PULSE ROBOTIC ARM

API REFERENCE GUIDE

ROZUM ROBOTICS

TABLE OF CONTENTS

REVISION LIST

Revision	Reason for change	Change details
9	Release 1.6.0 for the PULSE robot software	- PUT/jogging method added - GET/status method added - GET/robot/info method added - acceleration range increased up to 200%

WARNING SIGNS AND THEIR MEANINGS

Below are the warning symbols used throughout the manual and explanations of their meanings.

ⓘ *The sign denotes important information that is not directly related to safety, but that the user should be aware of.*

⚠ ***The sign indicates important safety precautions the user should follow.***

1 GENERAL DATA

This reference guide contains a detailed description of the PULSE REST Application Programming Interface (API). The API implements the functionality for monitoring and controlling the PULSE robotic arm (also, robotic arm or arm) and its work tool (also, tool).

API requests are in the *JSON* format. API responses are in the *JSON* and/or in the *plain text* format. All returned values are either *double numbers* or *text strings*.

API access is based on the *HTTP (v 2.0)* methods listed in TABLE 1-1. The **request address** for the HTTP methods is as follows: static or dynamic IP of the arm and port number 8081 (e. g., *10.10.10.20:8081*). For instructions to set the static or dynamic IP address of the arm, see the User Manual.

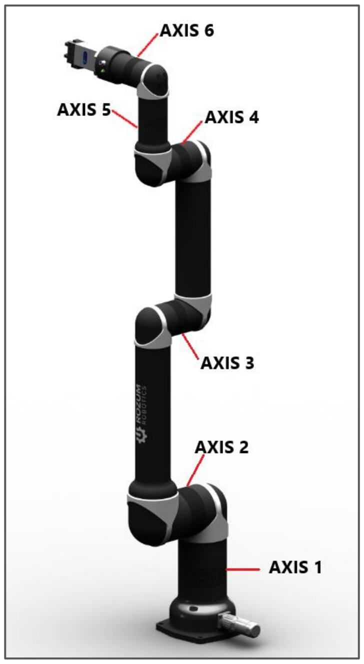
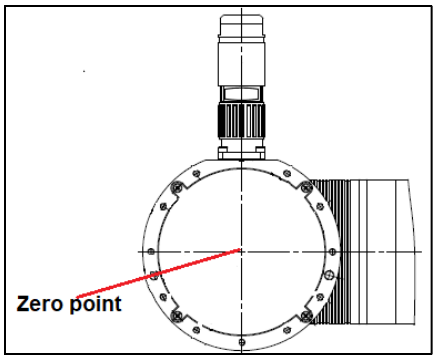
Table 1-1: **Supported HTTP methods**

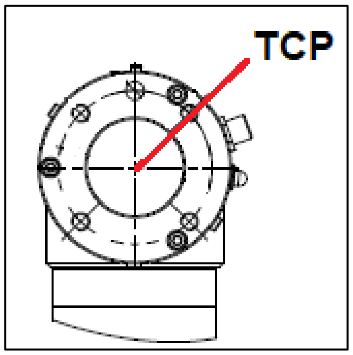
Method	Purpose
GET	• to get the actual pose / position of the robotic arm (rotation angles and coordinates of its joints) • to get the actual status of the arm motion (e.g., idle) (***deprecated***) • to get the actual state of the robotic arm (e.g., initializing) • to get the actual state of the servo motors in the arm joints (e.g., voltage, rotor velocity) • to get the actual properties (e.g., rotation angles, position coordinates) and shape of the work tool • to get the actual position of the arm base (rotation angles and coordinates) • to get the unique identifier (ID) of the robotic arm • to get the signal level (HIGH or LOW) on a digital output • to get the signal level (HIGH or LOW) on a digital input • to get data about a single or all obstacles within the arm environment • to get data about the hardware versions of the arm components • to get data about the software versions of the arm components • to get data about the arm software version • to get data about the arm (model, model version, and serial number)
PUT	• to set/change the pose/position of the robotic arm (rotation angles and coordinates of its joints) • to set/change the arm state (e.g., relax or freeze) • to set the arm in the jogging mode • to open the gripper • to close the gripper • to set the signal level on a digital output to HIGH or LOW • to recover the arm after an error • to add an obstacle to the robot environment for collision detection • to finish untwisting and quit the untwisting mode • to set the arm into the transportation pose
POST	• to set properties (e.g., rotation angles, coordinates) and shape of the work tool • to set a new position (rotation angles and coordinates) of the arm base
DELETE	• to remove a single or all obstacles from the arm environment

Glossary

TABLE 1-2 lists and defines essential terms used throughout the reference guide.

Table 1-2: **Essential REST API terms**

Term	Definition
Axis	An **axis** is a moveable structural component of the PULSE robotic arm comprising a servomotor to enable its rotation. In all, the PULSE robotic arm includes six links located on the robotic arm as illustrated below: 
Zero point	It is the origin point for measuring distances along the *x*, *y*, and *z* coordinate axes. Its original physical location is at the center of the arm base as shown below.  *It is possible to change the zero point location using the POST/Base request (see **Section**).*

Tool center point (TCP)	It is the point, relative to which all arm poses, positions, and movements are defined. Its original physical location is at the center of the arm wrist as shown below.  *Using the POST/tool/info request (see **Section**), you can relocate the TCP to any position within the tool or beyond it.*

2 ENABLING ACCESS TO API

You have to enable API access at least once — at the first start.

Before enabling API control, make sure the PULSE arm is:

- *connected to the control box, the work tool, the emergency button*
- *connected to a local network or a PC*
- *connected to a power supply*
- *switched on and ready for operation*

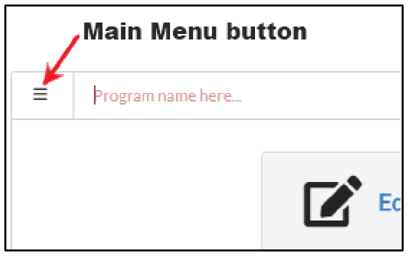
For connection and switching instructions, refer to the User Manual.

To enable API control of the PULSE arm, follow the instructions below:

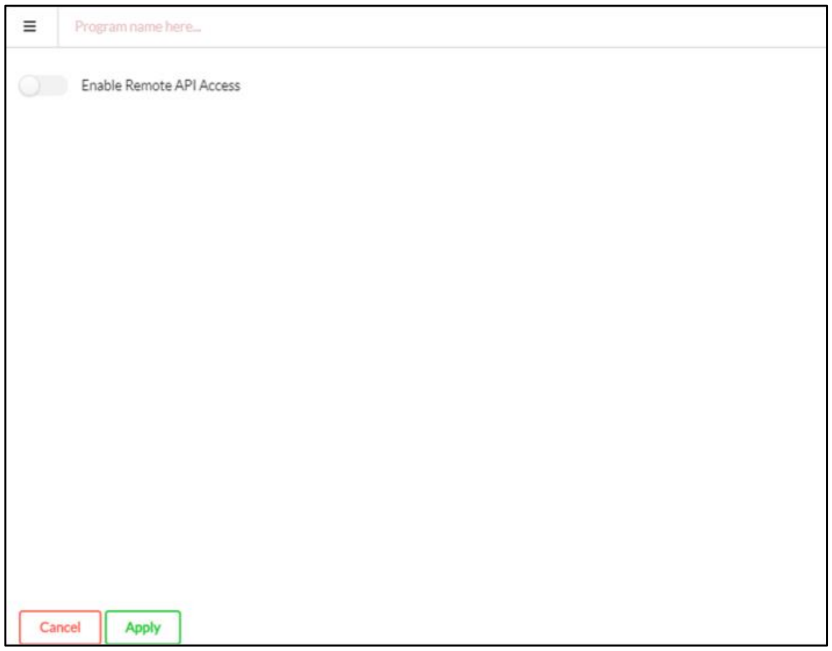
1. Check that the arm is ready for operation. The green LED on the control box should be on and the red one — off, whereas the LED on the arm wrist should be steady green.
2. Start the PULSE DESK user interface as described in the User Manual.

Note that the port number you are going to use API is always 8081!

3. In the displayed starting screen of the PULSE DESK interface, click the **Main Menu** button.



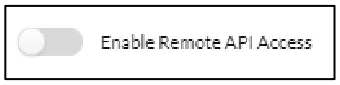
4. In the displayed menu, select **Configure**. PULSE DESK displays the **Configure** screen.



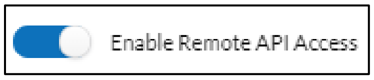
The Configure screen

5. In the **Configure** screen, switch the **Enable remote API access** toggle to the enabled state.

Disabled state



Enabled state



6. Click **Apply** to confirm.

Now, you can proceed to work with API methods.

3 DESCRIPTION OF API METHODS

The section describes in detail the PULSE REST API methods you can use to control the PULSE robotic arm and its work tool (a gripper), as well as to get information about the arm, its components, and parameters.

3.1 Requests to get parameters and states of the arm (GET)

3.1.1 Getting the actual arm position

Path:

`GET/position`

Description: The function returns the actual position of the PULSE robotic arm, which is described as a set of *x, y,* and *z* coordinates, as well as *roll*, *pitch*, and *yaw* rotation angles.

The coordinates define the actual distance (in meters) from the zero point of the robotic arm to the tool center point (TCP) along the *x, y,* and *z* axes accordingly. *Roll* stands for the TCP rotation angle around the *x* axis; *pitch*—the TCP rotation angle around the *y* axis; *yaw*—the TCP rotation angle around the *z* axis. All rotation angles are in radians and relative to the zero point.

Related REST API functions: PUT/POSITION, PUT/POSITIONS/RUN

Response content type: application/json, text/plain

Response body:

HTTP status code	Response schema/ type
200 OK	POSITION SCHEMA
500 Internal Server Error	Robot error string
503 Service Unavailable	Robot emergency string

Response examples:

- **200 OK**

```
{
  "point": {
    "x": 0.3,
    "y": -0.4,
    "z": 0.2
  },
  "rotation": {
    "roll": 3.14,
    "pitch": 0,
    "yaw": 0.5
  }
}
```

- **500 Internal Server Error**
```
[
  "Robot does not respond"
]
```
- **503 Service Unavailable**
```
[
  "Robot unavailable in emergency state"
]
```

3.1.2 Getting the actual arm pose

Path:

`GET/pose`

Description: The function returns the actual pose of the robotic arm. An arm *pose* is a set of output flange angles (in degrees) of the six servos in the arm joints.

