PULSE
COLLABORATIVE ROBOT

API REFERENCE GUIDE

ROZUM ROBOTICS

TABLE OF CONTENTS

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

The REST Application Programming Interface (API) described in this reference guide implements the functionality for monitoring and controlling motion of 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 for reading and writing motion parameters is based on the HTTP (v 2.0) methods listed in **Table 1-1**.

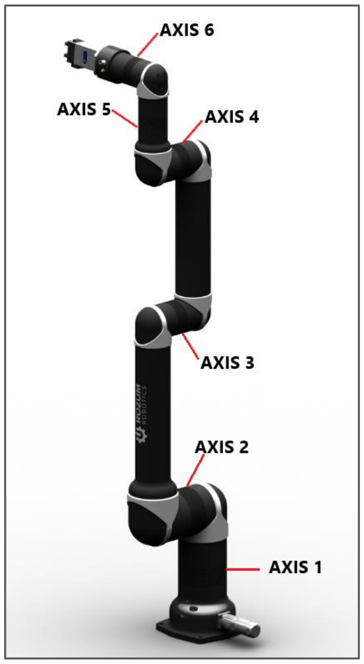
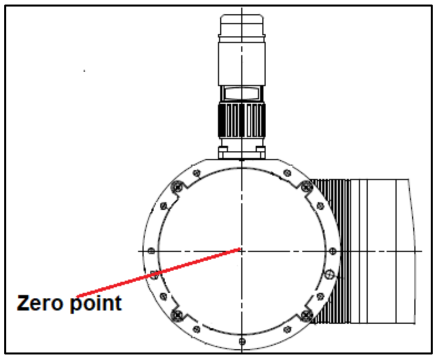
Table 1-1: **Supported HTTP methods**

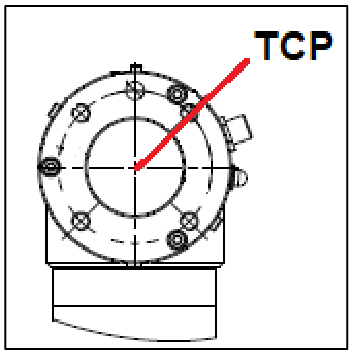
Method	Purpose
GET	• to get the actual pose / position of the robotic arm (rotation angles and coordinates of its joints) • to get the actual state of the robotic arm (e.g., idle) • to get the actual state of the servo motors in the arm joints (e.g., voltage, rotor velocity) • to get actual properties (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 version
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 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 axes 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 3.2.16****).*

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 3.2.15**), 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 first switching.

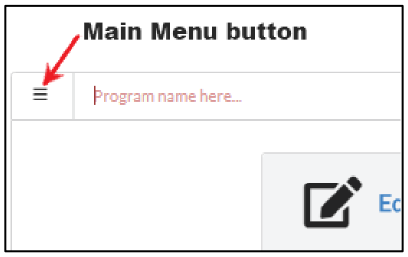
Before you attempt to enable API control, the PULSE arm should be:

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

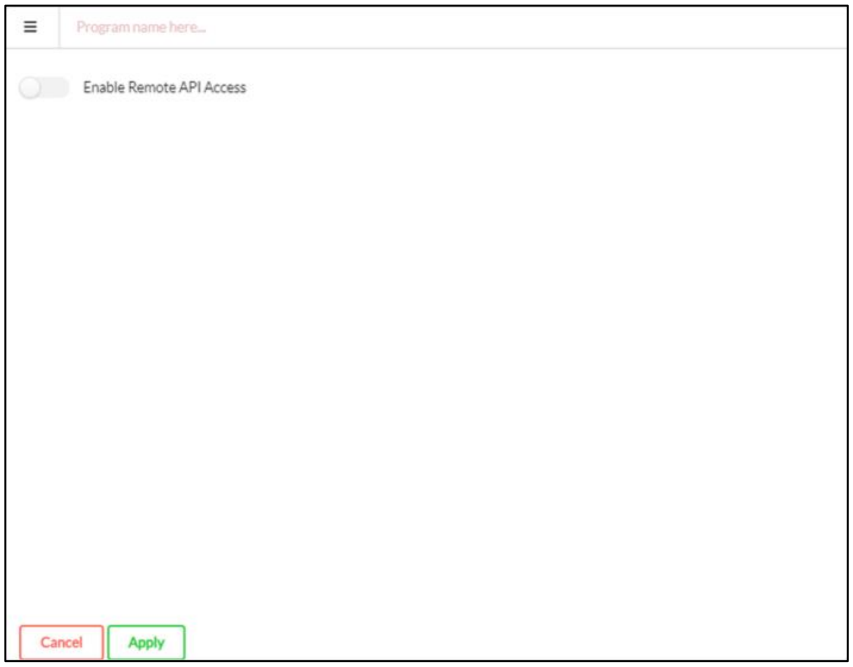
For connection and switching instructions, refer to HARDWARE INSTALLATION MANUAL.

