robart®

Robot Interface Protocol

V1.0.0

Disclaimer

The API / the robot interface protocol is provided as-is without warranty of any kind. Robart makes no warranty or representation, either express or implied, regarding the API / the robot interface protocol, including but not limited to, any implied warranties of fitness for a particular purpose and Robart disclaims any warranty that use of the API / the robot interface protocol will be uninterrupted or error free.

Revision History

Revision	Date	Author(s)	Description
1.0.0	23.01.2023	Wenigwieser, F.	Initial version

Contents

1. Introduction

To develop a user interface that can interact with the robot, a protocol is needed. To keep it simple and platform neutral, we decided to use a simple HTTP implementation that only supports GET requests and outputs JSON formatted response data.
Initially, we will not support POST request. However, we reserve the option of implementing it at a later point in time if it becomes necessary.

2. General Rules

2.1. Finding the Robot

To provide a satisfying user experience, the robot will be automatically discoverable. For this purpose, we are using ZeroConf. Robots will announce themselves as service type `_aicu-http._tcp.local.` Service name and type are temporary, and need to be changed to an officially registered service type before shipping a product. Using ZeroConf, you can get ip address and port of the http interface. Using the get request `get/robot_id` you can then identify the robot instance at this address. For more information about ZeroConf, see e.g.
http://en.wikipedia.org/wiki/Zero-configuration_networking
http://en.wikipedia.org/wiki/SRV_record

2.1.1. Alternative to ZeroConf based discovery

Extensive testing, and results from the field showed, that mdns(ZeroConf) might not be the best discovery service regarding reliability. There are a couple of reasons, why mdns is not reliable enough:

- Consumer grade WiFi gear (especially access points) are known to be buggy.
- Multicasts which are used by mdns are treated differently in WiFi, than normal frames. They are not reliable.
- mdns requires **at least** 2 successful multicast transmissions, for one discovery process. One is the query, the second one is the response. This essentially doubles the chance of packet loss.
- Especially multicasts seem to be implemented rather badly on many access points.

To resolve the outlined problems, an alternative solution has been implemented. It runs in conjunction with the default mdns discovery, and an application can use both discovery methods in parallel, to get more reliable results.

Protocol specification

When the robot is connected to a network, it will send repeated UDP datagram based messages in regular intervals to all nodes in the network using broadcasts. The interval is currently set to 5 seconds, which might however be modified in a later iteration of the implementation.

The protocol is rather simple:
UDP port = 10009
IP4 destination address = 255.255.255.255 (broadcast)
IP6 destination address is not a fixed value, but derived from the group id 0x80526F62 using *unicast prefix based ipv6 multicast address allocation* described in https://tools.ietf.org/html/rfc3306.

Message format:
One announcement message is (almost) only ASCII encoded text. The protocol is line based, separated

by the *’\n’* token. Each line represents a key-value pair, where the two components are separated by a *’=’* token.
The first line contains the robots unique id in the form *"unique_id=AACTJ0-ePHkyuZ5rS4QD8Q\n"*.
The following lines give information about the robots assigned IPv4 and IPv6 addresses. IP addresses can be declared in the following two forms:

- *"IP4=192.186.178.23\n"* => IPv4 addresses are always in the default *dot-notation*. IPv4 address count can be either 0 or 1.
- *"IP6=2001:470:6D:408:AEA:40FF:FE66:8167\n"* => IPv6 addresses are declared using the notation described in https://tools.ietf.org/html/rfc5952. Ipv6 address count can be 0 to many (in the current implementation at maximum 3, but do not rely on it).

Later protocol iterations can add more key-value pairs after the IP addresses, which means, that a parser has to take that into account. The default policy for unknown keys is to log a warning, dismiss the pair, and then continue parsing.

The last key-value pair contained in the message is terminated by an extra *’\n’* token. After this token the message is finished with a 16 bytes digest / signature, which is used to verify the message & also protect eventual protocol parser implementations against other applications sending udp broadcasts to port 10009.

The signature can be validated, by initializing a MD5 hasher with the seed *"Robarti"*, and then feeding all the data of the message including the *’\n’* tokens to it, except the last 16 bytes of the message, which contain the signature. After the message was fed to the hasher, generate a hash, and then compare it to the last 16 bytes of the message. When the two byte sequences are equal, then the announcement message is correct, when not, then it has to be discarded. It is recommended to do this verification before parsing any other parts of the protocol.

Example message

Note, that non printable characters and bytes are ’\’ escaped:

```
unique_id=AACTJ0-ePHkyuZ5rS4QD8Q\n
IP4=192.168.178.23\n
IP6=2001:470:6D:408:AEA:40FF:FE66:8167\n
\n
\x16j\x1d9\xe5v\x82\x80\x0e.z\xed\xa2\x9e<H
```

2.2. Number Formats

As the robot does not have a floating point unit, it will send and receive only integers (this is called fixed point math or FXP). The following table shows how to convert between floating point and fixed point.

Format	FXP 2 Float	Float 2 FXP	min Value	max Value
1.13.2	$Float = \frac{FXP}{2^2}$	$FXP = (int)(2^2 * Float)$	-8192	8191,75
1.4.11	$Float = \frac{FXP}{2^{11}}$	$FXP = (int)(2^{11} * Float)$	-16	15,99951171875

3. Requests

All requests are encoded in a standard http GET request of the form
`http://<ip-of-robot>/<variable>[?<param-name>=<param-value>[&<param-name>=<param-value>]*]`

In the request, strings should be first UTF-8 encoded and then use URL encoding. This is important for localized strings. For example, a parameter value containing the string “Küche” should be encoded as `K%C3%BCche`.

The ordering of parameters is important. Requests will only be accepted if the parameters are sent in the exact order defined in this document. Otherwise the server will issue a parameter error.

The response will be JSON formatted text. All strings are UTF-8 encoded (with special and control characters properly escaped as needed by the JSON format). In case of a valid request, http will answer with 200 OK and a message that is specific to the request. In case of an invalid request, an error message will be sent, as described in chapter 4. Starting with Protocol Version 3.0.0, the client is required to gracefully handle fields which are not defined. This means that a client must discard them silently. The reason for this requirement is to be able to extend the protocol without breaking backwards compatibility. On the other hand, all requests must strictly follow the specification of the current implemented protocol version on the robot. Therefore, the client needs to read out the protocol version from the robot and adhere to the specified commands of this version. Any commands with unknown or missing fields will be ignored by the robot.

3.1. Unlocking Requests

PFS & Parameters	Return Values	Description
`set/unlock_http`		
pass=<password>	`{}`	Unlocks the local http interface of the robot. When its unlocked you can control the robot with it. The password label is inside the robot under the dustbin
`set/lock_http`		
	`{}`	Locks the http interface again, so you can not control the robot via the local http interface anymore

3.2. General Requests

PFS & Parameters	Return Values	Description
`get/protocol_version`		

PFS:2|2L|3	`{`
	`"version_major": <integer>,`
	`"version_minor": <integer>,`
	`"patch_level": <integer>,`
	`}`
	or
	`{`
	`"version_major": <integer>,`
	`"version_minor": <integer>,`
	`"patch_level": <integer>,`
	`"tls": <integer>`
	`}`
		Return the current protocol version, and whether the local https interface is available; This version number is identical to the version noted in the document revision history at the beginning of this document. The "tls" field can be 0 for not available, or 1, which stands for available.
		Note that the SDK can assume, that no https interface is available, when the "tls" field is not included in the response(for older FW).

<table>
<tr><th>PFS & Parameters</th><th>Return Values</th><th>Description</th></tr>
<tr><td colspan="3"><b>get/status</b></td></tr>
<tr><td>PFS:2|2L|3</td><td>

```
{
  "voltage":<voltage>,
  "mode"   :"<Mode>",
  "cleaning_parameter_set": <set_id>,
  "active_cleaning_parameter_set": "<set>",
  "active_pump_volume": "<pump_volume>",
  "battery_level": <level>,
  "charging": <charge_state>,
  "time": {
    "year": <YYYY>,
    "month": <MM>,
    "day": <DD>,
    "hour": <hh24>,
    "min": <mm>,
    "sec": <ss>,
    "day_of_week": <DOW>
  },
  "startup_time": {
    "year": <YYYY>,
    "month": <MM>,
    "day": <DD>,
    "hour": <hh24>,
    "min": <mm>,
    "sec": <ss>,
    "day_of_week": <DOW>
  },
  "recovery_info": {
    "status_code": <status_code>,
    "validation_pattern": <val_pattern>
  }
}
```

</td><td>Voltage format:[V], 1.5.10
Mode example cleaning
For a list of supported modes, see chapter 3.10.1.
Cleaning Parameter Sets are described in chapter 3.10.4.
Battery Level: Est. %, [0, 100]
Charging: <charge_state>: charging, connected, unconnected
Time: Shows the current time on the robot
Startup_Time: Shows the time when the robot was turned on or restarted
See chapter 3.10.2 for time format specification</td></tr>
</table>

PFS & Parameters	Return Values	Description
get/robot_flags		

PFS:2|2L|3	{
	"not_ready":[<robot_flag>],
	"notification":[<robot_flag>],
	"error":[<robot_flag>]
	}
		If the robot is currently in "not-ready" (see get/status), then there will be flags in the "not-ready" or "error" section. Flags in "error" represent conditions from which the robot cannot recover normally, and it must be power-cycled. Flags in "not-ready" usually require some user interaction to make the robot operational again. The flags in "notification" describe other conditions of the robot that might require some user interaction, like cleaning the dustbin or removing objects from brushes or wheels. However, even without interaction the robot will allow to start new tasks.
		For a list of supported flags see 3.10.13.

get/execution_state		

PFS:2|2L|3	{
	"top_level_state":<top_level_state>,
	"operational_state":<operational_state>,
	"sub_states":[
	{
	"state":<sub_state>,
	"map_id":<map_id>,
	"area_ids":[<area_id>],
	"strategy":<string>
	}
	]
	}
		Shows the current execution state of the robot.
		For a list of supported states, see chapters 3.10.10, 3.10.11, and 3.10.12.
		Strategies are described in chapter 3.10.17.

