AVA ROBOTICS

Ava® Cloud Service Client API, Programmer's Guide March 2018

This document provides some description of the Client API to Ava Robotics' Robot Management System (RMS). This is meant to be used in conjunction with the "Ava Cloud Service Client API: Reference Guide".

The Robot Management System is the core software of the Ava Cloud Service, which provides connectivity between client applications (e.g. iOS clients) and Ava robots. This document describes how to use the client API, and provides many helpful examples. It is not intended to describe each API call, or all of the possible options for each API call, as that is described in the "Ava Cloud Service Client API: Reference Guide".

Table of Contents

RMS 101 - An Introduction to Basic Concepts

Okay, so the reason that you are reading this document is because you want to build a client application that connects to the RMS and controls robots, right? So what are the basics that you need to know?
First of all, you should know that there is an administrative UI to the RMS that allows you to configure items such as users, as well as perform other administrative functions. This is at **https://{rms-name}/cust_admin**. You need an administrative user account (username/password) in order to log in. Here you can see the status of robots, sessions, and users.

Users

The RMS is configured with user accounts. A user account is what defines a *user*. When a client application connects to the RMS, it must pass the credentials of a user (username/password), and that connection is then bound to that user. The following fields are defined within the RMS for a user account:

- **Username**: a unique string identifier for this user
- **Password**: a login password
- **Sites**: the sites (maps) that the user has access to

- **Video Endpoints**: video address(es) where the user can be reached (e.g. billybob@rp.example.com)

Sessions

The term *session* (e.g. "user session", or "robot session") is used to mean a reserved amount of time that a robot is allocated to a user. During a session the user has exclusive use of the robot. When a user requests a start time and duration for which they would like to have access to a robot, it is a session (sometimes called a reservation). If the user asks for a robot immediately, it is called an *ad-hoc session*. If the user requests a robot for some time in the future, it is considered a *future session*, or future reservation.
When starting an ad-hoc session (starting now), the client specifies whether the robot should autonomously drive to a preset location (/robot/tel/goToRoom) or whether the session should start wherever the robot currently is (/rms/goToMap), typically on its dock.
A single user can only have one session allocated to it at a time, and similarly a robot can only be assigned to a single user session at a time.

Initiating the Video Call

After a session has been created, the client application can initiate the video call. The robot always calls the user, and this is initiated using /robot/tel/dial.
The client can hang up the call with /robot/tel/hangup.

Autonomous Versus Manual (Teleop) Driving

There are two distinct ways that the robot drives, *autonomously* and *manually* (teleop). Autonomous driving is when the robot knows its intended destination, and will drive itself to that destination without any user input. Manual driving is when the user specifically tells the robot to move in a particular direction for a particular amount of time. Autonomous driving is a better user experience, as the movements are more smooth, and the user does not need to be concerned with driving the robot. Manual driving is most useful for making small movements, and less useful for driving from location to location.
Note that the robot has *obstacle detection and avoidance* (OD/OA), which means that the robot detects static (unmoving) and dynamic (moving) obstacles and will not drive into these obstacles. This is true both when the robot is driving autonomously, as well as when a user is manually driving the robot. This means that when a user manually drive's the robot it may not respond to movement requests if the robot believes that this will cause a collision (or will cause the robot to fall down stairs, etc.).
The main API function to have the robot drive autonomously to a location is via /robot/tel/goToRoom. For manual driving, /robot/drive/velocity is used.

Robot Pose: Standing, Sitting, and Looking Around

The robot has the ability to "stand" and "sit". This refers to the ability to move the videoconferencing head up and down. When the robot is interacting with a person that is standing, it is more natural if the robot is taller (standing). Similarly when the robot is interacting with a person who is sitting, it is more natural if the robot is shorter (sitting).
Having the robot "look around" is an important feature. The camera has the ability to tilt up and down, however it does not have the ability to pan (back and forth). Therefore, in order to pan, the robot must be commanded to rotate.
The URI /robot/drive/payloadPose provides the ability to set the robot pose (including standing/sitting, camera tilt, and rotating the robot). The standing/sitting mechanism is usually referred to as *Z-lift* in the

APIs.

