Living Documentation for WARA-PS Core System API Specifications: Version 0.7 Document Status: Internal

Tommy Persson (tommy.persson@liu.se)
LiU:AIICS
Combitech
Kockums

June 24, 2021

Contents

1 Intoduction **3**

2 WARAPS API Specifications **3**
2.1 APIs 4
2.2 Topics 4
2.3 Heartbeat/Info Data 4
2.4 Sensors 5
2.5 Tasks 5
2.6 Direct Execution 5
2.7 TST Execution 5
2.8 Delegation 5

3 Topics JSON/MQTT **5**
3.1 Topic Naming Scheme 5
3.2 Naming Conventions 6

4 Topics ROS **6**

5 Heartbeat/Info **6**
5.1 General Heartbeat Message 7
5.2 Sensor Level Message 7
5.3 Direct Execution Level Info 8

6 Sensor API JSON **9**
6.1 Mandatory Sensors 9
6.2 Optional Sensors 9

7 Sensor API ROS **9**
7.1 Mandatory Sensors 9

7.2 Optional Sensors 9
7.3 MQTT Bridge 10

8 General Information (AUTOGENERATED DO NOT EDIT) 10

9 Tasks 10
9.1 General Tasks 10
9.2 image-object-detection 10
9.3 look-at-bearing 11
9.4 look-at-position 11
9.5 look-at-positions 12
9.6 move-path 12
9.7 move-to 12
9.8 observe-position 12
9.9 search-area 13
9.10 search-areas 13
9.11 search-around-position 14
9.12 Vehicle Type Specific Tasks 14
9.13 land 14
9.14 navigate 14
9.15 take-off 15

10 JSON Examples for Tasks 15
10.1 General Tasks 15
10.1.1 look-at-bearing 15
10.1.2 look-at-position 15
10.1.3 look-at-positions 16
10.1.4 move-path 17
10.1.5 move-to 18
10.1.6 observe-position 18
10.1.7 search-area 19
10.1.8 search-areas 20
10.1.9 search-around-position 22

11 JSON Examples for TST and start-tst command 22
11.1 General Tasks 22
11.1.1 look-at-angle 22
11.1.2 look-at-position 23
11.1.3 look-at-positions 23
11.1.4 move-path 24
11.1.5 move-to 25
11.1.6 observe-position 26
11.1.7 search-area 26
11.1.8 search-areas 27
11.1.9 search-around-position 29

12 Direct Execution API JSON 30
12.1 Commands 30
12.2 Task Command Messages 31
12.2.1 ping 31

12.2.2 signal-task . . . 31
12.2.3 start-task . . . 31
12.3 Response Messages . . . 31
12.3.1 ping response . . . 31
12.3.2 signal-task response . . . 32
12.3.3 start-task response . . . 32
12.4 Feedback Messages . . . 32

13 Direct Execution API ROS 32
13.1 Plugin in C++ using the available libraries . . . 32
13.2 Use an action interface . . . 33
13.3 Use JSON Interface via MQTT bridge . . . 33
13.3.1 Start MQTT Bridge . . . 33
13.3.2 Topics JSON . . . 33
13.3.3 Service Call Sending Response and Feedback . . . 33

14 TST Execution API JSON 34
14.1 Topics . . . 34
14.2 TST Execution Commands . . . 35
14.3 TST Command Messages . . . 35
14.3.1 Signal a TST Node . . . 35
14.3.2 Signal a TST Tree . . . 35
14.3.3 Signal a Unit/Agent . . . 35
14.3.4 Start a TST start-tst . . . 35
14.4 Response Messages . . . 36
14.4.1 Signal TST Node response . . . 36
14.4.2 Signal TST Tree response . . . 36
14.4.3 Signal TST Unit/Agent response . . . 36
14.4.4 Start TST Response . . . 36
14.5 Feedback Messages . . . 36

15 TST Execution API ROS 37

1 Intoduction

To appear in later versions.

2 WARAPS API Specifications

This document has WORK IN PROGRESS/DRAFT status!

MAIN PRINCIPLE: Try to minimize work so do extensions in a general way but in a way that works with the existing system. So we can in the existing system do smaller refactoring to adapt. The important thing is to not disallow the general optimal solution. But start with some hard coding or assumptions of default just to be able to test and close the loop. Do not freeze unneccessary, be AGILE!

Some design decisions:

- Robot or Agents send regular heartbeat messages with some information on a topic
- Sending a heartbeat message allows for registering the availability of the agent to the rest of the system.
- Communication requiring a response should be asynchronous
- Use UUID to identify a communication message
- Unreliable communication is handled by resending commands with the same UUID until we give up or a response has arrived.

2.1 APIs

The following APIs will in the finished design be defined:

- Robot/agent/system JSON API heartbeat part
- Robot/agent/system ROS API heartbeat part
- Sensor API JSON
- Sensor API ROS
- Robot/agent/system JSON API direct execution part
- Robot/agent/system ROS API direct execution
- TST execution JSON API
- TST execution ROS API
- Robot/agent/system JSON API delegation part
- Robot/agent/system ROS API delegation part
- Delegation JSON API
- Delegation ROS API

In this document we are only describing the following subset of these APIs:

- Robot/agent/system JSON API heartbeat part
- Robot/agent/system ROS API heartbeat part
- Sensor API JSON
- Sensor API ROS
- Robot/agent/system JSON API direct execution part
- Robot/agent/system ROS API direct execution
- TST execution JSON API
- TST execution ROS API
- Delegation JSON API
- Delegation ROS API

2.2 Topics

- Topics JSON/MQTT[1]
- Topics ROS[2]

2.3 Heartbeat/Info Data

A robot/agent/system needs to regularly send out information about itself. The same JSON data is also sent in the ROS API but then send as std_msgs/String on a ROS topic.

[1]`<doc/topics_json.md>`
[2]`<doc/topics_ros.md>`

See: - Heartbeat/info[3]

2.4 Sensors

- Sensor API JSON/MQTT[4]
- Sensor API ROS[5]

2.5 Tasks

- Tasks that can be implemented by a robot/agent/system[6]

2.6 Direct Execution

- Direct Execution API JSON[7]
- Direct Execution API ROS[8]

2.7 TST Execution

- TST Execution API JSON[9]
- TST Execution API ROS[10]

2.8 Delegation

- Delegation API JSON[11]
- Delegation API ROS[12]

3 Topics JSON/MQTT

3.1 Topic Naming Scheme

Naming scheme PREFIX1/PREFIX2/PREFIX3/PREFIX4/UNIT/#

For sensors: PREFIX1/PREFIX2/PREFIX3/PREFIX4/UNIT/sensor/SENSOR

For heartbeat/info messages: PREFIX1/PREFIX2/PREFIX3/PREFIX4/UNIT/XXX

- PREFIX1: naming of context, can be used to have parallel experiments/demos. E.g: waraps-demo-2021, waraps, aiics-testing, and so on.
- PREFIX2: In this case unit to indicate that it is a agent/robot/system that have heartbeat.
- PREFIX3: type of unit: surface, air, subsurface, ground.
- PREFIX4: Indicate role. Possible values: real, simulation, playback...
- UNIT: Name like: usvc2, piraya0, dji1, airpelagosystem0, ...