PFS & Parameters	Return Values	Description
`get/power_status`		

PFS:2|2L|3	`{`
	`  "power_status": <string>`
	`}`
		Current power status of robot.
		Possible *power_status* values:
		• *"initializing"* Power management not initialized yet.
		• *"sleeping"* Robot is in sleep mode.
		• *"active"* Robot is not in sleep mode.

`get/command_result`		

PFS:2|2L|3	`{`
	`  "commands": [`
	`    {`
	`      "cmd_id" : "<command_id>",`
	`      "status" : "<status>",`
	`      "error_code" : "<int>"`
	`    }`
	`  ]`
	`}`
		Provides information about the outcome of the last user commands
		See chapter 3.10.5

`set/switch_cleaning_parameter_set`		
PFS:2|2L|3 [opt]cleaning_parameter_set=<set_id> [opt]pump_volume=<pump_volume>	`{}`	Switching immediately to the new parameter set. For <set_id> and <pump_volume> see chapters 3.10.4 and 3.10.19.
`get/cleaning_parameter_set`		

PFS:2|2L|3	`{`
	`  "cleaning_parameter_set": <set_id>,`
	`  "user_cleaning_parameter_set": "<cleaning_parameter_set>",`
	`  "user_pump_volume": "<pump_volume>"`
	`}`
		Returns the cleaning parameter set (see 3.10.4) currently set by the user

PFS & Parameters	Return Values	Description
`set/cleaning_parameter_default_settings`		
PFS:2|2L|3 [opt]cleaning_parameter_set=<cleaning_parameter_set> [opt]pump_volume=<pump_volume>	`{}`	Switching immediately to the new parameter set. For <cleaning_parameter_set> and <pump_volume> see chapters 3.10.4 and 3.10.19. Note that this is equivalent to setting the live parameters default_scm and default_pcm.
`get/cleaning_parameter_default_settings`		

PFS:2|2L|3	`{`
	`  "cleaning_parameter_set": "<set>",`
	`  "pump_volume": "<pump_volume>"`
	`}`
		Returns the default cleaning parameter set (see 3.10.4) and pump volume (see 3.10.19) currently set by the user

`get/robot_id`		

PFS:FACT|2|2L|3	`{`
	`  "name": "<string>",`
	`  "unique_id": "<string>",`
	`  "camlas_unique_id": "<string>",`
	`  "model": "<string>",`
	`  "firmware": "<string>",`
	`  "commit_id": "<string>",`
	`  "os_version": "<string>",`
	`  "devices": [`
	`    {"name": "<name>",`
	`    "model": "<model>",`
	`    "id": "<id>",`
	`    "firmware": "<firmware>"},`
	`    ...`
	`  ]`
	`}`
		Returns info about the robot including Name, Model, UniqueID of Robot and Sensors and the actual running firmware. The field *commit_id* identifies the exact version of the source code, from which the firmware was built. The field *os_version* identifies the exact yocto_apollo version, which was used to build the firmware image.
		Also includes an array of all the external / internal devices on the robot with information about the hardware model, unique identifier, and running firmware of the device. All information attached to a device is entirely optional and can always be an empty string.

PFS & Parameters	Return Values	Description
get/statistics		

PFS:2|2L|3	{
	  "total_distance_driven": <distance>,
	  "total_cleaning_time": <time>,
	  "total_area_cleaned": <area>,
	  "total_number_of_cleaning_runs": <int>
	}
		Distance format: 0.25.7 [m]
		Time format: 0.26.6 [h]
		Area format: 0.26.6 [m2]
		These statistics are reset with set/do_statistics_reset

get/operation_mode		
PFS:2|2L|3	{ "operation_mode" : <operation-mode>, "operation_state" : <operation-state>}	Operation-mode: dut or normal. Operation-state: booted (modules, drivers and services of Operating System are running), initialized (parameters of system have been loaded), running (firmware is up and running. If the operation mode is DUT, the board ready for testing) or exception (board is in error state).
get/permanent_statistics		

PFS:2|2L|3	{
	  "total_distance_driven": <distance>,
	  "total_cleaning_time": <time>,
	  "total_area_cleaned": <area>,
	  "total_number_of_cleaning_runs": <int>
	}
		Distance format: 0.25.7 [m]
		Time format: 0.26.6 [h]
		Area format: 0.26.6 [m2]
		These statistics are reset with set/do_factory_reset

get/lifetime_statistics		

PFS:2|2L|3	{
	  "total_distance_driven": <distance>,
	  "total_cleaning_time": <time>,
	  "total_area_cleaned": <area>,
	  "total_number_of_cleaning_runs": <int>
	}
		Distance format: 0.25.7 [m]
		Time format: 0.26.6 [h]
		Area format: 0.26.6 [m2]
		These statistics cannot be reset.

PFS & Parameters	Return Values	Description
get/file_system_status		

PFS:2|2L|3	{
	"data": {"mode": <string>}
	}
		Health information of the file system. Can be used to check whether the file system is still writable

get/pump_volume_settings		

PFS:3	{
	"mode": <pump_volume>,
	"priming": <int>,
	"cleaning": <int>
	}
		Get pump volume settings. See 3.10.19 for values of <pump_volume>. `priming` and `cleaning` represents a volume in multiples of 0.01ml. E.g. a value of 6000 represents 60ml.

set/stop		

PFS:2|2L|3	{
	"cmd_id":<command_id>
	}
		Stop the robot immediately

set/abort		

PFS:2|2L|3	{
	"cmd_id":<command_id>
	}
		Stop the robot immediately and disable the ability to continue the current task. If the robot was already idle, disable the ability to continue the previous task.

set/go_home		

PFS:2|2L|3	{
	"cmd_id":<command_id>
	}
		Go back to the position where the exploration started and search for the docking station

PFS & Parameters	Return Values	Description
set/explore		

PFS:3	{
	"cmd_id":<command_id>
	}
		Explores a map

set/clean_all		

PFS:2|2L|3
[opt] cleaning_parameter_set=<set_id>
[opt] cleaning_strategy_mode=<mode_id>
[opt] method=<method>
[opt] pump_volume=<pump_volume>
	{
	"cmd_id":<command_id>
	}
		Start cleaning mode, clean everything that is reachable
		<set_id> See chapter 3.10.4
		<mode_id> See chapter 3.10.17.
		<method>: See chapter 3.10.18.
		<pump_volume> See chapter 3.10.19.

set/clean_map		

PFS:3
map_id=<map-id>
[opt] area_ids=<array of ids, separated by comma>,
e.g., area_ids=45,4,123
[opt] cleaning_parameter_set=<set_id>
[opt] cleaning_strategy_mode=<mode_id>
[opt] method=<method>
[opt] pump_volume=<pump_volume>
	{
	"cmd_id":<command_id>
	}
		Cleans the permanent map specified by map_id according to the specified cleaning parameter set (see Chapter 3.10.4). If area_ids are provided, the corresponding sequence of areas is cleaned.
		<mode_id> See chapter 3.10.17.
		<method>: See chapter 3.10.18.
		<pump_volume> See chapter 3.10.19.

PFS & Parameters	Return Values	Description
set/continue		

PFS:3
[opt] cleaning_parameter_set=<set_id>
[opt] cleaning_strategy_mode=<mode_id>
[opt] method=<method>
[opt] pump_volume=<pump_volume>
	{
	"cmd_id":<command_id>
	}
		Continues clean all, clean map, or exploration from previous state. Ignored if no task can be continued.
		<set_id> See chapter 3.10.4.
		<mode_id> See chapter 3.10.17.
		<method>: See chapter 3.10.18.
		<pump_volume> See chapter 3.10.19.

set/clean_spot		

PFS:2|2L|3
map_id=<map-id>
[opt] x1=<coordinate>
[opt] y1=<coordinate>
[opt] cleaning_parameter_set=<set_id>
[opt] spot_type=<spot-type-id>
[opt] cleaning_strategy_mode=<mode_id>
[opt] method=<method>
[opt] pump_volume=<pump_volume>
	{
	"cmd_id":<command_id>
	}
		Moves to the given spot (if possible), and starts the spot cleaning program
		Coordinate format: [cm], 1.13.2
		Go to the given coordinates, and start spot cleaning mode x1, y1 and spot_type are optional, and need not be provided. If x1 or y1 are missing, the robot will clean at its current position.
		The map_id indicates to which map the location refers to.
		spot-type-id represents one of several, pre-defined spot-size definitions.
		<set_id> See chapter 3.10.4.
		<mode_id> See chapter 3.10.17.
		<method>: See chapter 3.10.18.
		<pump_volume> See chapter 3.10.19

PFS & Parameters	Return Values	Description
`set/clean_start_or_continue`		

PFS:2|2L|3
[opt] cleaning_parameter_set=<set_id>
[opt] cleaning_strategy_mode=<mode_id>
[opt] method=<method>
[opt] pump_volume=<pump_volume>
	{
	"cmd_id":<command_id>
	}
		Will continue clean all, clean map, or exploration from previous state if corresponding task has been interrupted (like set/continue). Will start a new clean all otherwise (like set/clean_all).
		<set_id> See chapter 3.10.4.
		<mode_id> See chapter 3.10.17.
		<method>: See chapter 3.10.18.
		<pump_volume> See chapter 3.10.19

`set/goto_sleep`		

PFS:FACT|2|2L|3
[opt]sleep_mode=<string>
	{
	"cmd_id":<command_id>
	}
		The robot goes to a sleep mode ("soft_sleep", "deep_sleep"). Currently only "soft_sleep" is supported.

`set/do_factory_reset`		
PFS:FACT|2|2L|3	{}	Deletes all Userdata from the robot
`set/do_statistics_reset`		
PFS:2|2L|3	{}	Resets robot statistics (cf. get/statistics)
`set/priming_test`		

PFS:3	{
	"cmd_id":<command_id>
	}
		Executes wet pad priming.

