Dronesmith API

This is the development documentation. Things are subject to change as the API is tested and features are requested and implemented, or removed. Those implementing documentation should use this document as the foundation.

Changelog

1.0.01: (2016-11-03)

- Various minor bug fixes and improvements.
- Added groups to drone objects.
- Added group updates to *PUT /api/drone/<id/name>*
- The online field now works when querying a drone.
- Added query strings to *GET /api/drone*

1.0.02: (2016-11-10)

- Minor bug fixes once more.
- Add "relativePos" and "relativeAlt" to goto and takeoff.
- Remove GPS lat/lon from land, not applicable in firmware.
- Add \`statusCode\` to command response object.
- Send a 400 error when trying to query information on a drone that is not online.
- Make field names in post bodies case insensitive.

Using the API

Each request is made to: ***api.dronesmith.io/api/***
All API requests are prefixed with **/api/**.

All request and response data are marshaled as JSON. Please include "Content-Type" and "application/json" in your request header when sending request data to the API.

A valid user email address and API key are required to use the Dronesmith API. These components must be included with every request made to the API within the *request headers.*

Your email address should be included in the request headers as "user-email" and your api key as "user-key". See the example below for more information.

To get an API key, simply go to cloud.dronesmith.io and create an account. Your API key will be sent to you in your confirmation email. You do not need to verify your account to use the API. You can also retrieve your API key at any time by logging into cloud.dronesmith.io and selecting "account" from the status bar.

An easy way to verify you can talk to the Dronesmith API is to simply send a GET request to ***api.dronesmith.io/api/***.

Here is an example request.

Headers:

user-email	*test@test.com*
user-key	*a1204f04-c2a9-477b-ae20-a8bba455da09*

GET **/api/**

Response:

403 - Forbidden, request failed to authenticate
204 - Success (no response content)

A response code of 204 indicates that you can talk to the Dronesmith API and your email/key were successfully authenticated. **All API requests will respond with 403 (Forbidden) if there was a failure to authenticate.** Any request path or method made that does not exist in the API will respond with a 404 (Not found) status code.

Successful responses: To retain consistency, all API requests return JSON data. Many responses, especially those of POST, PUT, or DELETE methods do not have any data to send back. These responses will send back the following, case-insensitive vernacular if they have completed a request successfully, but have no further content:

Status code: 200
JSON: {“status”: “OK”}

Error responses: If the API encounter an error handling a request, such as being unable to find a drone object with a specified ID or name, it will return the following:

Status code: 400
JSON: {“error”: “*description of error here*”}

The API always follow these conventions when dealing successful responses and errors; developers can use these to help determine the outcome of their API request. Please keep in these responses are *case insensitive*, so parsing for case may be needed, depending on the developer’s implementation.

Drone Objects

A Drone Object is the representation of your drone in the API. Every drone object is a unique drone that API requests can be made to gather telemetry information, send commands to, etc. You can think of drone objects as the API's representation of your physical or virtual drone.

Drone Objects can be created in two ways. First, any physical drone, such as a Luci, that is activated through DS Link properly, will automatically create a drone object associated with that Luci to your account. Secondly, you may create a virtual drone and corresponding object, by using the Virtual Drone API.

Anatomy of a drone object
Every drone object contains the following metadata.

id <string> <read-only>
The ID field is a unique identifier for a drone object. You can use this as a way to query your drone. Each ID is a string of 24 serialized letters and numbers.

name <string>
The name is a string that serves as an alias for your drone object, making it easier to work with. Your name must be unique for all of your drone objects on your account.

simId <string> <read-only>
If your drone is a virtual drone (see *Virtual Drones* below) then your drone object will have this field, which serves a unique Id for virtual drones. In general, the simId is not important to the end user, however you can use it as a convenient way of determining if your drone object is tied to a real or virtual drone.

created <date> <read-only>
A UTC date of when this drone object was created.

updated <date> <read-only>
A UTC date of when this drone object was last modified. Keep in mind this update is from live drone data, ***not*** updating the drone object metadata.

systemId <number>
Represents the drone object's MAVLink ID number. Is 1 by default. Generally does not need to be modified.

