": true,
            "command": 16,
            "doJumpId": 3,
            "frame": 3,
            "params": [
                0,
                0,
                0,
                null,
                46.596091095095595,
                7.155409574522529,
                80
            ],
            "type": "SimpleItem"
        },
        {
            "AMSLAltAboveTerrain": 1702,
            "WGS84Altitude": 1746.7657908230472,
            "AltitudeAboveTerrain": null,
            "Altitude": 80,
            "AltitudeMode": 1,
            "autoContinue": true,
            "command": 16,
            "doJumpId": 4,
            "frame": 3,
            "params": [
                0,
                0,
                0,
                null,
                46.594510235348935,
                7.1645893837283126,
                80
```

```
                ],
                "type": "SimpleItem"
            },
            {
                "AMSLAltAboveTerrain": 1702,
                "WGS84Altitude": 1746.7657908230472,
                "AltitudeAboveTerrain": null,
                "Altitude": 80,
                "AltitudeMode": 1,
                "autoContinue": true,
                "command": 16,
                "doJumpId": 5,
                "frame": 3,
                "params": [
                    0,
                    0,
                    0,
                    null,
                    46.5860165694928,
                    7.1700825477908126,
                    80
                ],
                "type": "SimpleItem"
            },
            {
                "AMSLAltAboveTerrain": 1702,
                "WGS84Altitude": 1746.7657908230472,
                "AltitudeAboveTerrain": null,
                "Altitude": 80,
                "AltitudeMode": 1,
                "autoContinue": true,
                "command": 16,
                "doJumpId": 6,
                "frame": 3,
                "params": [
                    0,
                    0,
                    0,
                    null,
                    46.57492561360571,
                    7.1618428016970626,
                    80
                ],
                "type": "SimpleItem"
            }
        ],
        "plannedHomePosition": [
            46.528106480035156,
            7.1166141033311225,
            1622,
            1666.7657908230472
        ],
        "vehicleType": 20,
        "version": 2
    },
```

```
    "rallyPoints": {
        "points": [],
        "version": 2
    },
    "version": 1
}
```

Get route list

Retrieves all available routes into the specified project

```
GET /api/1.0/routes?project={projectId}
```

Response

Model: application/json

```
ARRAY
[
      {
        "id": <Integer> Unique route id,
        "name": <String> Route name,
        "approved":  <Integer> Is the route approved or not (1 = aproved, 0 = not aproved),
      }
]
```

Query parameters

Some parameters are allowed for a filtered response, parameters with * are mandatory

> **CAUTION**
> parameters with * are mandatory

```
* project: <Integer> - Get routes for that project id
```

Example

```
const fetchData=async ()=>{
    let j,response;
    const serverResponse = await fetch(`https://cloud.rigi.tech/api/1.0/routes?project=3`, {
        mode: "cors",
        method: "GET",
        referrer: "no-referrer",
        headers: {"authorization": "Bearer " + token}
    })
    if(!serverResponse.ok){
```

```
        try {
            switch (serverResponse.status){
                case 400:
                     j = await serverResponse.json();
                     break;
                case 403:
                    response =  { message: "Forbidden access to data" };
                case 404:
                    response =  { message: "Address not found" };
                default:
                     j = await serverResponse.json()
            }
        } catch (e) {
            response =  { message: "Error fetching data" };
        }
    }else{
        j = await serverResponse.json()
    }

    response = { data:j };
    const {data, message} = response
    if(message) console.debug(message)
    if(data) console.debug(data)
}
fetchData()
```

Answer:

```
[
    {
        "id": 268,
        "name": "route1",
        "approved": 1
    },
    {
        "id": 290,
        "name": "route2",
        "approved": 1
    },
    {
        "id": 585,
        "name": "route3",
        "approved": 0
    }
]
```

Get route

Retrieves the route specified

```
GET /api/1.0/route/{routeId}
```

TIP

droneId can be obtained from the get route list endpoint. You can use the id field.

Response

Model: application/json

You can find the route specification at Flightplan format

Example

```
const fetchData=async ()=>{
    let j,response;
    const serverResponse = await fetch(`https://cloud.rigi.tech/api/1.0/route/290`, {
        mode: "cors",
        method: "GET",
        referrer: "no-referrer",
        headers: {"authorization": "Bearer " + token}
    })
    if(!serverResponse.ok){
        try {
            switch (serverResponse.status){
                case 400:
                     j = await serverResponse.json();
                     break;
                case 403:
                    response =  { message: "Forbidden access to data" };
                case 404:
                    response =  { message: "Address not found" };
                default:
                     j = await serverResponse.json()
            }
        } catch (e) {
            response =  { message: "Error fetching data" };
        }
    }else{
        j = await serverResponse.json()
    }
```

```
    response = { data:j };
    const {data, message} = response
    if(message) console.debug(message)
    if(data) console.debug(data)
}
fetchData()
```

Answer:


```
{
  "uuid": "1b273f50-edbe-4d20-abc6-e38d0ccd107f",
  "version": 2,
  "safetySettings": {},
  "safetyProfile": null,
  "meta": {
    "altitudeMode": 2
  },
  "geoFence": {
    "circles": [],
    "polygons": [
      {
        "uuid": "f810f84e-7f36-4719-88be-b8933a8f0ede",
        "inclusion": "inclusion",
        "type": "geocage",
        "altAmsl": 606,
        "vertices": [
          {
            "uuid": "3a787436-42d9-44d3-90a4-cf2a7b7a8515",
            "lat": 49.150077591022544,
            "lon": 16.825107180165574
          },
          {
            "uuid": "07597893-91dd-4b0a-a279-f1c06a103f2e",
            "lat": 49.1525792724473,
            "lon": 16.82125552802019
          },
          {
            "uuid": "ab91c46a-28e7-47e7-b70c-0cb7c1ad1a3b",
            "lat": 49.15376165013778,
            "lon": 16.81496843008928
          },
          {
            "uuid": "55139d20-8ffb-4c63-a95c-c91dcb22caf5",
            "lat": 49.159680131436424,
            "lon": 16.797056638287827
          },
          {
            "uuid": "bcd105a3-7eed-4cee-b204-167cc74d1f8a",
            "lat": 49.16061853343021,
            "lon": 16.78889467625551
          },
          {
            "uuid": "4794c72a-a557-4f84-83c2-603f33197177",
```

```
      "lat": 49.1588066122935,
      "lon": 16.781934343861863
    },
    {
      "uuid": "04ffc700-b70d-4cee-9649-bfa59d84dc03",
      "lat": 49.156665723624045,
      "lon": 16.763573282050416
    },
    {
      "uuid": "183ebe2c-8c7a-4740-a666-917fd7fdfb01",
      "lat": 49.154121708361615,
      "lon": 16.75222552626066
    },
    {
      "uuid": "df0e8fe1-3808-4377-92b0-202fd2a9152b",
      "lat": 49.15045478109202,
      "lon": 16.749031685876176
    },
    {
      "uuid": "79e43234-4b07-4ccf-b8da-3e2f3169dadb",
      "lat": 49.15087034473323,
      "lon": 16.73482838331394
    },
    {
      "uuid": "273fa702-c267-4d37-b884-bf0585bb4ab9",
      "lat": 49.15204377237618,
      "lon": 16.728736080812737
    },
    {
      "uuid": "67e2dc89-32b1-4712-844a-276b15ff8fe2",
      "lat": 49.15747909242239,
      "lon": 16.718050998287723
    },
    {
      "uuid": "5555b1f9-dbba-41ec-8ac0-e5ad7d19e44f",
      "lat": 49.15871855242119,
      "lon": 16.707182896906957
    },
    {
      "uuid": "78b11855-cd26-40c2-9a5d-ff59b485261a",
      "lat": 49.15592011697326,
      "lon": 16.697717433680296
    },
    {
      "uuid": "44867986-1926-41f3-b260-de6c48fb3804",
      "lat": 49.15532924349978,
      "lon": 16.69664389261881
    },
    {
      "uuid": "cbb195c9-c9ca-4c44-b684-cac1f63df8f7",
      "lat": 49.15410684189552,
      "lon": 16.699218155817576
    },
    {
      "uuid": "1a32a299-d6f6-44ab-a19e-afe233c8be69",
```

```
      "lat": 49.15394953255132,
      "lon": 16.705954549951926
    },
    {
      "uuid": "f497465e-b3a4-45b3-82aa-9a800681f062",
      "lat": 49.152632613814326,
      "lon": 16.71185015014045
    },
    {
      "uuid": "553e56de-85cc-4edc-b533-d082271068bf",
      "lat": 49.1453995376264,
      "lon": 16.723716661917493
    },
    {
      "uuid": "05a117d0-103f-4ad1-a13b-ffa003adf2b2",
      "lat": 49.144687954965214,
      "lon": 16.73247223033242
    },
    {
      "uuid": "64b8a136-760d-49c3-82ab-536e54ca2b85",
      "lat": 49.14278810311593,
      "lon": 16.738223305555508
    },
    {
      "uuid": "4d7ebe08-529c-4904-b2da-328022c91a7e",
      "lat": 49.14037824223416,
      "lon": 16.739596806118712
    },
    {
      "uuid": "9142156f-cfcb-4d4c-b628-9fdcae6d2c5a",
      "lat": 49.140354745397154,
      "lon": 16.74912422208719
    },
    {
      "uuid": "d1c5770b-30c3-4ce3-a1ed-7e8847824aab",
      "lat": 49.14489181364054,
      "lon": 16.757474609302804
    },
    {
      "uuid": "c1ff6edd-33a5-4afa-ae96-890fa17f3e1f",
      "lat": 49.1466213436577,
      "lon": 16.76951571612291
    },
    {
      "uuid": "e4d74178-6b04-4ccd-a89e-af4db5765e01",
      "lat": 49.142827966263894,
      "lon": 16.777396046208665
    },
    {
      "uuid": "c847d2de-b0dc-4d42-82f4-39819404b365",
      "lat": 49.146266933091326,
      "lon": 16.795914017247483
    },
    {
      "uuid": "d3796f54-f087-4956-a7eb-4af2af0d6089",
```

```
            "lat": 49.14568092194048,
            "lon": 16.80801614432268
          },
          {
            "uuid": "544203c0-a3ee-44cd-99bd-e609cbc4fd14",
            "lat": 49.14392284690498,
            "lon": 16.810891472386643
          },
          {
            "uuid": "2fd7f924-008d-4d21-9a90-49c6d07dfecb",
            "lat": 49.142629692022375,
            "lon": 16.81568189768724
          },
          {
            "uuid": "4ce5de83-4863-43cd-90e9-48592f81748e",
            "lat": 49.14238228676779,
            "lon": 16.821888529347703
          },
          {
            "uuid": "f0657467-a0ca-424e-a261-51c7b16d2ce5",
            "lat": 49.14256827948575,
            "lon": 16.825053535985276
          },
          {
            "uuid": "0acac6df-5a6a-4685-88e3-161035daff4b",
            "lat": 49.144357985057624,
            "lon": 16.82549341826372
          },
          {
            "uuid": "82363d36-4317-4054-bbdc-8952aa1fe142",
            "lat": 49.14684942832515,
            "lon": 16.82568653731279
          }
        ],
        "altConversions": {
          "altWgs84": 650.0502829949656
        },
        "meta": {
          "editMode": false,
          "color": "#ffa500",
          "visible": true
        }
      }
    ]
  },
  "mission": [
    {
      "uuid": "cb29a5cf-e98f-4885-8893-90eefb22326e",
      "command": 22,
      "altAmsl": 350,
      "lat": 49.15108304952246,
      "lon": 16.79625748449217,
      "altConversions": {
        "altWgs84": 394.11329777267474,
        "altAboveTakeoff": 16,
```

```
        "altAboveTerrain": 83,
        "altGroundAmsl": 267,
        "altGroundWgs84": 311.11329777267474
      },
      "padAltAmsl": 267
    },
    {
      "uuid": "17a2ba19-1c44-473c-8b2d-aec626c3ac0d",
      "command": 16,
      "altAmsl": 350,
      "lat": 49.15052165642498,
      "lon": 16.763470161494123,
      "altConversions": {
        "altWgs84": 394.18614799696365,
        "altAboveTakeoff": 16,
        "altAboveTerrain": 105.78,
        "altGroundAmsl": 244.22,
        "altGroundWgs84": 288.4061479969637
      }
    },
    {
      "uuid": "90b66945-eb65-41af-b632-b4731b3713ee",
      "command": 3000,
      "altAmsl": 350,
      "lat": 49.15052165642498,
      "lon": 16.763470161494123,
      "altConversions": {
        "altWgs84": 394.18614799696365,
        "altAboveTakeoff": 16,
        "altAboveTerrain": 105.78,
        "altGroundAmsl": 244.22,
        "altGroundWgs84": 288.4061479969637
      },
      "transitionType": "front"
    },
    {
      "uuid": "1f3f75e1-c149-4933-99fd-d1c221abf7d0",
      "command": 16,
      "altAmsl": 350,
      "lat": 49.14911814584975,
      "lon": 16.754372108515607,
      "altConversions": {
        "altWgs84": 394.20467686945904,
        "altAboveTakeoff": 16,
        "altAboveTerrain": 110.02000000000001,
        "altGroundAmsl": 239.98,
        "altGroundWgs84": 284.18467686945905
      }
    },
    {
      "uuid": "27b289ec-453c-439c-90b9-84017accfb5c",
      "command": 3000,
      "altAmsl": 350,
      "lat": 49.146872446229374,
      "lon": 16.745960701044904,
```

```
      "altConversions": {
        "altWgs84": 394.22110539882925,
        "altAboveTakeoff": 16,
        "altAboveTerrain": 119.68,
        "altGroundAmsl": 230.32,
        "altGroundWgs84": 274.54110539882925
      },
      "transitionType": "back"
    },
    {
      "uuid": "f8dbcd23-ba71-4e2b-8115-889ce06ff670",
      "command": 16,
      "altAmsl": 360,
      "lat": 49.147939166240384,
      "lon": 16.732141960200178,
      "altConversions": {
        "altWgs84": 404.2607765552109,
        "altAboveTakeoff": 16,
        "altAboveTerrain": 10,
        "altGroundAmsl": 350,
        "altGroundWgs84": 394.2607765552109
      }
    },
    {
      "uuid": "bdc3da24-6db1-40fa-a0ad-4486f211309e",
      "command": 21,
      "altAmsl": 360,
      "lat": 49.147939166240384,
      "lon": 16.732141960200178,
      "altConversions": {
        "altWgs84": 404.2607765552109,
        "altAboveTakeoff": 16,
        "altAboveTerrain": 10,
        "altGroundAmsl": 350,
        "altGroundWgs84": 394.2607765552109
      },
      "padAltAmsl": 350
    }
  ],
  "rallyPoints": [
    {
      "uuid": "829d915a-29f6-4d17-aa24-a208080fe9d9",
      "lat": 49.15916640477572,
      "lon": 16.80844544225584,
      "padAltAmsl": 266.64,
      "altAmsl": 286.64,
      "altConversions": {
        "altWgs84": 330.7479568737182,
        "altPadWgs84": 310.7479568737182
      }
    },
    {
      "uuid": "5e1eedd9-3c9a-4e4e-b9dc-447302ee6981",
      "lat": 49.16489131666681,
      "lon": 16.78097962194334,
```