PFS & Parameters	Return Values	Description
`set/pump_volume_settings`		

`PFS:3`
mode=<pump_volume>
[opt] priming=<int>
[opt] cleaning=<int>
	`{}`	Sets pump volume settings to low, medium or high. See 3.10.19 for values of <pump_volume>. Parameters `priming` and `cleaning` are only allowed if mode=`direct`, and set a specific water volume in 0.01ml. E.g. a value of 6000 represents 60ml.

3.3. Config Requests

PFS & Parameters	Return Values	Description
get/wifi_status		

PFS:FACT|2|2L|3	{
	  "status": <string>,
	  "ssid": <string>,
	  "raw_ssid": <string>,
	  "rssi": <integer>,
	  "mac_address": <string>,
	  "ip_address": <string>,
	  "type": <string>
	}
		Returns the current status of the WIFI interface.
		All non-ASCII, non-printable SSID characters are replaced with '?' in the `ssid` field.
		raw_ssid will show the SSID in base64 encoding, just like in `get/wifi_scan_results`
		MAC_address format: xx:xx:xx:xx:xx:xx
		where xx is a hexadecimal number
		IP_address format: xxx.xxx.xxx.xxx
		where xxx is a decimal number between 0 and 255
		IMPORTANT: ip_address is deprecated in favour of `get/network_status`
		type can be `wifi` if connected to a wifi, `uAP` if in access point mode, `wired` if connected via LAN, `undefined`

PFS & Parameters	Return Values	Description
`get/network_status`		

PFS:FACT|2|2L|3	`{`
	`    "mac_address": <string>,`
	`    "addresses": [`
	`        {`
	`            "type": <string>,`
	`            "ip_addr": <string>`
	`        },*`
	`    ]`
	`}`
		Returns the current network status of the robot. It provides information about the IP addresses, which are assigned to the robot‘s network interface in form of an array.
		Fields:
		`mac_address`: xx:xx:xx:xx:xx:xx where xx is a hexadecimal number
		`addresses[i]->type`: can be "v4", "v6-ll", or "v6-slaac", which determines the type of the assigned ip address.
		`addresses[i]->ip_addr`: Is either a string formatted IPv4 address in form of xxx.xxx.xxx.xxx where xxx is a decimal number between 0 and 255, or a string formatted IPv6 address, which is described in rfc5952.

PFS & Parameters	Return Values	Description
set/connect_wifi		

PFS:FACT|2|2L|3
ssid=<ssid>
passphrase=<passphrase>
	{ "cmd_id":<command_id> }	Connects immediately to the defined network. If this does not work it will fallback to AP-mode
		This command can fail with several error_codes via "get/command_result"
		0 - no error
		deprecated errors(blackfin)
		1 - deauthenticated
		2 - dissociated
		3 - not in range
		4 - wlan chip not responding
		5 - ssid len invalid
		6 - cipher not supported
		7 - psk len invalid
		8 - dhcp start error
		9 - dhcp timeout error
		10 - assoc error
		new error codes since posix/apollo
		1000 - DISCONNECTED + wpa supplicant reason_code 1001→ 1 = WLAN_REASON_UNSPECIFIED see Reason codes (IEEE Std 802.11-2016, 9.4.1.7, Table 9-45)
		2001 - NETWORK_NOT_FOUND
		2002 - COMMAND_ERROR
		2003 - LOCAL_DISCONNECT

PFS & Parameters	Return Values	Description
`set/create_uap`		

`PFS:FACT|2|2L|3`	{
	"cmd_id":<command_id>
	}
		creates a hotspot without the need of a factory reset or connection loss.

`set/start_scan`		

`PFS:FACT|2|2L|3`	{
	"cmd_id":<command_id>
	}
		Starts a scan of WIFI networks.

`get/wifi_scan_results`		

`PFS:FACT|2|2L|3`	{
	  "cmd_id": <int>,
	  "scanning": <boolean>,
	  "scan": [
	    {
	      "ssid" : <SSID>,
	      "raw_ssid": <RAWSSID-base64-encoded>,
	      "bssid": <mac-address>,
	      "channel": <chan_nr>,
	      "protocol": <string>,
	      "pairwiseciper": <string>,
	      "groupcipher": <string>,
	      "rssi": <int>
	    }
	  ]
	}
		All non-ASCII, non-printable SSID characters are replaced with '?' in the `ssid` field. The `raw_ssid` field will always contain the actual SSID in base64 encoded format, which means it can contain any character (even non ASCII unicode chars).

PFS & Parameters	Return Values	Description
get/stored_wifi_networks		

PFS:FACT|2|2L|3	`{`
	`  "saved_wifis": [`
	`    {`
	`      "ssid" : <SSID>,`
	`      "raw_ssid": <RAWSSID-base64-encoded>,`
	`      "bssid": <mac-address>,`
	`      "channel": <chan_nr>,`
	`      "protocol": <string>,`
	`      "pairwiseciper": <string>,`
	`      "groupcipher": <string>,`
	`      "rssi": <int>`
	`    }`
	`  ]`
	`}`
		All non-ASCII, non-printable SSID characters are replaced with '?' in the `ssid` field. The `raw_ssid` field will always contain the actual SSID in base64 encoded format, which means it can contain any character (even non ASCII unicode chars).

set/uap_ssid		

PFS:FACT|2|2L|3
ssid=<ssid>
	`{`
	`  "cmd_id":<command_id>`
	`}`
		overrides uap ssid;
		• This will only be effective for the next call to either set/pairing_on or set/create_uap.
		• It will not create an access point.
		• It will not change the SSID of a currently open access point.

PFS & Parameters	Return Values	Description
`set/wifi_region`		

`PFS:FACT|2|2L|3`
reg_domain=<ISO/IEC 3166-1 alpha2>
	`{}`	sets wireless regulatory domain;
		• The *<ISO/IEC 3166-1 alpha2>* are two character alphabetic country codes, defined in *ISO 3166*.
		• This should be called, before attempting to connect to any wifi network, and before scanning for wifi networks.
		• During the runtime of the robot (one reboot cycle), this call should only be called once. Preferably as early as possible after boot.

`set/wifi_power_save`		

`PFS:FACT|2|2L|3`
power_save=<int>
	`{}`	enable or disable wifi power save mode (enabled may lead to worse ping).

`get/wifi_power_save`		

`PFS:FACT|2|2L|3`	`{`
	`  "power_save": <integer>`
	`}`
		get wifi power save mode.

PFS & Parameters	Return Values	Description
`control/iot_status`		

`PFS:FACT|2|2L|3`	`{`
	` "iot_state":<iot_state>,`
	` "iot_server":<iot_server_name>,`
	` "confirmation_state":<confirmation_state>`
	`}`
		Returns the status of the IoT connection including the confirmation status of push button <iot_state> can be
		`unknown` (during starting process),
		`disabled` (when IoT is not enabled),
		`connected` or
		`disconnected`.
		<iot_server_name> is the iot-server name including the port number, to which the robot is connected (in *"<host>:<port>"* format).
		<confirmation_state> can be
		`none` (regular state),
		`waiting` (if waiting for confirmation via `control/confirmed_button`),
		`confirmed` (if `control/confirmed_button` was called)

`get/robot_name`		

`PFS:FACT|2|2L|3`	`{`
	`  "name": "<string>"`
	`}`
		returns the name of the Robot

`set/robot_name`		

`PFS:FACT|2|2L|3`
name=<String>
	`{}`	Set the user-defined name of the robot. Can be retrieved via `get/robot_name`

3.4. Map Requests

A map is always referenced by a map_id (see Chapter 3.10.7).

PFS & Parameters	Return Values	Description
get/feature_map		

PFS:2|3
[opt]map_id=<int>
	{"map":
	  {
	    "map_id": <map-id>,
	    "lines" : [
	      {
	        "x1":<coordinate>,
	        "y1":<coordinate>,
	        "x2":<coordinate>,
	        "y2":<coordinate>
	      }
	    ],
	    "docking_pose": {
	      "x" : <coordinate>,
	      "y" : <coordinate>,
	      "heading" : <angle>,
	      "valid" : <true/false>
	    },
	    "timestamp":<timestamp>
	  }
	}
		If map_id is not provided, data for the active map_id are shown.
		Coordinate format: [cm], 1.13.2
		A map consists of a set of lines, each going from x1/y1 to x2/y2.

PFS & Parameters	Return Values	Description
get/tile_map		

PFS:3
[opt]map_id=<int>
	{"map":
	    {
	    "map_id : <map–id>,
	    "areas":
	     [
	       {
	        "area_id" : <area−id>
	       }
	     ],
	    "lines" : [
	       {
	       "x1":<coordinate>,
	       "y1":<coordinate>,
	       "x2":<coordinate>,
	       "y2":<coordinate>
	       }
	      ],
	     "docking_pose": {
	      "x" : <coordinate>,
	      "y" : <coordinate>,
	      "heading" : <angle>
	      "valid" : <0/1>
	     },
	    "suggested_orientation": {
	       "x": <coordinate>,
	       "y": <coordinate>,
	       "angle": <angle>
	    },
	    "outline": [
	    {
	        "x": <coordinate>,
	        "y": <coordinate>
	    }
	    ],
	    "timestamp": <timestamp>
	  }
	}
		If map_id is not provided, data for the active map_id are shown. Coordinate format: [cm], 1.13.2 A map consists of a set of a set of area ids that belong to the simplified map. These areas can be retrieved by the get_area request. Additionally, the map consists of a set of simplified lines that indicated mayor obstacles (segments). All areas and segments are represented in the same (global) coordinate system. Finally, a transformation is suggested, which indicates a possible map transformation for simplified displying.