WebSocket Connection and Authentication

All communication between a client and the RMS happens through a secure WebSocket connection. For details on WebSockets, see the WebSocket Wiki page. A WebSocket connection is bidirectional, which means that messages can be sent asynchronously from either side of the connection. For the most part, the client makes calls to the RMS and the RMS replies to those calls, however there are some cases where the RMS sends unsolicited *push messages* to the client (normally to inform the client of some error event, e.g. the robot disconnected). The client can also *subscribe* to particular types of messages to get regular updates ("push" messages).
The client must make a WebSocket connection to the RMS using the following format:

```
  URI: "wss://{rms-name}/client" (e.g.
"wss://d010.rms.irobotava.com/client")
  Protocol: "irobot-bcr-scheduler"
```

In addition, the WebSocket connection must pass the username and password in a Basic Authentication Header. Because the WebSocket connection is secure (uses SSL), basic authentication is secure. The username and password credentials must have been previously configured within the RMS. If the credentials are not valid, the connection will be rejected by the RMS.

RMS Client API Message Format

Once a WebSocket connection is established, one may make RMS Client API calls to the RMS over this connection. All of the messages sent over the WebSocket connection use JSON, a human readable, attribute-value pair structure. Before describing the attributes in each message, let's look at an example:
Client requests details about a session:

```
{
  "op": "request",
  "uri": "/rms/getSession",
  "args": {
      "id": 8514
  }
}
```

Server responds with that session's details:

```
{
  "op": "response",
  "uri": "/rms/getSession",
  "response": {
      "session": {
              "id": 8514,
              "startTime": 1369848600000,
              "endTime": 1369852200000,
              "mapName": "Building-10-1",
              "company": "iRobot Corporation",
              "building": "Building 10",
              "floor": "First floor",
              "room": "CR-Brassneck-10-1-162",
              "user": "Eben",
              "defaultVoipID": "c593c471-efde-11e2-b778-0800200c9a66",
              "state": "RESERVED",
              "title": "Discuss TPS report format"
      }
  }
}
```

Note that the request and response messages are both in JSON. Every message is encoded in JSON. Additionally, every message contains the attributes "uri" and "op". "uri" defines the context of the message, and "op" defines the message's operation type. All of the possible "op" values include:

"op" Value	Description
"request"	Request from the client to the RMS to do something.
"response"	Response to a "request", sent from RMS to client.
"subscribe"	Client request to subscribe to receive a certain type of message from the RMS. For example, one can subscribe to receive session state updates by subscribing to the uri /rms/sessionStateUpdate. The RMS will reply with a "subscribed" message, and any time the state changes, the client will receive a "push" notification with the uri /rms/sessionStateUpdate from the RMS.
"subscribed"	Response to a "subscribe" request, sent from the RMS to the client.
"unsubscribe"	Client request to unsubscribe from a certain type of message from the RMS. RMS should reply with a "unsubscribed" message.
"unsubscribed"	Response to an "unsubscribe" request, sent from the RMS to the client.
"push"	Asynchronous messages sent from RMS to client. Some push messages may be sent even if the client hasn't subscribed to them. Other push messages must be subscribed to in order for the RMS to send them to the client.

The following sections describe the format of each type of message.

"request" Message Format

Following are the attributes and values in a "request" message.

"request" Attribute	Value
"op"	"request"
"uri"	URI that defines the message.
"args"	The arguments to be passed in this request.The value is a JSON structure, and the attribute-value pairs within the structure are dependent on the URI. Some URIs do not have any arguments, and therefore do not use "args" at all.

"response" Message Format

Following are the attributes and values in a "response" message. A response message is sent from the RMS to the client in response to a "request" message.

"request" Attribute	Value
"op"	"response"
"uri"	URI that defines the message.
"response"	A JSON structure containing the response information. The details of the response are URI specific.