online <boolean>
True if the drone is currently communicating with the API, false otherwise. Online resets to false after 90 seconds of inactivity.

type <string>
Is *"Drone"* when the drone object represents a physical drone. Is *"Virtual Drone"* when the drone object represents a virtual drone. You are free modify this type field for your own needs, such as classifying swarm drone groups.

firmwareId <string> <read-only>
A unique Id generated by physical drones from DS Link. This is "unknown" on virtual drones.

missions <array> <read-only>
A list of mission objects that your drone has uploaded. See *Mission Objects* for more details.

sensors <object> <read-only>
A set of user-defined sensors with their values. These allow users to upload their own custom sensor information, either locally from their drone using DS Link, or from another source. See *User Defined Sensors* section.

groups <array>
An array of strings of groups the drone is associated with. Groups serve as easy ways to logically organize drones.

Name/Id

All API endpoints beginning with *drone* have will have the syntax:

/api/drone/<id/name>/<endpoint>

Where *<id/name>* is the name or id of the drone, and *<endpoint>* is the name of the API request. In these cases, you can specify a drone by using its *ID* or *name*. The *ID* corresponds to the drone object's ID, and name corresponds to its name. For virtual drones, a name is always generated, but for real drones, a name must be updated to the drone object via a PUT request (see API below). All endpoints beginning with */api/drone* follow this convention.

Offline Drones

If the drone is no longer connected or responding to the API, any calls made to:

/api/drone/<id/name>/<endpoint>

Will return a 400 status error indicating the drone is not online.

Drone Object API

GET **/api/drone/**

Description: Lists all drone objects a user account has as a JSON array. See *Anatomy of a Drone Object* for more information.

Query Strings:

Size - the number of missions to include in the object. Default is 10 if no size is specified.

Page - the offset to return. Only applicable if size has been set. For instance, a size of 10 with a page of 2 would return missions starting from 10 * (2-1) = 10 to 20.

Sort - Order the missions by a field. Use a dash `-` (minus) to specify an inverted ordering.

Groups - Select drone objects by groups. You can select mutliple groups by separting each with a comment. Ex: *?groups=swarm-23,vegas,px4*

<Field> - Select drone objects by generic field name in drone object, with a specific value. *Ex: ?type=Virtual Drone&systemId=1*

Returned Fields:

Total - total number of missions

Size - Number of returned missions

Page - the specified page. Null if page is not applicable.

Missions - The mission object.

Example Response:

```
{
        [
        {
                id: “58069ee983a21d7b4f79ae91”,
                name: “tiny_swanson”,
                online: true,
                ...
        },
        {id: “57fc62b34f1565523bfad446” },
        …
        ]
}
```

GET /api/drone/<name/id>

<name/id>
Either your drone's unique ID, or its custom name.

Description: Returns information for a single drone object in a user's account. See *Anatomy of a Drone Object* for more information.

Example Response:

```
{
	"id": "58069ee983a21d7b4f79ae91",,
	"simId":
"e538a4f7c1e5884c51736ad482a0c166b0dd99239004cc0e26717a3587bb6a18",
	"updated": "2016-10-19T02:45:30.103Z",
	...
}
```

PUT /api/drone/<name/id>

<name/id>
Either your drone's unique ID, or its custom name.

Description: Update metadata in your drone object on all fields that are not *read-only*. See *Anatomy of a Drone Object* for more information.

Note: Using Groups
There is a special rule for updating groups in a drone object.

To update a group object, you must use an object, and preface it with one of the three fields: *$set, $push, $pull.* Then, include the array of groups. The field dictates how you wish to modify the group. You can have multiple *$set, $push, and $pull* operations in your group field, but the behavior of how it will affect the groups is undefined, as it depends on how the JSON object is iterated, which is arbitrary and implementation dependent. It is recommended to only use one of these fields per group update at a time.

$set is used to do a fresh update on a group. This remove the existing group names and add the new group names, or if an empty array is used, no groups.