PFS & Parameters	Return Values	Description

get/topo_map

PFS:3

[opt]map_id=<int>

```
{
    "map_id" : <map-id>,
    "nodes":
     [
       {
         "area_id" : <area-id>
       }
     ],
    "edges":
     [
       {
         "from" : <area-id>,
         "to"   : <area-id>,
         "line" : {
            "x1":<coordinate>,
            "y1":<coordinate>,
            "x2":<coordinate>,
            "y2":<coordinate>
         }
       }
     ]
}
```

If map_id is not provided, data for the active map_id are shown.
Coordinate format: [cm], 1.13.2
A topological map is a graph where the nodes are the map areas, and each edge represents a gap or door between the two areas it connects. The gap is a line going from x1/y1 to x2/y2.

get/n_n_polygons

PFS:2|3

[opt]map_id=<int>

```
{"map":
  {
    "map_id": <map-id>,
    "polygons": [
    { "segments" : [
       {
       "x1":<coordinate>,
       "y1":<coordinate>,
       "x2":<coordinate>,
       "y2":<coordinate>
       }
      ]
    }
    ],
    "timestamp": <timestamp>
  }

}
```

If map_id is not provided, data for the active map_id are shown.
Coordinate format: [cm], 1.13.2
A map consists of a set of polygons. Each polygon consists of a set lines, each going from x1/y1 to x2/y2.

PFS & Parameters	Return Values	Description
`get/seen_polygon`		

PFS:2|3
[opt]map_id=<int>
	`{`
	`    "seen_polygon": {`
	`        "map_id": <int>,`
	`        "polygons": [`
	`            {"segments":[`
	`                {"x1": <int>,`
	`                 "y1": <int>,`
	`                 "x2": <int>,`
	`                 "y2": <int>`
	`                }`
	`            ]}`
	`        ]`
	`    }`
	`}`
		If map_id is not provided, data for the active map_id are shown.
		Coordinate format: [cm], 1.13.2
		A seen_polygon consists of a set of polygons (some may be holes). Each polygon consists of a set segments, each going from x1/y1 to x2/y2.

`get/rob_pose`		

PFS:2|3
[opt]map_id=<int>
	`{`
	`   "map_id": <map-id>,`
	`   "target_map_id": <map-id>,`
	`   "x1": <coordinate>,`
	`   "y1": <coordinate>,`
	`   "heading": <angle>,`
	`   "valid": true,`
	`   "is_tentative": <true/false>,`
	`   "timestamp": <timestamp>`
	`}`
		Coordinate format: [cm], 1.13.2
		Angle format: [rad], 1.4.11
		0 rad means along the positive x-axis; angle > 0 is a rotation counter-clockwise, < 0 a rotation clockwise.
		The map_id result shows the current map (map that robot operates on). If the map_id parameter is provided and differs from the current map, target_map_id is set and the position represents the estimated position of the robot `if` it was on map target_map_id.
		The is_tentative flag indicates whether the given rob pose is tentative (i.e., a preliminary estimate of the robot pose), or if it is the actual (confirmed) robot pose on the map.

PFS & Parameters	Return Values	Description
`get/cleaning_grid_map`		

`PFS:2|3`
[opt] map_id=<int>
[opt] layer=0|1
[opt] minheat=1|2
	`{`
	`    "map_id": <map-id>,`
	`    "lower_left_x": <coordinate>,`
	`    "lower_left_y": <coordinate>,`
	`    "size_x": <int>,`
	`    "size_y": <int>,`
	`    "resolution": <resolution>,`
	`    "cleaned": [`
	`      <rle binary bitmap>`
	`    ],`
	`    "timestamp": <timestamp>`
	`}`
		Coordinate format: [cm], 1.13.2
		Resolution format: [cm], 1.13.2
		See separate point: Chapter 3.10
		`map_id` is currently ignored.
		`get/cleaning_grid_map` will currently always return the grid map of the currently used map.
		`layer` and `minheat` are only relevant for deep cleaning. Only one of them can be given. If neither is provided, `minheat` 1 is assumed.
		`layer` 0 returns the data for the first cleaning pass. `layer` 1 returns the data of the second (perpendicular) cleaning pass.
		`minheat` 1 returns a grid map where all cells are marked that have been cleaned in either cleaning pass. `minheat` 2 returns a grid map where all cells are marked that have been cleaned in both cleaning passes.

<table>
<tr><th>PFS &amp; Parameters</th><th>Return Values</th><th>Description</th></tr>
<tr><td colspan="3"><b><code>get/maps</code></b></td></tr>
<tr><td><code>PFS:3</code></td><td><pre>
{
  "maps":[
  {
    "map_id":map_id,
    "map_meta_data":<string>,
    "permanent_flag": <bool>,
    "statistics":{
      "area_size":<area_size>,
      "cleaning_counter" : <int>,
      "estimated_cleaning_time" : <dur>,
      "average_cleaning_time" : <dur>,
      "last_cleaned":{
        "year": <YYYY>,
        "month": <MM>,
        "day": <DD>,
        "hour": <hh24>,
        "min": <mm>,
        "sec": <ss>
      }
    }
  }
  ]
}
</pre></td><td>Area size format: [cm^2] 1.26.
Cleaning time fmt: [s] 0.22.10
The permanent flag indicates the permanent availability of the map.</td></tr>
<tr><td colspan="3"><b><code>get/map_status</code></b></td></tr>
<tr><td><code>PFS:3</code></td><td><pre>
{
    "operation_map_id": <map_id>,
    "active_map_id": <map_id>
}
</pre></td><td>Returns the current map status of the robot. Operation map id denotes the current map the robot is operating on (e.g., set by the user with a clean map command). The active map id denotes the current map used by the robot. The robot is localized in the operation map if and only if operation map id equals the active map id.</td></tr>
<tr><td colspan="3"><b><code>get/main_map_id</code></b></td></tr>
<tr><td><code>PFS:3</code></td><td><pre>
{
    "main_map_id" : <map-id>
}
</pre></td><td>If no main map was set, this will return with a main_map_id of 0.</td></tr>
</table>

PFS & Parameters	Return Values	Description
set/used_map		
PFS:3 map_id=<map_id>	{}	If map_id is non-zero, it will be used for all requests (like `/get/tile_map`) that take an optional map id.
set/save_map		

PFS:3
[opt]map_id=<map_id>
	{
	"cmd_id":<command_id>
	}
		Save the given map. (If map_id is omitted, it will save the "current" map.)

set/modify_map		

PFS:3
map_id=<map_id>
[opt] map_meta_data=<String>
[opt] docking_pose_x=<int>
[opt] docking_pose_y=<int>
[opt] docking_pose_heading=<i>
[opt] docking_station_available =<bool>
	{
	"cmd_id":<command_id>
	}
		Modifies the given map with map metadata and/or docking pose.
		docking_pose_x: [cm]1.13.2
		docking_pose_y: [cm]1.13.2
		docking_pose_heading: [rad]1.4.11

set/delete_map		

PFS:3
map_id=<map_id>
	{
	"cmd_id":<command_id>
	}
		Delete an existing map.

set/revert_map		

PFS:3
map_id=<map_id>
	{
	"cmd_id":<command_id>
	}
		Reject map or map changes after a completed exploration or map extension.

PFS & Parameters	Return Values	Description
`set/split_map`		
PFS:3 map_id=<map_id>	`{}`	Split the map with the given id.
`set/main_map_id`		
PFS:3 main_map_id=<map_id>	`{}`	Sets the main map for the robot. Clear the main map by setting main_map_id to 0.

3.5. Area Requests

PFS & Parameters	Return Values	Description
`get/areas`		

PFS:3
[opt]map_id=<int>
	{
	  "map_id": <map_id>,
	  "areas": [
	    {
	      "id": <int>,
	      "points": [
	        {
	          "x": <coordinate>,
	          "y": <coordinate>
	        }
	      ],
	      "area_type": <area_type>,
	      "area_state": <area_state>,
	      "area_meta_data": <string>,
	      "cleaning_parameter_set": <set_id>,
	      "floor_type": <floor_type>,
	      "room_type": <room_type>,
	      "strategy_mode": <strategy_mode>,
	      "method": <method>,
	      "pump_volume": <pump_volume>,
	      "statistics":{
	        "area_size":<area_size>,
	        "cleaning_counter": <int>,
	        "estimated_cleaning_time": <dur>,
	        "average_cleaning_time": <dur>,
	        "last_cleaned":{
	          "year": <YYYY>,
	          "month": <MM>,
	          "day": <DD>,
	          "hour": <hh24>,
	          "min": <mm>,
	          "sec": <ss>
	        }
	      }
	    }
	  ]
	}
		If map_id is not provided, data for the active map_id are shown.
		Coordinate format: [cm], 1.13.2.
		Area size format: [cm^2], 1.26.5
		Cleaning time format: [s], 0.22.10
		ID is an integer which identifies an area (can be used for deleting / modifying an area).
		Points are interconnected by lines, the last point is connected to the first one. A valid area must contain at least 3 points.
		The field area_type indicates the type of area (`room`, `to_be_cleaned`). The field area_state indicates the behavior state (`clean`, `blocking`, `inactive`, `proposed_blocking`, `declined_blocking`) of this particular area. The fields floor_type (e.g., `hardwood`, `carpet`) and room_type (`none`, `kitchen`, `sleeping_room`, etc.) describe the area in more detail (see Chapter 3.10.8). In case the user does not explicitly define the cleaning strategy, the field strategy_mode indicates the strategy mode for cleaning of the area (`normal`, `deep`, `walls_and_corners`).
		The field area_meta_data is any user defined string to name the Area, does not have to be unique.

PFS & Parameters	Return Values	Description
`set/add_area`		

PFS:3
map_id=<map_id>
[opt]area_meta_data=<string>
[opt]area_type=<area_type>
[opt]cleaning_parameter_set=<set-id>
[opt]area_state=<area_state>
[opt]floor_type=<floor_type>
[opt]room_type=<room_type>
[opt]strategy_mode=<strategy_mode>
[opt]method=<method>
[opt]pump_volume=<pump_volume>
x1=<coordinate>
y1=<coordinate>
...
xn=<coordinate>
yn=<coordinate>
	`{`
	`"cmd_id":<command_id>`
	`}`
		An area will be added to the current map. In the same request the points that define the area will be included, as well as certain area attributes (see Chapter 3.10.8). Also a user-defined name can be given.