"errorMessage"	If there is an error satisfying the request, the response will contain this attribute. The value is URI specific, but usually it is a text string to be displayed to the end user.

"subscribe" Message Format

Following are the attributes and values in a "subscribe" message. A client sends a subscribe message to the RMS in order to receive "push" notifications from the RMS for this URI. The RMS will reply with a "subscribed" message.

"subscribe" Attribute	Value
"op"	"subscribe"
"uri"	URI to subscribe to.

"subscribed" Message Format

Following are the attributes and values in a "subscribed" message. The RMS sends a "subscribed" message to the client in response to a "subscribe" request.

"subscribed" Attribute	Value
"op"	"subscribed"
"uri"	URI the user subscribed to.
"errorMessage"	If there is an error satisfying the request, the response will contain this attribute. The value is URI specific, but usually it is a text string to be displayed to the end user.

"unsubscribe" Message Format

Following are the attributes and values in an "unsubscribe" message. A client sends an unsubscribe message to the RMS to no longer receive "push" notifications from the RMS for this URI. The RMS will reply with a "unsubscribed" message.

"unsubscribe" Attribute	Value
"op"	"unsubscribe"
"uri"	URI to unsubscribe to.

"unsubscribed" Message Format

Following are the attributes and values in an "unsubscribed" message. The RMS sends an "unsubscribed" message to the client in response to an "unsubscribe" request.

"unsubscribed" Attribute	Value
"op"	"unsubscribed"
"uri"	URI the user unsubscribed to.
"errorMessage"	If there is an error in satisfying the request, the response will contain this attribute. The value is URI specific, but usually it is a text string to be displayed to the end user.

"push" Message Format

Following are the attributes and values in a "push" message. The RMS sends "push" messages to the client asynchronously (not in direct response to a message from the client). Push message are often used to push state (e.g. robot position) to the client on a regular basis without the client having to request it on a regular basis. Most push messages will only be sent if the user has subscribed to receive push notifications for this URI.

"push" Attribute	Value
"op"	"push"
"uri"	URI of the push notification.

"response"	A JSON structure containing the pushed information. The details of this attribute are URI specific.

Push Messages

Following are the push messages that the RMS may send to the client even if the client has not subscribed to them:

URI	Condition
/rms/sessionStateUpdate	Sent whenever the reservation state changes. Note that this can include a session being terminated by an administrator, in which case this message would be pushed with a state of "REMOVED".
/rms/sessionNotify	Sent whenever there is an important event relating to either the health of the robot, the connection of the robot to the RMS, or the connection of the client to the RMS.
/robot/tel/goToStatus	Sent periodically as the robot travels autonomously to a destination. The client should subscribe to this URI as well, as can be seen in the examples below, but it is possible these messages will be sent even without the subscription.

Keep-Alive (Ping) Messages

The RMS uses the standard WebSocket ping mechanism to periodically check that the client application is still connected. Most WebSocket libraries will respond to pings automatically at the library level, leaving the application developer free from the concern of generating pong frames. This can vary from implementation to implementation however, so the client must ensure that ping frames are responded to in some way, be it a feature of an included library, or explicitly implemented in the application code. If the client does not send a timely response to a ping message, the RMS will close the connection and disassociate the user with the robot. Note that this will not necessarily end the session, but the client must open a new WebSocket connection in order to re-embody the robot. Clients can also send WebSocket pings to the server. However, some WebSocket libraries do not provide an interface for sending ping frames. To circumvent this limitation, there is a JSON level URI (/rms/ping) that the client can use to ping the server. The client is not strictly required to send any type of ping at any particular interval, but it can be useful for determining if the client's connection to the RMS is still alive.

Examples

Here are some examples of how to perform common actions.