Response content type: application/json, text/plain

Related REST API functions: PUT/POSE, PUT/POSES/RUN

Response body:

HTTP status code	Response schema/ type
200 OK	POSE SCHEMA
500 Internal Server Error	Robot error string
503 Service Unavailable	Robot emergency string

Response examples:

- **200 OK**
```
{
  "angles": [
    61,
    -98,
    -122,
    -49,
    89,
    -28
  ]
```
- **500 Internal Server Error**
```
[
  "Robot does not respond"
]
```
- **503 Service Unavailable**
```
[
  "Robot unavailable in emergency state"
]
```

3.1.3 Getting the actual motion status (deprecated)

The function is deprecated starting from PULSE API release 1.5.0. Please, use the GET/STATUS *function instead.*

Path:

`GET/status/motion`

Description: The function returns the actual state of the arm motion. Possible arm states are as follows:

- **IDLE**
  The arm is not in motion, but is fully functional and ready for operation.
- **ZERO_GRAVITY**
  The arm is in the zero gravity mode, which means the user can move it by hand to set a motion trajectory.
- **RUNNING**
  The arm is in motion.
- **MOTION_FAILED**
  Motion is impossible due to incorrect motion settings.
- **ERROR**
  The arm stops moving due to an error and goes into the freeze mode, retaining its last position. The user can recover the arm, using the **PUT/RECOVER** function.
- **EMERGENCY**
  Motion is impossible due to an emergency. In this case, an emergency is a fatal failure that causes the control box to switch off and the arm to stop without retaining its position. Recovery with the **PUT/RECOVER** function is not possible.

Response content type: text/plain

Response body:

HTTP status code	Response schema/ type
200 OK	Enum. motion status [IDLE, ZERO_GRAVITY, RUNNING, MOTION_FAILED, EMERGENCY, ERROR]
500 Internal Server Error	Robot error string
503 Service Unavailable	Robot emergency string

Response examples:

- **200 OK**

```
[
  "IDLE",
  "ZERO_GRAVITY",
  "RUNNING",
  "MOTION_FAILED",
  "EMERGENCY",
  "ERROR"
]
```

- **500 Internal Server Error**

```
[
  "Robot does not respond"
]
```

- **503 Service Unavailable**
  ```
  [
    "Robot unavailable in emergency state"
  ]
  ```

3.1.4 Getting the actual state of the robotic arm

Path:

`GET/status`

Description: The function returns the actual state of the robotic arm. Possible arm states are as follows:

- **INITIALIZING**
  The arm starts initializing following successful initialization of its control box. The indicator on the arm wrist remains off until the initialization is successfully completed.
- **INITIALIZATION_FAILURE**
  The arm has failed to complete initialization and is not available for further operation. The arm wrist indicator is off.
- **TWISTED**
  A twist is detected on one or more motors in the arm joints, and the arm switches to the untwisting mode. In the mode, the PUT and other API requests to move the arm, as well as the PULSE DESK user interface are unavailable until the twist(s) is (are) eliminated. For details, refer to the description of the **PUT/UNTWISTING/FINISH** function and the User Manual.
- **ACTIVE**
  The arm has initialized successfully. It is ready for operation, but not in motion.
- **MOTION**
  The arm is in motion.
- **ZERO_GRAVITY**
  The arm is in the zero gravity mode. In the mode, the wrist buttons are enabled, and users can move the arm joints manually to set a motion trajectory. For details, refer to the User Manual.
- **JOGGING**
  The arm is in the jogging mode: it is moving along or rotating around each of the preset coordinate axes (*x, y, z*) at a pre-defined acceleration rate. For details, refer to the description of the **PUT/JOGGING** function.
- **BROKEN**
  Motion is impossible due to a system failure, such as a fatal error or a broken arm component. The arm stops without retaining its position. Recovery with the **PUT/RECOVER** function is not possible.
- **EMERGENCY**
  The arm stops moving due to a non-fatal error and goes into the freeze mode, retaining its last position. The user can recover the arm, using the **PUT/RECOVER** function.

In addition to the arm state, the function can also return a message to give more details about the state (e. g., a detailed failure description). The details (if any) are contained in the `message` string.

Response content type: text/plain

Response body:

HTTP status code	Response schema/ type
200 OK	SYSTEM STATUS SCHEMA
500 Internal Server Error	Robot error string

Response examples:

- **200 OK**

```
{
  "state": [
    "INITIALIZING",
    "INITIALIZATION_FAILURE",
    "TWISTED",
    "ACTIVE",
    "MOTION",
    "ZERO_GRAVITY",
    "JOGGING",
    "BROKEN",
    "EMERGENCY"
  ],
  "message": "string"
```

- **500 Internal Server Error**

```
[
  "Robot does not respond"
]
```

3.1.5 Getting the actual status of servo motors

Path:

```
GET/status/motors
```

Description: The function returns the actual states of the six servo motors integrated into the joints of the robotic arm. The states are described as an array of six objects—one for each servo motor. Each object includes the following properties:

- **Angle**—the actual angular position (degrees) of the servo's output flange
- **Rotor velocity**—the actual rotor velocity (RPM)
- **RMS current**—the actual input current (Amperes)
- **Phase current**—the actual magnitude of alternating current (Amperes)
- **Supply voltage**—the actual supply voltage (Volts)
- **Stator temperature**—the actual temperature (degrees C) as measured on the stator winding
- **Servo temperature**—the actual temperature (degrees C) as measured on the MCU PCB
- **Velocity setpoint**—the user-preset rotor velocity (RPM)
- **Velocity output**—the motor control current (Amperes) based on the preset velocity
- **Velocity feedback**—the actual rotor velocity (RPM)
- **Velocity error**—the difference between the preset and the actual rotor velocities (RPM)

- **Position setpoint**—the user-preset position of the servo flange (degrees)
- **Position output**—rotor velocity (RPM) based on the position setpoint
- **Position feedback**—the actual position of the servo flange (degrees) based on the encoder feedback
- **Position error**—the difference between the preset and the actual positions of the servo flange (degrees)

Response content type: application/json, text/plain

Response body:

HTTP status code	Response schema/ type
200 OK	MOTOR STATUS ARRAY SCHEMA
500 Internal Server Error	Robot error string
503 Service Unavailable	Robot emergency string

Response examples:

- **200 OK**

```
[
  {
    "angle": 168.89699,
    "rotorVelocity": -0.00064343837,
    "rmsCurrent": 0.01,
    "voltage": 47.795017,
    "phaseCurrent": 0.01,
    "statorTemperature": 27.990631,
    "servoTemperature": 31.739925,
    "velocityError": -0.022674553,
    "velocitySetpoint": -0.02331799,
    "velocityOutput": 0.01,
    "velocityFeedback": -0.00064343837,
    "positionError": 0.0385437,
    "positionSetpoint": 168.93799,
    "positionOutput": 0.01,
    "positionFeedback": 168.89944
  }
]
```

The example is one object containing properties for a single servo. In reality, the array in the response includes six similar objects.

- **500 Internal Server Error**

```
[
  "Robot does not respond"
]
```

- **503 Service Unavailable**

```
[
  "Robot unavailable in emergency state"
]
```

3.1.6 Getting actual tool properties

Path:

```
GET/tool/info
```

Description: The function returns actual properties of the last tool preset by the user, in particular:

- **name** — any random name of the work tool defined by the user (e.g., "gripper").
- **actual TCP position**, including:
  - **point —** *x, y,* and *z* coordinates defining the TCP offset (in meters) along the *x, y,* and *z* axes accordingly from its original location.
  - **rotation angles —** *roll*, *pitch*, and *yaw***.** *Roll* stands for the actual TCP rotation angle around the *x* axis; *pitch*—the actual TCP rotation angle around the *y* axis; *yaw*—the actual TCP rotation angle around the *z* axis. All rotation angles are in radians and relative to the physical center point of the arm base.

Related REST API functions: GET/TOOL/SHAPE, POST/TOOL/INFO, POST/TOOL/SHAPE

Response content type: application/json, text/plain

Response body:

HTTP status code	Response schema/ type
200 OK	TOOL INFO SCHEMA
500 Internal Server Error	Robot error string
503 Service Unavailable	Robot emergency string

Response examples:

- **200 OK**

```
{
  "name": "gripper",
  "tcp": {
    "point": {
      "x": 0.3,
      "y": -0.4,
      "z": 0.2
    },
    "rotation": {
      "roll": 3.14,
      "pitch": 0,
      "yaw": 0.5
    }
  }
}
```

- **500 Internal Server Error**

```
[
  "Robot does not respond"
]
```

- **503 Service Unavailable**

```
[
  "Robot unavailable in emergency state"
]
```

3.1.7 Getting the actual tool shape

Path:

```
GET/tool/shape
```

Description: The function returns the actual properties defined by the user for a specific tool to describe the tool shape, in particular:

- **radius** — radius of the work tool (in meters) measured from its physical center point.
- **begin** — the start *x, y,* and *z* coordinates of the work tool capsule measured as a distance (in meters) along the corresponding axes from the original TCP.
- **finish —** the end *x, y,* and *z* coordinates of the work tool capsule measured as a distance (in meters) along the corresponding axes from the original TCP.

Related REST API functions: GET/TOOL/INFO, POST/TOOL/INFO, POST/TOOL/SHAPE

Response content type: application/json, text/plain

Response body:

HTTP status code	Response schema/ type
200 OK	TOOL SHAPE SCHEMA, TOOL INFO SCHEMA
500 Internal Server Error	Robot error string
503 Service Unavailable	Robot emergency string

Response examples:

- **200 OK**

```
{
  "shape": [
    {
      "radius": 0.03,
      "begin": {
        "x": 0,
        "y": 0,
        "z": 0
      },
      "finish": {
        "x": 0,
        "y": 0,
        "z": 0.24
      }
    }
  ]
}
```

- **500 Internal Server Error**

```
[
  "Robot does not respond"
]
```

- **503 Service Unavailable**

```
[
  "Robot unavailable in emergency state"
]
```

3.1.8 Getting the actual position of the arm base

Path:

`GET/base`

Description: The function returns the actual position of the arm's zero point in the user environment. The actual zero point position is described as a set of *x, y,* and *z* coordinates, as well as *roll*, *pitch*, and *yaw* rotation angles.

The coordinates define the offset (in meters) from the physical center point of the arm base (original zero point) to the actual zero point position along the *x, y,* and *z* axes accordingly. *Roll* stands for the rotation angle around the *x* axis; *pitch*—the rotation angle around the *y* axis; *yaw*—the rotation angle around the *z* axis. All rotation angles are in radians and relative to the physical center point of the arm base.