```
      "padAltAmsl": 233.8,
      "altAmsl": 253.8,
      "altConversions": {
        "altWgs84": 297.98100594923847,
        "altPadWgs84": 277.98100594923847
      }
    }
  ]
}
```

Get edge node list

Retrieves all available edge nodes into the specified project

```
GET /api/1.0/edge_nodes?project={projectId}
```

Response

Model: application/json

```
ARRAY
[
    {
        "id": <Integer> the unique edge node id,
        "serialNumber": <String> the unique edge node serial serialNumber,
        "lastSynchronization": <String> last synchronization date,
        "version": <String> edge node software version,
        "lat": <Float> last known latitude,
        "lon": <Float> last known longitude,
    }
]
```

Query parameters

> **CAUTION**
>
> parameters with * are mandatory

```
* project: <Integer> - Get edge nodes for that project id
```

Example

```
const fetchData=async ()=>{
    let j,response;
    const serverResponse = await fetch(`https://cloud.rigi.tech/api/1.0/edge_nodes?project=3`, {
        mode: "cors",
        method: "GET",
        referrer: "no-referrer",
        headers: {"authorization": "Bearer " + token}
    })
```

```
    if(!serverResponse.ok){
        try {
            switch (serverResponse.status){
                case 400:
                     j = await serverResponse.json();
                     break;
                case 403:
                    response =  { message: "Forbidden access to data" };
                case 404:
                    response =  { message: "Address not found" };
                default:
                     j = await serverResponse.json()
            }
        } catch (e) {
            response =  { message: "Error fetching data" };
        }
    }else{
        j = await serverResponse.json()
    }

    response = { data:j };
    const {data, message} = response
    if(message) console.debug(message)
    if(data) console.debug(data)
}
fetchData()
```

Answer:


```
[
    {
        "id": 3,
        "serialNumber": "ED_01_0003",
        "lastSynchronization": "2023-09-26T08:30:00.000Z",
        "version": "1.9.4",
        "lat": 55.4718345,
        "lon": 10.325116166666668
    },
    {
        "id": 5,
        "serialNumber": "ED_01_0004",
        "lastSynchronization": "2023-09-26T11:25:00.000Z",
        "version": "1.9.4",
        "lat": 46.53064033333333,
        "lon": 6.602070333333334
    },
    {
        "id": 18,
        "serialNumber": "ED_01_5002",
        "lastSynchronization": "2023-09-05T14:15:06.000Z",
        "version": "1.8.10",
        "lat": 56.595916333333335,
        "lon": 11.152845166666667
```

```
    }
]
```

Get delivery requests list

Retrieves all available delivery requests into the specified project

```
GET /api/1.0/requests?project={projectId}
```

Response

Model: application/json

```
ARRAY
[
    {
        "id": <Integer> the unique id of the drone
        "urgent": <boolean> The request urgency,
        "from": <String> The request base uuid (from),
        "to": <String> The request base uuid (t),
        "comments": <String> Requst comments,
        "status": <String> Requst status ("In process", "Scheduled", "Out for delivery","Completed"),
        "containers": <Integer> Number of containers for the delivery,
        "operation": <String> Sync id corresponding to the operatin manging the delivery
    }
]
```

Query parameters

Some parameters are allowed for a filtered response

> **CAUTION**
> parameters with * are mandatory

```
* project: <Integer> - Get drones for that project id
status: <Integer> - will retunr operations with this status (1="In process", 2="Scheduled", 3="Out for
delivery", 4="Completed"). Multiple status are allowed separated by comma.
```

Example

```
const fetchData=async ()=>{
    let j,response;
```

```
    const serverResponse = await fetch(`https://cloud.rigi.tech/api/1.0/requests?project=3&status=1`, {
        mode: "cors",
        method: "GET",
        referrer: "no-referrer",
        headers: {"authorization": "Bearer " + token}
    })
    if(!serverResponse.ok){
        try {
            switch (serverResponse.status){
                case 400:
                     j = await serverResponse.json();
                     break;
                case 403:
                    response =  { message: "Forbidden access to data" };
                case 404:
                    response =  { message: "Address not found" };
                default:
                     j = await serverResponse.json()
            }
        } catch (e) {
            response =  { message: "Error fetching data" };
        }
    }else{
        j = await serverResponse.json()
    }

    response = { data:j };
    const {data, message} = response
    if(message) console.debug(message)
    if(data) console.debug(data)
}
fetchData()
```

Answer:


```
[
    {
        "id": 23,
        "urgent": false,
        "from": "40f98fcf-38f9-4a08-aea5-831bd5eb8fd8",
        "to": "40f98fcf-38f9-4a08-aea5-831bd5eb8fd8",
        "comments": "A description to add",
        "status": "In process",
        "containers": 2,
        "operation": null
    },
    {
        "id": 22,
        "urgent": false,
        "from": "40f98fcf-38f9-4a08-aea5-831bd5eb8fd8",
        "to": "40f98fcf-38f9-4a08-aea5-831bd5eb8fd8",
        "comments": "A description to add",
        "status": "In process",
        "containers": 2,
```

```
        "operation": null
    }
]
```

Create new delivery request

Creates a new delivery request with the specified values

```
POST /api/1.0/request?project={projectId}
```

CAUTION

parameters with * are mandatory

Body

```
* origin: <String> The origin base uuid for the delivery,
* destination: <String date> The destination base uuid for the delivery,
containers: <Integer> Number of containers. value between 0 and 10 (kg),
urgent: <Integer> 0 = not urgent, 1 = urgent,
comments: <String> Comments for the delivery,
```

TIP

You can obtain the base origin and destination uuid retrieving the base list from the project.

Response

Model: application/json

```
{
       "id": <Integer> the unique id of the drone
       "urgent": <boolean> The request urgency,
       "from": <String> The request base uuid (from),
       "to": <String> The request base uuid (t),
       "comments": <String> Requst comments,
       "status": <String> Requst status ("In process", "Scheduled", "Out for delivery","Completed"),
       "containers": <Integer> Number of containers for the delivery,
       "operation": <String> Sync id corresponding to the operatin manging the delivery
   }
```

TIP

When the delivery request is created the operation is not yet assigned, this value will be present when the deivery request is attached to the operation.

Example

```
const fetchData=async ()=>{
    let j,response;
    const serverResponse = await fetch(`https://cloud.rigi.tech/api/1.0/request`, {
        mode: "cors",
        method: "POST",
        referrer: "no-referrer",
        headers: {"authorization": "Bearer " + token}
        body: JSON.stringify({
          "urgent": 0,
          "containers": 2,
          "origin": "40f98fcf-38f9-4a08-aea5-831bd5eb8fd8",
          "destination":  "3e8bf724-b17a-4046-8eb1-dc63a378fc7c",
          "comments": "A description to add"
        }),
    })
    if(!serverResponse.ok){
        try {
            switch (serverResponse.status){
                case 400:
                     j = await serverResponse.json();
                     break;
                case 403:
                    response =  { message: "Forbidden access to data" };
                case 404:
                    response =  { message: "Address not found" };
                default:
                     j = await serverResponse.json()
            }
        } catch (e) {
            response =  { message: "Error fetching data" };
        }
    }else{
        j = await serverResponse.json()
    }

    response = { data:j };
    const {data, message} = response
    if(message) console.debug(message)
    if(data) console.debug(data)
}
fetchData()
```

Answer:

```
```

```
{
    "id": 26,
    "from": "40f98fcf-38f9-4a08-aea5-831bd5eb8fd8",
    "to": "40f98fcf-38f9-4a08-aea5-831bd5eb8fd8",
    "status": "In process",
    "urgent": false,
    "comments": "A description to add",
    "containers": 2,
    "operation": null
}
```

Update delivery request

Updates the specified values for a delivery request. Only "In process" and "scheduled" delivery request can be edited.

```
PATCH /api/1.0/request/{deliveryRequestId}
```

Body

All parameters are optional

```
containers: <Integer> Number of containers. value between 0 and 10 (kg),
urgent: <Integer> 0 = not urgent, 1 = urgent,
comments: <String> Comments for the delivery,
```

Response

Model: application/json

```
{
       "id": <Integer> the unique id of the drone
       "urgent": <boolean> The request urgency,
       "from": <String> The request base uuid (from),
       "to": <String> The request base uuid (t),
       "comments": <String> Requst comments,
       "status": <String> Requst status ("In process", "Scheduled", "Out for delivery","Completed"),
       "containers": <Integer> Number of containers for the delivery,
       "operation": <String> Sync id corresponding to the operatin manging the delivery
   }
```

TIP

When the delivery request is created the operation is not yet assigned, this value will be present when the deivery request is attached to the operation.

Example

```
const fetchData=async ()=>{
    let j,response;
    const serverResponse = await fetch(`https://cloud.rigi.tech/api/1.0/request/41`, {
        mode: "cors",
        method: "PATCH",
        referrer: "no-referrer",
        headers: {"authorization": "Bearer " + token}
         body: JSON.stringify({
          "comments":"test comments"
      }),
    })
    if(!serverResponse.ok){
        try {
            switch (serverResponse.status){
                case 400:
                     j = await serverResponse.json();
                     break;
                case 403:
                    response =  { message: "Forbidden access to data" };
                case 404:
                    response =  { message: "Address not found" };
                default:
                     j = await serverResponse.json()
            }
        } catch (e) {
            response =  { message: "Error fetching data" };
        }
    }else{
        j = await serverResponse.json()
    }

    response = { data:j };
    const {data, message} = response
    if(message) console.debug(message)
    if(data) console.debug(data)
}
fetchData()
```

Answer:

```
{
    "id": 26,
    "from": "40f98fcf-38f9-4a08-aea5-831bd5eb8fd8",
    "to": "40f98fcf-38f9-4a08-aea5-831bd5eb8fd8",
    "status": "In process",
    "urgent": false,
    "comments": "test comments",
    "containers": 2,
    "operation": null
}
```

Delete delivery request

Delete a delivery request. Only "In process" and "scheduled" delivery request can be deleted.

```
DELETE /api/1.0/request/{deliveryRequestId}
```

Response

Model: application/json

```
{
    "status": <String> Call status
}
```

Example

```
const fetchData=async ()=>{
    let j,response;
    const serverResponse = await fetch(`https://cloud.rigi.tech/api/1.0/request/60`, {
        mode: "cors",
        method: "DELETE",
        referrer: "no-referrer",
        headers: {"authorization": "Bearer " + token}
    })
    if(!serverResponse.ok){
        try {
            switch (serverResponse.status){
                case 400:
                     j = await serverResponse.json();
                     break;
                case 403:
                    response =  { message: "Forbidden access to data" };
                case 404:
                    response =  { message: "Address not found" };
                default:
                     j = await serverResponse.json()
            }
        } catch (e) {
            response =  { message: "Error fetching data" };
        }
    }else{
        j = await serverResponse.json()
```

```
    }

    response = { data:j };
    const {data, message} = response
    if(message) console.debug(message)
    if(data) console.debug(data)
}
fetchData()
```

Answer:

```
{
    "status":  "Request deleted"
}
```

Get safety profile list

Retrieves all available safety profiles for that project.

TIP

You will need the safety profile id from this endpoint to create a new route.

```
GET /api/1.0/safety_profiles?project={projectId}
```

Response

Model: application/json

```
ARRAY
[
    {
        "id": <String> the unique uuid of the safety profile
        "name": <String> Safety profile name,
        "description": <String> Safety profile description,
        "safetySettings": <Object> Object containing the safety settings for the profile,
                         (safety_setting_key: <Object>
                          {label: <String> safety setting label,
                           description: <String> Safety setting description,
                           value: safety setting value,
                           unit: safety setting unit (if applicable)}
                           )
    }
]
```

Query parameters

Some parameters are allowed for a filtered response

CAUTION

parameters with * are mandatory

```
description: <String> - It will return anything containing this string in description
```

Example

```
const fetchData=async ()=>{
    let j,response;
    const serverResponse = await fetch(`https://cloud.rigi.tech/api/1.0/safety_profiles?
project=3&serialNumber=GA`, {
        mode: "cors",
        method: "GET",
        referrer: "no-referrer",
        headers: {"authorization": "Bearer " + token}
    })
    if(!serverResponse.ok){
        try {
            switch (serverResponse.status){
                case 400:
                     j = await serverResponse.json();
                     break;
                case 403:
                    response =  { message: "Forbidden access to data" };
                case 404:
                    response =  { message: "Address not found" };
                default:
                     j = await serverResponse.json()
            }
        } catch (e) {
            response =  { message: "Error fetching data" };
        }
    }else{
        j = await serverResponse.json()
    }

    response = { data:j };
    const {data, message} = response
    if(message) console.debug(message)
    if(data) console.debug(data)
}
fetchData()
```

Answer:


```
[
    {
        "id": "08624f6e-482c-4531-8f89-6dfefc53faae",
        "name": "VLOS",
        "description": "Safety profile used in VLOS flights (demonstrations, trainings and flight
testing).",
        "safetySettings":{"safety_prefloor_warning":{"label": "Lower Deviation Warning", "description":
"Lower Deviation Warning", "value": "false",…}
    },
    {
        "id": "1bf0c3c0-5f51-43e9-9e0a-43e1c55bd314",
        "name": "Default profile",
```

```
        "description": "Default safety profile",
        "safetySettings":{"safety_prefloor_warning":{"label": "Lower Deviation Warning", "description":
"Lower Deviation Warning", "value": "false",…}
    },
    {
        "id": "8c6f08e5-f6d4-440f-ba66-fb97b400bec3",
        "name": "BVLOS",
        "description": "Safety profile for BVLOS operations requiring enhanced containment (scope of the
Design Verification).",
        "safetySettings":{"safety_prefloor_warning":{"label": "Lower Deviation Warning", "description":
"Lower Deviation Warning", "value": "true",…}
    }
]
```

Get settings list

Retrieves all available settings into the specified project

```
GET /api/1.0/settings?project={projectId}
```

Response

Model: application/json