Connect to the RMS

Open a WebSocket connection to the RMS. Note the use of basic authentication to authenticate the user. If the Authorization header is not included, or contains an invalid username or password, an HTTP 401 response will be returned and the connection will be closed.

```
Client:
GET /client HTTP/1.1
Upgrade: websocket
Connection: Upgrade
Host: wconrad.rms.irobot.com
```

```
Sec-WebSocket-Protocol: irobot-bcr-scheduler
Sec-WebSocket-Key: n4BsTkr+xyZhv3vcAFuQZg==
Sec-WebSocket-Version: 13
Origin: mydomain.com
Accept-Language: en
Authorization: Basic dGhpc2lzbXluYW1lOnRoaXNpc215cGFzc3dvcmQ=

Server:
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=
Sec-WebSocket-Protocol: irobot-bcr-scheduler
(WebSocket is now open)
```

Create an Ad-hoc Session

Once logged in, this is the fastest way for a user to get on to a robot if they don't care where the robot is. In most cases, the robot will be on its dock, but if there are multiple robots serving the same map, there is no way to guarantee you will get any particular robot, or that one will be available at all.

```
{
    "op": "request",
    "uri": "/rms/goToMap",
    "args": {
        "name": "iRobotBedford2ndFlr_12-12-14"
    }
}

{
    "op": "response",
    "uri": "/rms/goToMap",
    "response": {
        "session": {
            "id": 283,
            "startTime": 1422545100350,
            "endTime": 1422549000000,
            "mapName": "iRobotBedford2ndFlr_12-12-14",
            "company": "iRobot",
            "building": "Bedford",
            "floor": "2nd Floor",
            "room": null,
            "user": "bob",
            "defaultVoipID": "bob@rp.mydomain.com",
            "state": "ARRIVED_AND_EMBODIED",
            "robot": "bcr-sim-1-1",
            "title": "Ad-Hoc Session",
            "extend": true
        }
    }
}
```

Now that the session is ARRIVED_AND_EMBODIED, we can dial the user.

```
{
    "op": "request",
    "uri": "/robot/tel/dial"
}
{
    "op": "response",
    "uri": "/robot/tel/dial",
    "response": {
        "id": "35"
    }
}
```

If there weren't any robots available on the map, the response would contain an *errorMessage* field.

```
{
    "op": "response",
    "uri": "/rms/goToMap",
    "response": {
    },
    "errorMessage": "No robots available to satisfy request."
}
```

To prevent this error from happening in the first place, the client can call /rms/availabilityByMap to determine which maps have available robots on them.

```
{
    "op": "request",
    "uri": "/rms/availabilityByMap",
    "args": {
        "startTime": 1422555588403
    }
}

{
    "op": "response",
    "uri": "/rms/availabilityByMap",
    "response": {
        "startTime": 1422555540403,
        "availability": [
            {
                "available": true,
                "mapName": "Building10-2_11_26_14",
                "transitionTime": 1422598740403,
                "transitionTime2": null
            },
            {
                "available": false,
                "mapName": "iRobotBedford2ndFlr_12-12-14",
                "transitionTime": 1422598842908,
                "transitionTime2": null
            }
        ]
    }
}
```

In this case, we now know that the second map is not available, so we could hide or otherwise disable that option for the user.