Related REST API functions: POST/BASE

Response content type: application/json, text/plain

Response body:

HTTP status code	Response schema/ type
200 OK	POSITION SCHEMA
500 Internal Server Error	Robot error string
503 Service Unavailable	Robot emergency string

Response examples:

- **200 OK**
```
{
  "point": {
    "x": 0.3,
    "y": -0.4,
    "z": 0.2
  },
  "rotation": {
    "roll": 3.14,
    "pitch": 0,
    "yaw": 0.5
  }
```
- **500 Internal Server Error**
```
[
  "Robot does not respond"
]
```
- **503 Service Unavailable**
```
[
  "Robot unavailable in emergency state"
]
```

3.1.9 Getting the arm ID

Path:

`GET/robot/id`

Description: The function returns the unique identifier (ID) of the robotic arm. The ID is an alphanumeric designation that consists of individual servo motor identifications.

Response content type: text/plain

Response body:

HTTP status code	Response schema/ type
200 OK	String
500 Internal Server Error	Robot error string
503 Service Unavailable	Robot emergency string

Response examples:

- **200 OK**
```
"1346466AFG872"
```
- **500 Internal Server Error**
```
[
  "Robot does not respond"
]
```
- **503 Service Unavailable**
```
[
  "Robot unavailable in emergency state"
]
```

3.1.10 Getting the signal level on a digital output

Path:

`GET/signal/output/{port}`

Description: The function returns the actual signal level on the digital output specified in the `{port}` parameter of the request path.

ATTENTION! SPECIFYING THE `{port}` PARAMETER IS MANDATORY!

A digital output is a physical port on the back panel of the control box. Since the control box has two digital outputs, the parameter value can be either *1* (corresponds to Relay output 1) or *2* (corresponds to Relay output 2).

The function returns either of the following values:

- `LOW`—default user-defined state (e.g., LED off)
- `HIGH`—change of the user defined state (e.g., LED on)

For location of digital outputs, refer to the User Manual.

Related REST API functions: PUT/SIGNAL/OUTPUT/{PORT}/HIGH, PUT /SIGNAL/OUTPUT/{PORT}/LOW

Response content type: text/plain

Response body:

HTTP status code	Response schema/ type
200 OK	Signal enum. string
412 Precondition Failed	Port error string
500 Internal Server Error	Robot error string
503 Service Unavailable	Robot emergency string

Response examples:

- **200 OK**
```
[
  "HIGH",
  "LOW"
]
```
- **412 Precondition Failed**
```
[
  "Unable to use parameter value {13}"
]
```
- **500 Internal Server Error**
```
[
  "Robot does not respond"
]
```
- **503 Service Unavailable**
```
[
  "Robot unavailable in emergency state"
]
```

3.1.11 Getting the signal level on a digital input

Path:

`GET/signal/input/{port}`

Description: The function returns the actual signal level on the digital input specified in the `{port}` parameter of the request path.

ATTENTION! SPECIFYING THE `{port}` PARAMETER IS MANDATORY!

A digital input is a physical port on the back panel of the control box. Since the control box has four digital inputs (DI), the parameter can have any integral value between *1* (corresponds to DI1) and *4* (corresponds to DI4).

The function returns either of the following values:

- `LOW`—default user-defined state
- `HIGH`—change of the user defined state

For location of digital outputs, refer to the User Manual.

Response content type: text/plain

Response body:

HTTP status code	Response schema/ type
200 OK	Signal enum. string
412 Precondition failed	Port error string
500 Internal Server Error	Robot error string
503 Service Unavailable	Robot emergency string

Response examples:

- **200 OK**
```
[
  "HIGH",
  "LOW"
]
```
- **412 Precondition Failed**
```
[
  "Unable to use parameter value {13}"
]
```
- **500 Internal Server Error**
```
[
  "Robot does not respond"
]
```
- **503 Service Unavailable**
```
[
  "Robot unavailable in emergency state"
]
```

3.1.12 Getting data about obstacles in an arm environment

Path:

`GET/environment`

Description: The function returns data about all obstacles preset within the arm's environment.

An *obstacle* is any object, such as a control box or a wall, in the way of an arm to be taken into consideration for collision detection. An obstacle can be one of the following types:

- **BOX**—typically used to describe obstacles with a shape reminding that of a box.
- **CAPSULE**—preferred for objects of cylindrical shape or having complex structure and irregular outlines. To describe an obstacle of complex structure, it is possible to use multiple capsules.
- **PLANE**—recommended for describing plain-surface objects, such as a wall or a table.

Depending on the total quantity of obstacles preset in a given environment, the response of the function can contain one or more data arrays. Each array comprises the following data:

- **Obstacle type**—a geometric pattern, roughly describing the shape of an obstacle for collision detection purposes—**BOX, CAPSULE,** and **PLANE.**
- **Name**—any random name as defined by the user for a specific obstacle type (e.g., "`first_box`").
- **Obstacle properties—**spatial location and/or dimensions of a specific obstacle.

Each obstacle type has its own set of properties as described in the table below.

Type	Properties
BOX	- **sides**—the *x*, *y*, and *z* coordinates defining the dimensions of an obstacle (i.e., length, width, depth). - **position**—a set of the *x*, *y*, and *z* coordinates, as well as *roll, pitch* and *yaw* angles defining the location of an obstacle in space. The coordinate values are distances (in meters) along the *x*, *y*, and *z* axes accordingly, measured from the obstacle's center point relative to the zero point (see GLOSSARY). *Roll, pitch* and *yaw* are rotation angles (in radians) of the obstacle's center point relative to the zero point.
CAPSULE	- **radius**—the radius (in meters) of the capsule shape incorporating an obstacle, measured from the obstacle's center point - **start point—**the starting *x*, *y*, and *z* coordinates (in meters) of the capsule shape length relative to the zero point - **end point—**the end *x*, *y*, and *z* coordinates (in meters) of the capsule shape length relative to the zero point
PLANE	- **points**—at least three points constituting a single plane; each of the points is described as a set of *x*, *y*, and *z* coordinates (in meters) on the plane

Related REST API functions: GET/ENVIRONMENT/{OBSTACLE}, PUT/ENVIRONMENT, DELETE/ENVIRONMENT, DELETE/ENVIRONMENT/{OBSTACLE}

Response content type: application/json, text/plain

Response body:

HTTP status code	Response schema/ type
200 OK	OBSTACLE SCHEMA
500 Internal Server Error	Robot error string
503 Service Unavailable	Robot emergency string

Response examples:

- **200 OK**

```
[
  {
    "obstacleType": "BOX",
    "name": "example_box",
    "sides": {
      "x": 0.1,
      "y": 0.1,
      "z": 0.1
    },
    "position": {
      "point": {
        "x": 1,
        "y": 1,
        "z": 1
      },
      "rotation": {
        "roll": 0,
        "pitch": 0,
        "yaw": 0
      }
    }
  },
  {
    "obstacleType": "CAPSULE",
    "name": "example_capsule",
    "radius": 0.1,
    "startPoint": {
      "x": 0.5,
      "y": 0.5,
      "z": 0.2
    },
    "endPoint": {
      "x": 0.5,
      "y": 0.5,
      "z": 0.2
    }
  },
  {
    "obstacleType": "PLANE",
    "name": "example_plane",
    "points": [
      {
        "x": -0.5,
        "y": 0.2,
        "z": 0
      },
      {
        "x": -0.5,
        "y": 0,
        "z": 0
      },
      {
        "x": -0.5,
        "y": 0,
        "z": 0.1
      }
    ]
  }
]
```

- **500 Internal Server Error**

```
[
  "Robot does not respond"
]
```

- **503 Service Unavailable**

```
[
  "Robot unavailable in emergency state"
]
```

3.1.13 Getting data about a specific obstacle in the arm environment

Path:

`GET/environment/{obstacle}`

Description: The function returns data about the obstacle specified in the `{obstacle}` parameter of the request path.

ATTENTION! SPECIFYING THE `{obstacle}` PARAMETER IS MANDATORY!

An *obstacle* is any object, such as a control box or a wall, in the way of an arm to be taken into consideration for collision detection.

An obstacle can be one of the following types:

- **BOX—** typically used to describe obstacles with a shape reminding that of a box.
- **CAPSULE—**preferred for objects of cylindrical shape or having complex structure and irregular outlines. In the latter two cases, it is also possible to describe an obstacle using multiple capsules.
- **PLANE**—recommended for describing plain-surface objects, such as a wall or a table.

*For this REST API request, the `{obstacle}` parameter in the request path can contain **no more than a single object** (e.g., box 1).*

The response of the function contains a single data array comprising the following:

- **Obstacle type**—a geometric pattern, roughly describing the shape of an obstacle for collision detection purposes— **BOX, CAPSULE,** and **PLANE.**
- **Name**—any random name as defined by the user for a specific obstacle type (e.g., "`first_box`").
- **Obstacle properties—**spatial location and / or dimensions of a specific obstacle

Each obstacle type has its own set of properties as described in the table below.

Obstacle type	Obstacle properties
BOX	- **sides**—the *x*, *y*, and *z* coordinates defining the spatial dimensions of an obstacle (i.e., length, width, depth). - **position**—a set of the *x*, *y*, and *z* coordinates, as well as *roll, pitch* and *yaw* angles defining the location of an obstacle in space. The coordinate values are distances (in meters) along the *x*, *y*, and *z* axes accordingly, measured from the obstacle's center point relative to the zero point (see GLOSSARY). *Roll, pitch* and *yaw* are rotation angles (in radians) of the obstacle's center point relative to the zero point.

CAPSULE	- **radius**—the radius (in meters) of the capsule shape incorporating an obstacle, measured from the obstacle's center point - **start point—**the starting *x*, *y*, and *z* coordinates (in meters) of the capsule shape length relative to the zero point - **end point—**the end *x*, *y*, and *z* coordinates (in meters) of the capsule shape length relative to the zero point
PLANE	- **points**—at least three points constituting a single plane; each of the points is described as a set of *x*, *y*, and *z* coordinates (in meters) on the plane

Related REST API functions: GET/ENVIRONMENT, PUT/ENVIRONMENT, DELETE/ENVIRONMENT, DELETE/ENVIRONMENT/{OBSTACLE}

Response content type: application/json, text/plain

Response body:

HTTP status code	Response schema/ type
200 OK	OBSTACLE SCHEMA
500 Internal Server Error	Robot error string
503 Service Unavailable	Robot emergency string

Response examples:

- **200 OK**

```
[
  {
    "obstacleType": "BOX",
    "name": "example_box",
    "sides": {
      "x": 0.1,
      "y": 0.1,
      "z": 0.1
    },
    "position": {
      "point": {
        "x": 1,
        "y": 1,
        "z": 1
      },
      "rotation": {
        "roll": 0,
        "pitch": 0,
        "yaw": 0
      }
    }
  },
  {
    "obstacleType": "CAPSULE",
    "name": "example_capsule",
    "radius": 0.1,
    "startPoint": {
      "x": 0.5,
      "y": 0.5,
      "z": 0.2
    },
    "endPoint": {
      "x": 0.5,
      "y": 0.5,
      "z": 0.2
    }
  },
  {
    "obstacleType": "PLANE",
    "name": "example_plane",
    "points": [
      {
        "x": -0.5,
        "y": 0.2,
        "z": 0
      },
      {
        "x": -0.5,
        "y": 0,
        "z": 0
      },
      {
        "x": -0.5,
        "y": 0,
        "z": 0.1
      }
    ]
  }
]
```

- **500 Internal Server Error**

```
[
  "Robot does not respond"
]
```

- **503 Service Unavailable**

```
[
  "Robot unavailable in emergency state"
]
```

3.1.14 Getting the hardware versions of the arm components

Path:

`GET/version/hardware`

Description: The function returns the hardware versions for all motors in the arm joints, as well as hardware versions for the USB-CAN dongle, the safety board, and the wrist.