Example:

Old Drone object: *{groups: [swarm-12, vegas]}*

Update value: *groups: {$set: [swarm-32, kansas, multirotor] }*

New value: *{groups: [swarm-32, kansas, multirotor]}*

$pull is used to remove groups from a drone object. Any groups in the drone object that matches the array specified by $pull will be removed. If there is no match, no change is made.

Example:

Old Drone object: *{groups: [swarm-12, vegas]}*

Update value: *groups: {$pull: [swarm-32, vegas] }*

New value: *{groups: [swarm-12]}*

$push is used to add groups to a drone object. Any groups in the drone object that matches the array specified by $pull will be added. Existing groups remain unchanged.

Old Drone object: *{groups: [swarm-12, vegas]}*

Update value: *groups: {$push: [swarm-32, vegas] }*

New value: *{groups: [swarm-12, swarm-32, vegas]}*

Example Body:
```
{
        “name”: “gz-drone2”,
        “groups”: {$set: [simdrones-2]}
}
```

DELETE   /api/drone/<name/id>

<name/id>
Either your drone’s unique ID, or its custom name.

Description: Delete a drone object in your account. If this is a virtual drone, this forcefully terminates it and deletes all associated metadata. For a real drone, it only deletes associated metadata. If the real drone is successfully authenticated with the API, the next time it is online, a new drone object will be created. You must log out of your drone via DS Link to completely remove the drone object. This may take a few seconds for the request to send a response.

Virtual Drones

To make development easier, Dronesmith provides virtual drones and virtual Linux environments for developers to work with.

What is a virtual drone?
A virtual drone is a drone that runs a simulated environment. For all intents and purposes, the drone functions exactly like a real drone, only virtual drones have unlimited battery life, and lack any of the problems that makes working with real drones difficult. They are useful for developers who need to implement higher level solutions before going to hardware. Each virtual drone comes with a fake GPS and Optical Flow unit to ease development.

Virtual Drone API

POST **/api/drone/<name>**

<name> <optional>
An optional name. Must be unique. This will be the name of your new virtual drone.

Description: Creates a virtual environment and Drone Object associated. Please note this request may take a few seconds to complete. The drone exists in a virtual Linux environment. You may access this Linux environment using the *SSH API*. Returns the newly created Drone Object if successful. You can delete virtual drones by sending a DELETE request to the associated Drone Object. In order to begin using your virtual drone, you must send the /start request.

POST **/api/drone/<id/name>/start**

<name/id>
Either your drone's unique ID, or its custom name. The drone must be a virtual drone.

Description: Initializes or unpauses the virtual drone's Linux environment, allowing it to begin running. You must call this after creating a virtual drone to begin using it. This may take a few seconds for the request to send a response.

POST **/api/drone/<id/name>/stop**

<name/id>
Either your drone's unique ID, or its custom name. The drone must be a virtual drone.

Description: Pauses the virtual drone's Linux environment. This does not destroy your virtual drone, but it will keep it from running until a *start* request is sent. Sending a *stop* then *start* request effectively restarts your virtual drone. This can be useful as a reset if the virtual drone has been misconfigured or has crashed a flight and is unable to takeoff.

Drone Telemetry

To collect basic information about your drone while it's running, such as where it's located, what orientation it's at, etc, send one of the following GET requests.

All requests have the following structure:

GET **/api/drone/<name/id>/<telemetry>**

<name/id>
Either your drone's unique ID, or its custom name.

<telemetry>
The requested telemetry information.

GET **/api/drone/<id/name>/info**

Description: Returns basic information about your drone.

Type - The kind of drone this is. Examples include:
- Quadrotor
- Hexrotor
- Fixed Wing
- VTOL (vertical takeoff and land)

Firmware - The underlying flight control software. Examples include *PX4*, *APM.*
Protocol - The communications protocol. Currently only *MAVLink* is supported.
LastUpdate - a timestamp of when the last message was received.
LastOnline - a timestamp of when the drone last signed on with the API.

Example response:
```
{
  "Type": "Quadrotor",
  "Firmware": "PX4",
  "Protocol": "MAVLink v3",
  "LastUpdate": "2016-10-19T22:15:41.810529109-07:00",
  "LastOnline": "2016-10-19T20:02:34.292940188-07:00"
}
```

GET **/api/drone/<id/name>/status**

Description: Returns current information about your drone.