```
ARRAY
[
    {
        "key": <String> the unique key for the setting
        "value": <String> The value for the setting,
        "section": <String> Section where the setting is applied,
        "description": <string> Setting description,
    }
]
```

Query parameters

Some parameters are allowed for a filtered response

> **CAUTION**
> parameters with * are mandatory

```
section: <String> - It will return anything containing this string in section
description: <Integer> - It will return anything containing this string in the description
```

Example

```
const fetchData=async ()=>{
    let j,response;
    const serverResponse = await fetch(`https://cloud.rigi.tech/api/1.0/settings?project=3`, {
        mode: "cors",
        method: "GET",
        referrer: "no-referrer",
        headers: {"authorization": "Bearer " + token}
```

```
    })
    if(!serverResponse.ok){
        try {
            switch (serverResponse.status){
                case 400:
                     j = await serverResponse.json();
                     break;
                case 403:
                    response =  { message: "Forbidden access to data" };
                case 404:
                    response =  { message: "Address not found" };
                default:
                     j = await serverResponse.json()
            }
        } catch (e) {
            response =  { message: "Error fetching data" };
        }
    }else{
        j = await serverResponse.json()
    }

    response = { data:j };
    const {data, message} = response
    if(message) console.debug(message)
    if(data) console.debug(data)
}
fetchData()
```

Answer:


```
[
    {
        "key": "preflight_checklist",
        "value": "3",
        "section": "control",
        "description": ""
    },
    {
        "key": "ground_checklist",
        "value": "13",
        "section": "control",
        "description": ""
    },
    {
        "key": "rigi_map_selected_style",
        "value": "default",
        "section": "map",
        "description": ""
    },
    {
        "key": "rigi_map_selected_style_light",
        "value": "light",
        "section": "map",
        "description": ""
```

```
    },
    {
        "key": "daily_checklist",
        "value": "12",
        "section": "control",
        "description": "Daily checklist configured."
    },
    {
        "key": "vertical_distance_units",
        "value": "m",
        "section": "map",
        "description": "Defines the unit used for vertical distances."
    },
    {
        "key": "short_horizontal_distance_units",
        "value": "m",
        "section": "map",
        "description": "Defines the unit used for (short) horizontal distances."
    },
    {
        "key": "long_horizontal_distance_units",
        "value": "m",
        "section": "map",
        "description": "Defines the unit used for (long) horizontal distances."
    },
    {
        "key": "weight_units",
        "value": "kg",
        "section": "map",
        "description": "Defines the unit used for weights."
    },
    {
        "key": "vertical_speed_units",
        "value": "ms",
        "section": "map",
        "description": "Defines the unit used for vertical speeds."
    },
    {
        "key": "horizontal_speed_units",
        "value": "ms",
        "section": "map",
        "description": "Defines the unit used for horizontal speeds."
    },
    {
        "key": "horizontal_square_distance_units",
        "value": "km",
        "section": "map",
        "description": "Defines the unit used for surfaces."
    },
    {
        "key": "planner_maximum_telemetry_distance",
        "value": "30000",
        "section": "planner",
        "description": null
    },
```

```
    {
        "key": "planner_maximum_way_point_distance",
        "value": "10000",
        "section": "planner",
        "description": null
    },
    {
        "key": "planner_maximum_distance",
        "value": "100000",
        "section": "planner",
        "description": null
    },
    {
        "key": "planner_maximum_climb_angle",
        "value": "6",
        "section": "planner",
        "description": null
    },
    {
        "key": "enable_segment_planner",
        "value": "0",
        "section": "map",
        "description": "Enable segment planner only for admins"
    }
]
```

Create new route

Creates a new route into the specific project

```
POST /api/1.0/route
```

> **CAUTION**
> parameters with * are mandatory

Body

> **CAUTION**
> Follow correctly the flightplan format or the server will reject the plan creation.
>
> you can find the flightplan format in the flightplan-format section

```
* project: <Integer> The project id,
* routename: <String> The new route name,
* flightPlan: <Object> Json object containing the flight plan for the route,
routestate: <String> Route state for the route ("draft", "validated", "authorized"). "Draft" is set if
not specified
```

Response

Model: application/json

```
{
    "status": <Sting> Message with the call status ("accepted", "rejected")
    "route": <Object> Json object containing the route data
              id: <Integer> route id
              name: <String> route name
              routeState: <String> route state ("draft", "validated", "authorized")
              flightPlan: <Object> Json flightPlan
}
```

Example

```
const fetchData=async ()=>{
    let j,response;
    const serverResponse = await fetch(`https://cloud.rigi.tech/api/1.0/route`, {
  "project":"17",
  "routename":"Test route 3",
  "routestate":"validated",
  "flightPlan":{
    "geoFence":{
      "polygons": [
        {
          "type": "geocage",
          "altAmsl": 606,
          "inclusion": "inclusion",
          "vertices": [
            {
              "lat":49.150077591022544,
              "lon":16.825107180165574
            },
            {
              "lat":49.1525792724473,
              "lon":16.82125552802019
            },
            {
              "lat":49.15376165013778,
              "lon":16.81496843008928
            },
            {
              "lat":49.159680131436424,
              "lon":16.797056638287827
            },
            {
              "lat":49.16061853343021,
              "lon":16.78889467625551
            },
            {
              "lat":49.1588066122935,
              "lon":16.781934343861863
            },
            {
              "lat":49.156665723624045,
              "lon":16.763573282050416
            },
            {
              "lat":49.154121708361615,
              "lon":16.75222552626066
            },
            {
              "lat":49.15045478109202,
              "lon":16.749031685876176
            },
            {
              "lat":49.15087034473323,
```

```
      "lon":16.73482838331394
    },
    {
      "lat":49.15204377237618,
      "lon":16.728736080812737
    },
    {
      "lat":49.15747909242239,
      "lon":16.718050998287723
    },
    {
      "lat":49.15871855242119,
      "lon":16.707182896906957
    },
    {
      "lat":49.15592011697326,
      "lon":16.697717433680296
    },
    {
      "lat":49.15532924349978,
      "lon":16.69664389261881
    },
    {
      "lat":49.15410684189552,
      "lon":16.699218155817576
    },
    {
      "lat": 49.15394953255132,
      "lon":16.705954549951926
    },
    {
      "lat":49.152632613814326,
      "lon":16.71185015014045
    },
    {
      "lat":49.1453995376264,
      "lon":16.723716661917493
    },
    {
      "lat":49.144687954965214,
      "lon":16.73247223033242
    },
    {
      "lat":49.14278810311593,
      "lon":16.738223305555508
    },
    {
      "lat":49.14037824223416,
      "lon":16.739596806118712
    },
    {
      "lat":49.140354745397154,
      "lon":16.74912422208719
    },
    {
```

```
            "lat":49.14489181364054,
            "lon":16.757474609302804
          },
          {
            "lat":49.1466213436577,
            "lon":16.76951571612291
          },
          {
            "lat":49.142827966263894,
            "lon":16.777396046208665
          },
          {
            "lat":49.146266933091326,
            "lon":16.795914017247483
          },
          {
            "lat":49.14568092194048,
            "lon":16.80801614432268
          },
          {
            "lat":49.14392284690498,
            "lon":16.810891472386643
          },
          {
            "lat":49.142629692022375,
            "lon":16.81568189768724
          },
          {
            "lat":49.14238228676779,
            "lon":16.821888529347703
          },
          {
            "lat":49.14256827948575,
            "lon":16.825053535985276
          },
          {
            "lat":49.144357985057624,
            "lon":16.82549341826372
          },
          {
            "lat":49.14684942832515,
            "lon":16.82568653731279
          }
        ]
      }]
  },
  "mission": [
    {
      "altAmsl": 350,
      "padAltAmsl": 267,
      "command": 22,
      "lat":49.15108304952246,
      "lon":16.79625748449217

    },
```

```
      {
        "altAmsl": 350,
        "command": 16,
        "lat":49.15052165642498,
        "lon":16.763470161494123

      },
      {
        "altAmsl": 350,
        "command": 3000,
        "lat":49.15052165642498,
        "lon":16.763470161494123,
        "transitionType":"front"

      },
      {
        "altAmsl": 350,
        "command": 16,
        "lat":49.14911814584975,
        "lon":16.754372108515607
      },
      {
        "altAmsl": 350,
        "command": 3000,
        "lat":49.146872446229374,
        "lon":16.745960701044904,
        "transitionType":"back"

      },
      {
        "command": 21,
        "altAmsl": 360,
        "lat":49.147939166240384,
        "lon":16.732141960200178,
        "padAltAmsl": 350
      }
    ],

    "rallyPoints": [
      {
        "altAmsl": 350,
        "lat":49.15916640477572,
        "lon":16.80844544225584
      },
      {
        "lat":49.16489131666681,
        "lon":16.78097962194334
      }
    ]
  }
}
)
    if(!serverResponse.ok){
        try {
            switch (serverResponse.status){
```

```
                case 400:
                     j = await serverResponse.json();
                     break;
                case 403:
                    response =  { message: "Forbidden access to data" };
                case 404:
                    response =  { message: "Address not found" };
                default:
                     j = await serverResponse.json()
            }
        } catch (e) {
            response =  { message: "Error fetching data" };
        }
    }else{
        j = await serverResponse.json()
    }

    response = { data:j };
    const {data, message} = response
    if(message) console.debug(message)
    if(data) console.debug(data)
}
fetchData()
```

Answer:

> **CAUTION**
>
> The flightPlan returned will consist in the flightPlan sent for the creation populated with extra data like extra altitudes, uuid's etc. This is the final plan saved into the system


```
{
"status": true,
    "route":{
        "id": 1955,
        "name": "Test route 3 (3)",
        "routeState": "validated",
        "flightPlan":{"uuid": "fdf47a5f-63bc-48c0-be1f-36ced889ba67", "version": 2, "safetySettings":{},
"safetyProfile": null,…}
    }
}
```

Update route

Updates the specified values for a route.

```
POST /api/1.0/route/{routeId}
```

Body

> **CAUTION**
> Follow correctly the flightplan format or the server will reject the plan creation.
>
> you can find the flightplan format in the flightplan-format section

```
* project: <Integer> The project id,
routename: <String> The new route name,
flightPlan: <Object> Json object containing the flight plan for the route,
routestate: <String> Route state for the route ("draft", "validated", "authorized"). "Draft" is set if
not specified
```

Response

Model: application/json

```
{
    "status": <Sting> Message with the call status ("accepted", "rejected")
    "route": <Object> Json object containing the route data
              id: <Integer> route id
              name: <String> route name
              routeState: <String> route state ("draft", "validated", "authorized")
              flightPlan: <Object> Json flightPlan
}
```

Example

```
const fetchData=async ()=>{
    let j,response;
    const serverResponse = await fetch(`https://cloud.rigi.tech/api/1.0/route/1843`, {
        mode: "cors",
        method: "PATCH",
        referrer: "no-referrer",
        headers: {"authorization": "Bearer " + token}
         body: JSON.stringify({
          "name":"updated route",
      }),
    })
    if(!serverResponse.ok){
        try {
            switch (serverResponse.status){
                case 400:
                     j = await serverResponse.json();
                     break;
                case 403:
                    response =  { message: "Forbidden access to data" };
                case 404:
                    response =  { message: "Address not found" };
                default:
                     j = await serverResponse.json()
            }
        } catch (e) {
            response =  { message: "Error fetching data" };
        }
    }else{
        j = await serverResponse.json()
    }

    response = { data:j };
    const {data, message} = response
    if(message) console.debug(message)
    if(data) console.debug(data)
}
fetchData()
```

Answer:

```
    {
    "status": true,
    "route":{
        "id": 1843,
        "name": "Updated route",
        "routeState": "authorized",
        "flightPlan":{"fileType": "Plan", "safetySettings":{}, "geoFence":{"circles":[],…}
    }
}
```

Get operation list

Retrieves all available operations into the specified project. The operations ar ordered by scheduled time descendent

```
GET /api/1.0/operations?project={projectId}
```

Response

Model: application/json

```
ARRAY
[
    {
        "id": <Integer> the unique operation id
        "syncId": <Integer> unique id used to load the operation into the drone,
        "name": <String> The operation name,
        "takeOffTime": <String> The operation take off time,
        "scheduledTime": <String> The operation scheduled time,
        "comments": <String> The operation comments,
        "status": <Integer> The operation status, //(1:Scheduled ,2:In progress, 3:Delivered,
4:Incident)
        "deliveryTime": <String> The operation delivery time,
    }
]
```

Query parameters

Some parameters are allowed for a filtered response

CAUTION

parameters with * are mandatory

TIP

This endpoint return a maximum of 1000 operations, if it's returning this error try to filter for a shorter range of dates or other attributes

```
* project: <Integer> - Get drones for that project id
name: <String> - It will return operations containing this string in name
from: <String date> - It will return operations older than this date (YYY:MM:DD HH:mm:ss)
to: <String date> - It will return operations previous to this date (YYY:MM:DD HH:mm:ss)
status: <Integer> - It will return simulators or not simulators, (1=Scheduled ,2=In progress,
```

```
3=Delivered, 4=Incident)
drone: <Integer> - It will return operations for this drone id
```

Example

```
const fetchData=async ()=>{
    let j,response;
    const serverResponse = await fetch(`https://cloud.rigi.tech/api/1.0/operations?project=22&from=2021-
01-01 00:00:00&to=2022-01-01 00:00:00&status=1&drone=21&name=test`, {
        mode: "cors",
        method: "GET",
        referrer: "no-referrer",
        headers: {"authorization": "Bearer " + token}
    })
    if(!serverResponse.ok){
        try {
            switch (serverResponse.status){
                case 400:
                     j = await serverResponse.json();
                     break;
                case 403:
                    response =  { message: "Forbidden access to data" };
                case 404:
                    response =  { message: "Address not found" };
                default:
                     j = await serverResponse.json()
            }
        } catch (e) {
            response =  { message: "Error fetching data" };
        }
    }else{
        j = await serverResponse.json()
    }

    response = { data:j };
    const {data, message} = response
    if(message) console.debug(message)
    if(data) console.debug(data)
}
fetchData()
```

Answer:

```
[
    {
        "id": 220,
        "name": "test operation 1",
        "takeOffTime": null,
        "scheduledTime": "2021-11-16T17:47:42.000Z",
        "comments": null,
        "status": 1,
```

```
        "deliveryTime": null
    },
    {
        "id": 235,
        "name": "test operation 2",
        "takeOffTime": null,
        "scheduledTime": "2021-11-22T14:08:54.000Z",
        "comments": null,
        "status": 1,
        "deliveryTime": null
    },
    {
        "id": 296,
        "name": "test operation 3",
        "takeOffTime": null,
        "scheduledTime": "2021-12-16T08:52:47.000Z",
        "comments": null,
        "status": 1,
        "deliveryTime": null
    }
]
```

Get operation information

Retrieves operation information for the specified operation. You can get the operation id listing all the operations first

