Forge Cloud v0.02

API Documentation

Access & API Key

API keys are currently private and only given by special request. Any given API key should be kept secret. The following custom HTTP headers are required to make a valid request to the API.

```
curl -H user-email=<your-email> \
     -H user-key=<apikey> \
        stage.dronesmith.io/api/user/563c2507a1e444b50b0f8d95
```

Without a valid api key and account, the server will respond with code `401` (not authorized).

All api endpoints point to `stage.dronesmith.io/api/`.

All api requests are handled in standard practice with `GET/POST/PUT/DELETE` HTTP headers. The responses will always be in JSON format.

Gimme

To obtain API keys, please contact geoff@skyworksas.com.

Error Handling

User-defined errors besides `401`s will have the response code `400`. Server side error codes *should not* happen, but if they do, you will see a `500` code. `400` errors will respond the following JSON object:

Return **400**:

```
{
    "error" : "message about the error."
}
```

User `/api/user/`

The user endpoint provides information for querying user accounts. Currently, you may only query a restricted subset of user information via GET requests.

Here is the public schema:

```
id          : Unique identifier
email       : String,
fullName    : String <optional>,
company     : String <optional>,
kind        : Enumerated String,
Otherkind   : String
created     : Date,
lastLogin   : Date,
userAgent   : String,
ipAddr      : String,
drones      : Array of [Drone] <optional>
```

The userAgent property contains information about the browser and operating system the user used when they last logged in. ipAddr contains the last used IP address.

Query user(s) `GET /api/user/{id}?<querystring>`

Find a user by a specific id. `id` is not required. Without a `sort` specifier, the order of users returned is not defined. `total`, `page` and `size` are always returned. Total is the total number of entries, size is the number of entries returned in the query, and page is an offset (see pagination).

Example: Get a list of all users who identify as developers.

```
GET /api/user?kind=developer
```

Return **200**:

```
{
    total: 34,
    page: null,
    size: 34,
    users: [
        {
            "id": "563c2507a1e444b50b0f8d8b",
            "email": "amanda@skyworksas.com",
            "fullName": "Amanda",
            "company": "Skyworks Aerial Systems",
            "kind": "developer",
            ...
        },
        {
            "id": "563c2507a1e444b50b0f8d95",
            "email": "sethw@skyworksas.com",
            "fullName": "Seth",
            "company": "Skyworks Aerial Systems",
            "kind": "developer",
            ...
        },
        ...
    ]
}
```

Query a single user

```
GET /api/user/563c2507a1e444b50b0f8d95
```

Return **200**:

```
{
    "id": "563c2507a1e444b50b0f8d95",
    "email": "sethw@skyworksas.com",
    "fullName": "Seth",
    "company": "Skyworks Aerial Systems",
    "kind": "other",
    "Otherkind": "elevated",
    "created": "2015-11-06T03:56:55.878Z",
    "lastLogin": "2015-11-06T21:24:08.142Z",
    "userAgent": "Mozilla/5.0",
    "ipAddr": "127.0.0.1",
    "drones": [
        "5637f2451cf63be64145366c",
        ...
    ]
}
```

Pagination

When not querying by individual ids, you can also use **page** and **size** fields divide up a query into multiple pieces (pagination), suitable for large queries.

```
GET /api/user?offset=90&size=50
```

Return **200**:

```
{
    total: 200,
    size: 10,
    page: 90,
    users: [
        { <user90> },
        { <user91> },
        ...
        { <user109> }
    ]
}
```

Note that the size parameter is required to use page. The page paremeters begins with 1 and should go no longer than the total number of entries. A page of 0 does nothing.

Sorting

When not querying by individual ids you can use **sort** option to sort by one of the attributes of the field. For instance:

Asending order:

```
GET /api/user?size=50&sort=created
```

Descending order:

```
GET /api/user?size=50&sort=-created
```

Returns the first 50 entries of the user list who were created last sorted in ascending order. To sort in reverse order, prefix the attribute with a minus dash `-`.

Add a drone to a user `PUT /api/user/id/drone_id`

It is recommended though not required that a drone be associated with a specific user. This can be done by issuing a PUT request with the user id, followed by the drone id. `{ "status" : "ok" }` is returned. An error will be returned if the drone does not exist. The user's list of drones is a unique list, so any duplicates will be removed.

Remove a drone from a user `DELETE /api/user/id/drone_id`

By the same virtue, one can remove a drone from a user with a DELETE request. Again, `{ "status" : "ok"}` is returned on success.

Drone `/api/drone/`

The public schema contains information pertaining to a particular UAV in question.

```
id              : Unique Identifier
name            : String <optional>,
systemId        : Number,
manufacturer    : String <optional>,
created         : Date,
updated         : Date <optional>,
hardwareId      : String <optional>,
firmwareId      : String <optional>,
parameters      : Mixed Object,
missions        : Array of [Mission] <optional>
```

The parameters field is an associative array that contains a list of the drone's parameters. You can think of these as settings for the drone. This should be updated whenever the drone object is updated.