[3]`<doc/heartbeat_info.md>`
[4]`<doc/sensor_api_json.md>`
[5]`<doc/sensor_api_ros.md>`
[6]`<doc/tasks.md>`
[7]`<doc/direct_execution_api_json.md>`
[8]`<doc/direct_execution_api_ros.md>`
[9]`<doc/tst_execution_api_json.md>`
[10]`<doc/tst_execution_api_ros.md>`
[11]`<doc/delegation_api_json.md>`
[12]`<doc/delegation_api_ros.md>`

- SENSOR: Name or reference to real sensor virtual sensor. Examples: velocity, geopose, batteri_state, executing_tasks, executing_tst_nodes.

Examples:

- waraps/unit/air/simulation/dji0/heartbeat
- waraps/unit/air/simulation/dji0/sensor_info
- waraps/unit/air/simulation/dji0/direct_execution_info
- waraps/unit/air/simulation/dji0/tst_execution_info
- waraps/unit/air/simulation/dji0/sensor/position
- waraps/unit/air/simulation/dji0/sensor/spatial
- waraps/unit/air/simulation/dji0/sensor/speed
- waraps/unit/air/simulation/dji0/sensor/heading
- waraps/unit/air/simulation/dji0/sensor/course
- waraps/unit/air/simulation/dji0/sensor/camera_url
- waraps/unit/air/simulation/dji0/exec/command
- waraps/unit/air/simulation/dji0/exec/response
- waraps/unit/air/simulation/dji0/exec/feedback

3.2 Naming Conventions

- Topics starts with a name, e.g. waraps/ not /waraps/
- Topics have all lowercase and separates words with underscore _, e.g. .../sensor/camera_data.

4 Topics ROS

For sensor topics see Sensor API ROS[13]

The heartbeat/info messages in string format is sent on:

```
n.advertise<lrs_msgs_common::TopicMsg>("/heartbeat_info", 10);
```

The definition of TopicMsg is:

```
string topic
string msg
```

So the topic used in the JSON/MQTT API is put on the msg also. That is so that it can be forwarded to the JSON/MQTT part of the system.

WORK IN PROGRESS, TO BE EXTENDED.

5 Heartbeat/Info

Design decisions:

- One ”heartbeat” message indicating existence of agent.
- One info message on each API level, maybe less update rate the the general HeartBeat message.
- The ”heartbeat” message contains a list of levels it is registered on.

[13]`<sensor_api_ros.md>`

- The ”info” messages should have an agent-uuid field that is changed each time the agent reboots. So we can now if it for example remember all the tasks it has executed or is executing.
- Use different heartbeat/info messages for each level and send them for all levels you want to register on.

5.1 General Heartbeat Message

- Time stamp in double (time.time() in Python, seconds in double)
- Name
- Rate in Hz, expected rate the heartbeat is sent
- A type field indicates what type of message it is. -List of levels to register on: sensor, direct execution, tst execution, delegation. This means the agent is intended to work on these levels but the info messages then indicate if it is available on that level. For example when a boat is out of communication range it will not send info messages on sensor and direct execution level.
- Agent-uuid, should change when rebooting and losing state information
- The possible values for agent-type is: air, ground”, surface, subsurface”
- TOPIC: .../UNIT/heartbeat

```
{
    "agent-type": "surface",
    "agent-uuid": "b992552b-90b0-4911-8707-d6f808362bd2",
    "levels": [
        "sensor",
        "direct execution"
    ],
    "name": "piraya0",
    "rate": 1.0,
    "stamp": 1614080475.1084013,
    "type": "HeartBeat"
}
```

5.2 Sensor Level Message

- Name
- List of sensor data that is provided
- Rate in Hz, expected rate the SensorInfo is sent
- Works as a heartbeat for the sensor level.
- The string in sensor-data-provided is the string used in sensor topic. All sensor topics is under UNIT/sensor/. E.g: piraya0/sensor/position.
- TOPIC: UNIT/sensor_info.

```
{
    "name": "piraya0",
    "rate": 1.0,
    "sensor-data-provided": [
        "position",
        "speed",
        "executing_tasks"
    ],
    "stamp": 1614181737.0478,
    "type": "SensorInfo"
}
```

```
{
   "name" : "dji0",
   "rate" : 1,
   "sensor-data-provided" : [
      "position",
      "speed",
      "battery_state",
      "executing_tasks",
      "executing_tst_nodes"
   ],
   "stamp" : 1614181499.8100259,
   "type" : "SensorInfo"
}
```

5.3 Direct Execution Level Info

- Name
- List of tasks that are supported and for each task a list of signals it understands.
- Rate in Hz, expected rate the DirectExecutionInfo is sent
- List of tasks currently executing.
- Works as a heartbeat for the direct execution level.
- TOPIC: UNIT/direct_execution_info

```
{
    "name": "piraya0",
    "rate": 1.0,
    "type": "DirectExecutionInfo",
    "stamp": 1614080836.5142891,
    "tasks-available": [
        {
            "name": "navigate",
            "signals": [
                "$abort",
                "$enough",
                "$pause"
            ]
        },
        {
            "name": "look-at",
            "signals": [
                "$abort",
                "$enough"
            ]
        }
    ],
    "tasks-executing": [
        {
            "task-name": "navigate",
            "task-uuid": "b6161aee-a90f-4f5c-8a22-7d814725a6e2"
        }
    ]
}
```

6 Sensor API JSON

The robot/agent/system pushes out regular sensor data on MQTT topics.