Related REST API functions: GET/VERSION/SOFTWARE, GET/VERSION/SOFTWARE/ROBOT

Response content type: application/json, text/plain

Response body:

HTTP status code	Response schema/ type
200 OK	VERSION SCHEMA

500 Internal Server Error	Robot error string
503 Service Unavailable	Robot emergency string

Response examples:

- **200 OK**
```
{
  "motorsVersion": [
    "string"
  ],
  "safetyVersion": "string",
  "usbCanVersion": "string",
  "wristVersion": "string"
}
```
- **500 Internal Server Error**
```
[
  "Robot does not respond"
]
```
- **503 Service Unavailable**
```
[
  "Robot unavailable in emergency state"
]
```

3.1.15 Getting the software versions of the arm components

Path:

`GET/version/software`

Description: The function returns the software versions for all motors in the arm joint, as well as software versions for the USB-CAN dongle, the safety board, and the wrist.

Related REST API functions: GET/VERSION/HARDWARE, GET/VERSION/SOFTWARE/ROBOT, GET/ROBOT/INFO

Response content type: application/json, text/plain

Response body:

HTTP status code	Response schema/ type
200 OK	VERSION SCHEMA
500 Internal Server Error	Robot error string
503 Service Unavailable	Robot emergency string

Response examples:

- **200 OK**
```
    {
      "motorsVersion": [
        "string"
      ],
      "safetyVersion": "string",
      "usbCanVersion": "string",
      "wristVersion": "string"
    }
```

- **500 Internal Server Error**
```
[
  "Robot does not respond"
]
```
- **503 Service Unavailable**
```
[
  "Robot unavailable in emergency state"
]
```

3.1.16 Getting the arm software version

Path:

`GET/version/software/robot`

Description: The function returns the version of the arm's core software.

Related REST API functions: GET/VERSION/HARDWARE, GET/VERSION/SOFTWARE, GET/ROBOT/INFO

Response content type: application/json, text/plain

Response body:

HTTP status code	Response schema/ type
200 OK	String
500 Internal Server Error	Robot error string
503 Service Unavailable	Robot emergency string

Response examples:

- **200 OK**
```
"1.4.3-release"
```
- **500 Internal Server Error**
```
[
  "Robot does not respond"
]
```
- **503 Service Unavailable**
```
[
  "Robot unavailable in emergency state"
]
```

3.1.17 Getting information about the arm

Path:

`GET/robot/info`

Description: The function returns the unique serial number of the arm, as well as its model and model version data. In case any data is not available, the function returns "unknown" for the corresponding property.

Related REST API functions: GET/VERSION/HARDWARE, GET/VERSION/SOFTWARE, GET/ROBOT/ID, GET/VERSION/SOFTWARE/ROBOT

Response content type: application/json, text/plain

Response body:

HTTP status code	Response schema/ type
200 OK	ROBOT INFO SCHEMA
500 Internal Server Error	Robot error string

Response examples:

- **200 OK**
```
{
  "model": "pulse75",
  "version": "2.4.0",
  "serialNumber": "18-00604"
}
```
- **500 Internal Server Error**
```
[
  "Robot does not respond"
]
```

3.2 Requests to set parameters, states, and actions (PUT, POST)

3.2.1 Setting a new arm position

Path:

```
PUT/position
```

Description: The function commands the arm to move to a new position. The *position* is described as a set of *x, y,* and *z* coordinates, as well as *roll*, *pitch*, and *yaw* rotation angles.

The coordinates define the desired distance (in meters) from the zero point to the TCP along the *x, y,* and *z* axes accordingly. *Roll* stands for the desired TCP rotation angle around the *x* axis; *pitch* — the desired TCP rotation angle around the *y* axis; *yaw* — the desired TCP rotation angle around the *z* axis. All rotation angles are in radians and relative to the zero point.

Related REST API functions: GET/POSITION, PUT/POSITIONS/RUN

Request content type: application/json

Request parameters: It is **obligatory** to set one of the below parameters.

For each single request, you can set ONLY ONE of the below variants: speed, tcp_max_velocity, or velocity+acceleration.

- **speed**
- **tcp_max_velocity**
- **velocity+acceleration**

  Important! In the combination, **it is obligatory** to set both velocity and acceleration.

Parameter	Description
speed	The parameter sets the combination of velocity and acceleration (in % max. velocity and acceleration) at which the arm should move to a target position. The **admissible value range** is from 1 to 100. **Type:** *number (double)* **Included as:** *query*

velocity	The parameter sets the velocity (in % max. velocity) at which the arm should move to the target position. The **admissible value range** is from 1 to 100. ***ATTENTION! The velocity parameter is always used in combination with the acceleration parameter.*** **Type:** *number (double)* **Included as:** *query*
acceleration	The parameter sets the acceleration (in % max. acceleration) at which the arm should move to the target position. The **admissible value range** is from 1 to 200. ***ATTENTION! Use values greater than 100% with caution! They are admissible only with a relatively light load or without a load. Otherwise, the arm can fail to operate.*** ***ATTENTION! The acceleration parameter is obligatory, always used in combination with the velocity parameter.*** **Type:** *number (double)* **Included as:** *query*
tcp_max_velocity	The parameter defines the limit velocity in meters per second that an end effector can reach at its TCP while moving. The **admissible value range** is from 0.001 to 2 m/s. **Included as:** *query*
motionType	The parameter sets the type of motion the arm should use to get to the target position. **Admissible values** are as follows: • **JOINT** When set to this motion type, the arm moves to the specified position along a trajectory that has been calculated as the most convenient one. The trajectory can be described as a set of joint angles connected into a curve. • **LINEAR** When the motion type is LINEAR, the arm moves to the specified position along a straight line. This motion takes more time than with the *type* parameter set to JOINT. However, the trajectory is entirely predictable, unlike with the JOINT type motion. When the user specifies no value for the parameter, it is set to the default one—JOINT. **Included as:** *query*

Request body: The request body is in accordance with the POSITION SCHEMA. It specifies the coordinates (*x, y, z*) and rotation angles (*roll, pitch, yaw*) that describe the target arm position.

Make sure to specify all point (x, y, z) and rotation (roll, pitch, yaw) properties in the request body. Otherwise, the function returns a 400 Bad Request error.

Request example:

Request path: Depending on which of the above obligatory parameters the user chooses to set, the request path is as illustrated below.

- **speed**
  `PUT/position?speed=100&motionType=joint`
- **velocity+acceleration**
  `PUT/position?velocity=100&acceleration=10`
- **tcp_max_velocity**
  `PUT/position?tcp_max_velocity=1`

Request body:

```
{
  "point": {
    "x": 0.3,
    "y": -0.4,
    "z": 0.2
  },
  "rotation": {
    "roll": 3.14,
    "pitch": 0,
    "yaw": 0.5
  }
}
```

Response content type: application/json, text/plain

Response body:

HTTP status code	Description	Response schema/ type
200 OK	Actual arm position	POSITION SCHEMA
400 Bad Request	Message parsing error	String
412 Precondition Failed	Incorrect input parameters	String
500 Internal Server Error	Robot error	String
503 Service unavailable	Robot emergency	String

Response examples:

- **200 OK**

```
{
  "point": {
    "x": 0.3,
    "y": -0.4,
    "z": 0.2
  },
  "rotation": {
    "roll": 3.14,
    "pitch": 0,
    "yaw": 0.5
  }
}
```

- **400 Bad Request**
```
[
  "Incorrect format of input Message"
]
```
- **412 Precondition Failed**
```
[
  "Unreachable Position",
  "Collision detected",
  "Invalid velocity parameter: is not in (0, 2] range",
  "Not present"
]
```
- **500 Internal Server Error**
```
[
  "Robot does not respond"
]
```
- **503 Service Unavailable**
```
[
  "Robot unavailable in emergency state"
]
```

3.2.2 Setting a new arm pose

Path:

`PUT/pose`

Description: The function commands the arm to move to a new pose. A pose is as a set of output flange angles (in degrees) of the six servos in the arm joints.

Related REST API functions: GET/POSE, PUT/POSES/RUN

Request body: The request body is in accordance with the POSE SCHEMA. It specifies the angles that each of the six servos should reach to move the arm to the target pose.

Request type: application/json

Request parameters: It is **obligatory** to set one of the below parameters.

For each single request, you can set ONLY ONE of the below variants: speed, tcp_max_velocity, or velocity+acceleration.

- **speed**
- **tcp_max_velocity**
- **velocity+acceleration**

  Important! In the combination, **it is obligatory** to set both velocity and acceleration.

Parameter	Description
speed	The parameter sets the combination of velocity and acceleration (in % max. velocity and acceleration) at which the arm servos should move to the target pose angles. The **admissible value range** is from 1 to 100. **Type:** *number (double)* **Included as:** *query*

velocity	The parameter sets the velocity (in % max. velocity) at which the arm servos should move to the target pose angles. The **admissible value range** is from 1 to 100. ***ATTENTION! The velocity parameter is always used in combination with the acceleration parameter.*** **Type:** *number (double)* **Included as:** *query*
acceleration	The parameter sets the acceleration (in % max. acceleration) at which the arm servos should move to the required pose angles. The **admissible value range** is from 1 to 200. ***ATTENTION! Use values greater than 100% with caution! They are admissible only with a relatively light load or without a load. Otherwise, the arm can fail to operate.*** ***ATTENTION! The acceleration parameter is obligatory, always used in combination with the velocity parameter.*** **Type:** *number (double)* **Included as:** *query*
tcp_max_velocity	The parameter defines the limit velocity in meters per second that an end effector can reach at its TCP while moving. The **admissible value range** is from 0.001 to 2 m/s. **Included as:** *query*
motionType	The parameter sets the type of motion the arm should use to get into the specified pose. **Admissible values** are as follows: • **JOINT** When set to this motion type, the arm moves to the specified pose along a trajectory that has been calculated as the most convenient one. The trajectory can be described as a set of joint angles connected into a curve. • **LINEAR** When the motion type is LINEAR, the arm moves to the specified pose along a straight line. This motion takes more time than with the *type* parameter set to JOINT. However, the trajectory is entirely predictable, unlike with the JOINT type motion. When the user specifies no value for the parameter, it is set to the default one—JOINT. **Included as:** *query*

Request example:

Path: Depending on which of the above obligatory parameters the user chooses to set, the request path is as illustrated below.