PFS & Parameters	Return Values	Description
set/modify_area		

PFS:3
map_id=<map_id>
area_id=<area-id>
[opt]area_meta_data=<string>
[opt]area_type=<area_type>
[opt]cleaning_parameter_set=<set-id>
[opt]area_state=<area_state>
[opt]floor_type=<floor_type>
[opt]room_type=<room_type>
[opt]strategy_mode=<strategy_mode>
[opt]method=<method>
[opt]pump_volume=<pump_volume>
[opt]x1=<coordinate>
[opt]y1=<coordinate>
. . .
[opt]xn=<coordinate>
[opt]yn=<coordinate>
	{
	"cmd_id":<command_id>
	}
		An area will be modified to the specified given data. In the same request certain area attributes (see Chapter 3.10.8) can be modified.

set/merge_areas		

PFS:3
map_id=<map-id>
area_id1=<area-id>
area_id2=<area-id>
	{
	"cmd_id":<command_id>
	}
		Two areas will be merged into one. Properties of area_id1 will be used for the result.

PFS & Parameters	Return Values	Description
`set/split_area`		

PFS:3
map_id=<map_id>
area_id=<area-id>
x1=<coordinate>
y1=<coordinate>
. . .
xn=<coordinate>
yn=<coordinate>
	{
	"cmd_id":<command_id>
	}
		Splits an area in two or more areas along the poly-line defined by the points x1, y1, · · · , xn, yn.

`set/delete_area`		

PFS:3
map_id=<map-id>
area_id=<area-id>
	{
	"cmd_id":<command_id>
	}
		Delete the area identified by map and area id

`set/propose_nogo_areas`		

PFS:3
map_id=<map-id>
	{
	"cmd_id":<command_id>
	}
		Add nogo areas based on points-of-interest on the map specified by map_id. The added nogo areas will be added as areas with area_state proposed_blocking (see Chapter 3.10.8) and will not influence robot behavior until confirmed with `set/confirm_nogo_areas`. The proposed nogo areas are part of the output of `get/areas`.

PFS & Parameters	Return Values	Description
`set/confirm_nogo_areas`		

PFS:3
map_id=<map-id>
[opt] confirmed=<comma-separated area ids>
[opt] declined=<comma-separated area ids>
Example:
confirmed=4,7,9
declined=2,11,8
	{
	"cmd_id":<command_id>
	}
		Confirm or decline proposed nogo areas generated by `set/propose_nogo_areas`. The specified areas must have the area state proposed_blocking. All confirmed nogo areas will subsequentially be converted into standard blocking areas, while all declined nogo areas will have the area_state declined_blocking. Declined nogo areas have no influence on robot behavior, they are only relevant for the automatic nogo area creation with `set/propose_nogo_areas`. See Chapter 3.10.8 for a summary of all area attributes.

3.6. Points of Interest Requests

PFS & Parameters	Return Values	Description
get/points_of_interest		

PFS:3
[opt]map_id=<int>
[opt]points_ids=<array of ids, separated by comma> (e.g., points_ids=34,3,123
	`{`
	`"map_id": <map_id>,`
	`"points_of_interest": [`
	`{`
	`"id": <int>,`
	`"pose": {`
	`"x": <coordinate>,`
	`"y": <coordinate>,`
	`"heading": <angle>`
	`},`
	`"type": <point_of_interest_type>,`
	`"meta_data": <string>,`
	`"timestamp": {`
	`"year": <YYYY>,`
	`"month": <MM>,`
	`"day": <DD>,`
	`"hour": <hh24>,`
	`"min": <mm>,`
	`"sec": <ss>`
	`}`
	`}`
	`]`
	`}`
		If map_id is not provided, data for the active map_id are shown. If points_ids are not provided all points of interest for given map are shown.
		Coordinate format: [cm], 1.13.2
		Angle format: [rad], 1.4.11
		"id" is an integer which identifies a point of interest (can be used for deleting / modifying a point of interest).
		The field "type" indicates the type of point of interest (see Chapter 3.10.9).
		The field "meta_data" is any user defined string, does not have to be unique.

set/delete_points_of_interest		

PFS:3
map_id=<int> points_ids=<array of ids, separated by comma> (e.g., points_ids=34,3,123
	`{`
	`"cmd_id":<command_id>`
	`}`
		Delete the points of interest identified by map_id and its points_ids

3.7. Schedule Requests

<table>
<tr><th>PFS &amp; Parameters</th><th>Return Values</th><th>Description</th></tr>
<tr><td><b>get/schedule</b></td><td></td><td></td></tr>
<tr><td>PFS:2|2L|3</td><td><pre>
{
  "schedule": [
    {
      "task_id" : <task_id>
      "time": {
        "days_of_week": <List_Of_DOW>,
        "year": <YYYY>,
        "month": <MM>,
        "day": <DD>,
        "hour": <hh24>,
        "min": <mm>,
        "sec": <ss>
      },
      "repeated": <int>,
      "enabled": <int>,
      "task":{
        "map_id": <map_id>,
        "cleaning_parameter_set": <set_id>,
        "strategy_mode": <strategy_mode>,
        "method": <method>,
        "pump_volume": <pump_volume>,
        "cleaning_mode":<mode_id>,
        "parameter1":<string>,
        "parameter2":<string>,
        "parameters":[<string>],
      }
    }
  ]
}
</pre></td><td>The array <code>schedule</code> will contain 0 or more elements (0, if there is no scheduled task). <code>time</code> and <code>days_of_week</code> formats are explained in chapter 3.10.2; <code>repeated</code> defines the time in days before this task will be repeated (e.g. 0 for a one time task, 1 for a daily task and 7 for a weekly task). <code>enabled</code> is either 1 or 0. <code>task</code> defines what to do in the cleaning mode (see chapter 3.10.3). In case of clean all, parameter1 and 2 can be empty. <code>parameters</code> contains all parameters in a list.</td></tr>
</table>

PFS & Parameters	Return Values	Description
`set/add_scheduled_task`		

`PFS:2|2L|3`
cleaning_mode=<mode-id>
cleaning_parameter_set=<set-id>
[opt]strategy_mode=<strategy_mode>
[opt]method=<method>
[opt]pump_volume=<pump_volume>
[opt]days_of_week=<List-Of-DOW>
[opt]year=<YYYY>
[opt]month=<MM>
[opt]day=<DD>
hour=<hh24>
min=<mm>
[opt]repeated=<dd>
map_id=<map-id>
param1=<string>
param2=<string>
[opt]param3..16=<string>
[opt]enabled=<int>
	{
	"cmd_id":<command_id>
	}
		mode-id: See chapter 3.10.3 for available modes
		See chapter 3.10.2 for a description of the time format
		for the definitions of parameters see `get/schedule`
		EITHER days_of_week, OR year AND month AND day AND repeated must be provided.
		`Enabled` must be either 1 or 0. Its default value is 1.

PFS & Parameters	Return Values	Description
set/modify_scheduled_task		

PFS:2|2L|3
task_id=<task_id>
[opt]cleaning_mode=<mode-id>
[opt]cleaning_parameter_set=<set-id>
[opt]strategy_mode=<strategy_mode>
[opt]method=<method>
[opt]pump_volume=<pump_volume>
[opt]days_of_week=<List-Of-DOW>
[opt]year=<YYYY>
[opt]month=<MM>
[opt]day=<DD>
[opt]hour=<hh24>
[opt]min=<mm>
[opt]repeated=<dd>
[opt]map_id=<map-id>
[opt]param1=<string>
[opt]param2=<string>
[opt]param3..16=<string>
[opt]enabled=<int>
	{
	"cmd_id":<command_id>
	}
		Allows changing some or all of the fields of a cleaning task. Some of the fields can only be set as a group or not at all.
		Groups:
		• cleaning_mode, map_id, param1..16
		• cleaning_parameter_set
		• days_of_week OR year+month+day+repeated
		• hour, min
		• enabled
		mode-id: See chapter 3.10.3 for available modes
		See chapter 3.10.2 for a description of the time format.

set/delete_scheduled_task		

PFS:2|2L|3
task_id=<task_id>
	{
	"cmd_id":<command_id>
	}
		Will delete the selected task

PFS & Parameters	Return Values	Description
set/clear_schedule		

PFS:2|2L|3	{
	"cmd_id":<command_id>
	}
		Clears the whole schedule
		The request will not return an error if there is no current schedule

3.8. Logging Requests

PFS & Parameters	Return Values	Description
get/ui_cmd_log		

PFS:2|2L|3	`[{`
	`    "id": <int>,`
	`    "cmd": <string>,`
	`    "rtc": <rtc>,`
	`    "params": <string>,`
	`    "source": <int>`
	`}]`
		Get the list of the last 50 UI commands.

get/notifications		

PFS:2|2L|3
[opt] last_id=<integer>
	`{`
	`  "robot_notifications":[`
	`    {`
	`      "id":<int>,`
	`      "type":<notification_type>,`
	`      "type_id":<type_id>,`
	`      "timestamp":{`
	`        "year": <YYYY>,`
	`        "month": <MM>,`
	`        "day": <DD>,`
	`        "hour": <hh24>,`
	`        "min": <mm>,`
	`        "sec": <ss>`
	`      },`
	`      "current_status":<string>,`
	`      "map_id":<map_id>,`
	`      "area_id":<area_id>,`
	`      "source_type": <string>,`
	`      "source_id": <int>,`
	`      "hierarchy": <int>,`
	`      "info": <int>`
	`    }`
	`  ]`
	`}`
		last_id == 0 → get all, else get_id > last_id
		Currently we show at most 50 events and notifications.
		current_status={clean_all, clean_map, clean_map_areas,clean_area, clean_spot, go_to, go_home, *etc.* };
		source_type={user, calendar, operation_unit, unknown};
		source_id=request_id (task_id from schedule, or cmd_id respectively);
		hierarchy=1 (top level task), > 1 (lower level task);