6.1 Mandatory Sensors

The following sensors are mandatory: positon, heading, course, speed

For example the topics for a dji:

- waraps/unit/air/simulation/dji0/sensor/position: GeoPoint
- waraps/unit/air/simulation/dji0/sensor/heading : float
- waraps/unit/air/simulation/dji0/sensor/course : float
- waraps/unit/air/simulation/dji0/sensor/speed : float

6.2 Optional Sensors

Example of optionally sensors:

- waraps/unit/air/simulation/dji0/sensor/camera_url: string
- waraps/unit/air/simulation/dji0/sensor/videoserver_url: string

7 Sensor API ROS

The robot/agent/system pushes out regular sensor data on ROS topics. The name of the topic is:

- UNIT/sensor/SENSOR (e.g: /dji0/sensor/speed)

7.1 Mandatory Sensors

The following sensors are mandatory: positon, heading, course, speed

For example the topics for a dji:

- /dji0/sensor/position: geographic_msgs/GeoPoint
- /dji0/sensor/heading : std_msgs/Float32
- /dji0/sensor/course : std_msgs/Float32
- /dji0/sensor/speed : std_msgs/Float32

7.2 Optional Sensors

Example of optional sensors:

- /dji0/sensor/camera_url: std_msgs/String

- /dji0/sensor/videoserver_url: std_msgs/String
- /dji0/sensor/battery_state: sensor_msgs/BatteryState

7.3 MQTT Bridge

To get the sensor data to the MQTT broker and coded in JSON a bridge program has to be used. The one available can be started with the following docker-compose.yml file:

```
---
version: '3.3'
services:

  json-api:
    image: gitlab.liu.se:5000/lrs/waraps_docker_images/lrs-json-api-bridge:latest
    command: roslaunch lrs_json_api_bridge json_api.launch  mqtt_host:=broker.waraps.org m
    network_mode: host
    security_opt:
      - "apparmor:unconfined"
    environment:
      - ROS_MASTER_URI=http://localhost:11311
      - ROS_IP=127.0.0.1
```

But change the prefix flags to the correct value. NS is the namespace for the robot. So for example: ”/dji0”, ”/leo1”, ...

8 General Information (AUTOGENERATED DO NOT EDIT)

This page documents all task specifications used by the JSON/MQTT and ROS APIs. It is generated directly from the specifications files, and should serve as the most up-to-date specification of the tasks and their parameters.

9 Tasks

9.1 General Tasks

9.2 image-object-detection

Description: Run detection on an image input stream. Result is put out as a sensor. Send $enough to stop the current detection process. Also the result can be seen in an image stream whose address is found using sensors.

Tags: None

Task Parameters:

- **image-input-url** (string): If non empty the URL for the image stream to do the detections on.

- **image-input-ros-topic** (string): The ROS topic for the input image stream. if given it overrides the image-input-url
- **score** (float64): The minimum sore 0-1 to output a detection.
- **annotation** (string): The values: debug, standard
- **detections-sensor-topic** (string): The ROS topic name for the sensor. Given as full path or reltive to the namespace. Example: sensor/detections
- **object-types** (strings): A list of the object types to detect.
- **wanted-detection-rate** (int32): The wanted detection rate.
- **salient-points-topic** (string): The ROS topic name for the sensor. Given as full path or reltive to the namespace. Example: sensor/detections
- **salient-score** (float64): The minimum sore 0-1 to output a salient point.

9.3 look-at-bearing

Description: This task causes the platform with a camera to point that camera using the specified bearing. The task is intended for cases where one should for example capture video, so rather than pointing once, it encapsulates a process where the camera is continuously adjusted. The task will only terminate when a signal $enough is sent to the task.

Tags: payload

Task Parameters:

- **bearing** (float64): Bearing angle in degree. Positive is to the right (clockwise). If absoulte bearing is used zero degre is north
- **use-absolute-bearing** (bool): If true then bearing angle is relative to north. Otherwise it is relative to the vehicles direction.
- **tilt** (float64): Tilt angle in degree. Positive is up.
- **tilt-body-relative** (bool): If true then the tilt angle is relative to the body of the vehicle. Otherwise it is relative to the world (horisontal plane). If this flag is false and the camera cannot be controlled relative to the horisontal plane than act like the parameter was true.

9.4 look-at-position

Description: The agent will try to observe a position by moving the camera (it is only allowed to move the camera). It will try to observe it until it gets an $enough signal. It will send back status in the feedback message telling if the position is in view. Possible values for status relative to that: in-view-center, in-view, not-in-view.

Tags: None

Task Parameters:

- **geopoint** (geopoint): The position to observe.
- **named-position** (string): If defined look at this position (that can change). If the string start with / then it is the position of that unit that is used instead.

9.5 look-at-positions

Description: The agent will try to observe a position or a set of positions by moving the camera (it is only allowed to move the camera). It will try to observe it until it gets an $enough signal. It will send back status in the feedback message telling if the position is in view. Possible values for status relative to that: in-view-center, in-view, not-in-view. If more than one position is given as argument then the position nearest to the vehicle should be selected.

Tags: None

Task Parameters:

- **geopoints** (geopoints): The positions to observe.
- **named-positions** (strings): If defined look at this position (that can change). If the string start with / then it is the position of that unit that is used instead.

9.6 move-path

Description: This node moves the agent along a path. It is allowed to prepare for moving inside the node, for example doing take-off before flying. Also the specified altitudes is the minimal altitude, the flying should find postiions higher if the position is occupied with obstacles.

Tags: vehicle

Task Parameters:

- **waypoints** (geopoints): The positions to move to
- **speed** (string): Qualitative speed level. Possible values are 'fast', 'standard' and 'slow', and the meaning of these parameters is platform-dependent.

9.7 move-to

Description: This node moves the agent to a specific position. It is allowed to prepare for moving inside the node, for example doing take-off before flying. Also the specified altitude is the minimal altitude, the flying should find postiions higher if the position is occupied with obstacles.

Tags: vehicle

Task Parameters:

- **waypoint** (geopoint): The position to move to
- **speed** (string): Qualitative speed level. Possible values are 'fast', 'standard' and 'slow', and the meaning of these parameters is platform-dependent.

9.8 observe-position

Description: The agent will try to observe a given position. It will observe it until it gets an $enough signal.

Tags: None

Task Parameters:

- **geopoint** (geopoint): The position to observe.
- **speed** (string): Qualitative specification of speed. The meaning is platform dependent. Possible valuess: fast, standard, slow
- **distance** (float64): If given or if greater than 0.0 then use this radius for circling the obervation position for vehicles that needs to move while observing the position. For vehicles that can hover uses this is the distance from which to observe.
- **height** (float64): If vehicles can hover this is the wanted height above the position to observe.
- **duration** (int32): Time to observe the position in seconds. If negative never stop and an $enough signal have to be sent to finish the TST execution.

9.9 search-area

Description: This node specifies a search of an area given by a list of geopoints. The area is 'closed' by assuming a segment between the last and first position in the list of geopoints. It is allowed to prepare for moving inside the node, for example doing take-off before flying.

Tags: vehicle

Task Parameters:

- **area** (geopoints): The area to search.
- **speed** (string): Qualitative speed level. Possible values are 'fast', 'standard' and 'slow', and the meaning of these parameters is platform-dependent.
- **target-type** (string): The type of the search target. Possible values: human, boat, ship, car, ...
- **area-type** (string): The type of the search target. Possible values: water, beach, forrest, field
- **target-size** (float64): The estimates size of the search target in meter.