- **speed**
  `PUT/pose?speed=100&motionType=joint`

- **velocity+acceleration**
  `PUT/pose?velocity=100&acceleration=10`
- **tcp_max_velocity**
  `PUT/pose?tcp_max_velocity=1`

Request body:

```
{
  "angles": [
    61,
    -98,
    -122,
    -49,
    89,
    -28
  ]

}
```

Response body:

HTTP status code	Description	Response schema/ type
200 OK	Success	-
400 Bad Request	Message parsing error	String
412 Precondition Failed	Incorrect input parameters	String
500 Internal Server Error	Robot error	String
503 Service unavailable	Robot emergency	String

Response examples:

- **400 Bad Request**
```
[
  "Incorrect format of input Message"
]
```
- **412 Precondition Failed**
```
[
  "Unreachable Position",
  "Collision detected",
  "Invalid velocity parameter: is not in (0, 2] range",
  "Not present"
]
```
- **500 Internal Server Error**
```
[
  "Robot does not respond"
]
```
- **503 Service Unavailable**
```
[
  "Robot unavailable in emergency state"
]
```

3.2.3 Asking the arm to move to a position

Path:

`PUT/positions/run`

Description: The function allows for setting a trajectory of one or more waypoints to move the robotic arm smoothly from one position to another. In the trajectory, each waypoint (both intermediary and target positions) is described as a set of *x, y,* and *z* coordinates, as well as *roll*, *pitch*, and *yaw* rotation angles.

The coordinates define the desired distance (in meters) from the zero point to the TCP along the *x*, *y*, and *z* axes accordingly. *Roll* stands for the desired TCP rotation angle around the *x* axis; *pitch*—the desired TCP rotation angle around the *y* axis; *yaw*—the desired TCP rotation angle around the *z* axis. All rotation angles are in radians.

Note: Similarly, you can move the arm from one position to another through one or more waypoints, using the PUT/POSITION request. When the arm is executing a trajectory of PUT/POSITION waypoints, it stops for a short moment at each preset waypoint. With the PUT/POSITIONS/RUN function, the arm moves smoothly though all waypoints without stopping, which reduces the overall time of going from one position to another.

Related REST API functions: PUT/POSITION, PUT/POSITIONS/RUN

Request body: The request body is in accordance with the POSITION SCHEMA. It specifies the coordinates (*x, y, z*) and rotation angles (*roll, pitch, yaw*) of all the waypoints on the trajectory from the initial position to the target one.

Make sure to specify all point (x, y, z) and rotation (roll, pitch, yaw) properties in the request body. Otherwise, the function returns a 400 Bad Request error.

Request type: application/json

Request parameters: It is **obligatory** to set one of the below parameters.

For each single request, you can set ONLY ONE of the below variants: speed, tcp_max_velocity, or velocity+acceleration.

- **speed**
- **tcp_max_velocity**
- **velocity+acceleration**

Important! In the combination, **it is obligatory** to set both velocity and acceleration.

Parameter	Description
speed	The parameter sets the combination of velocity and acceleration (in % max. velocity and acceleration) at which the arm should move to a waypoint position. The **admissible value range** is from 1 to 100. **Type:** *number (double)* **Included as:** *query*

velocity	The parameter sets the velocity (in % max. velocity) at which the arm should move to a waypoint position. The **admissible value range** is from 1 to 100. ***ATTENTION! The velocity parameter is always used in combination with the acceleration parameter.*** **Type:** *number (double)* **Included as:** *query*
acceleration	The parameter sets the acceleration (in % max. acceleration) at which the arm should move to a waypoint position. The **admissible value range** is from 1 to 200. ***ATTENTION! Use values greater than 100% with caution! They are admissible only with a relatively light load or without a load. Otherwise, the arm can fail to operate.*** ***ATTENTION! The acceleration parameter is obligatory, always used in combination with the velocity parameter.*** **Type:** *number (double)* **Included as:** *query*
tcp_max_velocity	The parameter defines the limit velocity in meters per second that an end effector can reach at its TCP while moving. The **admissible value range** is from 0.001 to 2 m/s. **Included as:** *query*
motionType	The parameter sets the type of motion the arm should use to get to the specified position through one or more waypoints. **Admissible values** are as follows: • **JOINT** When set to this motion type, the arm moves from one waypoint to another along a trajectory that has been calculated as the most convenient one. The trajectory can be described as a set of joint angles connected into a curve. • **LINEAR** When the motion type is LINEAR, the arm moves from one waypoint to another along a straight line. This motion takes more time than with the *type* parameter set to JOINT. However, the trajectory is entirely predictable, unlike with the JOINT type motion. When the user specifies no value for the parameter, it is set to the default one—JOINT. **Included as:** *query*

Request example:

Path: Depending on which of the above obligatory parameters the user chooses to set, the request path is as illustrated below.

- **speed**
  `PUT/positions/run?speed=100&motionType=joint`
- **velocity+acceleration**
  `PUT/positions/run?velocity=100&acceleration=10`
- **tcp_max_velocity**
  `PUT/positions/run?tcp_max_velocity=1`

Request body:

```
[
  {
    "point": {
      "x": 0.3,
      "y": -0.4,
      "z": 0.2
    },
    "rotation": {
      "roll": 3.14,
      "pitch": 0,
      "yaw": 0.5
    }
  }
]
```

Response body:

HTTP status code	Description	Response schema/ type
200 OK	Success	-
400 Bad Request	Message parsing error	String
412 Precondition Failed	Incorrect input parameters	String
500 Internal Server Error	Robot error	String
503 Service Unavailable	Robot emergency	String

Response examples:

- **200 OK**
- **400 Bad Request**

```
[
  "Incorrect format of input Message"
]
```

- **412 Precondition Failed**

```
[
  "Unreachable Position",
  "Collision detected",
  "Invalid velocity parameter: is not in (0, 2] range",
  "Not present"
]
```

- **500 Internal Server Error**
```
[
  "Robot does not respond"
]
```
- **503 Service Unavailable**
```
[
  "Robot unavailable in emergency state"
]
```

3.2.4 Asking the arm to move to a pose

Path:

`PUT/poses/run`

Description: The function allows for setting a trajectory of one or more waypoints to move the robotic arm smoothly from one pose to another. In the trajectory, each waypoint is a set of output flange angles (in degrees) of the six servos in the arm joints.

Note: Similarly, you can move the arm from one pose to another through one or more waypoints, using the PUT/POSE function. However, when the arm is executing a trajectory of PUT/POSE waypoints, it stops for a short moment at each preset waypoint. With the PUT/POSES/RUN function, the arm moves smoothly though all waypoints without stopping, which reduces the overall time of going from one pose to another.

Related REST API functions: PUT/POSE, GET/POSE

Request body: The request body is in accordance with the POSE SCHEMA. It specifies the angles that each of the six servos should reach to move the arm to a target pose.

Request content type: application/json

Request parameters: It is **obligatory** to set one of the below parameters.

For each single request, you can set ONLY ONE of the below variants: speed, tcp_max_velocity, or velocity+acceleration.

- **speed**
- **tcp_max_velocity**
- **velocity+acceleration**

  Important! In the combination, **it is obligatory** to set both velocity and acceleration.

Parameter	Description
speed	The parameter sets the combination of velocity and acceleration (in % max. velocity and acceleration) at which the arm servos should move to a waypoint. The **admissible value range** is from 1 to 100. **Type:** *number (double)* **Included as:** *query*

velocity	The parameter sets the velocity (in % max. velocity) at which the arm servos should move to a waypoint. The **admissible value range** is from 1 to 100. ***ATTENTION! The velocity parameter is always used in combination with the acceleration parameter.*** **Type:** *number (double)* **Included as:** *query*
acceleration	The parameter sets the acceleration (in % max. acceleration) at which the arm servos should move to a waypoint. The **admissible value range** is from 1 to 200. ***ATTENTION! Use values greater than 100% with caution! They are admissible only with a relatively light load or without a load. Otherwise, the arm can fail to operate.*** ***ATTENTION! The acceleration parameter is obligatory, always used in combination with the velocity parameter.*** **Type:** *number (double)* **Included as:** *query*
tcp_max_velocity	The parameter defines the limit velocity in meters per second that an end effector can reach at its TCP while moving. The **admissible value range** is from 0.001 to 2 m/s. **Included as:** *query*
motionType	The parameter sets the type of motion the arm should use to get to the specified pose through one or more waypoints. **Admissible values** are as follows: • **JOINT** When set to this motion type, the arm moves from one waypoint to another along a trajectory that has been calculated as the most convenient one. The trajectory can be described as a set of joint angles connected into a curve. • **LINEAR** When the motion type is LINEAR, the arm moves from one waypoint to another along a straight line. This motion takes more time than with the *type* parameter set to JOINT. However, the trajectory is entirely predictable, unlike with the JOINT type motion. When the user specifies no value for the parameter, it is set to the default one—JOINT. **Included as:** *query*

Request example:

Path: Depending on which of the above obligatory parameters the user chooses to set, the request path is as illustrated below.

- **speed**
  `PUT/poses/run?speed=100&motionType=joint`
- **velocity+acceleration**
  `PUT/poses/run?velocity=100&acceleration=10`
- **tcp_max_velocity**
  `PUT/poses/run?tcp_max_velocity=1`

Request body: Specifies the angles describing a waypoint.

```
[
  {
    "angles": [
      61,
      -98,
      -122,
      -49,
      89,
      -28
    ]
  }
]
```

Response body:

HTTP status code	Description	Response schema/ type
200 OK	Success	-
400 Bad Request	Message parsing error	String
412 Precondition Failed	Incorrect input parameters	String
500 Internal Server Error	Robot error	String
503 Service Unavailable	Robot emergency	String

Response content type: text/plain

Response examples:

- **200 OK**
- **400 Bad Request**
```
[
  "Incorrect format of input Message"
]
```
- **412 Precondition Failed**
```
[
  "Unreachable Position",
  "Collision detected",
]
```
- **500 Internal Server Error**
```
[
  "Robot does not respond"
]
```
- **503 Service Unavailable**
```
[
  "Robot unavailable in emergency state"
]
```

3.2.5 Asking the arm to open the gripper

Path:

```
PUT/gripper/open
```

Description: The function commands the arm to open the gripper. It has no request body, but the user can optionally set one parameter—timeout.