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 constantly on, and the LED on the arm wrist should be steady green.
2. Start the PULSE DESK user interface as described in the USER MANUAL.
3. In the displayed starting screen of the PULSE DESK interface, click the **Main Menu** button.



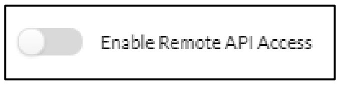
4. In the displayed menu, select **Configure**. PULSE DESK displays the **Configure** screen.



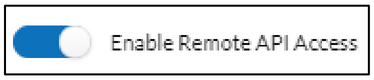
The Configure screen

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 available API functions.

3 DESCRIPTION OF API FUNCTIONS

The section describes in detail the **REST API functions** you can use to control the PULSE robotic arm and its work tool (a gripper), as well as to monitor the arm's motion 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	String
503 Service Unavailable	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 motion status

Path:

GET/status/motion

Description: The function returns the actual state of the robotic arm. 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	String enum: [IDLE, ZERO_GRAVITY, RUNNING, MOTION_FAILED, EMERGENCY, ERROR]
500 Internal Server Error	String
503 Service Unavailable	String

Response examples:

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

3.1.3 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	String
503 Service Unavailable	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"
```

3.1.4 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	String
503 Service Unavailable	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"
```

3.1.5 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	String
503 Service Unavailable	String

Response examples:

- **200 OK**

```
{
  "name": "gripper",
  "tcp": {
     "point": {
          "x": 0,
          "y": 0,
          "z": 0.31
        },
      "rotation": {
          "roll": 0,
          "pitch": 0,
          "yaw": 0
      }
    }
  }
```

- **500 Internal Server Error**

```
"Robot does not respond"
```

- **503 Service Unavailable**

```
"Robot unavailable in emergency state"
```

3.1.6 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 schemaTool info schema**
500 Internal Server Error	String
503 Service Unavailable	String

Response examples:

- **200 OK**

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

- **500 Internal Server Error**
  `"Robot does not respond"`

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

3.1.7 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	String
503 Service Unavailable	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"
```

3.1.8 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	String
503 Service Unavailable	String

Response examples:

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

3.1.9 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 Hardware Installation 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	string enum: [HIGH, LOW]
412 Precondition Failed	String
500 Internal Server Error	String
503 Service Unavailable	String

Response examples:

- **200 OK**
  `"HIGH"`
- **412 Precondition Failed**
  `"Unable parameter value {13}"`
- **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 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 Hardware Installation Manual.

Response content type: text/plain

Response body:

HTTP status code	Response schema/ type
200 OK	string enum: [HIGH, LOW]
412 Precondition failed	String
500 Internal Server Error	String
503 Service Unavailable	String

Response examples:

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

3.1.11 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	String
503 Service Unavailable	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"`

3.1.12 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	String
503 Service Unavailable	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 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	String
503 Service Unavailable	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"`

3.1.14 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

Response content type: application/json, text/plain

Response body:

HTTP status code	Response schema/ type
200 OK	**Version schema**
500 Internal Server Error	String
503 Service Unavailable	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 arm 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

Response content type: application/json, text/plain

Response body:

HTTP status code	Response schema/ type
200 OK	String
500 Internal Server Error	String
503 Service Unavailable	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"
```

3.2 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:

Parameter	Description
speed	The parameter sets the speed (in % max speed) at which the arm should move to the required position. The **admissible value range** is from 1 to 100. ***ATTENTION! SPECIFYING THE "speed" PARAMETER IS MANDATORY OTHERWISE AN ERROR IS GENERATED.*** **Type:** *number (double)* **Included as:** *query*
type	The parameter sets the type of motion the arm should use to get to the required 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*

tcp_max_velocity	The parameter defines the limit velocity in meters per second that an end effector can reach at its TCP while moving. It is not mandatory. When the user specifies no value for it, it is set to default. The default setting is 2 m/s. The admissible value range is from 0.001 to 2 m/s. **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 required TCP position.