PFS & Parameters	Return Values	Description

get/task_history

PFS:2|2L|3

[opt] last=<integer>
[opt] last_id=<integer>

```
{
  "task_history":[
    {
      "id":<int>,
      "task_type_id":<task_type_id>,
      "task_type":<string>,
      "strategy":<string>,
      "method": <method>,
      "pump_volume": <pump_volume>,
      "cleaning_parameter_set":<set_id>,
      "map_id":<map_id>,
      "area_ids":[<area_id>],
      "source":<string>,
      "source_id":<int>,
      "start_time":{
        "year": <YYYY>,
        "month": <MM>,
        "day": <DD>,
        "hour": <hh24>,
        "min": <mm>,
        "sec": <ss>
      },
      "end_time":<DATETIME structure, as above>,
      "state_id":<task_state_id>,
      "state":<string>,
      "area":<area_size>,
      "continuable":<int>,
      "event_history":[
        {
          "time":<DATETIME structure>,
          "state_id":<task_state_id>,
          "state":<string>
        }
      ],
      "area_history":[
        {
          "area_id":<area_id>,
          "start_time":<DATETIME structure>,
          "end_time":<DATETIME structure>,
          "state_id":<task_area_state_id>,
          "state":<string>
        }
      ],
      "firmware":<string>
    }
  ],
  "task_requires_map_confirmation":<int>,
  "task_requires_special_area_confirmation":<int>
}
```

Shows the status of the last few executed robot tasks.

(For last=N, returns at most the N last entries of the task history. For last_id=ID, skips the first tasks with id < ID.)

For a list of supported task types, task states and task area states, see chapters 3.10.14, 3.10.15 and 3.10.16.

Strategies are described in chapter 3.10.17.

Cleaning Parameter Sets are described in chapter 3.10.4.

For the format of start_time, end_time and time, see chapter 3.10.2.

If continuable is 1, that task can be continued via set/continue.

The event_history contains a list of previous interruptions, with time and interruption state.

The area_history contains a list of cleaned areas, with start and end time, and other area entries (e.g. areas created by reexploration). See 3.10.15 for possible states.

task_requires_map_confirmation points at the last exploration or extending map cleaning task that requires a user decision. It is 0, if no such decision is required. It is cleared if the map of that task is saved or reverted.

task_requires_special_area_confirmation points at the last carpet detecting task that has still at least one area marked as carpet_unprocessed. See 3.10.16

<table>
<tr><th>PFS &amp; Parameters</th><th>Return Values</th><th>Description</th></tr>
<tr><td><b>get/area_history</b></td><td></td><td></td></tr>
<tr><td>PFS:2|2L|3

map_id=<integer>
[opt] last=<integer&gt;</td><td>{
  "map_id": <int>,
  "area_history": [{
    "area_id": <int>,
    "cleaning_history": [{
      "state_id": <task_state_id>,
      "state": <string>,
      "area": <area_size>,
      "start_time": {
        "year": <YYYY>,
        "month": <MM>,
        "day": <DD>,
        "hour": <hh24>,
        "min": <mm>,
        "sec": <ss&gt;,
        "day_of_week": <DOW>
      },
      "end_time": {
        "year": <YYYY>,
        "month": <MM>,
        "day": <DD>,
        "hour": <hh24>,
        "min": <mm>,
        "sec": <ss&gt;,
        "day_of_week": <DOW>
      },
      "source":<string>,
      "source_id":<int>
    }
    ]
  }
  ]
}</td><td>Shows the area-specific cleaning history of all areas with area state clean (see chapter 3.10.8) in the map specified by map_id.
The area history for a specific area is only updated when the robot completes cleaning that area. It can have at most 50 entries; all entries older than 30 days are removed during the update process.
If the parameter last=N is given, then for each area at most N entries are returned.
The parameters state_id and state refer to the task area states (see chapter 3.10.16).
The parameter area reflects the actually cleaned area in cm^2 (format 1.26.5).
The parameters start_time and end_time give the start and end times of the cleaning process, see chapter 3.10.2 for time format specification.</td></tr>
</table>

Explanation of the output fields:

- "current_status" ('clean_all', 'go_home', etc) and "type" ('started', 'succeeded', etc) allow for a quick human interpretable reading of the events and notifications.
- "type_id" gives the strong association to the various events.
- "map_id" (if not equal to zero) gives the corresponding active map id for which the corresponding event is valid.
- "area_id" (if not equal to zero) the corresponding active area id if available.
- "source_type" or "source" ('user', 'calendar', 'operation_unit', 'unknown') describes the source of an event; if it is task related ('clean_all') it's the source of the task, which allows to distinguish user from calendar generated tasks. For example, "current_status"

= 'clean_all' with "source_type" = 'calendar' means the this clean all task has be initiated by the calendar. Events that are not related to tasks have "source_type" = 'operation_unit' (e.g., "battery_low").

- "source_id" allows the association to a specific "task_id" of a calendar entry or the "cmd_id" for a user generated task (internally this is the rob_task_id). During one rob_task (e.g., clean) all sub tasks (localize, go_home) have the same cmd_id.
- The "hierarchy" states the hierarchy level of task related events. A top level task will generate events with "hierarchy" = 1 while a lower level task will have events with "hierarchy" > 1. For example, a go home event corresponding to a task initiated by the user will have a "hierarchy" = 1 while a go home event in a clean all task will have a "hierarchy"= 2 (it's a lower level task within cleaning).
- "info" gives a numerical code with additional information for the particular event.

3.9. Direct Mode Requests

The purpose of this mode is to enable a lower level interface for more direct access to some command. This mode can be enabled and disabled by a "`direct/enable`" command.
While this mode is enabled only the here listed commands can be executed (instead of the normal list of set commands).

PFS & Parameters	Return Values	Description
`direct/enable`		

`PFS:2|2L|3`
enabled=<1/0>
	`{}`	Enables/disables direct control mode

`direct/stop`		

`PFS:2|2L|3`	`{`
	`"cmd_id":<command_id>`
	`}`
		Stops the robot

`direct/go_to`		

`PFS:2|2L|3`
x=<coordinate>
y=<coordinate>
[opt] heading=<angle>
	`{`
	`"cmd_id":<command_id>`
	`}`
		Plans a path to the specified coordinates, and moves there if possible, heading is optional

`direct/turn_to_meander`		

`PFS:2|2L|3`
x=<coordinate>
y=<coordinate>
heading=<angle>
	`{`
	`"cmd_id":<command_id>`
	`}`
		Executes a half circle turn to the position, if possible.

PFS & Parameters	Return Values	Description
direct/meander		

PFS:2|2L|3
x1=<coordinate>
y1=<coordinate>
x2=<coordinate>
y2=<coordinate>
	{
	"cmd_id":<command_id>
	}
		Follows a meander line between positions (x1,y1) and (x2,y2) as close as possible

direct/forward		

PFS:2|2L|3
dist=<dist>
speed=<speed>
	{
	"cmd_id":<command_id>
	}
		Moves <dist>(1.10.5) cm forward with speed <speed>(1.6.9) cm/sec

direct/back		

PFS:2|2L|3
dist=<dist>
speed=<speed>
	{
	"cmd_id":<command_id>
	}
		Moves <dist>(1.10.5) cm forward with speed <speed>(1.6.9) cm/sec

direct/turn_left		

PFS:2|2L|3
angle=<angle>
ang_speed=<ang_speed>
	{
	"cmd_id":<command_id>
	}
		Turns left by <angle>(1.4.11) rad using angular speed <ang_speed>(1.2.13) rad/sec

direct/turn_right		

PFS:2|2L|3
angle=<angle>
ang_speed=<ang_speed>
	{
	"cmd_id":<command_id>
	}
		Turns right by <angle>(1.4.11) rad using angular speed <ang_speed>(1.2.13) rad/sec

PFS & Parameters	Return Values	Description
direct/circular_move		

PFS:2|2L|3
radius=<dist>
turn_speed=<ang_speed>
[opt]turn_angle=<angle>
[opt]infinite_movement=0/1
	{
	"cmd_id":<command_id>
	}
		Move in a radius <radius>(1.10.5) [cm] with angular speed <turn_speed>(1.2.13) [rad/sec]. If not defined, <turn_angle> is 2π rad (1.4.11) and <infinite_movement> = 1. Note: if radius is positive, robot moves counter-clockwise, if negative, robot move clockwise.

direct/set_pwm		

PFS:2|2L|3
[opt]main_brush=<int>
[opt]side_brush=<int>
[opt]fan=<int>
[opt]pump=<int>
[opt]agitator=<int>
[opt]cleaning_parameter_set=<set_id>
	{
	"cmd_id":<command_id>
	}
		Sets the PWM values in the cleaning control. PWM values can be in range of <0, 100>. For <set_id> see chapter 3.10.4. If a cleaning parameter set other than `none` is given, the remaining PWM input values are ignored.

3.10. Data types

3.10.1. Modes

The list of possible modes is still evolving, but here are the ones that are currently available:

Name	Description
not_ready	Robot is in not ready mode and does not accept any tasks.
ready	Robot is in ready mode. It is fully operational and accepts all user tasks. If robot is sufficiently charged (and if defined connected to docking station), it will also accept all calendar tasks.
exploring	Robot is exploring due to user task. Robot accepts all user but no calendar tasks.
cleaning	Robot is cleaning due to user or calendar task (clean all, clean map). Robot accepts all user but no calendar tasks.
target_point	Robot has been sent to a target point by the user
go_home	Robot is going home due to user task. Robot accepts all user but no calendar tasks.
lifted	Robot is lifted. Robot accepts neither user nor calendar tasks.
direct_control	Robot is currently in direct control.
recovery	Boot failure. Firmware is corrupt. Re-flashing of the robot is required.
pairing	Robot is in Bluetooth and/or Open-Access-Point-pairing mode.
unknown	Robot is currently in unknown state.