Related REST API functions: PUT/GRIPPER/CLOSE

Request parameter:

Parameter	Description
timeout	The parameter specifies how long (in milliseconds) the arm should remain idle, waiting for the gripper to open. The default manufacturer-preset value is 500 ms. **Admissible value range:** integers from 1 and above *When the parameter setting is out of the admissible range, it is replaced automatically with the default value.* **Type:** *number (int32)* **Included as:** *query*

Response content type: text/plain

Response body:

HTTP status code	Response schema/ type
200 OK	-
500 Internal Server Error	Robot error string
503 Service Unavailable	Robot emergency string

Response examples:

- **200 OK**
- **500 Internal Server Error**

```
[
  "Robot does not respond"
]
```

- **503 Service Unavailable**

```
[
  "Robot unavailable in emergency state"
]
```

3.2.6 Asking the arm to close the gripper

Path:

```
PUT/gripper/close
```

Description: The function commands the arm to close the gripper. It has no request body, but the user can optionally set one parameter—timeout.

Related REST API functions: PUT/GRIPPER/OPEN

Request parameter:

Parameter	Description
timeout	The parameter specifies how long (in milliseconds) the arm should remain idle, waiting for the gripper to close. The default manufacturer-preset value is 500 ms. **Admissible value range:** integers from 1 and above *When the parameter setting is out of the admissible range, it is replaced automatically with the default value.* **Type:** *number (int32)* **Included as:** *query*

Response content type: text/plain

Response body:

HTTP status code	Response schema/ type
200 OK	-
500 Internal Server Error	Robot error string
503 Service Unavailable	Robot emergency string

Response examples:

- **200 OK**
- **500 Internal Server Error**
```
[
  "Robot does not respond"
]
```
- **503 Service Unavailable**
```
[
  "Robot unavailable in emergency state"
]
```

3.2.7 Asking the arm to relax

Path:

`PUT/relax`

Description: The function sets the arm in the "relaxed" state. The arm stops moving without retaining its last position. In this state, the user can move the robotic arm by hand (e. g., to verify/ test a motion trajectory).

Related REST API functions: PUT/FREEZE

Response content type: text/plain

Response body:

HTTP status code	Response schema/ type
200 OK	-
500 Internal Server Error	Robot error string
503 Service Unavailable	Robot emergency string

Response examples:

- **200 OK**
- **500 Internal Server Error**
```
[
  "Robot does not respond"
]
```
- **503 Service Unavailable**
```
[
  "Robot unavailable in emergency state"
]
```

3.2.8 Asking the arm to go to the freeze state

Path:

`PUT/freeze`

Description: The function sets the arm in the "freeze" state. The arm stops moving, retaining its last position.

⚠ ***In the state, it is not advisable to move the arm by hand as this can cause damage.***

Related REST API functions: **PUT/RELAX**

Response content type: text/plain

Response body:

HTTP status code	Response schema/ type
200 OK	-
500 Internal Server Error	Robot error string
503 Service Unavailable	Robot emergency string

Response examples:

- **200 OK**
- **500 Internal Server Error**
```
[
  "Robot does not respond"
]
```
- **503 Service Unavailable**
```
[
  "Robot unavailable in emergency state"
]
```

3.2.9 Controlling the arm in the jogging mode

Path:

`PUT/jogging`

Description: The function sets the arm in the *jogging mode*, while also enabling users to specify jogging motion parameters. In the mode, the arm moves along or rotates around one or more preset coordinate axes (*x, y, z*) at a pre-defined acceleration rate.

The arm continues moving in the mode at preset parameters until the mode is disabled. There are **two way to disable the mode** (see the request body examples below):

- to send a `PUT/jogging` request with a void "`acceleration`" object in the request body; or
- to set all properties of the "`acceleration`" object to 0.

To have a thorough understanding how you can best implement the jogging mode in your environment, we recommend contacting our support specialists.

Request body: The request body is in accordance with the JOGGING ACCELERATION SCHEMA. The schema defines acceleration of the arm along the coordinate axes (*x, y, z*) and at *roll, pitch,* and *yaw* rotation angles (*rx, ry, rz*). **Note** that all the coordinates and rotation angles are relative to the base of the robot arm.

All the properties in the schema are optional — you can set and omit them at your discretion. The **admissible value range** for all properties is from -1 to 1. When no value is specified for a property, the property is set to the default 0. Accordingly, when no values are specified for any of the properties, all of them are set to the default 0, in which case the arm stops and the jogging mode is disabled.

Request content type: application/json

Request examples:

- **To set jogging parameters**

```
{
    "acceleration" : {
        "x" : 1
        "y" : -0.5
        "z" : 0
        "rx" : 0
        "ry" : 0.5
        "rz" : -0.1
    }
}
```

- **To disable the jogging mode (alternative 1)**

```
PUT /jogging '{"acceleration": {}}
```

- **To disable the jogging mode (alternative 2)**

```
PUT /jogging

{
    "acceleration" : {
        "x" : 0
        "y" : 0
        "z" : 0
        "rx" : 0
        "ry" : 0
        "rz" : 0
    }
}
```

Response content type: text/plain

Response body:

HTTP status code	Response schema/ type
200 OK	-
500 Internal Server Error	Robot error string
503 Service Unavailable	Robot emergency string

Response examples:

- **200 OK**
- **500 Internal Server Error**
```
[
  "Robot does not respond"
]
```
- **503 Service Unavailable**
```
[
  "Robot unavailable in emergency state"
]
```

3.2.10 Setting high signal level on a digital output

Path:

`PUT/signal/output/{port}/high`

Description: The function sets the digital output specified in the `{port}` parameter of the request path to the HIGH signal level.

ATTENTION! SPECIFYING THE `{port}` PARAMETER IS MANDATORY!

A digital output is a physical port on the back panel of the control box. Since the control box has two digital outputs, the parameter value can be either *1* (corresponds to Relay output 1) or *2* (corresponds to Relay output 2).

For location of the digital outputs, refer to the document User Manual.

Related REST API functions: GET/SIGNAL/OUTPUT/{PORT}, PUT /SIGNAL/OUTPUT/{PORT}/LOW

Response content type: text/plain, application/json

Response body:

HTTP status code	Response schema/ type
200 OK	-
412 Precondition Failed	Incorrect input parameters
500 Internal Server Error	Robot error string
503 Service Unavailable	Robot emergency string

Response examples:

- **200 OK**

- **412 Precondition Failed**
```
[
  "Unable to use parameter value {13}"
]
```
- **500 Internal Server Error**
```
[
  "Robot does not respond"
]
```
- **503 Service Unavailable**
```
[
  "Robot unavailable in emergency state"
]
```

3.2.11 Setting low signal level on a digital output

Path:

```
PUT /signal/output/{port}/low
```

Description: The function sets the digital output specified in the *{port}* parameter of the request path to the LOW signal level.

ATTENTION! SPECIFYING THE* {port} *PARAMETER IS MANDATORY!

A digital output is a physical port on the back panel of the control box. Since the control box has two digital outputs, the parameter value can be either *1* (corresponds to Relay output 1) or *2* (corresponds to Relay output 2). For location of the digital outputs and their detailed description, refer to the User Manual.

Related REST API functions: PUT/SIGNAL/OUTPUT/{PORT}/HIGH, GET/SIGNAL/OUTPUT/{PORT}

Response content type: text/plain, application/json

Response body:

HTTP status code	Response schema/ type
200 OK	String
412 Precondition Failed	Incorrect input parameters
500 Internal Server Error	Robot error string
503 Service Unavailable	Robot emergency string

Response examples:

- **200 OK**
- **412 Precondition Failed**
```
[
  "Unable to use parameter value {13}"
]
```
- **500 Internal Server Error**
```
[
  "Robot does not respond"
]
```

- **503 Service Unavailable**
  ```
  [
    "Robot unavailable in emergency state"
  ]
  ```

3.2.12 Recovering the arm after an emergency

Path:

`PUT/recover`

Description: The function recovers the arm after an emergency, setting its motion status to IDLE. Recovery is possible only after an emergency that is not fatal (a non-fatal error corresponds to the ERROR status) (see GET/STATUS/MOTION).

With the 200 OK status code, the function returns either of two values:

- `SUCCESS`—the recovery has been completed as appropriate
- `FAILED`—the recovery has failed

Response content type: text/plain

Response body:

HTTP status code	Response schema/ type
200 OK	String enum: [SUCCESS, FAILED]
500 Internal Server Error	Robot error string
503 Service Unavailable	Robot emergency string

Response examples:

- **200 OK**
  ```
  [
    "SUCCESS",
    "FAILED"
  ]
  ```
- **500 Internal Server Error**
  ```
  [
    "Robot does not respond"
  ]
  ```
- **503 Service Unavailable**
  ```
  [
    "Robot unavailable in emergency state"
  ]
  ```

3.2.13 Adding an obstacle to the arm's environment

Path:
`PUT/environment`

Description: The function enables adding obstacles to the environment of a robotic arm for collision detection purposes.

An *obstacle* is any object, such as a control box or a wall, in the way of an arm to be taken into consideration for collision detection. An obstacle can be one of the following types:

- **BOX—**typically used to describe obstacles with a shape reminding that of a box.

- **CAPSULE—**preferred for objects of cylindrical shape or having complex structure and irregular outlines. In the latter two cases, it is also possible to describe an obstacle using multiple capsules.
- **PLANE**—recommended for describing plain-surface objects, such as a wall or a table.

After a power-off, any obstacle settings for a specific environment are reset to defaults (cleared from the device memory).

Note: With ***a single*** ***`PUT/environment`*** ***request***, it is possible to add ***only one obstacle***. To add multiple obstacles, create and send the required quantity of `PUT/environment` requests.

Request body: The request body is in accordance with the OBSTACLE SCHEMA and contains a single data array comprising the following:

- **Obstacle type**—a geometric pattern, roughly describing the shape of an obstacle for collision detection purposes— **BOX, CAPSULE,** and **PLANE.**
- **Name**—any random name as defined by the user for a specific obstacle type (e.g., "`first_box`").
- **Obstacle properties—**spatial location in space and / or dimensions of a specific obstacle.

  Each obstacle type has its own set of properties as described in the table below.