Create an Ad-hoc Session at a Preset

Presets are pre-defined locations that the robot knows how to autonomously drive to. Each map has multiple rooms (also called named spaces), and each room can have a number of different presets (also called waypoints). For example, a large conference room may have one preset near the back of the room facing a projector screen (a meeting participant), and another preset near the projector screen facing the audience (a presenter). In most cases, a room will only have one preset.
The /robot/tel/listPresets URI can be used to retrieve all of the presets that the user has access to.

```
{
    "op": "request",
    "uri": "/robot/tel/listPresets"
}

{
    "op": "response",
    "uri": "/robot/tel/listPresets",
    "response": [
        {
          "mapName": "Building10-2_11_26_14",
            "floor": {
                "company": "iRobot Corporation",
                "building": "Building 10",
                "floor": "2nd Floor"
            },
            "rooms": [
                {
                    "name": "CR-Mach5",
                    "waypoints": [
                        {
                            "id": 1477,
                            "name": "Mach5",
                            "location": {
                                "theta": 2.1359725,
                                "x": 2316,
                                "y": 4658
                            },
                            "topLevel": true
                        }
                    ]
                },
                {
                    "name": "CR-Pebbles",
                    "waypoints": [
                        {
                            "id": 1200,
                            "name": "CR-Pebbles-10-2-333",
                            "location": {
                                "theta": 0.66104317,
                                "x": 2379,
                                "y": 4650
                            },
                            "topLevel": true
                        }
                    ]
                },
                {
                    "name": "IT Service Desk",
                    "waypoints": [
                        {
                            "id": 2161,
                            "name": "IT Service Desk",
                            "location": {
                                "theta": -3.0172377,
                                "x": 2951,
                                "y": 1858
                            },
                            "topLevel": true
                        }
                    ]
                }
            ]
        },
        {
            "mapName": "iRobotBedford2ndFlr_12-12-14",
            "floor": {
                "company": "iRobot",
                "building": "Bedford",
                "floor": "2nd Floor"
            },
            "rooms": [
                {
                    "name": "CR-Attila-6-2-300",
                    "waypoints": [
                        {
                            "id": 5266,
                            "name": "CR-Attila-6-2-300",
                            "location": {
                                "theta": 1.7956071,
                                "x": 5418,
                                "y": 4999
                            },
                            "topLevel": true
                        }
                    ]
                },
                {
                    "name": "CR-B9-8-2-305",
                    "waypoints": [
                        {
                            "id": 16450,
                            "name": "CR-B9-8-2-305",
                            "location": {
                                "theta": 2.5318904,
                                "x": 2286,
                                "y": 5454
                            },
                            "topLevel": true
                        }
                    ]
                },
                {
                    "name": "CR-Braava-12-2",
                    "waypoints": [
                        {
                            "id": 23196,
                            "name": "CR-Braava-12-2",
                            "location": {
                                "theta": -0.7299051,
                                "x": 2040,
                                "y": 2784
                            },
                            "topLevel": true
                        }
                    ]
                }
            ]
        }
    ]
}
```

Once the user selects a preset, the client can call /robot/tel/goToRoom using the map name, room name, and waypoint id from /robot/tel/listPresets. This will create a session at the requested preset, and command robot to drive there autonomously.

```
{
    "op": "request",
    "uri": "/robot/tel/goToRoom",
    "args": {
        "waypoint": {
            "id": 1477
        },
        "map": "Building10-2_11_26_14",
        "room": "CR-Mach5"
    }
}

{
    "uri": "/robot/tel/goToRoom",
    "op": "response",
    "response": {
        "session": {
            "id": 284,
            "startTime": 1422551400167,
            "endTime": 1422554400000,
            "mapName": "Building10-2_11_26_14",
            "company": "iRobot Corporation",
            "building": "Building 10",
            "floor": "2nd Floor",
            "room": "CR-Mach5",
            "user": "bob",
            "defaultVoipID": "bob@rp.mydomain.com",
            "state": "DISPATCHED_AND_EMBODIED",
            "robot": "bcr-sim-1-1",
            "title": "Ad-Hoc Session",
            "extend": true
        }
    }
}
```

At this point, a robot is being dispatched to the requested destination. To keep track of its status, the client should subscribe to /robot/tel/goToStatus. It's also possible to poll the /robot/tel/goToStatus URI using the request/response model instead of a subscription, but the subscription is the preferred method.

```
{
    "op": "subscribe",
    "reliable": true,
    "uri": "/robot/tel/goToStatus",
    "rate": 0.2
}

{
    "op": "subscribed",
    "uri": "/robot/tel/goToStatus",
    "response": {}
}

{
    "op": "push",
    "uri": "/robot/tel/goToStatus",
    "response": {
        "destination": "CR-Mach5",
        "status": "IN_PROGRESS",
        "detail": "",
        "eta": "64.195259337"
    },
    "time": "6917369.245158297"
}
```

```
{
    "op": "push",
    "uri": "/robot/tel/goToStatus",
    "response": {
        "destination": "CR-Mach5",
        "status": "IN_PROGRESS",
        "detail": "",
        "eta": "67.896928088"
    },
    "time": "6917374.247609167"
}

  ... (More push messages) ...

{
    "op": "push",
    "uri": "/robot/tel/goToStatus",
    "response": {
        "destination": "CR-Mach5",
        "status": "IN_PROGRESS",
        "detail": "",
        "eta": "0.242018261"
    },
    "time": "6917409.248892287"
}

{
    "op": "push",
    "uri": "/robot/tel/goToStatus",
    "response": {
        "destination": "CR-Mach5",
        "status": "SUCCEEDED",
        "detail": "",
        "eta": "0.225689358"
    },
    "time": "6917414.249175004"
}
```