Each drone is required to have systemId. This should correspond to its `MAVLink` id and is defaulted to 1. It is advised but not required that each drone have a different systemId.

Querying a drone is similar to querying a user, but offers a more advanced query format to facilitate querying by different parameters.

Query drones(s) `GET /api/drone/{id}?<querystring>`

Find a drone by a specific id. `id` is not required. Without a `sort` specifier, the order of drones returned is not defined. `total`, `page` and `size` are always returned. Total is the total number of entries, size is the number of entries returned in the query, and page is an offset (see pagination).

Example: Get a list of all drones with UAVCAN disabled:

```
GET /api/drone?parameters.UAVCAN_ENABLE=0
```

Return **200**:

```
{
    total: 5,
    size: 1,
    page: null,
    drones: [
        {
            "_id": "5637f2451cf63be64145366c",
            parameters: {
                UAVCAN_ENABLE: 0,
                ...
            },
            "updated": "2015-11-05T06:04:57.701Z",
            "missions": [
                "563bc8b36b33b16b5f3112b1",
                "563bf15ed6dc383e719dc006"
            ],
            "firmwareId": "1.01",
            "hardwareId": "LUCI REV B",
            "created": "2015-11-02T23:31:17.175Z",
            "systemId": 1,
            "name": "",
            "manufacturer": ""
        }
    ]
}
```

Query a single drone

```
GET /api/drone/5637f2451cf63be64145366c
```

Return **200**:

```
{
    "_id": "5637f2451cf63be64145366c",
    parameters: {
        UAVCAN_ENABLE: 0,
        ...
    },
    "updated": "2015-11-05T06:04:57.701Z",
    "missions": [
        "563bc8b36b33b16b5f3112b1",
        "563bf15ed6dc383e719dc006"
    ],
    "firmwareId": "1.01",
    "hardwareId": "LUCI REV B",
    "created": "2015-11-02T23:31:17.175Z",
    "systemId": 1,
    "name": "",
    "manufacturer": ""
}
```

Pagination

When not querying by individual ids, you can also use **page** and **size** fields divide up a query into multiple pieces (pagination), suitable for large queries.

```
GET /api/drone?page=90&size=50
```

Return **200**:

```
{
    total: 200,
    size: 10,
    page: 90,
    users: [
        { <drone90> },
        { <drone91> },
        ...
        { <drone109> }
    ]
}
```

Note that the size parameter is required to use page. The page paremeters begins with 1 and should go no longer than the total number of entries. A page of 0 does nothing.

Sorting

When not querying by individual ids you can use **sort** option to sort by one of the attributes of the field. For instance:

Asending order:

```
GET /api/drone?size=50&sort=created
```

Descending order:

```
GET /api/drone?size=50&sort=-created
```

Returns the first 50 entries of the drone list who were created last sorted in ascending order. To sort in reverse order, prefix the attribute with a minus dash `-`.

Add a drone `POST /api/drone/`

Allows you to create a new drone. The returned object will echo the parameters sent along with the server's filled in fields (such as the drone id)

```
POST /api/drone/

POST Body
{
    "systemId": 254 <required>,
    "parameters": {} <required>,
    ...
}
```

Remove a drone `DELETE /api/drone/id`

```
DELETE /api/drone/5637f2451cf63be64145366c
```

Return **200**:

```
{"status": "OK"}
```

This status message indicates the drone was successfully deleted.

Update a drone `PUT /api/drone/id`

```
PUT /api/drone/5637f2451cf63be64145366c

PUT Body
{
    "drone": {
        "parameters": {}
    }
}
```

Add a mission to a drone `PUT /api/ droneId/`

It is recommended though not required that a mission be associated with a specific drone. This can be done by issuing a PUT request with the drone id, and including the middion id in the body. An error will be returned if the mission does not exist. The drone's list of missions is a unique list, so any duplicates will be removed.

```
PUT /drone/addMission/5637f2451cf63be64145366c

PUT Body
{
    "missionId": "563bc8b36b33b16b5f3112b1"
}
```

Response **200**:

```
{
    "status": "OK"
}
```

Remove a mission from a drone `DELETE`

`/api/drone/removeMission/droneId`

By the same virtue, one can remove a mission from a from with a DELETE request. Again, `{ "status" : "ok"}` is returned on success.

```
DELETE /api/drone/removeMission/5637f2451cf63be64145366c

DELETE Body
{
    "missionId": "563bc8b36b33b16b5f3112b1"
}
```

Response **200**:

```
{
    "status": "OK"
}
```

Mission `/api/mission/`

The mission schema is organized as a sequence of MAVLink log format entries.

```
name        : String <optional>,
kind        : Enumerated String,
created     : Date,
start       : Date <optional>,
end         : Date <optional>,
user        : User <optional>,
errCount    : Number,
flight: Array of [
    {
        time            : Date <optional>,
        message         : Number or String,
        systemID        : Number,
        componentID     : Number <optional>,
        payload         : Mixed Object <optional>
    }
] <optional>,
parameters: Mixed Object <optional>
```

The mission schema is designed to be flexible to allowing many different flight logging formats.