Type	Properties
BOX	- **sides**—the *x*, *y*, and *z* coordinates defining the spatial dimensions of an obstacle (i.e., length, width, depth). - **position**—a set of the *x*, *y*, and *z* coordinates, as well as *roll, pitch* and *yaw* angles defining the location of an obstacle in space. The coordinate values are distances (in meters) along the *x*, *y*, and *z* axes accordingly, measured from the obstacle's center point relative to the zero point (see GLOSSARY). *Roll, pitch* and *yaw* are rotation angles (in radians) of the obstacle's center point relative to the zero point.
CAPSULE	- **radius**—the radius (in meters) of the capsule shape incorporating an obstacle, measured from the obstacle's center point - **start point—**the starting *x*, *y*, and *z* coordinates (in meters) of the capsule shape length relative to the zero point - **end point—**the end *x*, *y*, and *z* coordinates (in meters) of the capsule shape length relative to the zero point
PLANE	- **points**—at least three points constituting a single plane; each of the points is described as a set of *x*, *y*, and *z* coordinates (in meters) on the plane

Related REST API functions: GET/ENVIRONMENT, GET/ENVIRONMENT/{OBSTACLE}, DELETE/ENVIRONMENT, DELETE/ENVIRONMENT/{OBSTACLE}

Request example:

```
[
  {
    "obstacleType": "BOX",
    "name": "example_box",
    "sides": {
      "x": 0.1,
      "y": 0.1,
      "z": 0.1
    },
    "position": {
      "point": {
        "x": 1,
        "y": 1,
        "z": 1
      },
      "rotation": {
        "roll": 0,
        "pitch": 0,
        "yaw": 0
      }
    }
  },
  {
    "obstacleType": "CAPSULE",
    "name": "example_capsule",
    "radius": 0.1,
    "begin": {
      "x": 0.5,
      "y": 0.5,
      "z": 0.2
    },
    "finish": {
      "x": 0.5,
      "y": 0.5,
      "z": 0.2
    }
  },
  {
    "obstacleType": "PLANE",
    "name": "example_plane",
    "points": [
      {
        "x": -0.5,
        "y": 0.2,
        "z": 0
      },
      {
        "x": -0.5,
        "y": 0,
        "z": 0
      },
        {
          "x": -0.5,
          "y": 0,
          "z": 0.1
        }
      ]
    }
]
```

Response content type: text/plain

Response body:

HTTP status code	Response schema/ type
200 OK	OBSTACLE SCHEMA
412 Precondition Failed	Incorrect input parameters
500 Internal Server Error	Robot error string
503 Service Unavailable	Robot emergency string

Response examples:

- **200 OK**
- **412 Precondition Failed**

```
[
  "Unable to use parameter value {13}"
]
```

- **500 Internal Server Error**

```
[
  "Robot does not respond"
]
```

- **503 Service Unavailable**

```
[
  "Robot unavailable in emergency state"
]
```

3.2.14 Setting the arm into a transportation pose

Path:

```
PUT/pack
```

Description: The function sets the arm into a preset pose for transportation.

Response content type: text/plain

Response body:

HTTP status code	Response schema/ type
200 OK	-
500 Internal Server Error	Robot error string
503 Service Unavailable	Robot emergency string

Response examples:

- **200 OK**

- **500 Internal Server Error**
```
[
  "Robot does not respond"
]
```
- **503 Service Unavailable**
```
[
  "Robot unavailable in emergency state"
]
```

3.2.15 Quitting the untwisting mode

Path:

`PUT/untwisting/finish`

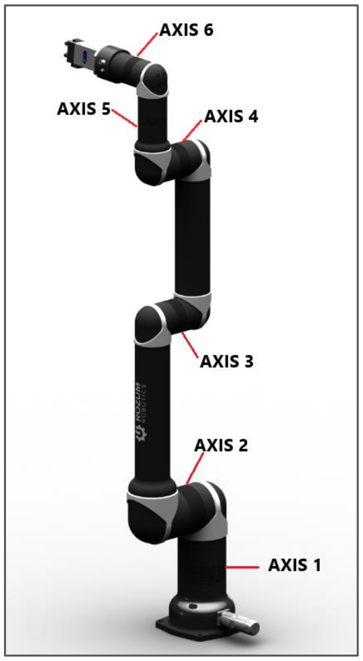
Description: The function enables users to verify the results of untwisting and quit the untwisting mode. In the untwisting mode, PUT and other API requests to move the arm are unavailable, until untwisting is completed. Users can only work with GET requests.

The arm goes into the untwisting mode after an emergency shutdown if a *twist* is detected on one or more motors in its joints during initialization. Simultaneously, a twist detection alert is generated, containing the following information:

- which axis (one or more) has a motor with a twist
- how many turns to make to untwist the axis (axes)
- in which direction to make the turns

A twist is when a motor has made more than 360° turn. Multiple twists can lead to wire breaks and other irreparable damages.

Attention! Before applying the function, you have to untwist motor(s) manually as instructed in the associated twist detection alert and taking into consideration the location of the arm axes.



Request body: The function has no request body.

Response body: The function either notifies about successful completion of manual untwisting or returns a twist detection alert.

HTTP status code	Description	Response schema/ type
200 OK	Success	String
500 Internal Server Error	Robot error string	String
503 Service Unavailable	Robot emergency string	String

Response content type: text/plain

Response examples:

- **200 OK**
- **500 Internal Server Error**
```
[
  "Robot does not respond"
]
```
- **503 Service Unavailable**
```
[
  "Robot unavailable in emergency state"
]
```

3.2.16 Setting tool properties

Path:

```
POST/tool/info
```

Description: The function enables setting tool properties for collision detection purposes, in particular:

- **name** — any random name of the work tool defined by the user (e.g., "gripper").
- **actual TCP position** described as a set of the following properties:
  - **point—***x, y,* and *z* coordinates defining the offset (in meters) along the *x, y,* and *z* axes accordingly from the original TCP (see GLOSSARY) after adding / changing the work tool.
  - **rotation angles—***roll*, *pitch*, and *yaw***.** *Roll* stands for the actual TCP rotation angle around the *x* axis; *pitch*—the actual TCP rotation angle around the *y* axis; *yaw*—the actual TCP rotation angle around the *z* axis. All rotation angles are in radians and relative to the physical center point of the arm base.

Related REST API functions: GET/TOOL/INFO**,** GET/TOOL/SHAPE, POST/TOOL/SHAPE

Request content type: application/json, text/plain

Request body: The request body is in accordance with the TOOL INFO SCHEMA.

Request example:

```
{
  "name": "gripper",
  "tcp": {
    "point": {
      "x": 0.3,
      "y": -0.4,
      "z": 0.2
    },
    "rotation": {
      "roll": 3.14,
      "pitch": 0,
      "yaw": 0.5
    }
  }
}
```

Response content type: application/json, text/plain

Response body:

HTTP status code	Description	Response schema/ type
200 OK	TOOL INFO SCHEMA	TOOL INFO SCHEMA
400 Bad Request	Message parsing error	String
412 Precondition Failed	Incorrect input parameters	String
500 Internal Server Error	Robot error string	String

Response examples:

- **200 OK**

```
{
  "name": "gripper",
  "tcp": {
    "point": {
      "x": 0.3,
      "y": -0.4,
      "z": 0.2
    },
    "rotation": {
      "roll": 3.14,
      "pitch": 0,
      "yaw": 0.5
    }
  }
}
```

- **400 Bad Request**

```
[
  "Incorrect format of input Message"
]
```

- **412 Precondition Failed**

```
[
  "Unreachable Position",
  "Collision detected",
]
```

- **500 Internal Server Error**

```
[
  "Robot does not respond"
]
```

3.2.17 Setting the tool shape

Path:

```
POST/tool/shape
```

Description: The function enables setting tool shape for collision detection purposes by defining the following properties:

- **radius** — radius of the work tool (in meters) measured from its physical center point.
- **begin** — the start *x, y,* and *z* coordinates of the work tool capsule measured as a distance (in meters) from the original TCP.
- **finish** — the end *x, y,* and *z* coordinates of the work tool capsule measured as a distance (in meters) from the original TCP.

Related REST API functions: GET/TOOL/INFO, GET/TOOL/SHAPE, POST/TOOL/INFO

Request content type: application/json, text/plain

Request body: The request body is in accordance with the TOOL SHAPE SCHEMA.

Request example:

```
{
  "shape": [
    {
      "radius": 0.5,
      "begin": {
        "x": 0.3,
        "y": -0.4,
        "z": 0.2
      },
      "finish": {
        "x": 0.3,
        "y": -0.4,
        "z": 0.2
      }
    }
  ]
}
```

Response content type: application/json, text/plain

Response body:

HTTP status code	Description	Response schema/ type
200 OK	TOOL SHAPE SCHEMA	TOOL SHAPE SCHEMA
400 Bad Request	Message parsing error	String
412 Precondition Failed	Incorrect input parameters	String
500 Internal Server Error	Robot error string	String

Response examples:

- **200 OK**

```
{
  "shape": [
    {
      "radius": 0.5,
      "begin": {
        "x": 0.3,
        "y": -0.4,
        "z": 0.2
      },
      "finish": {
        "x": 0.3,
        "y": -0.4,
        "z": 0.2
      }
    }
  ]
}
```

- **400 Bad Request**

```
[
  "Incorrect format of input Message"
]
```

- **412 Precondition Failed**

```
[
  "Unreachable Position",
  "Collision detected",
]
```

- **500 Internal Server Error**

```
[
  "Robot does not respond"
]
```

3.2.18 Setting a new zero point position

Path:

`POST/base`

Description: The function enables setting a new zero point position of the robotic arm as required for the current user environment (e.g., considering the surrounding obstacles). The new zero point position is described as a set of *x, y,* and *z* coordinates, as well as *roll*, *pitch*, and *yaw* rotation angles.

The coordinates define the desired offset (in meters) from the physical center point of the arm base (original zero point) along the *x, y,* and *z* axes accordingly. *Roll* stands for the rotation angle around the *x* axis; *pitch*—the rotation angle around the *y* axis; *yaw*—the rotation angle around the *z* axis. All rotation angles are in radians and relative to the physical center point of the arm base.

Related REST API functions: GET/BASE

Request content type: application/json

Request body: The request body is in accordance with the POSITION SCHEMA. It specifies the coordinates and rotation angles of the new zero point.

Request example:

```
{
  "point": {
    "x": 0.3,
    "y": -0.4,
    "z": 0.2
  },
  "rotation": {
    "roll": 3.14,
    "pitch": 0,
    "yaw": 0.5
  }
}
```

Response content type: application/json, text/plain

Response body:

HTTP status code	Description	Response schema/ type
200 OK	Success	String
400 Bad Request	Message parsing error	String
412 Precondition Failed	Incorrect input parameters	String
500 Internal Server Error	Robot error string	String

Response examples:

- **200 OK**
- **400 Bad Request**

```
[
  "Incorrect format of input Message"
]
```

- **412 Precondition Failed**

```
[
  "Unreachable Position",
  "Collision detected",
]
```

- **500 Internal Server Error**

```
[
  "Robot does not respond"
]
```

3.3 Requests to delete parameters of the arm (DELETE)

3.3.1 Removing all obstacles from the arm environment

Path:

```
DELETE/environment
```

Description: The function removes preset obstacles from the environment of a robotic arm. An *obstacle* is any object, such as a control box or a wall, in the way of an arm to be taken into consideration for collision detection.