When you see the "SUCCEEDED" status, you know that the robot has arrived at its destination. We can now unsubscribe from goToStatus and dial the user.

```
{
    "op": "unsubscribe",
    "uri": "/robot/tel/goToStatus"
}

{
    "op": "unsubscribed",
    "uri": "/robot/tel/goToStatus",
    "response": {}
}
```

```
{
    "op": "request",
    "uri": "/robot/tel/dial"
}

{
    "op": "response",
    "uri": "/robot/tel/dial",
    "response": {
        "id": "35"
    }
}
```

Issue Manual Drive Commands to the Robot

These commands should only be issued if the user is currently in a session with a state of ARRIVED_AND_EMBODIED. The session state is available under the URI /rms/sessionStateUpdate, which can be either subscribed to or polled.
Example: Drive forward

```
    {
        "op": "request",
        "args": {
            "sidestep": 0,
            "translate": 0.3888889,
            "rotate": 0,
            "duration": 0.35
        },
        "uri": "/robot/drive/velocity",
        "t": "13:21:16.3200"
    }

    {
        "uri": "/robot/drive/velocity",
        "op": "response",
        "response": {
            "traj": [
                {
                    "xd": "0.2322222143411636",
                    "yd": "8.67680416405392e-09",
                    "td": "1.735360832810784e-08",
                    "dt": "0.1178511"
                },
                {
                    "xd": "0.3888888955116272",
                    "yd": "1.453053322109099e-08",
                    "td": "2.906106644218198e-08",
                    "dt": "4.882149"
                }
            ]
        },
        "time": "6921596.767994934"
    }
```

Example: Drive forward while turning

```
    {
        "op": "request",
        "args": {
            "sidestep": 0,
            "translate": 0.4244444,
            "rotate": -0.03571429,
            "duration": 0.35
        },
        "uri": "/robot/drive/velocity",
        "t": "13:21:16.5200"
    }

    {
        "uri": "/robot/drive/velocity",
        "op": "response",
        "response": {
            "traj": [
                {
                    "xd": "0.2648636698722839",
                    "yd": "-0.119117297232151",
                    "td": "1.873741695135323e-08",
                    "dt": "0.02242093"
                },
                {
                    "xd": "0.302662581205368",
                    "yd": "-0.1164528131484985",
                    "td": "-0.008928538300096989",
                    "dt": "0.008017328"
                },
                {
                    "xd": "0.3648857176303864",
                    "yd": "-0.05955864489078522",
                    "td": "-0.01785710826516151",
                    "dt": "0.08158761"
                },
                {
                    "xd": "0.4217799007892609",
                    "yd": "-0.002664467552676797",
                    "td": "-0.026785708963871",
                    "dt": "0.008017346"
                },
                {
                    "xd": "0.4244443774223328",
                    "yd": "1.585903675049849e-08",
                    "td": "-0.0357142873108387",
                    "dt": "4.879956"
                }
            ]
        },
        "time": "6921596.96879839"
    }
```

Example: Change pose so robot is "sitting"

```
    {
        "op": "request",
        "args": {
            "zLift": -0.0021
        },
        "uri": "/robot/drive/payloadPose",
        "t": "12:11:59.2770"
    }

    {
        "uri": "/robot/drive/payloadPose",
        "op": "response",
        "response": {},
        "time": "6917439.715056558"
    }
```