Online - true if the drone's *datalink* is active, false otherwise. Please note that this field represents the *datalink* between the drone and DS Link, the translator that communicates with the API. If, for instance, the online is false, this indicates that DS Link is communicating with the API, but the drone itself is not communicating with DS Link.

Armed - true if the drone's motors are enabled to spin, false if a drone's motors are locked, preventing it from flying.

State - The drone's current flight state. The following are a list of states:
"Initializing" - Drone is currently starting up.
"Calibrating" - Drone is in a calibration mode.
"Standby" - Drone is awaiting commands.
"Active" - Drone is armed and flying.
"Failsafe" - Drone encountered an error is performing a failsafe operation, such as an emergency landing.

VTOLMode - Applies the VTOL (Vertical takeoff / land) drones only. Indicates if the vehicle is operating as a *fixed wing* or *multirotor* vehicle, or *transitioning* between such.

InAir - true if the drone has not detected any surface. When the drone detects a surface below it, this setting is false.

Power - the drone's current battery level as a percent. By default, the drone will enter a failsafe mode when the battery is below 17%. Virtual drones will simulate battery life, however, they will hold a drone's power at 17% to avoid entering failsafe mode.

Example response:
```
{
  "Online": true,
  "Armed": false,
  "State": "Standby",
  "VTOLMode": "Not a VTOL vehicle",
  "InAir": false,
  "Power": 100
}
```

GET **/api/drone/<id/name>/mode**

Description: Current operating mode of the drone. Default is “manual.” This setting can be changed with the *POST /mode* API request. The following are a list of modes:

Manual - Drone may be controlled with a standard RC controller, or with *POST /input* commands. No position or altitude holding is done.

Altitude - Drone may be controlled with a standard RC controller, or with *POST /input* commands. The throttle RC input is relative to altitude.

Position - Drone may be controlled with a standard RC controller, or with *POST /input* commands. All RC input is relative to the drone’s position and altitude.

Takeoff - Drone is currently in an automated takeoff mode. Set when *POST /takeoff* is requested. Remains in takeoff mode until a new command is issued.

Hold - Drone is currently moving to a position and will hold at that location. Set when *POST /goto* is requested.

Land - Drone is currently landing at a specified location. Set when *POST /land* is requested. Remains in land mode until a new command is issued.

RTL or RTGS - Drone is currently in failsafe mode.

Example response:
"Manual"

GET /api/drone/<id/name>/gps

Description: Gets the drone’s GPS position.

Satellites - Number of GPS satellites the drone’s GPS has detected. Useful for verifying GPS signal strength.
Lattitude - Global position in degrees latitude.
Longitude - Global position in degrees longitude.
Altitude - Mean atmospheric sea level altitude in meters relative to the drone’s global position. Basically, the average elevation from sea level in meters of the ground area the drone is in.

Example response:
```
{
  "Satellites": 10,
  "Latitude": 47.39774,
  "Longitude": 8.545593,
  "Altitude": 488.104
}
```

GET /api/drone/<id/name>/attitude

Description: Gets the drone's orientation. The range is -180 to 180 degrees Pitch and roll are relative to Earth's surface, with 0 being balanced. Yaw is relative to Earth's north pole, with 0 pointing directly north.

Example response:

```
{
  "Roll": -1.8285842,
  "Pitch": -1.2411544,
  "Yaw": 3.5657773
}
```

GET /api/drone/<id/name>/position

Description: Gets the drone's current position. X, Y, and Z are the drone's local position, relative to where its home location is, in meters. Latitude and longitude are the drone's global position, same as *GET /gps*. Altitude is the drone's relative altitude from the ground in meters. Heading is the drone's attitude from 0 to 360 degree, where 0 points to Earth's north pole.

Example response:

```
{
  "X": 0.004026684,
  "Y": -0.02161496,
  "Z": -0.11607925,
  "Latitude": 47.39774,
  "Longitude": 8.545593,
  "Altitude": 0.047,
  "Heading": 3.95
}
```

GET /api/drone/<id/name>/motors

Description: Gets the current PWM (the raw signal that goes to the motor) values being sent to the drone. The array represents the PWM value of each motor channel.