```
GET /api/1.0/operation/{operationId}
```

Response

Model: application/json

```
{
    "id": <Integer> The unique operation id
    "syncId": <Integer> The unique id for the operation, this id is what matches with the telemetry
    "name": <String> The operation name
    "route": <Integer> The route id,
    "takeOffTime": <String> The operation takeoff time,
    "scheduledTime": <string> The operation scheduled time,
    "comments": <String> The operation comments,
    "status": <Itneger> The operation status, //(1:Scheduled ,2:In progress, 3:Delivered, 4:Incident)
    "deliveryTime": <string> The operation delivery time,
    "drone": <Integer> The drone id
    "batteries": <Array> Array of all the batteries (serial number) for this operation
    "pic": <Integer> Pilot in command (user id)
    "groundOperators": <Array> Array of user id of the operation ground operators
    "payload": <Float> Operation payloag (kg)
    "duration": <String> Operation duration (HH:mm:ss)
    "distance": <Integer> Operation flight distance (m)

}
```

Example

```
const fetchData=async ()=>{
    let j,response;
    const serverResponse = await fetch(`https://cloud.rigi.tech/api/1.0/operation/257`, {
        mode: "cors",
        method: "GET",
        referrer: "no-referrer",
        headers: {"authorization": "Bearer " + token}
    })
    if(!serverResponse.ok){
        try {
```

```
            switch (serverResponse.status){
                case 400:
                     j = await serverResponse.json();
                     break;
                case 403:
                    response =  { message: "Forbidden access to data" };
                case 404:
                    response =  { message: "Address not found" };
                default:
                     j = await serverResponse.json()
            }
        } catch (e) {
            response =  { message: "Error fetching data" };
        }
    }else{
        j = await serverResponse.json()
    }

    response = { data:j };
    const {data, message} = response
    if(message) console.debug(message)
    if(data) console.debug(data)
}
fetchData()
```

Answer:

```
    {
        "id": 257,
        "name": "Test operation",
        "takeOffTime": "2021-11-24T12:00:49.000Z",
        "scheduledTime": "2021-11-24T12:30:21.000Z",
        "comments": null,
        "status": 3,
        "deliveryTime": "2021-11-24T12:38:15.000Z",
        "drone": 29,
        "batteries":["BAT-MT-01-00102", "BAT-MT-01-00103"],
        "pic": 31,
        "groundOperators":[4,6],
        "payload": 1.5,
        "duration": "00:37:26",
        "distance": 52866
    }
```

Get operation flightlog

Retrieves operation flightlog file. The endpoint returns a blob witht the file content.

```
GET /api/1.0/operation/{operationId}/flightlog
```

Response

Model: application/json

```
'application/x-binary'

Blob {
  [Symbol(type)]: 'application/x-binary',
  [Symbol(buffer)]: <Buffer> "Binnary buffer"
}
```

Example

Example retrieving the file and saving it with nodejs

```
    const fetchData=async ()=>{
    let j,response;
    const serverResponse = await fetch(`http://127.0.0.1:1337/api/1.0/operation/145/flightlog`, {
        mode: "cors",
        method: "GET",
        referrer: "no-referrer",
        headers: {"authorization": "Bearer " + token}
    })
    if(!serverResponse.ok){
        try {
            switch (serverResponse.status){
                case 400:
                     j = await serverResponse.json();
                     break;
                case 403:
                    response =  { message: "Forbidden access to data" };
                case 404:
                    response =  { message: "Address not found" };
                default:
                     j = await serverResponse.json()
```

```
            }
        } catch (e) {
            response =  { message: "Error fetching data" };
        }
    }else{
        j = await serverResponse.blob()
    }

    response = { data:j };
    const {data, message} = response
    if(message) console.debug(message)
    if(data) console.debug(data)
    const buffer=await data.text()
    fs.createWriteStream("flightlog.ulg").write(buffer);
}
fetchData()
```

Answer:

The file is created in the specified directory

Create new operation

Creates a new scheduled operation with the specified values. The operation can be updated only if it's in scheduled state

```
POST /api/1.0/operation
```

CAUTION

parameters with * are mandatory

Body

```
* project: <Integer> The project id,
* name: <String> The new operation name,
* scheduledTime: <String date> The operation scheduled time date (YYY-MM-DD HH:mm:ss),
* route: <Integer> The route id,
drone: <Integer> The drone id,
batteries: <Array> Array of all the batteries (serial number) for this operation,
payload: <Float> Float value between 0 and 2 (kg),
pic: <Integer> User id for the piclot in command,
groundOperators: <Array> Array of user id representing the ground operators for the operation
comments: <String> Comments for the operation,
```

Response

Model: application/json

```
{
    "status": <Sting> Message with the call status
    "id": <Integer> the unique id of the new created operation
    "warning": <Array> Array of messages with warnings on the operation creation
    "error": <Array> Array of error messages (only if the creation failed, if the call works this key is
not returned)
}
```

Example

```
const fetchData=async ()=>{
    let j,response;
    const serverResponse = await fetch(`https://cloud.rigi.tech/api/1.0/operation`, {
        mode: "cors",
        method: "POST",
        referrer: "no-referrer",
        headers: {"authorization": "Bearer " + token}
        body: JSON.stringify({
          "project":22,
          "name":"test operation api",
          "scheduledTime":"2022-07-19 11:00:06",
          "route":14,
          "batteries":["BAT-MT-01-00100", "BAT-MT-01-00104"],
          "payload":"1",
          "pic":34,
          "groundOperators":[32,39]
      }),
    })
    if(!serverResponse.ok){
        try {
            switch (serverResponse.status){
                case 400:
                     j = await serverResponse.json();
                     break;
                case 403:
                    response =  { message: "Forbidden access to data" };
                case 404:
                    response =  { message: "Address not found" };
                default:
                     j = await serverResponse.json()
            }
        } catch (e) {
            response =  { message: "Error fetching data" };
        }
    }else{
        j = await serverResponse.json()
    }

    response = { data:j };
    const {data, message} = response
    if(message) console.debug(message)
    if(data) console.debug(data)
}
fetchData()
```

Answer:

```
{
    "status": "Operation created",
    "id": 1042,
    "warining":[
      "There is no drone assigned. You will need to set it before the operation starts"
```

```
    ]
}
```

Update operation

Updates the specified values for a scheduled operation. If the operation is not scheduled an error will be returned

```
PATCH /api/1.0/operation/{operationId}
```

Body

All parameters are optional

```
name: <String> The new operation name,
scheduledTime: <String date> The operation scheduled time date (YYY-MM-DD HH:mm:ss),
drone: <Integer> The drone id,
route: <Integer> The route id,
payload: <Float> Float value between 0 and 2 (kg),
pic: <Integer> User id for the piclot in command,
groundOperators: <Array> Array of user id representing the ground operators for the operation
deliveryRequests: <Array> Array of delivery requests id for the operation
comments: <String> Comments for the operation,
```

Response

Model: application/json

```
{
    "status": <Sting> Message with the call status
    "id": <Integer> The unique id of the updated operation
    "syncId": <String> The unique syncId of the updated operation
}
```

Example

```
const fetchData=async ()=>{
    let j,response;
    const serverResponse = await fetch(`https://cloud.rigi.tech/api/1.0/operation/41`, {
        mode: "cors",
        method: "PATCH",
```

```
        referrer: "no-referrer",
        headers: {"authorization": "Bearer " + token}
         body: JSON.stringify({
          "name":"test operation api",
          "scheduledTime":"2022-07-19 11:00:06",
          "route":14,
          "payload":"1",
          "pic":34,
          "groundOperators":[32,39]
      }),
    })
    if(!serverResponse.ok){
        try {
            switch (serverResponse.status){
                case 400:
                     j = await serverResponse.json();
                     break;
                case 403:
                    response =  { message: "Forbidden access to data" };
                case 404:
                    response =  { message: "Address not found" };
                default:
                     j = await serverResponse.json()
            }
        } catch (e) {
            response =  { message: "Error fetching data" };
        }
    }else{
        j = await serverResponse.json()
    }

    response = { data:j };
    const {data, message} = response
    if(message) console.debug(message)
    if(data) console.debug(data)
}
fetchData()
```

Answer:

```
{
   "status: "Operation updated",
   "id": 41
   "syncId": 16584665126
}
```

Delete oparation

Delete a scheduled operation. If the operation is not scheduled an error will be returned

```
POST /api/1.0/operation/{operationId}
```

Response

Model: application/json

```
{
    "status": <String> Call status
}
```

Example

```
const fetchData=async ()=>{
    let j,response;
    const serverResponse = await fetch(`https://cloud.rigi.tech/api/1.0/operation/60`, {
        mode: "cors",
        method: "DELETE",
        referrer: "no-referrer",
        headers: {"authorization": "Bearer " + token}
    })
    if(!serverResponse.ok){
        try {
            switch (serverResponse.status){
                case 400:
                     j = await serverResponse.json();
                     break;
                case 403:
                    response =  { message: "Forbidden access to data" };
                case 404:
                    response =  { message: "Address not found" };
                default:
                     j = await serverResponse.json()
            }
        } catch (e) {
            response =  { message: "Error fetching data" };
        }
    }else{
        j = await serverResponse.json()
```

```
    }

    response = { data:j };
    const {data, message} = response
    if(message) console.debug(message)
    if(data) console.debug(data)
}
fetchData()
```

Answer:

```
{
    "status":  "Operation deleted"
}
```

Start simulator

This endpoint starts a simulator with the specified Id. The get the id you can list all the simulators first with the get-drones endpoint. Once started the simulator will start sending telemetry to the corresponding topic. The simulator will be stopped automatically after 3 hours. We recommend to stop the simulator after use.

```
POST /api/1.0/simulator/{simulatord}/start
```

Body

The speed parameter is optional

```
lat: <Float> Latitude where to start the simulator,
lon: <Float> Longitude where to start the simulator,
alt: <Float> AMSL Absolute altitude where to start the simulator,
speed: <Integer> Integer value between 1-5,
```

Response

Model: application/json

```
{
    "status": <Boolean> True if the simulator start request is accepted,
}
```

Example

```
const fetchData=async ()=>{
    let j,response;
    const serverResponse = await fetch(`/api/1.0/simulator/41/start`, {
        mode: "cors",
        method: "POST",
        referrer: "no-referrer",
        headers: {"authorization": "Bearer " + token}
         body: JSON.stringify({
          "lat": "46.58589776852092",
          "lon":"6.550499126884768",
```

```
          "alt":"506.2",
          "speed":"1",
      }),
    })
    if(!serverResponse.ok){
        try {
            switch (serverResponse.status){
                case 400:
                     j = await serverResponse.json();
                     break;
                case 403:
                    response =  { message: "Forbidden access to data" };
                case 404:
                    response =  { message: "Address not found" };
                default:
                     j = await serverResponse.json()
            }
        } catch (e) {
            response =  { message: "Error fetching data" };
        }
    }else{
        j = await serverResponse.json()
    }

    response = { data:j };
    const {data, message} = response
    if(message) console.debug(message)
    if(data) console.debug(data)
}
fetchData()
```

Answer:

```
{
   "status: true,
}
```

Stop simulator

This endpoint stops a simulator with the specified Id. The get the id you can list all the simulators first with the get-drones endpoint. The stop request might take some seconds to be processed.

```
POST /api/1.0/simulator/{simulatord}/stop
```

Response

Model: application/json

```
{
    "status": <Boolean> True if the simulator stop request is accepted,
}
```

Example

```
const fetchData=async ()=>{
    let j,response;
    const serverResponse = await fetch(`/api/1.0/simulator/41/stop`, {
        mode: "cors",
        method: "POST",
        referrer: "no-referrer",
        headers: {"authorization": "Bearer " + token}
        body: "",
    })
    if(!serverResponse.ok){
        try {
            switch (serverResponse.status){
                case 400:
                     j = await serverResponse.json();
                     break;
                case 403:
                    response =  { message: "Forbidden access to data" };
                case 404:
                    response =  { message: "Address not found" };
                default:
                     j = await serverResponse.json()
            }
        } catch (e) {
            response =  { message: "Error fetching data" };
        }
```

```
    }else{
        j = await serverResponse.json()
    }

    response = { data:j };
    const {data, message} = response
    if(message) console.debug(message)
    if(data) console.debug(data)
}
fetchData()
```

Answer:

```
{
   "status: true,
}
```

Move simulator

This endpoint moves a running simulator with the specified Id. The get the id you can list all the simulators first with the get-drones endpoint. The action request might take some seconds to be processed. We recommend to use the move endpoint to move the simulator if the drone is not correctly placed before to stop and star it in the correct location, moving the drone will be processed faster.

```
POST /api/1.0/simulator/{simulatord}/move
```

Body

```
lat: <Float> Latitude where to start the simulator,
lon: <Float> Longitude where to start the simulator,
alt: <Float> AMSL Absolute altitude where to start the simulator,
```

Response

Model: application/json

```
{
    "status": <Boolean> True if the simulator start request is accepted,
}
```

Example

```
const fetchData=async ()=>{
    let j,response;
    const serverResponse = await fetch(`/api/1.0/simulator/41/start`, {
        mode: "cors",
        method: "POST",
        referrer: "no-referrer",
        headers: {"authorization": "Bearer " + token}
         body: JSON.stringify({
          "lat": "46.58589776852092",
          "lon":"6.550499126884768",
          "alt":"506.2"
      }),
    })
```

```
    if(!serverResponse.ok){
        try {
            switch (serverResponse.status){
                case 400:
                     j = await serverResponse.json();
                     break;
                case 403:
                    response =  { message: "Forbidden access to data" };
                case 404:
                    response =  { message: "Address not found" };
                default:
                     j = await serverResponse.json()
            }
        } catch (e) {
            response =  { message: "Error fetching data" };
        }
    }else{
        j = await serverResponse.json()
    }

    response = { data:j };
    const {data, message} = response
    if(message) console.debug(message)
    if(data) console.debug(data)
}
fetchData()
```

Answer:

```
{
   "status: true,
}
```

Reset simulator battery

This endpoint resets the battery of a running simulator with the specified Id. The get the id you can list all the simulators first with the get-drones endpoint. Stopping and starting the simulator will also reset its battery, but we recommend to use this endpoint as the processing ime will be shorter.