9.10 search-areas

Description: This node specifies a search of a set of area. The areas are 'closed' by assuming a segment between the last and first position in the list of geopoints. It is allowed to prepare for moving inside the node, for example doing take-off before flying.

Tags: vehicle

Task Parameters:

- **areas** (geopointarrays): The areas to search.
- **speed** (string): Qualitative speed level. Possible values are 'fast', 'standard' and 'slow', and the meaning of these parameters is platform-dependent.
- **target-type** (string): The type of the search target. Possible values: human, boat, ship, car, ...
- **area-type** (string): The type of the search area given as a list of tags. Possible tag values: water, beach, forrest, field
- **target-size** (float64): The estimates size of the search target in meter.

9.11 search-around-position

Description: This node specifies a search of an area given by a list of geopoints. It is allowed to prepare for moving inside the node, for example doing take-off before flying.

Tags: vehicle

Task Parameters:

- **geopoint** (geopoint): The area to search.
- **radius** (float64): The radius around the geopoint to focus on.
- **speed** (string): Qualitative speed level. Possible values are ’fast’, ’standard’ and ’slow’, and the meaning of these parameters is platform-dependent.
- **target-type** (string): The type of the search target. Possible values: human, boat, ship, car, ...
- **target-size** (float64): The estimates size of the search target in meter.

9.12 Vehicle Type Specific Tasks

9.13 land

Description: Causes the platform to land.

Tags: vehicle

Task Parameters:

- **p** (geopoint): The position at which the platform should land, if land-at-current-position-flag is false.
- **land-at-current-position-flag** (bool): If this flag is true, land at the current position of the aircraft. Otherwise, land at the position given by p
- **z** (float64): The offset from the default landing altitude to use in simulations. Default value is 0.0
- **heading** (float64): Specify this for fixed-wing platforms that require a heading in order to land. Helicopters may ignore this parameter.

9.14 navigate

Description: Navigate through or to a sequence of waypoints. Depending on parameter flags repeat the sequence or end in a holding pattern or just end. Currently there is just one speed, that will be changed to one start speed and one speed per waypoint.

Tags: None

Task Parameters:

- **waypoints** (geopoints): Way points to navigate through or to.
- **commanded-speed** (float64s): If the commanded-speed parameter is specified use the value in the list for commanded speed for the corresponding waypoint.
- **speed** (strings): Qualitative specification of speed. The meaning is platform dependent. Possible values for the list elements: fast, standard, slow

- **continue-flag** (bool): If true amd loop flag is false and hold flag is false then continue straight ahead after reaching the end waypoint.
- **loop-flag** (bool): If true the loop the waypoints.
- **hold-flag** (bool): If true then do a holding pattern after the waypoints are finished or the TST have been enoughed.
- **straight-line-flag** (bool): If true then try to follow the line between waypoints. Not used from start position to first waypoint.

9.15 take-off

Description: This node ensures that the platform is in the air. If this is already true, nothing will happen. If the platform is on the ground, it will take off to a vehicle-specific altitude.

Tags: vehicle

Task Parameters:

- **height-above-takeoff** (float64): If specified, then climb to the specified altitude after taking off (the value is relative to the takeoff position). Otherwise, the altitude after take-off is unspecified and platform-specific.
- **path** (geopoints): Used in proposal and execution. A sequence of way points to fly through or to.

10 JSON Examples for Tasks

10.1 General Tasks

10.1.1 look-at-bearing

```
{
    "com-uuid": "06143d7a-ceae-4274-8ad5-7df5f75157c8",
    "command": "start-task",
    "execution-unit": "dji0",
    "sender": "commander",
    "task": {
        "name": "look-at-bearing",
        "params": {
            "bearing": 45,
            "tilt": -45,
            "tilt-body-relative": false,
            "use_absoiute_bearing": false
        }
    },
    "task-uuid": "0f2ae2c5-6fc1-465d-b94b-c1adc25a587f"
}
```

10.1.2 look-at-position

```
{
    "com-uuid": "173ba99f-cb4e-4c7f-87d0-c4888b1f1e9d",
    "command": "start-task",
    "execution-unit": "dji0",
    "sender": "commander",
    "task": {
        "name": "look-at-position",
        "params": {
            "geopoint": {
                "altitude": 40.0,
                "latitude": 57.7611363,
                "longitude": 16.6805011,
                "rostype": "GeoPoint"
            }
        }
    },
    "task-uuid": "fded481d-f3a9-46c1-a44e-af205b0190d9"
}
```

10.1.3 look-at-positions

```
{
    "com-uuid": "05b87b68-c361-43f7-b0b8-b1b3543ee908",
    "command": "start-task",
    "execution-unit": "dji0",
    "sender": "commander",
    "task": {
        "name": "look-at-positions",
        "params": {
            "geopoints": [
                {
                    "altitude": 40.0,
                    "latitude": 57.7611363,
                    "longitude": 16.6805011,
                    "rostype": "GeoPoint"
                },
                {
                    "altitude": 40.0,
                    "latitude": 57.7615541,
                    "longitude": 16.6798574,
                    "rostype": "GeoPoint"
                },
                {
                    "altitude": 40.0,
                    "latitude": 57.7615255,
                    "longitude": 16.6784841,
                    "rostype": "GeoPoint"
                },
                {
                    "altitude": 40.0,
                    "latitude": 57.7609703,
                    "longitude": 16.6780335,
                    "rostype": "GeoPoint"
                }
            ]
        }
    },
    "task-uuid": "67c9526d-4abd-49cb-8146-100eba998e04"
}
```

10.1.4 move-path

```
{
    "com-uuid": "0bac359a-7b77-467b-95f9-bbaed320aadd",
    "command": "start-task",
    "execution-unit": "dji0",
    "sender": "commander",
    "task": {
        "name": "move-path",
        "params": {
            "speed": "standard",
            "waypoints": [
                {
                    "altitude": 40.0,
                    "latitude": 57.7611363,
                    "longitude": 16.6805011,
                    "rostype": "GeoPoint"
                },
                {
                    "altitude": 40.0,
                    "latitude": 57.7615541,
                    "longitude": 16.6798574,
                    "rostype": "GeoPoint"
                },
                {
                    "altitude": 40.0,
                    "latitude": 57.7615255,
                    "longitude": 16.6784841,
                    "rostype": "GeoPoint"
                },
                {
                    "altitude": 40.0,
                    "latitude": 57.7609703,
                    "longitude": 16.6780335,
                    "rostype": "GeoPoint"
                },
                {
                    "altitude": 40.0,
                    "latitude": 57.760398,
                    "longitude": 16.6787201,
                    "rostype": "GeoPoint"
                },
                {
                    "altitude": 40.0,
                    "latitude": 57.7604781,
                    "longitude": 16.6799003,
                    "rostype": "GeoPoint"
                }
            ]
        }
    },
    "task-uuid": "15c42e27-6c43-4291-bf10-d55334148e35"
}
```