After a power-off, any obstacle settings for a specific environment are reset to defaults (cleared from the device memory).

Related REST API functions: GET/ENVIRONMENT, GET/ENVIRONMENT/{OBSTACLE}, PUT/ENVIRONMENT, DELETE/ENVIRONMENT/{OBSTACLE}

Request body: The function has no request body.

Response content type: text/plain

Response body:

HTTP status code	Response schema/ type
200 OK	Success
500 Internal Server Error	Robot error string
503 Service Unavailable	Robot emergency string

Response examples:

- **200 OK**
- **500 Internal Server Error**

```
[
  "Robot does not respond"
]
```

- **503 Service Unavailable**

```
[
  "Robot unavailable in emergency state"
]
```

3.3.2 Removing a specific obstacle from the arm environment

Path:

`DELETE/environment/{obstacle}`

Description: The function enables removing a single preset obstacle as specified in the *{obstacle}* parameter from the environment of a robotic arm.

ATTENTION! SPECIFYING THE `{obstacle}` ***PARAMETER IS MANDATORY!***

An *obstacle* is any object, such as a control box or a wall, in the way of an arm to be taken into consideration for collision detection.

Related REST API functions: DELETE/ENVIRONMENT, GET/ENVIRONMENT, GET/ENVIRONMENT/{OBSTACLE}, PUT/ENVIRONMENT

Request body: The function has no request body.

Response content type: text/plain

Response body:

HTTP status code	Response schema/ type
200 OK	Success
500 Internal Server Error	Robot error string
503 Service Unavailable	Robot emergency string

Response examples:

- **200 OK**
- **500 Internal Server Error**
```
[
  "Robot does not respond"
]
```
- **503 Service Unavailable**
```
[
  "Robot unavailable in emergency state"
]
```

ANNEX 1. RESPONSE/ REQUEST SCHEMAS

The Annex contains schemas for structuring the API requests and responses described in the above sections.

Position schema

Object	Properties	Example
Point	x: double number (meters) y: double number (meters) z: double number (meters)	`{ "x": 0.3, "y": -0.4, "z": 0.2 }`
Rotation	roll: double number (radians) pitch: double number (radians) yaw: double number (radians)	`{ "roll": "3.14", "pitch": "0", "yaw": "0.5" }`

Pose schema

Object	Property	Example
Angles	Double numbers (degrees)	`{ "angles": [ "61", "-98", "-122", "-49", "89", "-28" ] }`

Motor status array schema

Property	Property content	Example
Angle	Double number (degrees)	`{ "angle": "168.89699", "rotorVelocity": "-0.00064343837", "rmsCurrent": "0.01", "voltage": "47.795017", "phaseCurrent": "0.01", "statorTemperature": "27.990631", "servoTemperature": "31.739925", "velocityError": "-0.022674553", "velocitySetpoint": "-0.02331799", "velocityOutput": "0.01", "velocityFeedback": "-0.00064343837", "positionError": "0.0385437", "positionSetpoint": "168.93799", "positionOutput": "0.01", "positionFeedback": "168.89944", }`
Rotor velocity	Double number (RPM)	
RMS current	Double number (Amperes)	
Voltage	Double number (Volts)	
Phase current	Double number (Amperes)	
Stator temperature	Double number (degrees C)	
Servo temperature	Double number (degrees C)	
Velocity error	Double number (RPM)	
Velocity setpoint	Double number (RPM)	
Velocity output	Double number (Amperes)	
Velocity feedback	Double number (RPM)	
Position error	Double number (degrees)	
Position setpoint	Double number (degrees)	
Position output	Double number (RPM)	
Position feedback	Double number (degrees)	

Tool info schema

Object (property)	Property content	Examples

Name	String	`{`
		`"name": "gripper",`
		`}`

TCP		TCP

Point	x: double number (meters)
	y: double number (meters)
	z: double number (meters)
		`{`
		`"x": "0.3",`
		`"y": "-0.4",`
		`"z": "0.2"`
		`}`

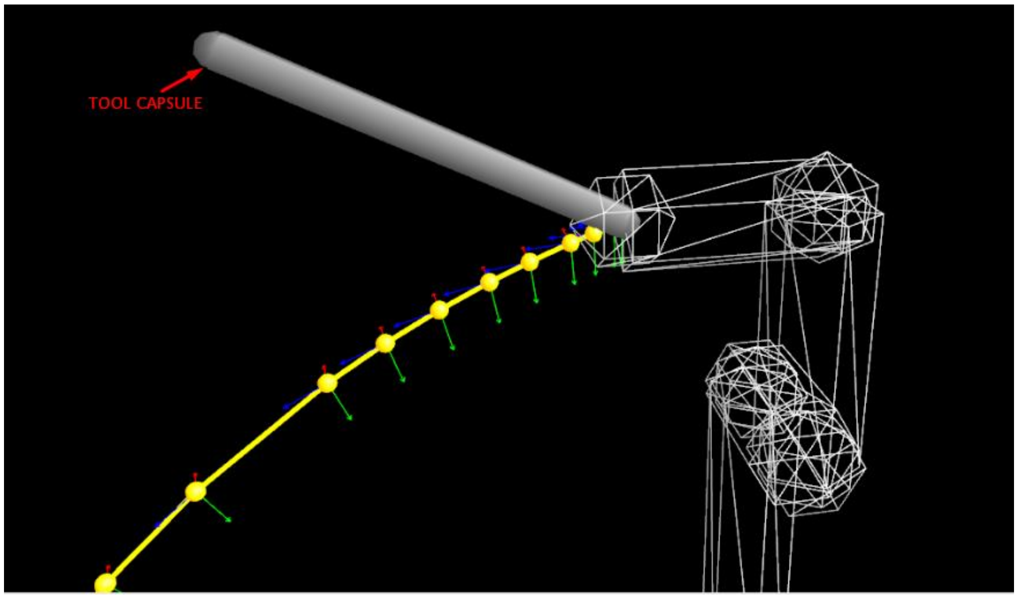
Rotation	roll: double number (radians)
	pitch: double number (radians)
	yaw: double number (radians)
		`{`
		`"roll": "3.14",`
		`"pitch": "0",`
		`"yaw": "0.5"`
		`}`

Tool shape schema

Object	Property content	Examples

Shape	radius: double number (meters)
	begin:
	x: double number (meters)
	y: double number (meters)
	z: double number (meters)
	finish:
	x: double number (meters)
	y: double number (meters)
	z: double number (meters)
		`{`
		`“shape”: [`
		`{`
		`"radius": 0.5,`
		`"begin": {`
		`"x": 0.3,`
		`"y": -0.4,`
		`"z": 0.2`
		`},`
		`"finish": {`
		`"x": 0.3,`
		`"y": -0.4,`
		`"z": 0.2`
		`}`
		`}`

The figure below is an illustration of a defined tool shape — CAPSULE.



Obstacle schema

Object (property)	Property content	Examples

Obstacle type
Name
	String enum: [BOX, CAPSULE, PLANE] String	{
		"obstacleType": "BOX",
		"name": "workspace"
		}

BOX		

Obstacle type
Name
 	
Sides
Point
 	
Center position
Point
 	
Rotation
	obstacleType: string
	name: string
	
	x: double number (meters)
	y: double number (meters)
	z: double number (meters)
	
	x: double number (meters)
	y: double number (meters)
	z: double number (meters)
	
	roll: double number (radians)
	pitch: double number (radians)
	yaw: double number (radians)
		{
		"obstacleType": "BOX",
		"name": "first_box",
		"sides": {
		"x": 0.3,
		"y": -0.4,
		"z": 0.2
		},
		"centerPosition": {
		"point": {
		"x": 0.3,
		"y": -0.4,
		"z": 0.2
		},
		"rotation": {
		"roll": 3.14,
		"pitch": 0,
		"yaw": 0.5
		}
		}
		}

CAPSULE		

Obstacle type
Name
Radius
Point
 	
Point
	obstacleType: string
	name: string
	radius: double number (meters)
	startPoint:
	x: double number (meters)
	y: double number (meters)
	z: double number (meters)
	
	endPoint:
	x: double number (meters)
	y: double number (meters)
	z: double number (meters)
		{
		"obstacleType": " CAPSULE",
		"name": "first_capsule",
		"radius": 0.5,
		"startPoint": {
		"x": 0.3,
		"y": -0.4,
		"z": 0.2
		},
		"endPoint": {
		"x": 0.3,
		"y": -0.4,
		"z": 0.2
		}
		}

PLANE		

Obstacle type
Name
Point
	obstacleType: string
	name: string
	points:
	x: double number (meters)
	y: double number (meters)
	z: double number (meters)
		{
		"obstacleType": "PLANE",
		"name": "first_plane",
		"points": [
		{
		"x": 0.3,
		"y": -0.4,
		"z": 0.2
		}
		]
		}

Version schema

Object	Property content	Examples
Version	{ "motorsVersion": ["string"], "safetyVersion": "string", "usbCanVersion": "string", "wristVersion": "string" }	{ "motorsVersion": ["string"], "safetyVersion": "string", "usbCanVersion": "string", "wristVersion": "string" }

System status schema

Object	Property content	Examples

State	"state":
	string enum. [INITIALIZING, INITIALIZATION_FAILURE, TWISTED, ACTIVE, MOTION, ZERO_GRAVITY, JOGGING, BROKEN, EMERGENCY]
	"message": "string"
	}
		{
		"state": [
		"INITIALIZING",
		"INITIALIZATION_FAILURE",
		"TWISTED",
		"ACTIVE",
		"MOTION",
		"ZERO_GRAVITY",
		"JOGGING",
		"BROKEN",
		"EMERGENCY"
		],
		"message": "string"
		}

Jogging acceleration schema

Object	Property content	Examples

Acceleration	x: number (double)
	y: number (double)
	z: number (double)
	rx: number (double)
	ry: number (double)
	rz: number (double)
		{
		"acceleration" : {
		"x" : 1
		"y" : -0.5
		"z" : 0
		"rx" : 0
		"ry" : 0.5
		"rz" : -0.1
		}
		}

Robot info schema

Object	Property content	Examples

Robot info	model: string
	version: string
	serialNumber: string
		{
		"model": "pulse75",
		"version": "2.4.0",
		"serialNumber": "18-00604"
		}