```
POST /api/1.0/simulator/{simulatord}/reset_battery
```

Response

Model: application/json

```
{
    "status": <Boolean> True if the simulator start request is accepted,
}
```

Example

```
const fetchData=async ()=>{
    let j,response;
    const serverResponse = await fetch(`/api/1.0/simulator/41/reset_battery`, {
        mode: "cors",
        method: "POST",
        referrer: "no-referrer",
        headers: {"authorization": "Bearer " + token}
        body: "",
    })
    if(!serverResponse.ok){
        try {
            switch (serverResponse.status){
                case 400:
                     j = await serverResponse.json();
                     break;
                case 403:
                    response =  { message: "Forbidden access to data" };
                case 404:
                    response =  { message: "Address not found" };
                default:
                     j = await serverResponse.json()
            }
        } catch (e) {
            response =  { message: "Error fetching data" };
```

```
        }
    }else{
        j = await serverResponse.json()
    }

    response = { data:j };
    const {data, message} = response
    if(message) console.debug(message)
    if(data) console.debug(data)
}
fetchData()
```

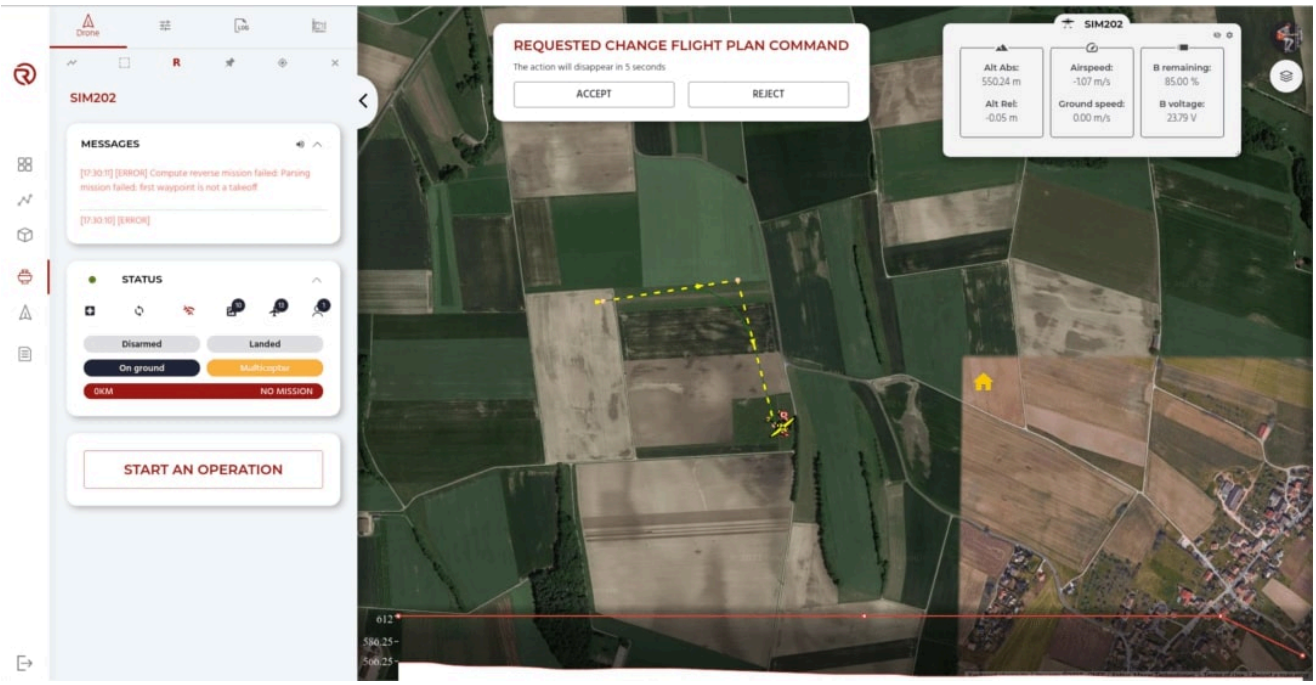
Answer:

```
{
   "status: true,
}
```

Controlling a drone

The API offers a range of REST endpoints to enable basic command control of the drone. However, these commands are not directly executed on the drone and require pilot approval through RigiCloud.

When an API initiates a command request, it triggers a command request notification in RigiCloud while the drone is under control. The notification screen appears as follows:



The pilot is allotted a predefined time frame to either accept or reject the command. This time limit is internally configured and can be customized to suit specific requirements.

In cases where two pilots simultaneously receive the same command request, only the response from the first pilot (whether it's an acceptance or rejection) will be considered valid. The outcome, whether accepted or rejected, will generate a corresponding message transmitted through the relevant message topic.

Start a mission

This endpoint request a start mission to the pilot. The drone needs to have a correctly loaded operation in order to start a new mission.

If the drone is already in a mission and pilot accepts it the drone will send a warning in the status_message topic. If the drone was loitering, this command will resume the mission.

```
POST /api/1.0/drone/{droneId}/startMission
```

TIP

Remember to subscribe to the status_message topic to receive feedback from pilot accept/reject or drone messages

Response

CAUTION

If the drone is not connected to the system, this call will return an error.

```
{status:"Command received, pending for pilot approval"}
```

Example

```
const fetchData=async ()=>{
    let j,response;
    const serverResponse = await fetch(`https://cloud.rigi.tech/api/1.0/drone/41/startMisssion`, {
        mode: "cors",
        method: "POST",
        referrer: "no-referrer",
        headers: {"authorization": "Bearer " + token}
        body: JSON.stringify({}),
    })
    if(!serverResponse.ok){
        try {
            switch (serverResponse.status){
                case 400:
                     j = await serverResponse.json();
                     break;
                case 403:
                    response =  { message: "Forbidden access to data" };
                case 404:
                    response =  { message: "Address not found" };
```

```
                default:
                    j = await serverResponse.json()
            }
        } catch (e) {
            response =  { message: "Error fetching data" };
        }
    }else{
        j = await serverResponse.json()
    }

    response = { data:j };
    const {data, message} = response
    if(message) console.debug(message)
    if(data) console.debug(data)
}
fetchData()
```

Cancel mission start

This endpoint request a cancel start mission to the pilot

This endpoint it's valid only during the start cawntdown. If the operator accept the command after the countdown the start will not be canceled

```
POST /api/1.0/drone/{droneId}/startMissionCancel
```

TIP

Remember to subscribe to the status_message topic to receive feedback from pilot accept/reject or drone messages

Response

CAUTION

If the drone is not connected to the system, this call will return an error.

```
{status:"Command received, pending for pilot approval"}
```

Example

```
const fetchData=async ()=>{
    let j,response;
    const serverResponse = await fetch(`https://cloud.rigi.tech/api/1.0/drone/41/startMissionCancel`, {
        mode: "cors",
        method: "POST",
        referrer: "no-referrer",
        headers: {"authorization": "Bearer " + token}
        body: JSON.stringify({}),
    })
    if(!serverResponse.ok){
        try {
            switch (serverResponse.status){
                case 400:
                     j = await serverResponse.json();
                     break;
                case 403:
                    response =  { message: "Forbidden access to data" };
                case 404:
                    response =  { message: "Address not found" };
```

```
                default:
                    j = await serverResponse.json()
            }
        } catch (e) {
            response =  { message: "Error fetching data" };
        }
    }else{
        j = await serverResponse.json()
    }

    response = { data:j };
    const {data, message} = response
    if(message) console.debug(message)
    if(data) console.debug(data)
}
fetchData()
```

Pause mission (hold)

This endpoint request a pause mission to the pilot

When this is accepted the drone will start loitering(when is in fix wing) or it will have (when is in multicopted mode)

```
POST /api/1.0/drone/{droneId}/pauseMission
```

TIP

Remember to subscribe to the status_message topic to receive feedback from pilot accept/reject or drone messages

Response

CAUTION

If the drone is not connected to the system, this call will return an error.

```
{status:"Command received, pending for pilot approval"}
```

Example

```
const fetchData=async ()=>{
    let j,response;
    const serverResponse = await fetch(`https://cloud.rigi.tech/api/1.0/drone/41/pauseMisssion`, {
        mode: "cors",
        method: "POST",
        referrer: "no-referrer",
        headers: {"authorization": "Bearer " + token}
        body: JSON.stringify({}),
    })
    if(!serverResponse.ok){
        try {
            switch (serverResponse.status){
                case 400:
                     j = await serverResponse.json();
                     break;
                case 403:
                    response =  { message: "Forbidden access to data" };
                case 404:
                    response =  { message: "Address not found" };
                default:
```

```
                    j = await serverResponse.json()
            }
        } catch (e) {
            response =  { message: "Error fetching data" };
        }
    }else{
        j = await serverResponse.json()
    }

    response = { data:j };
    const {data, message} = response
    if(message) console.debug(message)
    if(data) console.debug(data)
}
fetchData()
```

Continue operation

This endpoint request a start continueOperation to the pilot

This action will send a continue operation command. If the drone is in a different mode than the nominal mission, the continue operation will switch the drone to the original mission mode and will continue the mission from the current position.

```
POST /api/1.0/drone/{droneId}/continueOperation
```

TIP

Remember to subscribe to the status_message topic to receive feedback from pilot accept/reject or drone messages

Response

CAUTION

If the drone is not connected to the system, this call will return an error.

```
{status:"Command received, pending for pilot approval"}
```

Example

```
const fetchData=async ()=>{
    let j,response;
    const serverResponse = await fetch(`https://cloud.rigi.tech/api/1.0/drone/41/continueOperation`, {
        mode: "cors",
        method: "POST",
        referrer: "no-referrer",
        headers: {"authorization": "Bearer " + token}
        body: JSON.stringify({}),
    })
    if(!serverResponse.ok){
        try {
            switch (serverResponse.status){
                case 400:
                     j = await serverResponse.json();
                     break;
                case 403:
                    response =  { message: "Forbidden access to data" };
                case 404:
                    response =  { message: "Address not found" };
```

```
                default:
                    j = await serverResponse.json()
            }
        } catch (e) {
            response =  { message: "Error fetching data" };
        }
    }else{
        j = await serverResponse.json()
    }

    response = { data:j };
    const {data, message} = response
    if(message) console.debug(message)
    if(data) console.debug(data)
}
fetchData()
```

Return to takeoff

This endpoint request a return to home to the pilot

When this is accepted the drone will start the reverted mission from the current position

```
POST /api/1.0/drone/{droneId}/returnToHome
```

TIP

Remember to subscribe to the status_message topic to receive feedback from pilot accept/reject or drone messages

Response

CAUTION

If the drone is not connected to the system, this call will return an error.

```
{status:"Command received, pending for pilot approval"}
```

Example

```
const fetchData=async ()=>{
    let j,response;
    const serverResponse = await fetch(`https://cloud.rigi.tech/api/1.0/drone/41/returnToHome`, {
        mode: "cors",
        method: "POST",
        referrer: "no-referrer",
        headers: {"authorization": "Bearer " + token}
        body: JSON.stringify({}),
    })
    if(!serverResponse.ok){
        try {
            switch (serverResponse.status){
                case 400:
                     j = await serverResponse.json();
                     break;
                case 403:
                    response =  { message: "Forbidden access to data" };
                case 404:
                    response =  { message: "Address not found" };
                default:
```

```
                    j = await serverResponse.json()
            }
        } catch (e) {
            response =  { message: "Error fetching data" };
        }
    }else{
        j = await serverResponse.json()
    }

    response = { data:j };
    const {data, message} = response
    if(message) console.debug(message)
    if(data) console.debug(data)
}
fetchData()
```

Land

This endpoint request a land in place to the pilot

When this is accepted the drone will land at the current position. If the drone is in fix wing it will transition to multicopter to land

```
POST /api/1.0/drone/{droneId}/land
```

TIP

Remember to subscribe to the status_message topic to receive feedback from pilot accept/reject or drone messages

Response

CAUTION

If the drone is not connected to the system, this call will return an error.

```
{status:"Command received, pending for pilot approval"}
```

Example

```
const fetchData=async ()=>{
    let j,response;
    const serverResponse = await fetch(`https://cloud.rigi.tech/api/1.0/drone/41/land`, {
        mode: "cors",
        method: "POST",
        referrer: "no-referrer",
        headers: {"authorization": "Bearer " + token}
        body: JSON.stringify({}),
    })
    if(!serverResponse.ok){
        try {
            switch (serverResponse.status){
                case 400:
                     j = await serverResponse.json();
                     break;
                case 403:
                    response =  { message: "Forbidden access to data" };
                case 404:
                    response =  { message: "Address not found" };
                default:
```

```
                    j = await serverResponse.json()
            }
        } catch (e) {
            response =  { message: "Error fetching data" };
        }
    }else{
        j = await serverResponse.json()
    }

    response = { data:j };
    const {data, message} = response
    if(message) console.debug(message)
    if(data) console.debug(data)
}
fetchData()
```

Kill

This endpoint request a kill to the pilot

When this is accepted the drone will be killed (launching a parachute)

```
POST /api/1.0/drone/{droneId}/kill
```

TIP

Remember to subscribe to the status_message topic to receive feedback from pilot accept/reject or drone messages

Response

CAUTION

If the drone is not connected to the system, this call will return an error

```
{status:"Command received, pending for pilot approval"}
```

Example

```
const fetchData=async ()=>{
    let j,response;
    const serverResponse = await fetch(`https://cloud.rigi.tech/api/1.0/drone/41/land`, {
        mode: "cors",
        method: "POST",
        referrer: "no-referrer",
        headers: {"authorization": "Bearer " + token}
        body: JSON.stringify({}),
    })
    if(!serverResponse.ok){
        try {
            switch (serverResponse.status){
                case 400:
                     j = await serverResponse.json();
                     break;
                case 403:
                    response =  { message: "Forbidden access to data" };
                case 404:
                    response =  { message: "Address not found" };
                default:
```

```
                    j = await serverResponse.json()
            }
        } catch (e) {
            response =  { message: "Error fetching data" };
        }
    }else{
        j = await serverResponse.json()
    }

    response = { data:j };
    const {data, message} = response
    if(message) console.debug(message)
    if(data) console.debug(data)
}
fetchData()
```

Upload operation

This endpoint request to the operator to upload a new operation. The operation will be uploaded to the operator and will be available to be accepted or rejected.

This endpoint can be called only when the drone is landed or it will be rejected

```
POST /api/1.0/drone/{droneId}/uploadOperation
```

TIP

Remember to subscribe to the status_message topic to receive feedback from pilot accept/reject or drone messages

Response

CAUTION

If the drone is not connected to the system, this call will return an error

```
{status:"Command received, pending for pilot approval"}
```

Example

```
const fetchData=async ()=>{
    let j,response;
    const serverResponse = await fetch(`https://cloud.rigi.tech/api/1.0/drone/41/uploadOperation`, {
        mode: "cors",
        method: "POST",
        referrer: "no-referrer",
        headers: {"authorization": "Bearer " + token}
        body: JSON.stringify({opId: 1961}),
    })
    if(!serverResponse.ok){
        try {
            switch (serverResponse.status){
                case 400:
                     j = await serverResponse.json();
                     break;
                case 403:
                    response =  { message: "Forbidden access to data" };
                case 404:
                    response =  { message: "Address not found" };
```

```
                default:
                    j = await serverResponse.json()
            }
        } catch (e) {
            response =  { message: "Error fetching data" };
        }
    }else{
        j = await serverResponse.json()
    }

    response = { data:j };
    const {data, message} = response
    if(message) console.debug(message)
    if(data) console.debug(data)
}
fetchData()
```

Go to target

This endpoint sends a "Go to" command to the drone to navigate to a specific point.

When accepted, the drone will fly to the specified coordinates.