Example response:

```
[
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0
]
```

GET **/api/drone/<id/name>/rates**

Description: Returns the speed (rates) of the drone.

Airspeed - the drone’s measured airspeed. Normally 0 on multirotors, as they lack an airspeed sensor.

Groundspeed - the drone’s velocity relative to the ground in meters per second.

Throttle - the drone’s throttle value, a measurement of input force from 0 (no throttle) to 100 (maximum possible throttle).

Climb - the drone’s climb rate in meters per second. Negative value indicates the drone is moving downward, and positive indicates the drone is moving upward.

Example response:

```
{
  "Airspeed": 0,
  "Groundspeed": 0.030534232,
  "Throttle": 0,
  "Climb": 0.0637856
}
```

GET **/api/drone/<id/name>/target**

Description: Target represents a set of values the drone *wants* to reach. These are the control values.

Attitude - The intended attitude target; the orientation the drone *would like to be at*. Expressed as a quaternion.

Thrust - The intended thrust vector.

X, Y, Z - the drone’s target local position, relative to home position.

Latitude / Longitude - The drone’s target global position.

Altitude - The drone’s target relative altitude.

Example response:
```
{
  "Attitude": [
    0.9999932,
    0,
    0,
    -0.003681607
  ],
  "Thrust": 0.08,
  "X": 0,
  "Y": 0,
  "Z": 0,
  "Latitude": 0,
  "Longitude": 0,
  "Altitude": 0
}
```

GET /api/drone/<id/name>/sensors

Description: Gets the raw sensor values from the drone’s Inertial Measurement Unit (IMU).

AccX/Y/Z - The accelerometer’s X, Y, and Z measurements, in milli-Gs (a G is 9.8 meters per second squared, or the acceleration of Earth’s gravity). The range is typically 4G’s, but changes from drone to drone.

GyroX/Y/Z - The gyroscope’s X, Y, and Z measurements, in milliradians per second. The range is typically 6mrad, but changes depending on the drone’s implementation.

MagX/Y/Z - The magnometer (compass) measurements, in milliguass. The range is typically 8milligauss, but changes depending on the drone’s implementation.

Baro - The measurement of the drone’s barometric pressure unit, in millibars. It should be noted that the barometer is notoriously bad for measuring altitude, and direct measurement should be avoided.

Temp - The IMU’s operating temperature in degrees celsius, if available.

Example response:
```
{
  "AccX": -0.27144316,
  "AccY": 0.24669562,
  "AccZ": -9.84977,
  "GyroX": -0.007041006,
  "GyroY": -0.0033752206,
  "GyroZ": -0.0016560211,
  "MagX": 0.18976593,
  "MagY": 0.026640372,
  "MagZ": -0.40200245,
  "Baro": 0.104980275,
  "Temp": 32
}
```

GET /api/drone/<id/name>/home

Description: The drone's home location, in global and local coordinates. The home is set by the drone's initial location when it turns on, but can be changed with a *POST /home* command.

Example response:

```
{
  "X": 0.0000022542315,
  "Y": -0.0000025720442,
  "Z": -0.068796135,
  "Latitude": 47.39774,
  "Longitude": 8.545593,
  "Altitude": 488.104
}
```

GET /api/drone/<id/name>/log

Description: An output of the drone's status log. The array contains a set of objects with the message, the time it came, and the level, which determines its importance. The higher the level the more important the message is. A maximum of 200 log entries is retained. If there are more than 200 log entries, the oldest are removed. *Once you have queried /log, the entries are dequeued, so the next request will* ***not*** *contain the previous entries.*

Example response:

```
[
  {
    "Msg": "[inav] GPS signal found",
    "Time": "2016-10-28T13:03:13.857225257-04:00",
    "Level": 6
  },
  {
    "Msg": "[inav] init ref: 47.3977418, 8.5455939, 488.1040",
    "Time": "2016-10-28T13:03:14.839179169-04:00",
    "Level": 6
  }
]
```

GET **/api/drone/<id/name>/input**

Description: Gets the current input information that was last sent to the drone via a *POST /input* message. Currently only radio is supported. The channels represent the raw radio control values, from 1000 - 2000. A value of 65535 indicates there is no input being sent on that channel.