Handle an Obstructed Path or Robot Failure

First, make a request to go to a specific preset.

```
{
    "op": "request",
    "args": {
        "waypoint": {
            "id": 4082
        },
        "map": "iRobotBedford2ndFlr_12-12-14",
        "room": "10-2 IT Service Desk"
    },
    "uri": "/robot/tel/goToRoom"
}

{
    "uri": "/robot/tel/goToRoom",
    "op": "response",
    "response": {
        "session": {
            "id": 288,
            "startTime": 1422558060900,
            "endTime": 1422561600000,
            "mapName": "iRobotBedford2ndFlr_12-12-14",
            "company": "iRobot",
            "building": "Bedford",
            "floor": "2nd Floor",
            "room": "10-2 IT Service Desk",
            "user": "bob",
            "defaultVoipID": "bob@rp.mydomain.com",
            "state": "DISPATCHED_AND_EMBODIED",
            "robot": "bcr-sim-1-2",
            "title": "Ad-Hoc Session",
            "extend": true
        }
    }
}
```

Subscribe to /robot/tel/goToStatus so that we can be notified when the robot arrives. It's also possible to poll the URI using requests instead of a subscription, but the subscription is the preferred method. Note that the first push message may arrive before the response to the subscription.

```
{"op":"subscribe","reliable":true,"uri":"/robot/tel/goToStatus","rate":0.2
}
{"op":"push","uri":"/robot/tel/goToStatus","response":{"destination":"10-2
IT Service
Desk","status":"","detail":"","eta":"-2147483649"},"time":"6923983.8013047
58"}
{"op":"subscribed","uri":"/robot/tel/goToStatus","response":{}}
{"op":"push","uri":"/robot/tel/goToStatus","response":{"destination":"10-2
IT Service
Desk","status":"IN_PROGRESS","detail":"","eta":"9.80128139"},"time":"69239
88.80161664"}
{"op":"push","uri":"/robot/tel/goToStatus","response":{"destination":"10-2
IT Service
Desk","status":"IN_PROGRESS","detail":"","eta":"11.246342665"},"time":"692
3993.801804805"}
{"op":"push","uri":"/robot/tel/goToStatus","response":{"destination":"10-2
IT Service
Desk","status":"IN_PROGRESS","detail":"","eta":"11.752203717"},"time":"692
3998.801642721"}
{"op":"push","uri":"/robot/tel/goToStatus","response":{"destination":"10-2
IT Service
Desk","status":"IN_PROGRESS","detail":"","eta":"11.752203717"},"time":"692
4003.802069661"}
{"op":"push","uri":"/robot/tel/goToStatus","response":{"destination":"10-2
IT Service
Desk","status":"FAILED","detail":"PATH_OBSTRUCTED","eta":"-2147483649"},"t
ime":"6924008.8020877","server":{"ignores_remaining":3}}
```

After receiving a "FAILED" status, the user should be prompted as to whether they would like to ignore the error, or teleport in (i.e. connect video) to assess the situation. If the "ignores_remaining" field is zero, then the user has no option but to teleport in.
We can also unsubscribe from goToStatus since we know the status won't be changing unless we take some action. In this example, assume the user elects to ignore the failure.

```
{"op":"unsubscribe","uri":"/robot/tel/goToStatus"}
{"op":"unsubscribed","uri":"/robot/tel/goToStatus","response":{}}
{"op":"request","args":{"id":288,"intervene":false,"ignores_remaining":3},
"uri":"/rms/goToIntervene","t":"14:01:33.1630"}
```

```
{"op":"response","uri":"/rms/goToIntervene","response":{"results":2}}
```