```
POST /api/1.0/drone/{droneId}/goTo
```

TIP

Remember to subscribe to the status_message topic to receive feedback from pilot accept/reject decisions or drone messages.

Body

The expected body is a stringified JSON with the following keys:

```
{
    latitude: number,      // Latitude of the target point in decimal degrees
    longitude: number,     // Longitude of the target point in decimal degrees
    altitude: number,      // Altitude of the target point in meters above sea level
    token?: string         // (Optional) Validation token from a previous go to command
}
```

Response

The command will return a status message indicating that the command is pending pilot approval, or a rejection depending on certain validations:

CAUTION

The system performs the following checks and will throw errors if they fail:

- Minimum clearance of 20m
- Target position is within the allowed geofence

The system will also return a validation token that can be used to send subsequent go to commands without pilot approval. This token will expire if another go to command is accepted, when the drone changes its mode by an operator, or after one hour.

```
{
    "status": "Command received, pending for pilot approval",
```

```
    "token": "<validation_token>"
}
```

Example

```
const fetchData = async () => {
    let response;
    try {
        const serverResponse = await fetch(`https://cloud.rigi.tech/api/1.0/drone/41/goTo`, {
            mode: "cors",
            method: "POST",
            referrer: "no-referrer",
            headers: {"authorization": "Bearer " + token},
            body: JSON.stringify({
                latitude: 40.4168,
                longitude: -3.7038,
                altitude: 100,
                token: "<validation_token>"
            }),
        });

        if (!serverResponse.ok) {
            switch (serverResponse.status) {
                case 400:
                    response = await serverResponse.json();
                    break;
                case 403:
                    response = { message: "Forbidden access to data" };
                    break;
                case 404:
                    response = { message: "Address not found" };
                    break;
                default:
                    response = await serverResponse.json();
            }
        } else {
            response = await serverResponse.json();
        }

        console.debug(response);
    } catch (e) {
        console.debug("Error fetching data:", e);
    }
}
fetchData();
```

Validate go to target

This endpoint performs the validations for a previously sent go to command. These validations are the same as those executed for the go to command, but in this case it only tests the validations without executing the command.

```
POST /api/1.0/drone/{droneId}/validate_goTo
```

TIP

Remember to subscribe to the status_message topic to receive feedback from pilot accept/reject decisions or drone messages.

Body

The expected body is a stringified JSON with the following keys:

```
{
  latitude: number,      // Latitude of the target point in decimal degrees
  longitude: number,     // Longitude of the target point in decimal degrees
  altitude: number       // Altitude of the target point in meters above sea level
}
```

Response

The command will return a status message indicating that the target point is accepted, or a rejection depending on certain validations:

CAUTION

The system performs the following checks and will throw errors if they fail:

- Minimum clearance of 20m
- Target position is within the allowed geofence

```
{
    "status": "rejected",
    "errors": "The errors encountered during validation"
}
```

Example

```javascript
const fetchData = async () => {
    let response;
    try {
        const serverResponse = await fetch(`https://cloud.rigi.tech/api/1.0/drone/41/validate_goTo`, {
            mode: "cors",
            method: "POST",
            referrer: "no-referrer",
            headers: {"authorization": "Bearer " + token},
            body: JSON.stringify({
                latitude: 40.4168,
                longitude: -3.7038,
                altitude: 100,
            }),
        });

        if (!serverResponse.ok) {
            switch (serverResponse.status) {
                case 400:
                    response = await serverResponse.json();
                    break;
                case 403:
                    response = { message: "Forbidden access to data" };
                    break;
                case 404:
                    response = { message: "Address not found" };
                    break;
                default:
                    response = await serverResponse.json();
            }
        } else {
            response = await serverResponse.json();
        }

        console.debug(response);
    } catch (e) {
        console.debug("Error fetching data:", e);
    }
}
fetchData();
```

Connection to the socket

To connect to the socket, you'll need a security token. Instructions for obtaining this token can be found in the 'Getting Started' section.

Our subscription service operates through the Socket.io library, which leverages websockets as a transport protocol. While it's possible to connect using plain websockets, please be aware that connecting with plain websockets would require you to implement a ping-pong mechanism to avoid potential disconnections. We strongly recommend utilizing a Socket.io library for a smoother experience. Not all socket.io client libraries will work as expected, the server uses verion 4.6, check first compatibilities with the selected client library.

It's important to note that not all Socket.io client libraries will function as expected. Our server utilizes version 4.6, so we advise you to first check for compatibility with your chosen client library to ensure a seamless connection."

Javascript - client versioon 4.x https://socket.io/docs/v4/client-api/

Python - client versioon 4.x https://python-socketio.readthedocs.io/en/latest/client.html

Step 1: Connect to the socket

```
SOCKET URL: https://cloud.rigi.tech
```

Js example:

```
const socketIOClient = require("socket.io-client");
var io = socketIOClient;
const url="https://cloud.rigi.tech";
const
token="eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJjcmVhdGVkQXQiOjE1NTgwMTc0MTkzNzQsInVwZGF0ZWRBdCI6MTU1ODAxNz
iNy3o_RG5Fzwqu80tNzVltKddbB09RTx9aEOTaeI"
const socket = io(url+"?token="+token, {
    transports: ['websocket']
});
socket.on('connect', (s) => {
    console.debug("connected")
});
```

Step 2: Using the socket

Once you have successfully established a connection, you can begin subscribing to topics to receive real-time drone data and send commands to the drone. Detailed information on these actions can be found in the following sections.

Telemetry topic

Subscribing to the telemetry topic a JSON with drone telemetry will be received every 0.5s

```
TOPIC telemetry
```

Data reeived:

```
{
  lat: <latitude>,
  lon: <longitude>,
  heading:  <heading in degree>,
  drone:  <drone_id>,
  flightId: <flight_id>,
  battery: <remaining_battery>, //percentage
  timestamp: <timestamp in ms>,
  altitudeAbs: <Absolute altitude amsl in m>,
  altitudeRel: <Relative altitude in m>,
  inAir: <Drone in air>, //boolean
  mode: <Flight mode>,
  state: <Drone state>,
  missionDistance: <Mission distance in m>, //Distane flown for the mission
  missionTime: <Mission time>,
  airspeed: <Aispeed in m/s>,
  groundSpeed: <Groundspeed in m/s>,
  pitch: <pitch in degrees>,
  roll: <roll in degrees>,
  yaw: <yaw in degrees>,
  rocd: <Rocd in m/s>,
  windSpeed: <Wind speed in m/s>,
  windDirection: <Wind direction in degrees>, //direction into which the wind is blowing. 0° means
North, 90° means East
}
```

Js example:

```
 socket.on("telemetry", data => {
   console.debug("tlemetry received", data);
 }
```

Download plan

Subscribing to the download_plan topic a JSON with a complete flight plan will be received every time there is any update on the current drone flight plan

The flightplan follows a JSON specification compatible with https://dev.qgroundcontrol.com/master/en/file_formats/plan.html

```
TOPIC download_plan
```

```
{
  response: <flight_plan>,  //Stringified JSON
  id: <drone_id>
}
```

Js example:

```
socket.on("download_plan", data => {
  console.debug("download_plan", data);
}
```

Status message

Subscribing to the status_message topic a JSON with drone status messages will be received. Subscribing to this topic is important to understand wht's going on in the drone, specially if controling the drone through the API as is the publication channel for feedback on accepted or rejected commands by the operator

```
TOPIC status_message
```

```
{
  TYPE_MESSAGE_ID: <Type>, //integer (0: debug, 1: info, 2: warning, 3: error, 4: fatal)
  STATUS_TEXT: <message>, //string
  SYS_ID: <drone_id>, //integer
  TIMESTAMP: <timestamp>, //timestamp
}
```

Js example:

```
 socket.on("status_message", data => {
   console.debug("status_message", data);
 }
```

Controlling a drone

When the drone is connected to the socket, you can perform various actions to control it. These actions are sent directly to the drone and do not necessitate operator validation.

> **CAUTION**
> To use any action you need first to connect to the server. Please refer to the connect action to see how to connect to the socket.

Please be aware that any actions sent using this method will be the responsibility of the customer. While the API does provide methods to create and execute missions, along with some validation for these actions, the ultimate responsibility lies with the user. For a safer method, we recommend using the REST API for drone control, which involves human validation to control the drone.

Start a mission

> **CAUTION**
> To use any action you need first to connect to the server. Please refer to the connect action to see how to connect to the socket.

This action will send a mission start command to the drone. If the drone met the correct conditions a cowntdown will start and the drone will start the mission.

```
"/action/mission_start"
```

Body

```
rigiId: serial number of the drone/simulator
```

> **TIP**
> The rigiId (serial Number) can be obtained from the Rest api. You can list all the drones with the /drones endpoint and get the serial number from there.

Response

> **CAUTION**
> There is no direct response from the action. Any feedback from the drone will be sent as a status_message

Example

```
socket.emit("/action/mission_start", { rigiId: "SIM_223" });
```

Cancel mission start

CAUTION

To use any action you need first to connect to the server. Please refer to the connect action to see how to connect to the socket.

Once you have sent a command to start a mission, the drone will start a countdown. You can cancel the mission start during this countdown using this command.

```
"/action/mission_start_cancel"
```

Body

```
rigiId: serial number of the drone/simulator
```

TIP

The rigiId (serial Number) can be obtained from the Rest api. You can list all the drones with the /drones endpoint and get the serial number from there.

Response

CAUTION

There is no direct response from the action. Any feedback from the drone will be sent as a status_message

Example

```
socket.emit("/action/mission_start_cancel", { rigiId: "SIM_223" });
```

Pause mission (hold)

CAUTION

To use any action you need first to connect to the server. Please refer to the connect action to see how to connect to the socket.

This action will send a hold command to the drone. The drone will stop the mission and will hold its position by loitering if it's in fix wing mode or by hovering if it's in multirotor mode.

```
"/action/pause_mission"
```

Body

```
rigiId: serial number of the drone/simulator
```

TIP

The rigiId (serial Number) can be obtained from the Rest api. You can list all the drones with the /drones endpoint and get the serial number from there.

Response

CAUTION

There is no direct response from the action. Any feedback from the drone will be sent as a status_message

Example

```
socket.emit("/action/pause_mission", { rigiId: "SIM_223" });
```

Continue operation

CAUTION

To use any action you need first to connect to the server. Please refer to the connect action to see how to connect to the socket.

This action will send a continue operation command. If the drone is in a different mode than the nominal mission, the continue operation will switch the drone to the original mission mode and will continue the mission from the current position.

```
"/action/continue_operation"
```

Body

```
rigiId: serial number of the drone/simulator
```

TIP

The rigiId (serial Number) can be obtained from the Rest api. You can list all the drones with the /drones endpoint and get the serial number from there.

Response

CAUTION

There is no direct response from the action. Any feedback from the drone will be sent as a status_message

Example

```
socket.emit("/action/continue_operation", { rigiId: "SIM_223" });
```

Return to takeoff

CAUTION

To use any action you need first to connect to the server. Please refer to the connect action to see how to connect to the socket.

This action will a return to takeoff command to the drone. The drone will go back to the takeoff position with a reversed from the original mission.

```
"/action/reverse_misson_rtl"
```

Body

```
rigiId: serial number of the drone/simulator
```

TIP

The rigiId (serial Number) can be obtained from the Rest api. You can list all the drones with the /drones endpoint and get the serial number from there.

Response

CAUTION

There is no direct response from the action. Any feedback from the drone will be sent as a status_message

Example

```
socket.emit("/action/reverse_misson_rtl", { rigiId: "SIM_223" });
```

Land

CAUTION

To use any action you need first to connect to the server. Please refer to the connect action to see how to connect to the socket.

This action will send a land command to the drone. The drone will land at the current position.

```
"/action/land"
```

Body

```
rigiId: serial number of the drone/simulator
```

TIP

The rigiId (serial Number) can be obtained from the Rest api. You can list all the drones with the /drones endpoint and get the serial number from there.

Response

CAUTION

There is no direct response from the action. Any feedback from the drone will be sent as a status_message

Example

```
socket.emit("/action/land", { rigiId: "SIM_223" });
```

Kill

> **CAUTION**
> To use any action you need first to connect to the server. Please refer to the connect action to see how to connect to the socket.

This action will send a kill command to the drone. The drone will be killed displaying the parachute or doing the configured kill action.

```
"/action/kill"
```

Body

```
rigiId: serial number of the drone/simulator
```

> **TIP**
> The rigiId (serial Number) can be obtained from the Rest api. You can list all the drones with the /drones endpoint and get the serial number from there.

Response

> **CAUTION**
> There is no direct response from the action. Any feedback from the drone will be sent as a status_message

Example

```
socket.emit("/action/kill", { rigiId: "SIM_223" });
```

Upload an operation

CAUTION

To use any action you need first to connect to the server. Please refer to the connect action to see how to connect to the socket.

This action will upload an operation to the drone.You can obtain the operation ID by using the REST API.

CAUTION

The operation id is the field syncId comming from the API. You need to use this id here and not the field called id from the REST api.