10.1.5 move-to

```
{
    "com-uuid": "193f5b87-2c88-495e-941e-80182be4451b",
    "command": "start-task",
    "execution-unit": "dji0",
    "sender": "commander",
    "task": {
        "name": "move-to",
        "params": {
            "speed": "standard",
            "waypoint": {
                "altitude": 40.0,
                "latitude": 57.7611363,
                "longitude": 16.6805011,
                "rostype": "GeoPoint"
            }
        }
    },
    "task-uuid": "27ce3be0-972c-4130-b252-4e3f74832497"
}
```

10.1.6 observe-position

```
{
    "com-uuid": "e74ea2f3-db13-442d-a2ee-11c5baba0145",
    "command": "start-task",
    "execution-unit": "dji0",
    "sender": "commander",
    "task": {
        "name": "observe-position",
        "params": {
            "distance": 10.0,
            "duration": 20,
            "geopoint": [
                {
                    "altitude": 40.0,
                    "latitude": 57.7611363,
                    "longitude": 16.6805011,
                    "rostype": "GeoPoint"
                }
            ],
            "height": -1.0,
            "speed": [
                7.0
            ]
        }
    },
    "task-uuid": "46eab756-e0e7-47e3-948b-ad78c795295c"
}
```

10.1.7 search-area

```
{
    "com-uuid": "e16b95da-e3ef-4ba7-99f5-6db1680ad078",
    "command": "start-task",
    "execution-unit": "dji0",
    "sender": "commander",
    "task": {
        "name": "search-area",
        "params": {
            "area": [
                {
                    "altitude": 40.0,
                    "latitude": 57.7611363,
                    "longitude": 16.6805011,
                    "rostype": "GeoPoint"
                },
                {
                    "altitude": 40.0,
                    "latitude": 57.7615541,
                    "longitude": 16.6798574,
                    "rostype": "GeoPoint"
                },
                {
                    "altitude": 40.0,
                    "latitude": 57.7615255,
                    "longitude": 16.6784841,
                    "rostype": "GeoPoint"
                },
                {
                    "altitude": 40.0,
                    "latitude": 57.7609703,
                    "longitude": 16.6780335,
                    "rostype": "GeoPoint"
                }
            ],
            "speed": "standard",
            "target-size": 2.0
        }
    },
    "task-uuid": "dc5adc1f-eb01-40a9-be36-040b5e143e70"
}
```

10.1.8 search-areas

```
{
    "com-uuid": "c6929a0e-ad60-40d9-8f60-9b05d632c0fe",
    "command": "start-task",
    "execution-unit": "op0",
    "sender": "commander",
    "task": {
        "name": "search-areas",
        "params": {
            "areas": [
                [
                    {
                        "altitude": 40.0,
                        "latitude": 57.7611363,
                        "longitude": 16.6805011,
                        "rostype": "GeoPoint"
                    },
                    {
                        "altitude": 40.0,
                        "latitude": 57.7615541,
                        "longitude": 16.6798574,
                        "rostype": "GeoPoint"
                    },
                    {
                        "altitude": 40.0,
                        "latitude": 57.7615255,
                        "longitude": 16.6784841,
                        "rostype": "GeoPoint"
                    },
                    {
                        "altitude": 40.0,
                        "latitude": 57.7609703,
                        "longitude": 16.6780335,
                        "rostype": "GeoPoint"
                    }
                ],
                [
                    {
                        "altitude": 40.0,
                        "latitude": 57.7611363,
                        "longitude": 16.6805011,
                        "rostype": "GeoPoint"
                    },
                    {
```

```
                    "altitude": 40.0,
                    "latitude": 57.7615541,
                    "longitude": 16.6798574,
                    "rostype": "GeoPoint"
                },
                {
                    "altitude": 40.0,
                    "latitude": 57.7615255,
                    "longitude": 16.6784841,
                    "rostype": "GeoPoint"
                },
                {
                    "altitude": 40.0,
                    "latitude": 57.7609703,
                    "longitude": 16.6780335,
                    "rostype": "GeoPoint"
                }
            ],
            [
                {
                    "altitude": 40.0,
                    "latitude": 57.7611363,
                    "longitude": 16.6805011,
                    "rostype": "GeoPoint"
                },
                {
                    "altitude": 40.0,
                    "latitude": 57.7615541,
                    "longitude": 16.6798574,
                    "rostype": "GeoPoint"
                },
                {
                    "altitude": 40.0,
                    "latitude": 57.7615255,
                    "longitude": 16.6784841,
                    "rostype": "GeoPoint"
                },
                {
                    "altitude": 40.0,
                    "latitude": 57.7609703,
                    "longitude": 16.6780335,
                    "rostype": "GeoPoint"
                }
            ]
        ],
        "speed": "standard",
        "target-size": 4.0
    }
},
"task-uuid": "3f800056-7340-4607-8660-930297adc8b7"
```

```
}
```

10.1.9 search-around-position

```
{
    "com-uuid": "d37a2e04-c739-4691-a667-de8de8f3e7e8",
    "command": "start-task",
    "execution-unit": "dji0",
    "sender": "commander",
    "task": {
        "name": "search-around-position",
        "params": {
            "geopoint": {
                "altitude": 28.9,
                "latitude": 57.7611363,
                "longitude": 16.6805011,
                "rostype": "GeoPoint"
            },
            "radius": 25.0,
            "speed": "standard",
            "target-size": 2.0,
            "target-type": "human"
        }
    },
    "task-uuid": "f0dbb936-0c64-4b73-b2c5-85f718e1d7ba"
}
```

11 JSON Examples for TST and start-tst command

11.1 General Tasks

11.1.1 look-at-angle

```
{
    "com-uuid": "13023f33-3eff-4348-b6ef-dfd169ee4692",
    "command": "start-tst",
    "receiver": "op0",
    "sender": "commander",
    "tst": {
        "children": [],
        "common_params": {
            "execunit": "/op0",
            "node-uuid": "5184d315-15c4-42d4-a647-fa74f794c03b"
        },
        "name": "look-at-angle",
        "params": {
            "pan": 45,
            "pan-body-relative": false,
            "tilt": -45,
            "tilt-body-relative": false
        }
    }
}
```