Example response:
```
{
  "Channels": [
    1500, 1500, 1750, 1500, 65535, 65535, 65535, 65535, 65535, 65535, 65535, 65535, 65535,
65535, 65535, 65535, 65535, 65535
  ],
  "Signal": 100,
  "Type": "Radio Control"
}
```

Parameters

Parameters are the configuration settings of your drone. These allow you to configure your drone for various different flight settings, they are typically an important part of the setup procedure. The Parameters API allows you to set and get parameters without directly interfacing with your drone.

Note: Due to the structure of parameters and the way that they are loaded, the API cannot guarantee that all params will always be loaded during initialization without perpetually requesting them from the drone infinitely. The API reports any params that it failed to load by their index numbers.

Params API

GET **/api/drone/<id/name>/params**

Description: Lists all params. “Current” represents the current number of loaded parameters. “Total” is the total number of params the drone contains. “Missing” is an array of indexes that the API could not load. You may request these manually by using the GET /param request, or by using the params/refresh request to manually reload all of the params.

Example Response:

```
{
        “Current”: 399,
        “Total”: 399,
        “Missing”: null,
        “Params”: {
           "ATT_BIAS_MAX": 0.05,
           "ATT_EXT_HDG_M": 0,
           "ATT_MAG_DECL": 0,
           "ATT_MAG_DECL_A": 0,
           "ATT_VIBE_THRESH": 0.2,
           …
        }
}
```

GET **/api/drone/<id/name>/params/refresh**

Description: Force API to get all params again. Useful if params were changed externally. Returns ***{status: “OK”}*** if worked. You will need to use GET /params to see the updated list.

GET **/api/drone/<id/name>/param/<paramname/index>**

<paramname/index>
The name of the param, such as “ATT_EXT_HDG_M” or the index number, from 1 to the total param number.

Description: Get a single param value.

Example Response:
0.15

POST **/api/drone/<id/name>/param/<paramname/index>**

<paramname/index>
The name of the param, such as “ATT_EXT_HDG_M” or the index number, from 1 to the total param number.

Description: Updates the value of a param.

Additional Fields:

Value (required): The new value to change the parameter to. Always a number.

Example:
```
{
	“value”: 0.16
}
```

Command and Control

The command and control aspect of the API is used to remotely control your drone through the cloud. All requests are POST requests, and follow the standard drone/id convention.

Command Acknowledgement

With the exception of the */input* command, all command and control messages return a command ACK response, such as the following:

```
{
	“Command”: <command number>,
	“Status”: <description of the command result>
	“StatusCode”: <value, see below>
}
```

Command is a MAVLink number of the vehicle command. Status is the a string that provides information about how the command was handled. It will usually be accepted, but could be rejected was simply not received by the drone.

Status Codes

0: Command was accepted successfully.

1: Command was denied/rejected, this mean it could not execute the command, usually because the drone is already doing the command.

2: Command was permanently denied. This means the drone must be reconfigured in order to use this command, and cannot be in flight. (See params API for configuration).

3: Command unsupported. This means the command was recognized, but is not supported by the internal firmware.

4: Command failed. There was an internal error with the drone firmware itself.

10: Command unknown. Means that the API could not retrieve the status of the command.

Command API

POST **/api/drone/<id/name>/arm**

Description: Attempts to arm the drone, allowing the motors to run. *GET /motors* can be used to see the raw signal value of the motors.

Additional fields: None.

POST /api/drone/<id/name>/disarm

Description: Attempts to disarm the drone, disallowing the motors to run.

Additional fields: None.

POST /api/drone/<id/name>/mode

Description: Change the drone's control mode and arm or disarm it, such as "Manual", "Position", or "Auto". Please note the mode value is case sensitive. If mode field is blank, the drone will not attempt to change mode. "arm" may be set to true or false, or left blank. If blank, it will not attempt to change the drone's arming state.