3.10.2. Time Format

All date and time formats in the interface are based on a 24 hour time format. The valid range of the fields is as follows:

Abbreviation	Name	Valid Range	Note
YYYY	Year	[0000, 9999]	
MM	Month	[1, 12]	From Jan to Dec
DD	Day in Month	[1, 31]	The valid Range depends also on the Month and the Year
DOW	Day of Week	[1, 7]	From Mon to Sun
hh24	Hour of the day	[0, 23]	
mm	Minute	[0, 59]	
List-of-DOW	List of days of week	[1, 7]{,[1, 7]}*	E.g.: 1,2,3,4,5 for monday to friday, and 6,7 for the weekend
UTCOFFSET	UTC time zone offset	[%2B,%2D][00,23][00,59]	E.g.: %2B0200. %2B and %2D represent the sign in URL-encoded form (%2B=+, %2D=-). Direct encoded signs (+,-) only work on some platforms. The following number is exactly 4 digits. The first two digits represent the hours, the second two the minutes of the UTC offset.

`/set/time` will also accept abbreviated week day names instead of [1,7]. The allowed names are: mon, tue, wed, thu, fri, sat and sun. Therefore, `/set/time?day_of_week=wed&hour=12&min=0` and

/set/time?day_of_week=3&hour=12&min=0 are equivalent.
For /set/add_scheduled_task and /set/modify_scheduled_task, names and digits can be freely mixed. /set/add_scheduled_task?...&days_of_week=1,2,wed,thu,5&... can be used to schedule from monday to friday.

3.10.3. Cleaning Modes

This is a list of currently available cleaning modes that can be scheduled for automatic execution:

ID	Description
1	Same as `set/clean_all`. Cleans all reachable area in automatic mode.
2	Same as `set/clean_map`. If additional `areas` parameters are specified, `clean_map` with areas will be executed.

3.10.4. Cleaning Parameter Sets

A cleaning parameter set combines a number of parameter values (e.g. duty cycles for main brush, side brush, fan, the meander distance and so on) into a set with a unique identifier. The set of included parameters depends on the robot model. The parameter cleaning_parameter_set is an integer value which specifies the unique identifier. The robot will then use the parameter values associated with this id for the requested task. The associated parameter values can be configured through the parameter interface GUI. Since revision 6.47.0, it is also possible to pass the corresponding ID-string as argument in the http request.

ID	ID-string	Description
0	`none`	Default (none) mode
1	`normal`	Normal Mode
2	`silent`	Silent Mode
3	`intensive`	Intensive Mode
4	`super_silent`	Super-silent mode
5	`high`	High mode
6	`auto`	Can be used in place of `none` if `none` would imply that the parameter should be ignored.

3.10.5. Command Results

For certain commands that do not yield and instantaneous result, the outcome of these commands can be read via this `get/command_result`. It will return an array of the last commands containing the numerical `cmd_id`, and the status, and the error code of the command.

When you issue a set command you will get a numerical command id (cmd_id) in the return value, this number can be used to match the command with the matching status in the array which is returned by `get/command_result`.

Commands are kept in the memory until they are not finished (e.g., "executing" and "queued" are kept forever). When command_id is in a finished state, it is removed from memory and disappears from the console after 60 seconds (parameter par.algo.user_interface_manager.time_keep_comand_results).

The possible values for the status are:

Status	Description
queued	the command is inside the robots command queue

skipped	the command was skipped due to a higher priority command in the command queue
executing	the command is currently being executed
done	the command was successfully executed
error	the command execution was aborted cause of an error
interrupted	the command execution was interrupted by an unforeseen event (depending on command this could be obstacles, low battery,...)
aborted	the command execution was aborted to start a new user command or a new higher priority scheduled command

If a command fails, its error code will be non zero. For most commands this is 1, but some commands like wifi_connect encode some additional information into the error_code.

3.10.6. Cleaning Grid Map

The cleaning map is represented as a grid map. A grid map is a 2-dimensional mesh constructed out of square elements. The length of the borders of these squares is defined by <resolution>. In the cleaning coordinate system, the center of the lower left square is located at [lower_left_x; lower_left_y]. From there, elements are lined up along the x-axis (and y-axis resp.). x indices run from 0..size_x 1, y indices from 0..size_y 1. This means that in general the center of a cell in the feature map coordinate system can be calculated like this:
[lower_left_x;lower_left_y] + [x_index; y_index] * resolution.
The state of each individual cell is defined by a binary map, which is encoded by a simple RLE encoding to save bandwidth. It just outputs the number of 0s or 1s that follow. The decoded number of cells always equals size_x * size_y. The first element is the lower left element. From there, elements in x-direction follow row by row. The very first element in the encoding specifies the initial state (0 or 1), and does not represent a number in the resulting decoded map. After that, all numbers mean switch state and repeat for n times.

Example

Take for example the following simple grid map which represents the current cleaning state. The upper two rows are mixed, and the bottom row is fully cleaned:

```
0 0 0 0 0
1 1 0 0 1
1 1 1 1 1
```

size_x would be 5, size_y 3. The resulting rle encoded string would look like this: `0,7,2,1,5`

3.10.7. Map Identifier

The map-id (16 bit integer) needs to be provided with any location based command for synchronization purposes. Consider the following example: if the robot is in the robot relocalises itself in a map, it switches to the original with a different id. If during that phase a location based command is received, the map id is required. If the received map id does not coincide with the map id in the robot, the command is ignored. The client retrieves the most up to date map-id from any map related request (tile_map, door_map, cleaning_grid_map). If received map-ids do not match (e.g. during a map switching in a localization scenario), the client is responsible to clear the situation, i.e. using the map-ids.

3.10.8. Area Attributes

An area has the following attributes:

- area_id: Unique id of the area of a particular map.
- array of points: Describes the form of the area as a polygon points are interconnected by lines, the last point is connected to the first one.
- area_type: Specifies the (hierarchy) type of an area as to_be_cleaned or room. Areas of type room areas are generated by the robot after initial exploration. The collection of all room type areas for a given map are cleaned for a clean_map command. Areas of type to_be_cleaned are user defined and are only cleaned separately if specifically requested.
- area_state: Specifies the area state in terms of clean, blocking, inactive, proposed_blocking or declined_blocking. Inactive areas are not considered for cleaning. Areas with state proposed_blocking or declined_blocking have no influence on robot behavior and are only relevant for the automatic nogo area handling.
- area_meta_data: User-defined string, UTF-8 encoded name of area.
- cleaning_parameter_set: Specifies the active cleaning parameter set for this area (see Chapter 3.10.4).
- floor_type: Specifies the floor type of the area: none (unspecified), hard_wood, carpet, tiles, "low_pile_carpet"
- room_type: Specifies the room type of the area: none (unspecified), kitchen, office, sleeping, kids, bath, corridor, living, dining, lavatory, storage, hallway.
- strategy_mode: Specifies the cleaning strategy mode (see Section 3.10.17), which will be used if the user does not specify a cleaning strategy.
- method: Specifies the preferred cleaning method (dry or wet, see section 3.10.18), which will be used if the user did not specify dry or wet cleaning explicitly. A value of none will use the default method.
- pump_volume: Specifies the preferred pump volume mode (low, medium, high, see section 3.10.19), which will be used if the user did not specify a volume for the task explicitly. A volume of none will use the default pump settings.
- statistics: Statistic results for the area.

3.10.9. Points of interest Attributes

A point of interest has the following attributes:

- map_id: unique id of a particular map
- id: unique id of the point of interest
- pose: position of the point of interest in the map, given by `[x, y, heading]` triple.
- meta_data: user-defined string, UTF-8 encoded name of point of interest.

- timestamp: timestamp when the point of interest has been stored. Format <YYYY>, <MM>, <DD>, <HH>, <MM>, <SS>.
- type: specifies the type of a point of interest. Currently existing types:

`system_stuck_side_brush`	100
`system_stuck_main_brush`	101
`system_stuck_wheel`	102
`system_stuck_behavior`	103

3.10.10. Execution top level states

The following flags represents the possible top level states as reported in get/execution_state.

Name	Information
init	Robot is initializing. Wait until finished.
error	Robot is in an error condition from which it cannot recover. Robot must be power-cycled.
not_ready	Robot is in an abnormal condition. Please check get/robot_flags.
operational	Robot is in a normal condition.
test	Robot is doing an End-Of-Line calibration, verification or a demonstration.
pairing	Robot is in Bluetooth and/or Open-Access-Point-pairing mode. Send set/pairing_done when finished.

3.10.11. Execution operational states

The following flags represents the possible operational states as reported in get/execution_state. Apart from **none**, these states can only occur if the top level state is **operational**.

Name	Information
none	No further information is available.
ready	Robot is not doing anything in particular right now, but is ready for action.
busy	Robot is currently processing a task.
direct_control	Robot is in a mode that allows manual movement control.

3.10.12. Execution sub states

The following flags represents the possible sub states as reported in get/execution_state.