11.1.2 look-at-position

```
{
    "com-uuid": "60ab5b23-b6a8-4b1a-8b9f-1f4c9178aa1d",
    "command": "start-tst",
    "receiver": "op0",
    "sender": "commander",
    "tst": {
        "children": [],
        "common_params": {
            "execunit": "/op0",
            "node-uuid": "e912ac32-d1a9-4e58-946a-a9a67989e849"
        },
        "name": "look-at-position",
        "params": {
            "geopoint": {
                "altitude": 40.0,
                "latitude": 57.7611363,
                "longitude": 16.6805011,
                "rostype": "GeoPoint"
            }
        }
    }
}
```

11.1.3 look-at-positions

```
{
    "com-uuid": "deac6dff-eeb4-43f9-b404-85ce3d1f154d",
    "command": "start-tst",
    "receiver": "op0",
    "sender": "commander",
    "tst": {
        "children": [],
        "common_params": {
            "execunit": "/op0",
            "node-uuid": "cb038c7c-853d-4728-8a28-1e8b1b906742"
        },
        "name": "look-at-positions",
        "params": {
            "geopoints": [
                {
                    "altitude": 40.0,
                    "latitude": 57.7611363,
                    "longitude": 16.6805011,
                    "rostype": "GeoPoint"
                },
                {
                    "altitude": 40.0,
                    "latitude": 57.7615541,
                    "longitude": 16.6798574,
                    "rostype": "GeoPoint"
                },
                {
                    "altitude": 40.0,
                    "latitude": 57.7615255,
                    "longitude": 16.6784841,
                    "rostype": "GeoPoint"
                },
                {
                    "altitude": 40.0,
                    "latitude": 57.7609703,
                    "longitude": 16.6780335,
                    "rostype": "GeoPoint"
                }
            ]
        }
    }
}
```

11.1.4 move-path

```
{
    "com-uuid": "328ffdb3-6dc9-4540-8ab2-21f8ac1262be",
    "command": "start-tst",
    "receiver": "op0",
    "sender": "commander",
    "tst": {
        "children": [],
        "common_params": {
            "execunit": "/op0",
            "node-uuid": "1a292ec4-67ec-4fef-92ae-384246c2431c"
        },
        "name": "move-path",
        "params": {
            "speed": "standard",
            "waypoints": [
                {
                    "altitude": 40.0,
                    "latitude": 57.7611363,
                    "longitude": 16.6805011,
                    "rostype": "GeoPoint"
                },
                {
                    "altitude": 40.0,
                    "latitude": 57.7615541,
                    "longitude": 16.6798574,
                    "rostype": "GeoPoint"
                },
                {
                    "altitude": 40.0,
                    "latitude": 57.7615255,
                    "longitude": 16.6784841,
                    "rostype": "GeoPoint"
                },
                {
                    "altitude": 40.0,
                    "latitude": 57.7609703,
                    "longitude": 16.6780335,
                    "rostype": "GeoPoint"
                },
                {
                    "altitude": 40.0,
                    "latitude": 57.760398,
                    "longitude": 16.6787201,
                    "rostype": "GeoPoint"
                },
                {
                    "altitude": 40.0,
                    "latitude": 57.7604781,
                    "longitude": 16.6799003,
                    "rostype": "GeoPoint"
                }
            ]
        }
    }
}
```

11.1.5 move-to

```
{
    "com-uuid": "e680c1f5-af53-43f2-9d73-bb4cf74d21c4",
    "command": "start-tst",
    "receiver": "op0",
    "sender": "commander",
    "tst": {
        "children": [],
        "common_params": {
            "execunit": "/op0",
            "node-uuid": "f2d8abee-5080-4844-a710-8fd4d136e765"
        },
        "name": "move-to",
        "params": {
            "speed": "standard",
            "waypoint": {
                "altitude": 40.0,
                "latitude": 57.7611363,
                "longitude": 16.6805011,
                "rostype": "GeoPoint"
            }
        }
    }
}
```

11.1.6 observe-position

```
{
    "com-uuid": "8cd3944d-d767-43c5-92eb-55e3ce8cba2b",
    "command": "start-tst",
    "receiver": "op0",
    "sender": "commander",
    "tst": {
        "children": [],
        "common_params": {
            "execunit": "/op0",
            "node-uuid": "a4aafcee-9090-4fef-87f8-12fddf362237"
        },
        "name": "observe-position",
        "params": {
            "distance": 10.0,
            "duration": 20,
            "geopoint": [
                {
                    "altitude": 40.0,
                    "latitude": 57.7611363,
                    "longitude": 16.6805011,
                    "rostype": "GeoPoint"
                }
            ],
            "height": -1.0,
            "speed": [
                7.0
            ]
        }
    }
}
```

11.1.7 search-area

```
{
    "com-uuid": "79d118b0-501a-4156-83c2-95fd9aaafbf2",
    "command": "start-tst",
    "receiver": "op0",
    "sender": "commander",
    "tst": {
        "children": [],
        "common_params": {
            "execunit": "/op0",
            "node-uuid": "405f246b-3101-45e2-ac36-cf8176850082"
        },
        "name": "search-area",
        "params": {
            "area": [
                {
                    "altitude": 40.0,
                    "latitude": 57.7611363,
                    "longitude": 16.6805011,
                    "rostype": "GeoPoint"
                },
                {
                    "altitude": 40.0,
                    "latitude": 57.7615541,
                    "longitude": 16.6798574,
                    "rostype": "GeoPoint"
                },
                {
                    "altitude": 40.0,
                    "latitude": 57.7615255,
                    "longitude": 16.6784841,
                    "rostype": "GeoPoint"
                },
                {
                    "altitude": 40.0,
                    "latitude": 57.7609703,
                    "longitude": 16.6780335,
                    "rostype": "GeoPoint"
                }
            ],
            "speed": "standard",
            "target-size": 2.0
        }
    }
}
```