Additional fields:

mode - A string with the drone mode to change to. Optional. If not included, mode will not be changed. See *GET /mode* for a list of valid modes.

armed - A boolean to arm the drone or not. Optional. If not included, the arming state of the drone will not be changed.

POST /api/drone/<id/name>/takeoff

Description: Tells the drone to take off and hold at a certain altitude and/or location, if specified in the request body. A GPS unit is required.

Additional fields:

relativeAlt - true for relative coordinate frame for altitude, false for MSL (mean seal level) altitude. This field is true by default.

relativePos - true for relative coordinate frame for GPS position. False for absolutely GPS coordinates. This field is false by default.

lat - the input latitude to takeoff and hold position at. Optional. If not included, holds at its present latitude.

lon - the input longitude to takeoff and hold position at. Optional. If not included holds at its present longitude.

altitude - the relative altitude to takeoff and hold at. Optional. If not included, defaults to 10 meters.

POST /api/drone/<id/name>/land

Description: Tells the drone to land at a specified location. If no location is specified or a GPS unit is not installed, it will land where it is. Land by extension also forces the drone into a failsafe mode, if there are any issues with performing a controlled descent.

Additional fields:

No Additional fields.

POST /api/drone/<id/name>/goto

Description: Instructs the drone to fly to a specific location. If no location is specified, it will simply hold its current position. A GPS unit is required. Drone must be in air to use.

Additional fields:

relativeAlt - true for relative coordinate frame for altitude, false for MSL (mean seal level) altitude. This field is true by default.

relativePos - true for relative coordinate frame for GPS position. False for absolutely GPS coordinates. This field is false by default.

lat - the relative input latitude to go to and hold position at. Optional. If not included, holds at its present latitude.

lon - the relative input longitude to go to and hold position at. Optional. If not included holds at its present longitude.

altitude - the relative altitude to go to and hold at. Optional. If not included, defaults to present altitude.

POST /api/drone/<id/name>/home

Description: Update your drone's home location.

Additional fields:

lat - the relative input latitude for home. If not included, defaults to current home latitude.

lon - the relative input longitude for home. If not included, defaults to current home longitude.

POST /api/drone/<id/name>/input

Description: Programmatically control your drone's movements.

Additional fields:

Type - Required. Currently must be set to "radio."

Enabled - Set to true to allow manual input updates to the drone. Set to false to stop them.

Timeout - Number of milliseconds to specify for the API to disable manual input if no new *POST /input* is received in this time frame. Optional. If not specified, the API will never timeout.

Channels: The input values for each radio channel. Range is from 1000 to 2000 on virtual drones.

Example:
```
{
	"type": "radio",
	"enabled": true,
	"channels": [1500, 1500, 1750, 1500]
}
```

Channels are arranged as an array, where the first index represents the first channel. For a virtual drone, the mapping is default as follows:

Channel 1: Roll (1000 - 2000, 1500 is no roll)
Channel 2: Pitch (1000 - 2000, 1500 is no pitch)
Channel 3: Throttle (1000 - 2000, 1750 is enough throttle to hold the drone at its present altitude)
Channel 4: Yaw (1000 - 2000, 1500 is no yaw)

POST /api/drone/<id/name>/command

Description: Send a generic MAVLink command to your drone.

Additional fields:

Command - the MAVLink command number. Required.

Args - from 1 to 7, each argument. Depends on the command type. Optional.

Example:
```
{
	"command": 176,
	"args": [29, 4, 2]
}
```

User Defined Sensors/Payloads

One of the biggest applications of drones is being able to gather and collect data for specialized application in areas that are dirty, dangerous, or dull. The Sensor API can help you achieve this by providing access points to our API.

POST /api/drone/<id/name>/sensor/<name>

Description: Allows you to upload user defined sensor data to your drone, specified by name. Uploading new sensor data overrides the old data. The body of the request contains the data you wish to upload. This data is completely generic and allows you to upload whatever you want. This request can be called by DS Link, or by any app using this API.

Example Body:

GET **/api/drone/<id/name>/sensor/<name>**

Description: Returns user defined sensor information. To get all sensors, simply request the drone object.