Make sure to specify all point (x, y, z coordinates) and rotation (roll, pitch, yaw) properties in the request body. When at least one of the properties is not specified, the function returns a 400 Bad Request error.

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	Actual arm position	**Position schema**
400 Bad Request	Message parsing error	String
412 Precondition Failed	Incorrect input parameters	String
500 Internal Server Error	Arm error	String
503 Service unavailable	Arm 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 integrated into 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 required pose.

Request type: application/json

Request parameters:

Parameter	Description
speed	The parameter sets the speed (in % max. speed) at which servos should move to the required angles. The **admissible value range** is from 1 to 100. ***ATTENTION! SPECIFYING THE "speed" PARAMETER IS MANDATORY OTHERWISE AN ERROR IS GENERATED.*** **Type:** *number (double)* **Included as:** *query*
type *(for continuation, see the next page)*	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.

type *(continued)*	When the user specifies no value for the parameter, it is set to the default one—JOINT. **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. It is not mandatory. When the user specifies no value for it, it is set to default. The default setting is 2 m/s. The admissible value range is from 0.001 to 2 m/s. **Included as:** query

Request example:

```
{
  "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	Arm error	String
503 Service unavailable	Arm emergency	String

Response examples:

- **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"`

3.2.3 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	String
503 Service Unavailable	String

Response examples:

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

3.2.4 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 incorrect, put of the admissible range, to comply with the request, 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	String
500 Internal Server Error	String
503 Service Unavailable	String

Response examples:

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

3.2.5 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	String
503 Service Unavailable	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 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	String
503 Service Unavailable	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 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. 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 through a number of waypoints to a new pose.

Request content type: application/json

Request parameters:

Parameter	Description
speed	The parameter sets the speed (in % max. speed) at which servos should move to the required angles. The **admissible value range** is from 1 to 100. ***ATTENTION! SPECIFYING THE “speed” PARAMETER IS MANDATORY OTHERWISE AN ERROR IS GENERATED.*** **Type:** *number (double)* **Included as:** *query*
type	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*
tcp_max_velocity	The parameter defines the limit velocity in meters per second that an end effector can reach at its TCP while moving. It is not mandatory. When the user specifies no value for it, it is set to default. The default setting is 2 m/s. The **admissible value range** is from 0.001 to 2 m/s. **Included as:** query

Request example:

```
[
  {
    "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	Arm error	String
503 Service Unavailable	Arm emergency	String

Response content type: text/plain

Response examples:

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

3.2.8 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 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 TCP position to the required 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:

Parameter	Description
speed	The parameter sets the speed (in % max speed) at which the arm should move to the required position. The **admissible value range** is from 1 to 100. ***ATTENTION! SPECIFYING THE "speed" PARAMETER IS MANDATORY OTHERWISE AN ERROR IS GENERATED.*** **Type:** *number (double)* **Included as:** *query*
type	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*
tcp_max_velocity	The parameter defines the limit velocity in meters per second that an end effector can reach at its TCP while moving. It is not mandatory. When the user specifies no value for it, it is set to default. The default setting is 2 m/s. The **admissible value range** is from 0.001 to 2 m/s. **Included as:** *query*

Request example:

```
[
  {
    "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	Arm error	String
503 Service Unavailable	Arm 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"
  ```

3.2.9 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 Hardware Installation 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	String
503 Service Unavailable	String

Response examples:

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

3.2.10 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 document Hardware Installation 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	String
503 Service Unavailable	String

Response examples:

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

3.2.11 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	String
503 Service Unavailable	String

Response examples:

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

3.2.12 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,
    "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
    },
   ]
  }
]
```

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	String
503 Service Unavailable	String

Response examples:

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

```
"Unable parameter value {13}"
```

- **500 Internal Server Error**

```
"Robot does not respond"
```

- **503 Service Unavailable**

```
"Robot unavailable in emergency state"
```

3.2.13 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	String
500 Internal Server Error	String
503 Service Unavailable	String

Response examples:

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