11.1.8 search-areas

```
{
    "com-uuid": "1ae14698-551e-4f18-8414-a3c6679c1983",
    "command": "start-tst",
    "receiver": "op0",
    "sender": "commander",
    "tst": {
        "children": [],
        "common_params": {
            "execunit": "/op0",
        "node-uuid": "0eafd9c8-4ee6-4023-b537-84b48e630753"
    },
    "name": "search-areas",
    "params": {
        "areas": [
            [
                {
                    "altitude": 40.0,
                    "latitude": 57.7611363,
                    "longitude": 16.6805011,
                    "rostype": "GeoPoint"
                },
                {
                    "altitude": 40.0,
                    "latitude": 57.7615541,
                    "longitude": 16.6798574,
                    "rostype": "GeoPoint"
                },
                {
                    "altitude": 40.0,
                    "latitude": 57.7615255,
                    "longitude": 16.6784841,
                    "rostype": "GeoPoint"
                },
                {
                    "altitude": 40.0,
                    "latitude": 57.7609703,
                    "longitude": 16.6780335,
                    "rostype": "GeoPoint"
                }
            ],
            [
                {
                    "altitude": 40.0,
                    "latitude": 57.7611363,
                    "longitude": 16.6805011,
                    "rostype": "GeoPoint"
                },
                {
                    "altitude": 40.0,
                    "latitude": 57.7615541,
                    "longitude": 16.6798574,
                    "rostype": "GeoPoint"
                },
                {
                    "altitude": 40.0,
                    "latitude": 57.7615255,
                    "longitude": 16.6784841,
                    "rostype": "GeoPoint"
                },
                    {
                        "altitude": 40.0,
                        "latitude": 57.7609703,
                        "longitude": 16.6780335,
                        "rostype": "GeoPoint"
                    }
                ],
                [
                    {
                        "altitude": 40.0,
                        "latitude": 57.7611363,
                        "longitude": 16.6805011,
                        "rostype": "GeoPoint"
                    },
                    {
                        "altitude": 40.0,
                        "latitude": 57.7615541,
                        "longitude": 16.6798574,
                        "rostype": "GeoPoint"
                    },
                    {
                        "altitude": 40.0,
                        "latitude": 57.7615255,
                        "longitude": 16.6784841,
                        "rostype": "GeoPoint"
                    },
                    {
                        "altitude": 40.0,
                        "latitude": 57.7609703,
                        "longitude": 16.6780335,
                        "rostype": "GeoPoint"
                    }
                ]
            ],
            "speed": "standard",
            "target-size": 4.0
        }
    }
}
```

11.1.9 search-around-position

```
{
    "com-uuid": "55c92356-4451-46af-b74c-65e8ca1ae3ee",
    "command": "start-tst",
    "receiver": "op0",
    "sender": "commander",
    "tst": {
        "children": [],
        "common_params": {
            "execunit": "/op0",
            "node-uuid": "36b77b0b-587d-464c-8f85-ef1e555dab26"
        },
        "name": "search-around-position",
        "params": {
            "geopoint": {
                "altitude": 28.9,
                "latitude": 57.7611363,
                "longitude": 16.6805011,
                "rostype": "GeoPoint"
            },
            "radius": 25.0,
            "speed": "standard",
            "target-size": 2.0,
            "target-type": "human"
        }
    }
}
```

12 Direct Execution API JSON

The communication uses MQTT. The commands are JSON, that is a text format.

12.1 Commands

There are four commands:

- ping
- signal-task
- start-task
- query-status : return the status of the task (running, failed, success, unknown)

The commands are sent in a command message which:

- Can optionally contain start and end time (more in later versions).
- If no start time or end time is given, just execute directly and as fast as possible but you do not need to optimize. Regular as fast as possible.

Response message: - Returns agent-uuid. - A response string field. Possible values: running, failed, finished - A string for failure reason

Feedback messages: - Update of the status of the task - Should contain agent-uuid, task-uuid and status

Protocol: - task-uuid should be optionally sent in start-task, if not sent a generated task-uuid will be returned in the response message. - If a response is not arriving re-send the command with the same com-uuid. - The receiver of commands has to keep track of the history of commands so it can answer if a command is unknown or finished when the command is query-status. - The changing agent-uuid when rebooting is used to detect that the agent has lost its history of commands.

Naming of the topics to use: - Commands: dji0/exec/command - Response: dji0/exec/response - Feedback: dji0/exec/feedback

12.2 Task Command Messages

12.2.1 ping

A response message with pong as response should be sent when this message arrives.

```
{
    "com-uuid": "7300d049-176f-4dcb-b0bf-151980d5611a",
    "command": "ping",
    "sender": "commander"
}
```

12.2.2 signal-task

The buildtin signals are $enough, $pause, $continue and $abort.

Signal names starting with $ is system signals. Other names are specific for the task.

```
{
    "com-uuid": "7300d049-176f-4dcb-b0bf-151980d5611a",
    "command": "signal-task",
    signal: $abort,
    "sender": "commander",
    "task-uuid": "9b4c701c-e8a2-4a92-9370-87c8f57e0dcd"
}
```

```
{
    "com-uuid": "13b5e90e-a048-4adb-9015-b4629dbf8f38",
    "command": "signal-executor",
    signal: battery-is-charged,
    "sender": "commander",
    "uuid": "9b4c701c-e8a2-4a92-9370-87c8f57e0dcd"
}
```

12.2.3 start-task

For examples see: start-task examples[14].

12.3 Response Messages

12.3.1 ping response

```
{
    "agent-uuid": "3887ced0-4e2d-4be4-aeff-6ba7242853a1",
```

[14]<tasks.md#json-examples-for-tasks>

```
    "com-uuid": "1c5284da-b803-49d1-8c39-b96ef00603e7",
    "response": "pong",
    "response-to": "29c24bbf-88f0-4de2-97e9-4bd0d5f6bf23"
}
```

12.3.2 signal-task response

12.3.3 start-task response

```
{
    "agent-uuid": "3887ced0-4e2d-4be4-aeff-6ba7242853a1",
    "com-uuid": "1c5284da-b803-49d1-8c39-b96ef00603e7",
    "fail-reason": "",
    "response": "running",
    "response-to": "29c24bbf-88f0-4de2-97e9-4bd0d5f6bf23",
    "task-uuid": "5c0ed702-2f6a-46b4-a1a4-ea2e86e04282"
}
```

12.4 Feedback Messages

Update of the status for a task. The possible status strings are: - running - failed - aborted - paused - finished - starting - task specific strings

```
{
    "agent-uuid": "8309d3ef-56fc-4ab0-86b0-9d9705ccf043",
    "com-uuid": "b3c90b08-622c-41b5-948c-80ca0abb49dc",
    "status": "running",
    "task-uuid": "49fb88b8-4d4a-4268-a69a-35e620a1a5f4"
}
```

13 Direct Execution API ROS

More details in later versions of this document.

Talk with tommy.persson@liu.se for deatils about this if you want to plugin a robit using ROS.

Main methods:

- Plugin in C++ using the available libraries
- Use an action interface.
- Using the JSON messages provided on ROS topics via a MQTT Bridge.

13.1 Plugin in C++ using the available libraries

This will be available for people asking for this. It is not possible to describe meaningfully here. You need to check out some code and look at some examples. In the repo lrs_examples therer are code examples for this.

13.2 Use an action interface

This is work in progress with SMARC. A first prototype exists.

More documenation later.

13.3 Use JSON Interface via MQTT bridge

13.3.1 Start MQTT Bridge

```
---
version: '3.3'
services:

  json-api:
    image: gitlab.liu.se:5000/lrs/waraps_docker_images/lrs-json-api-bridge:latest
    command: roslaunch lrs_json_api_bridge json_api.launch  mqtt_host:=broker.waraps.org m
    network_mode: host
    security_opt:
      - "apparmor:unconfined"
    environment:
      - ROS_MASTER_URI=http://localhost:11311
      - ROS_IP=127.0.0.1
```

But change the prefix flags to the correct value. NS is the namespace for the robot. So for example: "/dji0", "/leo1", ...