```
"/action/upload_mission"
```

Body

```
rigiId: serial number of the drone/simulator
opId: syncId of the operation to be uploaded
```

TIP

The rigiId (serial Number) can be obtained from the Rest api. You can list all the drones with the /drones endpoint and get the serial number from there.

Response

CAUTION

There is no direct response from the action. Any feedback from the drone will be sent as a status_message

Example

```
socket.emit("/action/upload_mission", { rigiId: "SIM_223", opId: opId });
```

Connection to the socket

To connect to the socket, you'll need a security token. Instructions for obtaining this token can be found in the 'Getting Started' section.

Our subscription service operates through the Socket.io library, which leverages websockets as a transport protocol. While it's possible to connect using plain websockets, please be aware that connecting with plain websockets would require you to implement a ping-pong mechanism to avoid potential disconnections. We strongly recommend utilizing a Socket.io library for a smoother experience. Not all socket.io client libraries will work as expected, the server uses verion 4.6, check first compatibilities with the selected client library.

It's important to note that not all Socket.io client libraries will function as expected. Our server utilizes version 4.6, so we advise you to first check for compatibility with your chosen client library to ensure a seamless connection."

Javascript - client versioon 4.x https://socket.io/docs/v4/client-api/

Python - client versioon 4.x https://python-socketio.readthedocs.io/en/latest/client.html

Step 1: Connect to the socket

```
SOCKET URL: https://cloud.rigi.tech
```

Js example:

```
const socketIOClient = require("socket.io-client");
var io = socketIOClient;
const url="https://cloud.rigi.tech";
const
token="eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJjcmVhdGVkQXQiOjE1NTgwMTc0MTkzNzQsInVwZGF0ZWRBdCI6MTU1ODAxNz
iNy3o_RG5Fzwqu80tNzVltKddbB09RTx9aEOTaeI"
const socket = io("https://cloud.rigi.tech:3000",{
    transports: ['websocket'],
    auth: {
        token: token,
    },
});
socket.on('connect', (s) => {
    console.debug("connected")
     //We set the boundries or the returning value will be an ampty array
     s.emit("boundries", {"latBetween":[46.05591073938665,46.81869159487271],"lngBetween":[4.0324613616162
});
```

Step 2: Using the socket

CAUTION

> After a connection is correctly stablished you need to subscribe to the socket but also set the boundries from where you want to get data. If the boundries are not set the result will be an empty array

Once you have successfully established a connection, you can begin subscribing to topics to receive real-time drone data and send commands to the drone. Detailed information on these actions can be found in the following sections.

Live traffic topic

Subscribing to the "live-traffic" topic a JSON with live traffic telemetry will be received every 0.5s

```
TOPIC live-traffic
```

CAUTION

After a connection is correctly established you need to subscribe to the socket but also set the boundaries from where you want to get data. If the boundaries are not set the result will be an empty array

Data reeived:

```
ARRAY
[
    {
     altitude: 10363.2,
     heading: 208.13,
     hor_velocity: 176.76,
     icao: "300393",
     latitude: 46.913576287738344,
     longitude: 11.007103465708497,
     onGround: false,
     rigisource: "involi",
     source: "transponder",
     timestamp: 1698941968579,
     vel_velocity: -2.6
    }
]
```

Js example:

```
socket.on("live-traffic", data => {
  console.debug("live-traffic", data);
}
```

Filter the air traffic boundaries

With this action you can set the boundaries from where you want the air traffic

By default, once connected to the topic, the boundaries are not set

```
"boundaries"
```

Body

```
{
  "latBetween": ARRAY [lat1,lat2]
  "lngBetween": ARRAY [lng1,lng2]

}
```

Response

> **CAUTION**
> There is no direct response from the action. Once sent the topic will start filtering depending on the filters sent.

Example

```
 socket.emit("boundries", {"latBetween":[46.05591073938665,46.81869159487271],"lngBetween":
[4.032461361616266,8.446218685835015]});
```

Filter the air traffic source

With this action you can filter what is received in the live-traffic topic

By default, once connected to the topic, all the traffic is detected from all the sources.

```
"filters"
```

Body

```
ARARY of strings
The possible valies are "involi", "rigitech" or "all"

involi - All traffic comming from involi socket
rigitech - All traffic detected by rigitech receivers in drones
```

Response

> **CAUTION**
> There is no direct response from the action. Once sent the topic will start filtering depending on the filters sent.

Example

```
socket.emit("filters", ["involi","rigitech"]);
```

Flightplan format

The flightplan consist in a JSON with all the needed information. This page describes this format. Not all the keys are needed in order to create a new flightplan, some of the fields are automatically filled by the server

Flightplan specification

The flight plan specification is defined using a JSON Schema. For more details, please refer to https://json-schema.org/.

It's important to note that there are primarily two distinct formats outlined on this page. One format is used for creating plans, and it slightly differs from the format used for retrieving plans from the server. While the variations are minimal, it is crucial to pay close attention to the details.

Full plan creation

The full plan schema provided here is essential for creating a route or making deviations to an existing plan. Please be aware that the JSON schema presented is not an example but a template for your flight plan. An actual JSON example can be found at the end of this document.

```
{
    id: "/simpleflightplan",
    type: "object",
    safetyProfile: { type: "string" },  #this is the uuid of the safety proffile to be applied to the
route. If no safety profile is added the default settings will be applied
    properties: {
      mission: {
        type: "array",
        items: { $ref: "/simpleWaypoint" },
        minItems: 4,
      },
      geoFence: {
        type: "object",
        properties: {
          polygons: {
            type: "array",
            items: {
              type: "object",
              properties: {
                inclusion: { type: "string", enum: ["inclusion", "exclusion"] },
                type: { type: "string", enum: ["gournd_buffer", "geocage", "pregeocage", "polygon"] },
                altAmsl: { type: "number", minimum: -100 },
                vertices: { $ref: "/polygon" },
              },
            },
          },
        },
      },
      rallyPoints: {
```

```
        type: "array",
        items: { $ref: "/simpleRallypoint" },
      },
    },
    required: ["mission"],
  }
```

Waypoint

This specification corresponds to a waypoint and it's referenced from the full plan

> **CAUTION**
> Notice that the commands are just constants. You need to send the number of the command, not the string

```
MAV_CMD_NAV_TAKEOFF = 22
MAV_CMD_DO_VTOL_TRANSITION = 3000
MAV_CMD_NAV_WAYPOINT = 16
MAV_CMD_NAV_LAND = 21

{
    id: "/simpleWaypoint",
    type: "object",
    properties: {
      command: { type: "number", enum: [MAV_CMD_NAV_TAKEOFF, MAV_CMD_DO_VTOL_TRANSITION,
MAV_CMD_NAV_WAYPOINT, MAV_CMD_NAV_LAND] },
      altAmsl: { type: "number", minimum: -100 },
      padAltAmsl: { type: "number", minimum: -100 },
      lat: { type: "number", minimum: -90, maximum: 90 },
      lon: { type: "number", minimum: -180, maximum: 180 },
      groundAltitude: { type: "number", minimum: -100 },
      transitionType: { type: "string", enum: ["front", "back"] },
      precision: { type: "number", enum: [1, 0] },
    },
    required: ["command", "lat", "lon", "altAmsl"],
    allOf: [
      {
        if: {
          properties: { command: { const: MAV_CMD_DO_VTOL_TRANSITION } },
        },
        then: {
          required: ["transitionType"],
        },
      },
    ],
}
```

polygon

This specification corresponds to a geocage polygon and it's referenced from the full plan

```
{
    id: "/polygon",
    type: "array",
    minItems: 3,
    items: {
      type: "object",
      properties: {
        lat: { type: "number", minimum: -90, maximum: 90 },
        lon: { type: "number", minimum: -180, maximum: 180 },
      },
      required: ["lat", "lon"],
    },
},
```

Rallypoint

This specification corresponds to a rally point polygon and it's referenced from the full plan

```
{
    id: "/simpleRallypoint",
    type: "object",
    properties: {
      approachAltAmsl: { type: "number", minimum: -100 },
      altAmsl: { type: "number", minimum: -100 },
      lat: { type: "number", minimum: -90, maximum: 90 },
      lon: { type: "number", minimum: -180, maximum: 180 },
    },
    required: ["lat", "lon"],
  }
```

Example of a plan to request a route creation

```
{
    "geoFence":{
      "polygons": [
        {
          "type": "geocage",
          "altAmsl": 606,
          "inclusion": "inclusion",
          "vertices": [
            {
              "lat":49.150077591022544,
              "lon":16.825107180165574
            },
            {
              "lat":49.1525792724473,
              "lon":16.82125552802019
            },
            {
              "lat":49.15376165013778,
              "lon":16.81496843008928
```

```
            },
            {
              "lat":49.159680131436424,
              "lon":16.797056638287827
            },
            {
              "lat":49.16061853343021,
              "lon":16.78889467625551
            },
            {
              "lat":49.1588066122935,
              "lon":16.781934343861863
            },
            {
              "lat":49.156665723624045,
              "lon":16.763573282050416
            },
            {
              "lat":49.154121708361615,
              "lon":16.75222552626066
            },
            {
              "lat":49.15045478109202,
              "lon":16.749031685876176
            },
            {
              "lat":49.15087034473323,
              "lon":16.73482838331394
            },
            {
              "lat":49.15204377237618,
              "lon":16.728736080812737
            },
            {
              "lat":49.15747909242239,
              "lon":16.718050998287723
            },
            {
              "lat":49.15871855242119,
              "lon":16.707182896906957
            },
            {
              "lat":49.15592011697326,
              "lon":16.697717433680296
            },
            {
              "lat":49.15532924349978,
              "lon":16.69664389261881
            },
            {
              "lat":49.15410684189552,
              "lon":16.699218155817576
            },
            {
              "lat": 49.15394953255132,
```

```
      "lon":16.705954549951926
    },
    {
      "lat":49.152632613814326,
      "lon":16.71185015014045
    },
    {
      "lat":49.1453995376264,
      "lon":16.723716661917493
    },
    {
      "lat":49.144687954965214,
      "lon":16.73247223033242
    },
    {
      "lat":49.14278810311593,
      "lon":16.738223305555508
    },
    {
      "lat":49.14037824223416,
      "lon":16.739596806118712
    },
    {
      "lat":49.140354745397154,
      "lon":16.74912422208719
    },
    {
      "lat":49.14489181364054,
      "lon":16.757474609302804
    },
    {
      "lat":49.1466213436577,
      "lon":16.76951571612291
    },
    {
      "lat":49.142827966263894,
      "lon":16.777396046208665
    },
    {
      "lat":49.146266933091326,
      "lon":16.795914017247483
    },
    {
      "lat":49.14568092194048,
      "lon":16.80801614432268
    },
    {
      "lat":49.14392284690498,
      "lon":16.810891472386643
    },
    {
      "lat":49.142629692022375,
      "lon":16.81568189768724
    },
    {
```

```
              "lat":49.14238228676779,
              "lon":16.821888529347703
            },
            {
              "lat":49.14256827948575,
              "lon":16.825053535985276
            },
            {
              "lat":49.144357985057624,
              "lon":16.82549341826372
            },
            {
              "lat":49.14684942832515,
              "lon":16.82568653731279
            }
          ]
        }]
    },
    "mission": [
      {
        "altAmsl": 350,
        "padAltAmsl": 267,
        "command": 22,
        "lat":49.15108304952246,
        "lon":16.79625748449217

      },
      {
        "altAmsl": 350,
        "command": 16,
        "lat":49.15052165642498,
        "lon":16.763470161494123

      },
      {
        "altAmsl": 350,
        "command": 3000,
        "lat":49.15052165642498,
        "lon":16.763470161494123,
        "transitionType":"front"

      },
      {
        "altAmsl": 350,
        "command": 16,
        "lat":49.14911814584975,
        "lon":16.754372108515607
      },
      {
        "altAmsl": 350,
        "command": 3000,
        "lat":49.146872446229374,
        "lon":16.745960701044904,
        "transitionType":"back"
```

```
      },
      {
        "command": 21,
        "altAmsl": 360,
        "lat":49.147939166240384,
        "lon":16.732141960200178,
        "padAltAmsl": 350
      }
    ],

    "rallyPoints": [
      {
        "altAmsl": 350,
        "lat":49.15916640477572,
        "lon":16.80844544225584
      },
      {
        "lat":49.16489131666681,
        "lon":16.78097962194334
      }
    ]
}
```

Full plan retrving

When getting plans from the server you will notice some extra information into it. This information can be metadata or other extra information needed for the correct functioning of the flight.

```
{
    id: "/flightplan",
    type: "object",
    properties: {
      uuid: { type: "string" },
      version: { type: "number", minimum: 1 },
      safetySettings: {
        type: "object",
      },
      safetyProfile: { type: "string" },
      meta: {
        type: "object",
        properties: {
          altitudeMode: { type: "number", enum: [0, 1, 2] },
        },
        required: ["altitudeMode"],
      },
      mission: {
        type: "array",
        items: { $ref: "/fullWpSchema" },
        minItems: 4,
      },
      geoFence: {
```

```
        type: "object",
        properties: {
                    circles: {
            type: "array",
            items: {
              type: "object",
              properties: {
                uuid: { type: "string" },
                inclusion: { type: "string", enum: ["inclusion", "exclusion"] },
                type: { type: "string", enum: ["ground_buffer", "geocage", "pregeocage", "polygon"] },
                altAmsl: { type: "number", minimum: -100 },
                center: {
                  type: "object",
                  properties: {
                    lat: { type: "number", minimum: -90, maximum: 90 },
                    lon: { type: "number", minimum: -180, maximum: 180 },
                  },
                  required: ["lat", "lon"],
                },
                radius: { type: "number", minimum: 0 },
                altConversions: {
                  type: "object",
                  properties: {
                    altWgs84: { type: "number" },
                  },
                },
                meta: {
                  type: "object",
                  properties: {
                    editMode: { type: "boolean" },
                    color: { type: "string" },
                    visible: { type: "boolean" },
                  },
                },
                required: ["uuid", "inclusion", "type", "altAmsl", "radius", "center",
"altConversions"],
              },
            },
          },
          polygons: {
            type: "array",
            items: {
              type: "object",
              properties: {
                uuid: { type: "string" },
                inclusion: { type: "string", enum: ["inclusion", "exclusion"] },
                type: { type: "string", enum: ["ground_buffer", "geocage", "pregeocage", "polygon"] },
                altAmsl: { type: "number", minimum: -100 },
                vertices: { $ref: "/polygon" },
                altConversions: {
                  type: "object",
                  properties: {
                    altWgs84: { type: "number" },
                  },
                },
```

```
                meta: {
                  type: "object",
                  properties: {
                    editMode: { type: "boolean" },
                    color: { type: "string" },
                    visible: { type: "boolean" },
                  },
                },
              },
              required: ["uuid", "inclusion", "type", "altAmsl", "vertices", "altConversions"],
            },
          },
        },
      },
      rallyPoints: {
        type: "array",
        items: { $ref: "/fullRallypointSchema" },
      },
    },
    required: ["uuid", "version", "safetySettings", "safetyProfile", "mission", "geoFence",
"rallyPoints"],
  }
```

Waypoint

This specification corresponds to a waypoint and it's referenced from the full plan

> **CAUTION**
>
> Notice that the commands are just constants. You need to send the number of the command, not the string

```
MAV_CMD_NAV_TAKEOFF = 22
MAV_CMD_DO_VTOL_TRANSITION = 3000
MAV_CMD_NAV_WAYPOINT = 16
MAV_CMD_NAV_LAND = 21
MAV_CMD_DO_JUMP= 177
MAV_CMD_DO_CHANGE_SPEED= 178