```
"Robot does not respond"
```

- **503 Service Unavailable**

```
"Robot unavailable in emergency state"
```

3.2.14 Quitting the untwisting mode

Path:

PUT/untwisting/finish

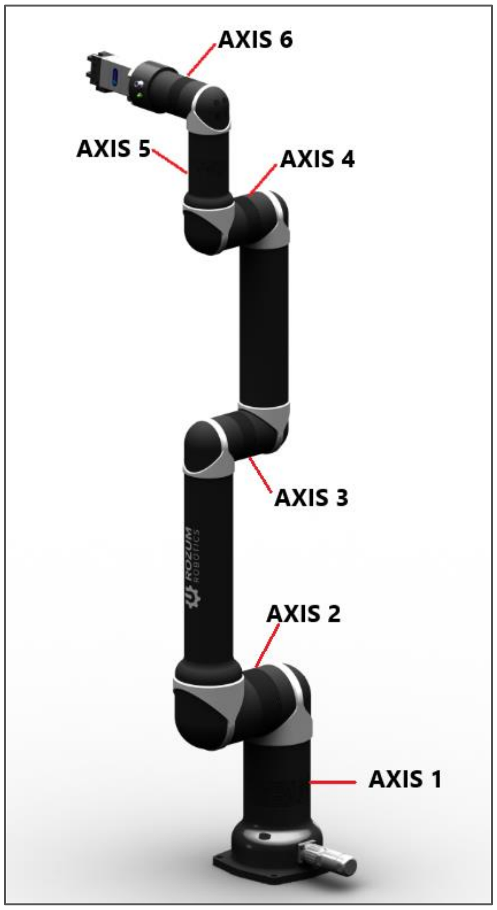
Description: The function enables users to verify the results of untwisting and quit the untwisting mode. In the untwisting mode, PUT and other API requests to move the arm are unavailable, until untwisting is completed. Users can only work with GET requests.

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	Arm error	String
503 Service Unavailable	Arm emergency	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"
```

3.2.15 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: 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	Arm error	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 Pose",`
  `"Collision detected"`

- **500 Internal Server Error**
  `"Robot does not respond"`

3.2.16 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: 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	Arm error	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 Pose",`
  `"Collision detected"`

- **500 Internal Server Error**
  `"Robot does not respond"`

3.2.17 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	Arm error	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	String
503 Service Unavailable	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	String
503 Service Unavailable	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

Schema (property)	Property content	Examples
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

Property	Property content	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

Schema (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"
		}

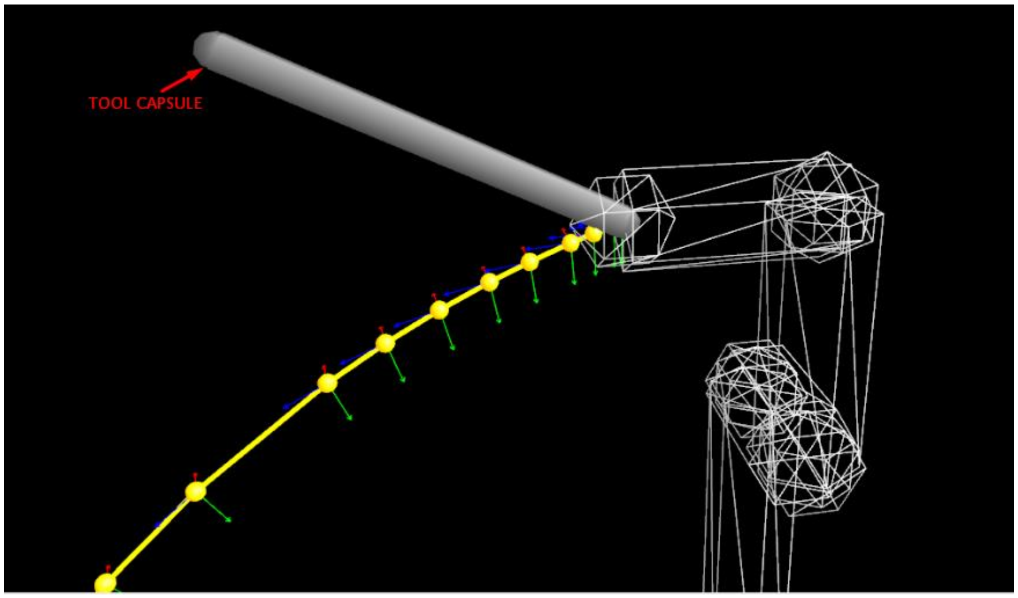
Rotation	roll: double number (radians)
	pitch: double number (radians)
	yaw: double number (radians)
		{
		"roll": "3.14",
		"pitch": "0",
		"yaw": "0.5"
		}

Tool shape schema

Schema	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.



Obstacle schema

Schema (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

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