The current two that are being used are `sdlog` and `mavlink log` formats. Both of them are encoded into the following schema, but have different formats.

The **mavlink log** format is a telemetry based ascii implementation that uses MAVLink messages as the basis of its datapoints. Such data includes a systemId, componentId, message number (which corresponds to type), and a time stamp at the end of each mesage. The payload is determined by the message type. Most of the data on Forge will be in the this format. You read more about it here.

The **sdlog** format is a high-resolution binary based implementation that uses a dynamic look-up table mapped by its header. In essence, this format's actual data is defined on the fly by the header itself; however, to stay consistant, most of the messages are uniformly supported by different UAVs and ground control stations. This format does not time stamp each data point, and uses strings to represent message names. You can view more information about it here.

Each mission will append a UTC Date formatted timestap at the beginning of a mission. The MAVLink log data is represented as a sequence of MAVLink packet entries. Typical flight times are within the range of 5-15 minutes. As MAVLink packets tend to output around 60 times a second, you can expect this array will be extremely large, even on smaller flights.

The parameters object contains a copy of the settings that were made for the drone during the flight. This is different from the parameters field in the drone itself typically.

The end timestamp is optional; however, all completed flights will have an end timestamp. The lack of this stamp indicates the flight is ongoing. If the mission failed to receive any data from the drone as a connection error, it will fill this field in with `null`.

Query missions(s) `GET /api/mission/{id}?<querystring>`

Find a mission by a specific id. `id` is not required. Without a `sort` specifier, the order of mission returned is not defined. `total`, `page` and `size` are always returned. Total is the total number of entries, size is the number of entries returned in the query, and page is an offset (see pagination).

Example: Get a list of all missions with UAVCAN disabled:

```
GET /api/mission?parameters.UAVCAN_ENABLE=0
```

Return **200**:

```
{
    total: 2,
    page: null,
    size: 2,
    missions: [
        {
            "_id": "563bc8b36b33b16b5f3112b1",
            "name": "",
            "user": "5637c39bc276bf4225a0bc3b",
            "kind": "mavlink",
            "end": "2015-11-04T04:41:37.801Z",
            "start": "2015-11-04T04:41:07.798Z"
        },
        {
            "_id": "563bf15ed6dc383e719dc006",
            "name": "",
            "user": "5637c39bc276bf4225a0bc3b",
            "kind": "sdlog",
            "start": "2015-11-06T00:16:30.486Z"
        }
    ]
}
```

Please note that you will **not** be able to view raw flight data points when querying multiple missions. You **must** query a single mission to view its datapoints.

Query a single mission

```
GET /api/mission/563bc8b36b33b16b5f3112b1
```

Return **200**:

```
{
    "_id": "563bc8b36b33b16b5f3112b1",
    "name": "",
    "user": "5637c39bc276bf4225a0bc3b",
    "kind": "mavlink",
    "end": "2015-11-04T04:41:37.801Z",
    "flight": [
        {
            "message": 0,
            "payload": {
                "custom_mode": 65536,
                "type": 2,
                "autopilot": 12,
                "base_mode": 81,
                "system_status": 0,
                "mavlink_version": 3
            },
            "_id": "563bc8b36b33b16b5f3113ae",
            "componentId": 1,
            "systemId": 1,
            "time": "2015-11-04T04:41:07.803Z"
        },
        ...
    ],
    "start": "2015-11-04T04:41:07.798Z"
}
```

Pagination

When not querying by individual ids, you can also use **page** and **size** fields divide up a query into multiple pieces (pagination), suitable for large queries.

```
GET /api/mission?offset=90&size=50
```

Return **200**:

```
{
    total: 200,
    size: 50,
    offset: 90,
    drones: [
        { <mission90> },
        { <mission91> },
        ...
        { <mission139> }
    ]
}
```

Note that the size parameter is required to use page. The page paremeters begins with 1 and should go no longer than the total number of entries. A page of 0 does nothing.

Sorting

When not querying by individual ids you can use **sort** option to sort by one of the attributes of the field. For instance:

Asending order:

```
GET /api/mission?size=50&sort=start
```

Descending order:

```
GET /api/mission?size=50&sort=-start
```

Returns the first 50 entries of the mission list of when the mission started sorted in ascending order. To sort in reverse order, prefix the attribute with a minus dash `-`.

Add a Mission `POST /api/mission/kind`

To add a mission, simply send your mavlink log (in JOSN format) or an sdlog (in binary format). `kind` should be `mavlink` for the former and `sdlog` for the latter, else parsing errors may occur.

`{status: "OK"}` JSON object is returned if successful. Please note that with high volume uploads, this request may take a while, so do not use short timeouts. Forge's maximum upload size is **50MB**. Please do not abuse this liberally large request size, or I will shorten it.

Remove a Mission `DELETE /api/mission/id`

Deleting missions simply involves entering the mission id. `{status: "OK"}` JSON object is returned if successful.