{
    id: "/fullWpSchema",
    type: "object",
    properties: {
      uuid: { type: "string" },
      command: { type: "number", enum: [MAV_CMD_NAV_TAKEOFF, MAV_CMD_DO_VTOL_TRANSITION,
MAV_CMD_NAV_WAYPOINT, MAV_CMD_NAV_LAND, MAV_CMD_DO_JUMP, MAV_CMD_DO_CHANGE_SPEED] },
      altAmsl: { type: "number", minimum: -100 },
      padAltAmsl: { type: "number", minimum: -100 },
      lat: { type: "number", minimum: -90, maximum: 90 },
      lon: { type: "number", minimum: -180, maximum: 180 },
      transitionType: { type: "string", enum: ["front", "back"] },
      repeat: { type: "number", minimum: 1 },
      speed: { type: "number", minimum: 0 },
      jumpToUuid: { type: "string" },
```

```
      precision: { type: "number", enum: [1, 0] },
      altConversions: {
        type: "object",
        properties: {
          altWgs84: { type: "number", minimum: -100 },
          altPadWgs84: { type: "number", minimum: -100 },
          altAboveTakeoff: { type: "number", minimum: -100 },
          altAboveTerrain: { type: "number", minimum: -100 },
          altGroundAmsl: { type: "number", minimum: -100 },
          altGroundWgs84: { type: "number", minimum: -100 },
        },
        required: ["altWgs84", "altAboveTakeoff", "altAboveTerrain", "altGroundAmsl", "altGroundWgs84"],
      },
    },
    required: ["uuid", "command", "lat", "lon", "altAmsl", "altConversions"],
    allOf: [
      {
        if: {
          properties: { command: { const: [MAV_CMD_NAV_TAKEOFF, MAV_CMD_NAV_LAND] } },
        },
        then: {
          required: ["padAltAmsl"],
        },
      },
      {
        if: {
          properties: { command: { const: MAV_CMD_DO_JUMP } },
        },
        then: {
          required: ["repeat", "jumpToUuid"],
        },
      },
      {
        if: {
          properties: { command: { const: MAV_CMD_DO_VTOL_TRANSITION } },
        },
        then: {
          required: ["transitionType"],
        },
      },
      {
        if: {
          properties: { command: { const: MAV_CMD_DO_CHANGE_SPEED } },
        },
        then: {
          required: ["speed"],
        },
      },
    ],
}
```

polygon

This specification corresponds to a geocage polygon and it's referenced from the full plan

```
{
    id: "/polygon",
    type: "array",
    minItems: 3,
    items: {
      type: "object",
      properties: {
        uuid: { type: "string" },
        lat: { type: "number", minimum: -90, maximum: 90 },
        lon: { type: "number", minimum: -180, maximum: 180 },
      },
      required: ["lat", "lon"],
    },
},
```

Rallypoint

This specification corresponds to a rally point polygon and it's referenced from the full plan

```
{
    id: "/fullRallypointSchema",
    type: "object",
    properties: {
      uuid: { type: "string" },
      lat: { type: "number", minimum: -90, maximum: 90 },
      lon: { type: "number", minimum: -180, maximum: 180 },
      altAmsl: { type: "number", minimum: -100 },
      padAltAmsl: { type: "number", minimum: -100 },
      altConversions: {
        type: "object",
        properties: {
          altWgs84: { type: "number", minimum: -100 },
          altPadWgs84: { type: "number", minimum: -100 },
        },
        required: ["altWgs84", "altPadWgs84"],
      },
      required: ["uuid", "lat", "lon", "altAmsl", "padAltAmsl", "altConversions"],
    },
  }
```

Example of a retrieved plan from the server

```
{
    "uuid": "b2713cb8-87a6-45e1-9a4c-28004703fe69",
    "version": 2,
    "safetySettings": {
    },
    "safetyProfile": null,
    "meta": {
```

```
        "altitudeMode": 2
    },
    "geoFence": {
        "circles": [
        ],
        "polygons": [
            {
                "uuid": "78f08b61-c945-4d82-ae56-47ea48d5ccf3",
                "inclusion": "inclusion",
                "type": "geocage",
                "altAmsl": 606,
                "vertices": [
                    {
                        "uuid": "68442a88-53ec-470d-839f-95742c57f3e8",
                        "lat": 49.150077591022544,
                        "lon": 16.825107180165574
                    },
                    {
                        "uuid": "ad120520-0519-422d-bb11-105bebabca5b",
                        "lat": 49.1525792724473,
                        "lon": 16.82125552802019
                    },
                    {
                        "uuid": "54f0db2b-a5e7-42ea-b3d8-7d4c9eea0c7f",
                        "lat": 49.15376165013778,
                        "lon": 16.81496843008928
                    },
                    {
                        "uuid": "bc222f24-b791-4e3a-8015-f3afa175183b",
                        "lat": 49.159680131436424,
                        "lon": 16.797056638287827
                    },
                    {
                        "uuid": "42f4afe2-f7cd-42f7-8e98-2b3b5e74c743",
                        "lat": 49.16061853343021,
                        "lon": 16.78889467625551
                    },
                    {
                        "uuid": "4288233c-18de-41ce-b607-0bffb0657629",
                        "lat": 49.1588066122935,
                        "lon": 16.781934343861863
                    },
                    {
                        "uuid": "e51ea7d5-81b9-47fe-add0-b2e65db72551",
                        "lat": 49.156665723624045,
                        "lon": 16.763573282050416
                    },
                    {
                        "uuid": "57396284-694f-4dcf-a420-989aaa2e4635",
                        "lat": 49.154121708361615,
                        "lon": 16.75222552626066
                    },
                    {
                        "uuid": "1ba3b694-07ea-4432-af0e-411fdb6db73d",
                        "lat": 49.15045478109202,
```

```
            "lon": 16.749031685876176
        },
        {
            "uuid": "c57b86ac-1b6a-4e8c-a099-f85fe530e3dd",
            "lat": 49.15087034473323,
            "lon": 16.73482838331394
        },
        {
            "uuid": "f24bebab-213c-4c11-827b-2b741e4d6c19",
            "lat": 49.15204377237618,
            "lon": 16.728736080812737
        },
        {
            "uuid": "0a226c5d-ddde-48e8-b459-7a7d4887e498",
            "lat": 49.15747909242239,
            "lon": 16.718050998287723
        },
        {
            "uuid": "8d65f43f-59e7-47ba-b76d-a993a4f2d374",
            "lat": 49.15871855242119,
            "lon": 16.707182896906957
        },
        {
            "uuid": "2ebf2fdd-7bea-4bec-b8d0-51ac46cbd3bc",
            "lat": 49.15592011697326,
            "lon": 16.697717433680296
        },
        {
            "uuid": "316bd508-aa33-44bc-af1c-b3fcc94f23cf",
            "lat": 49.15532924349978,
            "lon": 16.69664389261881
        },
        {
            "uuid": "53027d73-f92f-4d62-8799-82e194b4cf39",
            "lat": 49.15410684189552,
            "lon": 16.699218155817576
        },
        {
            "uuid": "4d7cea4c-80a2-4ab2-9e0b-86fd736ad85c",
            "lat": 49.15394953255132,
            "lon": 16.705954549951926
        },
        {
            "uuid": "56e9fa55-8cc6-46d2-9e0d-364349149a92",
            "lat": 49.152632613814326,
            "lon": 16.71185015014045
        },
        {
            "uuid": "ae229592-fb3b-4bbe-b677-dd97a026d1e9",
            "lat": 49.1453995376264,
            "lon": 16.723716661917493
        },
        {
            "uuid": "3a23be4f-5501-42e5-97a3-0b9ea4212ab7",
            "lat": 49.144687954965214,
```

```
            "lon": 16.73247223033242
        },
        {
            "uuid": "64da64c9-5752-47fa-b6d5-e19a9e5ea492",
            "lat": 49.14278810311593,
            "lon": 16.738223305555508
        },
        {
            "uuid": "921b80b8-a41a-43c2-8850-1fd754b8768e",
            "lat": 49.14037824223416,
            "lon": 16.739596806118712
        },
        {
            "uuid": "cab6d551-6f9a-4985-b58a-4214d641f041",
            "lat": 49.140354745397154,
            "lon": 16.74912422208719
        },
        {
            "uuid": "c5d5f9c2-ca83-4b04-b427-d0d788072f9b",
            "lat": 49.14489181364054,
            "lon": 16.757474609302804
        },
        {
            "uuid": "4916ff88-dd04-4f91-b641-e11bbdbf65ae",
            "lat": 49.1466213436577,
            "lon": 16.76951571612291
        },
        {
            "uuid": "d2c4fabb-6755-4f88-beb0-828d8bf86062",
            "lat": 49.142827966263894,
            "lon": 16.777396046208665
        },
        {
            "uuid": "744f2f38-8a44-4941-a587-1a7a586f38e9",
            "lat": 49.146266933091326,
            "lon": 16.795914017247483
        },
        {
            "uuid": "daeff90e-c1d5-45d0-bced-a63d6af10325",
            "lat": 49.14568092194048,
            "lon": 16.80801614432268
        },
        {
            "uuid": "c438ee8a-0d6a-4ae0-a253-091704526175",
            "lat": 49.14392284690498,
            "lon": 16.810891472386643
        },
        {
            "uuid": "63a899e6-9204-462a-b232-e7a74747b798",
            "lat": 49.142629692022375,
            "lon": 16.81568189768724
        },
        {
            "uuid": "91ea64b1-ed9f-4e58-87b6-33ade8d64835",
            "lat": 49.14238228676779,
```

```
                        "lon": 16.821888529347703
                    },
                    {
                        "uuid": "dbc42693-2866-44ba-b95a-c4567d4d9e29",
                        "lat": 49.14256827948575,
                        "lon": 16.825053535985276
                    },
                    {
                        "uuid": "8302174c-e63a-445a-a7c4-1c1bf9775812",
                        "lat": 49.144357985057624,
                        "lon": 16.82549341826372
                    },
                    {
                        "uuid": "76b78e9c-f524-45fc-942c-66719aad3509",
                        "lat": 49.14684942832515,
                        "lon": 16.82568653731279
                    }
                ],
                "altConversions": {
                    "altWgs84": 650.0502829949656
                },
                "meta": {
                    "editMode": false,
                    "color": "#ffa500",
                    "visible": true
                }
            }
        ]
    },
    "mission": [
        {
            "uuid": "6dc2dc6e-b932-49c9-a9a9-688d141b9a04",
            "command": 22,
            "altAmsl": 350,
            "lat": 49.15108304952246,
            "lon": 16.79625748449217,
            "altConversions": {
                "altWgs84": 394.11329777267474,
                "altAboveTakeoff": 16,
                "altAboveTerrain": 83,
                "altGroundAmsl": 267,
                "altGroundWgs84": 311.11329777267474
            },
            "padAltAmsl": 267
        },
        {
            "uuid": "d7d7696b-195f-485f-8ca5-e78c85d76803",
            "command": 16,
            "altAmsl": 350,
            "lat": 49.15052165642498,
            "lon": 16.763470161494123,
            "altConversions": {
                "altWgs84": 394.18614799696365,
                "altAboveTakeoff": 16,
                "altAboveTerrain": 105.78,
```

```
                "altGroundAmsl": 244.22,
                "altGroundWgs84": 288.4061479969637
            }
        },
        {
            "uuid": "330f0c8b-900d-43fe-bdcd-05c6686968a4",
            "command": 3000,
            "altAmsl": 350,
            "lat": 49.15052165642498,
            "lon": 16.763470161494123,
            "altConversions": {
                "altWgs84": 394.18614799696365,
                "altAboveTakeoff": 16,
                "altAboveTerrain": 105.78,
                "altGroundAmsl": 244.22,
                "altGroundWgs84": 288.4061479969637
            },
            "transitionType": "front"
        },
        {
            "uuid": "49b89cad-8354-456b-ab96-49ef983be771",
            "command": 16,
            "altAmsl": 350,
            "lat": 49.14911814584975,
            "lon": 16.754372108515607,
            "altConversions": {
                "altWgs84": 394.20467686945904,
                "altAboveTakeoff": 16,
                "altAboveTerrain": 110.02000000000001,
                "altGroundAmsl": 239.98,
                "altGroundWgs84": 284.18467686945905
            }
        },
        {
            "uuid": "dece8e01-bfe9-48a0-8f71-164fac36025f",
            "command": 3000,
            "altAmsl": 350,
            "lat": 49.146872446229374,
            "lon": 16.745960701044904,
            "altConversions": {
                "altWgs84": 394.22110539882925,
                "altAboveTakeoff": 16,
                "altAboveTerrain": 119.68,
                "altGroundAmsl": 230.32,
                "altGroundWgs84": 274.54110539882925
            },
            "transitionType": "back"
        },
        {
            "uuid": "ab561e5c-cc75-4a11-8719-14069ed94f6e",
            "command": 16,
            "altAmsl": 360,
            "lat": 49.147939166240384,
            "lon": 16.732141960200178,
            "altConversions": {
```

```
                "altWgs84": 404.2607765552109,
                "altAboveTakeoff": 16,
                "altAboveTerrain": 10,
                "altGroundAmsl": 350,
                "altGroundWgs84": 394.2607765552109
            }
        },
        {
            "uuid": "ddbcf338-99b5-4f4e-8fc6-dfae1a16214a",
            "command": 21,
            "altAmsl": 360,
            "lat": 49.147939166240384,
            "lon": 16.732141960200178,
            "altConversions": {
                "altWgs84": 404.2607765552109,
                "altAboveTakeoff": 16,
                "altAboveTerrain": 10,
                "altGroundAmsl": 350,
                "altGroundWgs84": 394.2607765552109
            },
            "padAltAmsl": 350
        }
    ],
    "rallyPoints": [
        {
            "uuid": "681ee0b6-f94f-4e4b-a7e7-244e3b10d9a1",
            "lat": 49.15916640477572,
            "lon": 16.80844544225584,
            "padAltAmsl": 266.64,
            "altAmsl": 286.64,
            "altConversions": {
                "altWgs84": 330.7479568737182,
                "altPadWgs84": 310.7479568737182
            }
        },
        {
            "uuid": "b2f1b70b-bd2a-49ed-ad33-3088d07da1e8",
            "lat": 49.16489131666681,
            "lon": 16.78097962194334,
            "padAltAmsl": 233.8,
            "altAmsl": 253.8,
            "altConversions": {
                "altWgs84": 297.98100594923847,
                "altPadWgs84": 277.98100594923847
            }
        }
    ]
}
```