The goToIntervene call has responded with a code of 2, meaning it allowed us to ignore the error and it is now sending the robot to its destination again. We re-subscribe to goToStatus to track its progress.

```
{"op":"subscribe","reliable":true,"uri":"/robot/tel/goToStatus","rate":0.2
}
{"op":"push","uri":"/robot/tel/goToStatus","response":{"destination":"10-2
IT Service
Desk","status":"","detail":"","eta":"-2147483649"},"time":"6924012.4885515
07"}
{"op":"subscribed","uri":"/robot/tel/goToStatus","response":{}}
{"op":"push","uri":"/robot/tel/goToStatus","response":{"destination":"10-2
IT Service
Desk","status":"IN_PROGRESS","detail":"","eta":"11.752203717"},"time":"692
4017.48863665"}
{"op":"push","uri":"/robot/tel/goToStatus","response":{"destination":"10-2
IT Service
Desk","status":"IN_PROGRESS","detail":"","eta":"12.396778481"},"time":"692
4022.488882194"}
{"op":"push","uri":"/robot/tel/goToStatus","response":{"destination":"10-2
IT Service
Desk","status":"IN_PROGRESS","detail":"","eta":"14.164085647"},"time":"692
4027.488953226"}
{"op":"push","uri":"/robot/tel/goToStatus","response":{"destination":"10-2
IT Service
Desk","status":"IN_PROGRESS","detail":"","eta":"15.002615482"},"time":"692
4032.489893568"}
{"op":"push","uri":"/robot/tel/goToStatus","response":{"destination":"10-2
IT Service
Desk","status":"IN_PROGRESS","detail":"","eta":"17.405158644"},"time":"692
4037.489881301"}
{"op":"push","uri":"/robot/tel/goToStatus","response":{"destination":"10-2
IT Service
Desk","status":"IN_PROGRESS","detail":"","eta":"20.093355768"},"time":"692
4042.489911545"}
{"op":"push","uri":"/robot/tel/goToStatus","response":{"destination":"10-2
IT Service
Desk","status":"IN_PROGRESS","detail":"","eta":"22.748297121"},"time":"692
4047.489912012"}
{"op":"push","uri":"/robot/tel/goToStatus","response":{"destination":"10-2
IT Service
Desk","status":"IN_PROGRESS","detail":"","eta":"22.780289195"},"time":"692
4052.490256835"}
{"op":"push","uri":"/robot/tel/goToStatus","response":{"destination":"10-2
IT Service
Desk","status":"IN_PROGRESS","detail":"","eta":"22.780289195"},"time":"692
4057.490782974"}
{"op":"push","uri":"/robot/tel/goToStatus","response":{"destination":"10-2
IT Service
Desk","status":"FAILED","detail":"PATH_OBSTRUCTED","eta":"-2147483649"},"t
ime":"6924062.490442207","server":{"ignores_remaining":2}}
```

If we get another "FAILED" status, we have encountered another failure. Note that the value of "ignores_remaining" has been decremented. Let's say that this time the user decides to teleport in and assess the situation. You might see this:

```
{"op":"unsubscribe","uri":"/robot/tel/goToStatus"}
{"op":"unsubscribed","uri":"/robot/tel/goToStatus","response":{}}
{"op":"request","args":{"id":288,"intervene":true},"uri":"/rms/goToInterve
ne"}
{"op":"response","uri":"/rms/goToIntervene","response":{"results":1}}
```

The intervention was approved. The session now has a status of "ARRIVED_AND_EMBODIED" and we can now initiate the VOIP call to the user. The user is now responsible for getting to the destination, or terminating the session if they are unable to get there.
Start the video call:

```
{
    "op": "request",
    "uri": "/robot/tel/dial"
}
{
    "op": "response",
    "uri": "/robot/tel/dial",
    "response": {
        "id": "35"
    }
}
```

Ending a Session

Once you have created a session, it is quite easy to end it, simply call /rms/endSession. This will hangup the call (if one is open), have the robot autonomously return to its dock, and make the robot available to service another session.

```
{
    "op": "request",
    "uri": "/rms/endSession"
}

{
    "op": "response",
    "uri": "/rms/endSession",
    "response": {
    }
}
```