13.3.2 Topics JSON

Subscribe to in ROS:

- exec_command (String): This is tje JSON command send to this agent.

13.3.3 Service Call Sending Response and Feedback

There are service calls for sending JSON response and feedback messages:

Sending Response

Code example Python:

```
def send_response(unit, response_uuid, task_uuid, response, fail_reason=""):
    msg = {}
    msg["com-uuid"] = str(uuid.uuid4())
    msg["response-to"] = response_uuid
    msg["agent-uuid"] = agent_uuid
    msg["task-uuid"] = task_uuid
    msg["response"] = response
    msg["fail-reason"] = fail_reason
    msgstr = json.dumps(msg, sort_keys=True, indent=4, separators=(',', ': '))
    topic = "exec/response"
    print("send_response:", unit, topic, msgstr)
    try:
        client = rospy.ServiceProxy('send_topic_msg_to_unit', SendTopicMsgToUnit)
        resp = client(unit.lstrip("/"), topic, msgstr)
        print("send_response send_topic_msg RESP:", resp)
    except rospy.ServiceException as e:
        print("Service call failed: %s"%e)
```

Sending Feedback

Code example Python:

```
def send_feedback(unit, task_uuid, status):
    msg = {}
    msg["com-uuid"] = str(uuid.uuid4())
    msg["agent-uuid"] = agent_uuid
    msg["task-uuid"] = task_uuid
    msg["status"] = status
    msgstr = json.dumps(msg, sort_keys=True, indent=4, separators=(',', ': '))
    # print(msgstr)
    topic = "exec/feedback"
    print("Send feedback on topic:", topic)
    try:
        client = rospy.ServiceProxy('send_topic_msg_to_unit', SendTopicMsgToUnit)
        resp = client(unit.lstrip("/"), topic, msgstr)
        print("send_feedback send_topic_msg RESP:", resp)
    except rospy.ServiceException as e:
        print("Service call failed: %s"%e)
```

14 TST Execution API JSON

The communication uses MQTT. The commands are JSON, that is a text format.

Note that this API is towards the TST/Delegation system.

- Execution of fully instantiated TST trees
- The Task execution is the same as for level 2 and used internally in the TST system.
- The communication with command and control is different.
- Fully instantiated TST trees (represented in JSON) are sent to the system and the result is: start of execution possibly on more than one platform or direct failure to start.
- Update of progress during execution including report of success or failure for each node in the tree.
- During execution abort/pause/continue/enough can be sent to specific nodes.
- We might have help command to send to all nodes in a tree.

14.1 Topics

- name/tst/command
- name/tst/response

- name/tst/feedback

The commands are usually sent to op0. So the most common topicas to use are: - op0/tst/command - op0/tst/response - op0/tst/feedback

But it is possible to send it directly to an agent also but the agent might not be running so it is safer to send it to the op0.

14.2 TST Execution Commands

- start-tst

14.3 TST Command Messages

14.3.1 Signal a TST Node

```
{
    "com-uuid": "2b01c8f9-437f-49d1-869b-cd5e3d3bb395",
    "command": "signal-tst-node",
    "node-uuid": "ccb261a7-3446-4e7e-9aea-d9d57eaed793",
    "sender": "commander",
    "signal": "$abort"
}
```

14.3.2 Signal a TST Tree

Send the signal to all TST nodes in the tree with the root node-uuid.

```
{
    "com-uuid": "2b01c8f9-437f-49d1-869b-cd5e3d3bb395",
    "command": "signal-tst-tree",
    "node-uuid": "ccb261a7-3446-4e7e-9aea-d9d57eaed793",
    "sender": "commander",
    "signal": "$abort"
}
```

14.3.3 Signal a Unit/Agent

Send the signal to all running TST nodes on the unit.

```
{
    "com-uuid": "2b01c8f9-437f-49d1-869b-cd5e3d3bb395",
    "command": "signal-unit",
    "unit": "/dji21",
    "sender": "commander",
    "signal": "$abort"
}
```

14.3.4 Start a TST start-tst

See examples[15]

[15] `<tasks.md#json-examples-for-tst-and-start-tst-command>`

It is best to set the node-uuid for the nodes and at least for the top node. The feedback till well about the nodes execution status and a node is then identified using the node-uuid.

14.4 Response Messages

14.4.1 Signal TST Node response

```
{
    "com-uuid": "9dfde776-9d80-4fe9-9e34-32761f5ed545",
    "response": "ok",
    "response-to": "638a433d-07fb-4bcf-af97-228114638907"
}
```

14.4.2 Signal TST Tree response

```
{
    "com-uuid": "9dfde776-9d80-4fe9-9e34-32761f5ed545",
    "response": "ok",
    "response-to": "638a433d-07fb-4bcf-af97-228114638907"
}
```

14.4.3 Signal TST Unit/Agent response

```
{
    "com-uuid": "9dfde776-9d80-4fe9-9e34-32761f5ed545",
    "response": "ok",
    "response-to": "638a433d-07fb-4bcf-af97-228114638907"
}
```

14.4.4 Start TST Response

```
{
    "com-uuid": "38cca78e-a19c-4773-9349-ffbd80ba15e6",
    "fail-reason": "",
    "response": "started",
    "response-to": "a1587d98-b253-4e8a-bb8c-c937693881c0",
    "root-uuid": "fe1fbcee-1cae-4a69-b9f7-83914454a7a7"
}
```

14.5 Feedback Messages

The possible values of the status is: - not-active - active - waiting - running - failed - revoked - paused - aborted - succeeded

Examples:

```
{
    "com-uuid": "ed3717a6-ffef-4129-ad21-ea577117c706",
    "fail-reason": "",
    "node-uuid": "ff200942-73a2-401f-83e2-8397d28facf8",
    "root-flag": true,
    "status": "active"
}
{
    "com-uuid": "068d8c73-9f83-4ac7-9fd0-84b8ba6e80da",
    "fail-reason": "",
    "node-uuid": "ff200942-73a2-401f-83e2-8397d28facf8",
    "root-flag": true,
    "status": "waiting"
}
{
    "com-uuid": "9a1118bb-3b50-4d74-b9e3-deb2c33078f8",
    "fail-reason": "",
    "node-uuid": "ff200942-73a2-401f-83e2-8397d28facf8",
    "root-flag": true,
    "status": "running"
}
```

15 TST Execution API ROS

See the tutorials. They describe how to create a TST tree and how to execute it.