Directly Access a Drone

For more advanced users, one may want to directly access their drone's Linux environment. Our API provides two requests for doing this. Note that this API is currently not supported on simulated drones due to container access issues.

POST **/api/drone/<id/name>/ssh/open**

Description: Creates an SSH connection with the drone in question. The username and password are the same as the ones you used to set the drone up.

Example Response:

```
{
  "Info": {
    "port": 16229,
    "url": "0.tcp.ngrok.io"
  },
  "Status": "OK"
}
```

POST **/api/drone/<id/name>/ssh/close**

Description: Closes an SSH connection to the drone.

Missions

Missions represent flight data objects that have been synced with the API. DS Link generates flight data objects whenever the drone is armed, to the time it is disarmed, and stores them locally on Luci's SD Card. When a drone connects to the API, it will automatically sync these mission objects to the cloud, making the accessible.

GET **/api/mission/**

Description: Get all missions associated with a user. You may use query strings to change the amount to fetch at once, or which page. Please note this query does not grab raw recorded data. You must GET /api/mission/<mission_id> in order to get detailed data.

Query Strings:

Size - the number of missions to include in the object. Default is 10 if no size is specified.

Page - the offset to return. Only applicable if size has been set. For instance, a size of 10 with a page of 2 would return missions starting from 10 * (2-1) = 10 to 20.

Sort - Order the missions by a field. Use a dash \`-\` (minus) to specify an inverted ordering.

Returned Fields:

Total - total number of missions

Size - Number of returned missions

Page - the specified page. Null if page is not applicable.

Missions - The mission object.

Example:

GET /api/mission?size=2&page=2&sort=-start

```
{
 "total": 32,
 "size": 2,
 "page": 2,
 "missions": [
   {
     "_id": "5758e654e3c6f411093d631c",
     "kind": "mavlinkBinary",
     "start": "2016-06-09T03:45:23.802Z",
     "end": "2016-06-09T05:51:35.561Z",
     "user": "5637c39bc276bf4225a0bc3b",
     "created": "2016-06-09T03:45:24.913Z",
     "name": ""
   },
   {
     "_id": "5758e8a9765aefa209bbd500",
     "kind": "mavlinkBinary",
     "start": "2016-06-09T06:11:20.819Z",
     "end": "2016-06-09T06:11:32.395Z",
     "user": "5637c39bc276bf4225a0bc3b",
     "created": "2016-06-09T03:55:21.885Z",
     "name": ""
```

```
    }
  ]
}
```

GET **/api/mission/<id>**

Description: Get a single mission, and its internal flight data. Here are the following fields for a mission object:

Id - the mission Id. A unique ID for each mission object.
Kind - The type of format. 3 formats are supported:
    Mavlink - MAVLink in JSON format.
    MavlinkBinary - MAVLink in binary format.
Start - the start time of the flight.
End - the end time of the flight.
User - an associated user id with the mission.
Flight - An array of messages that were recorded during the flight. Each message represents a MAVLink message. The Number is the number of the MAVLink message, and the payload is the data fields in the message. In addition, the system and component IDs of the MAVLink message are returned, and each message as a timestamp.

Example:
    /api/mission/581251be8f775cb114e3834a

```
{
  "_id": "581251be8f775cb114e3834a",
  "kind": "mavlinkBinary",
  "start": "2016-10-27T19:15:13.985Z",
  "end": "2016-10-27T19:20:06.214Z",
  "user": null,
  "flight": [
    {
      "message": 30,
      "payload": {
        "time_boot_ms": 132001,
        "roll": -0.006278814747929573,
        "pitch": 0.032164089381694794,
        "yaw": 0.01402966771274805,
        "rollspeed": -0.022026557475328445,
        "pitchspeed": -0.08455482125282288,
        "yawspeed": -0.05751141905784607
      },
      "componentId": 1,
    "systemId": 1,
    "time": "2016-10-27T19:15:13.985Z"
  },
        …
  ]
}
```

DELETE /api/mission/<id>

Description: Removes a mission object from the cloud, and deassociates the mission from the drone.