Name	Occurs in	Information
clean_all	**operational/busy**	**set/clean_all** is being executed, possibly started by the scheduler.
clean_map	**operational/busy**	**set/clean_map** is being executed, possibly started by the scheduler.
clean_spot	**operational/busy**	**set/clean_spot** is being executed.
explore	**operational/busy**	**set/explore** is being executed.
go_home	**operational/busy**	Robot is execution **set/go_home** or drives to the docking station/start point as part of another task.
go_to	**operational/busy**	**set/target_point** is being executed.

localize	**operational/busy**	Robot is trying to localize in a particular map as part of another task.
recharge_and_continue	**operational/busy**	Robot is going home to recharge and will continue its current task later. This happens as part of another task.
docking_search	**operational/busy**	Robot is searching for the docking station as part of another task.
undocking	**operational/busy**	Robot is undocking from the docking station as part of another task.
wet_pad_priming	**operational/busy**	Robot is priming its wet pad.
redocking	**operational/busy**	Robot is trying to redock to the docking station after connection loss.
test_calibration	**test**	Robot is executing one of the End-Of-Line calibrations or verifications.
test_endurance_test	**test**	Robot is executing a demonstration (**test/run_test?test_type=203**).
test_box_test	**test**	Robot is executing a demonstration (**test/run_test?test_type=201**).
test_rectangle_odo_test	**test**	Robot is executing a demonstration (**test/run_test?test_type=204**).
test_unlimited_cleaning	**test**	Robot is executing a demonstration (**test/run_test?test_type=200**).
test_straight_line	**test**	Robot is executing a demonstration (**test/run_test?test_type=202**).

3.10.13. Robot flags

The following flags represents conditions of the robot that might or might not inhibit it from operating normally.

Name	**Type**	**Required action**
lifted	not-ready	Put down robot
battery_low	notification	None, however cleaning is disabled until the robot is recharged
battery_critical	not-ready	Put robot on docking station or connect it via power cable, and wait until charged
connected_to_cable	not-ready	Disconnect power cable from robot
stuck_drop_sensor	not-ready	Lift robot and put it down again
dustbin_missing	not-ready	Reinsert dustbin. (Note: for some products, this is only a notification.)
dustbin_full	notification	Empty dustbin at next opportunity
toplid_open	notification	Close the lid. Cleaning and exploration are disabled in this state
water_tank_inserted	notification	Behaviour differs when water tank is docked
water_tank_empty	notification	Cannot clean with empty water tank inserted
main_brush_missing	notification	None, however, if this happens often, the cause might be a damaged main brush component

battery_temp_critical	not-ready	None. Note: when this happens during charging, it is only a notification
pairing	not-ready	Finish pairing or power-cycle robot
layers_not_running	error	Power-cycle robot
safety_supervisor_eternal_stop	error	Power-cycle robot
safety_supervisor_safety_stop	not-ready	Lift robot and put it down again
timestamp_overflow	error	Power-cycle robot
missing_camera_data	error	Power-cycle robot
missing_imu_data	error	Power-cycle robot
stuck_main_brush	notification	Clear main brush of obstructions if there are some. This can be caused by thick carpets.
stuck_side_brush	notification	Clear side brush of obstructions if there are some. This can be caused by thick carpets.
stuck_wheel	notification	Clear wheels of obstructions if there are some.
stuck_fan	notification	Clear fan of obstructions if there are some.
stuck_behavior	notification	Robot failed to plan its movement due to its current environment. Please remove obstacles, or place the robot somewhere else.
stuck_bumper	notification	Check bumper for foreign objects.
stuck_wheelswitch	not-ready	Lift robot and put it down again.
stuck_water_pump	notification	Check water pump if it happens repeatedly.
missing_water_pump	notification	Check water pump if it happens repeatedly.
stuck_wet_pad_agitator	notification	Check agitator if it happens repeatedly.
missing_wet_pad_agitator	notification	Check agitator if it happens repeatedly.
missing_sensor_data	notification	Some unspecified sensor data is missing.

3.10.14. Task types

The following task types can be shown in the task history.

Type	**Type id**
clean_all	0
clean_map	1
clean_spot	2
explore	3
go_home	4
go_to	5
skipped_task	6
reexplore	7

3.10.15. Task states

A task in the task history may be in one of the following states.

State	**State id**	**Cause of interruption**
executing	0	No interruption, task is running
done	1	Task was completed successfully

failed	2	Failure for unspecified reasons
interrupted_by_user	3	Another user command was issued
interrupted_system_reinitialization_pending	15	Robot was still reinitializing
interrupted_system_reinitialization_failure	17	Robot reinitialization failed, requires power-cycle
interrupted_map_missing	18	Map initialization problem
interrupted_battery_critical	20	Insufficient battery level to operate
interrupted_battery_low	21	Insufficient battery level to continue cleaning
event_docked	25	NOTE: Appears only in the event history
interrupted_cable_connected	27	A connected power-cable disallowed robot movement
interrupted_lifted	30	Robot was lifted off the ground
interrupted_toplid_open	35	Top lid was opened
interrupted_stuck_main_brush	40	Main brush appeared to be stuck
interrupted_stuck_side_brush	41	Side brush appeared to be stuck
interrupted_stuck_wheels	42	One of the wheels appeared to be stuck
interrupted_stuck_fan	43	Fan appeared to be stuck
interrupted_stuck_by_behaviour	44	Robot could no longer plan its movements, probably because of difficult environment.
interrupted_stuck_by_dropsensor	45	Drop sensor reports suggested that it was unsafe to navigate, or drop sensors were active at start of task.
interrupted_stuck_bumper	46	Bumper was not released despite considerable movement of the robot, or was active at start of task.
interrupted_stuck_wheelswitch	47	Wheelswitch was active at start of task.
interrupted_missing_dustbin	50	Dustbin appeared to be missing
interrupted_safety_supervisor	60	Safety supervisor chip blocked all movements, requires power-cycle
interrupted_safety_stop	62	Safety supervisor chip blocked all movements, requires lift
interrupted_missing_main_brush	70	Main brush appeared to be missing
interrupted_battery_temp_critical	75, 77	Battery was apparently overheating (75 = while charging, 77 = while discharging/cleaning)
interrupted_water_tank_removed	80	Water tank was removed
interrupted_water_tank_inserted	81	Water tank was inserted
interrupted_water_tank_empty	82	Cleaning was interrupted due to an empty water tank
interrupted_missing_water_pump	85	Water pump reported undercurrent
interrupted_stuck_water_pump	86	Water pump reported overcurrent
interrupted_missing_wet_pad_agitator	87	Wet pad agitator reported undercurrent
interrupted_stuck_wet_pad_agitator	88	Wet pad agitator reported overcurrent
interrupted_power_switch_shutdown	100	Robot was shutdown by the power switch

interrupted_low_battery_shutdown	101	Robot was shutdown due to a critical battery condition
interrupted_reboot	102	Robot was shutdown to execute a reboot
interrupted_firmware_update_reboot	103	Robot was shutdown to execute a reboot after a firmware update
event_dry_cleaning	110	Robot started dry cleaning
event_wet_cleaning	111	Robot started wet cleaning
started_docking_search	142	Search for the docking station was started as part of the task
started_recharge_and_continue	143	Recharging was started as part of the task
started_redocking	147	Redocking after connection loss was started as part of the task
started_reexplore	148	Reexploration / map extension was started as part of the task
started_auto_deep_clean	149	Robot started second cleaning pass after finishing the first
succeeded_docking_search	152	Search for the docking station was successfully finished as part of the task
succeeded_recharge_and_continue	153	Recharging was successfully finished as part of the task
succeeded_redocking	157	Redocking after connection loss was successfully finished as part of the task
succeeded_reexplore	158	Reexploration / map extension was successfully finished as part of the task
failed_go_home	160	Going home as part of the task failed
failed_localization	161	Robot failed to confirm its position in the map, and stopped to avoid causing damage in the no-go areas
failed_docking_search	162	Finding the docking station as part of the task failed
failed_recharge_and_continue	163	Recharging as part of the task failed
failed_go_to	164	Going to a target point as part of the task failed
failed_localization_due_to_timeout	165	Robot failed to confirm its position in the map within the given time limit
failed_relocalization	166	Robot failed subsequent confirmations of its position in the map within the given time limit
failed_redocking	167	Redocking after connection loss failed as part of the tas
failed_reexplore	168	Reexploration / map extension failed as part of the task
failed_go_home_due_to_blocking_area	169	Failed go-home due to path being blocked by blocking area (or carpet area if using wet-clean)

failed_target_unreachable_due_to_blocking_area	170	Failed to drive to target due to path being blocked by blocking area (or carpet area during wet-clean)
interrupted_stuck_by_behaviour_due_to_drop_sensor	215	Robot couldn't free itself due to drop sensor
interrupted_stuck_by_behaviour_due_to_self_lift	216	Robot couldn't free itself due to self lift
interrupted_stuck_by_behaviour_due_to_complicated_terrain	217	Robot couldn't free itself due to difficult terrain (carpet, obstacles, ...)
interrupted_stuck_by_behaviour_due_to_blocking_area	218	Robot couldn't free itself due to no-go/blocking area blocking the path
interrupted_stuck_by_behaviour_due_to_carpet	219	Robot couldn't free itself due to carpet
skipped_due_to_not_docked	220	Calendar task was skipped because robot was not docked
skipped_due_to_insufficient_battery_level	221	Calendar task was skipped because robot battery level was insufficient
skipped_due_to_timeout	222	Task was skipped because it could not be started in a timely manner
skipped_due_to_open_toplid	223	Task was skipped because the top lid was open
skipped_continue	224	A set/continue could not be executed
skipped_start_or_continue	225	A set/clean_start_or_continue could not be executed
skipped_by_clean_all	230	A task was skipped because a clean_all was already running
skipped_by_clean_map	231	A task was skipped because a clean_map was already running
skipped_by_clean_spot	232	A task was skipped because a clean_spot was already running
skipped_by_explore	233	A task was skipped because explore was already running
skipped_by_reexplore	234	A task was skipped because reexplore was already running
skipped_by_go_home	235	A task was skipped because a go_home was already running
skipped_by_go_to	236	A task was skipped because a go_to was already running
skipped_by_task	237	A task was skipped because some other task was already running
skipped_by_test_mode	238	A task was skipped because robot was in test mode
skipped_by_direct_ctrl_mode	239	A task was skipped because robot was in direct control mode
skipped_by_pairing_mode	240	A task was skipped because robot was in pairing mode

skipped_by_error_mode	241	A task was skipped because robot was in error state
skipped_by_not_ready_mode	242	A task was skipped because robot was not ready
skipped_by_init_mode	243	A task was skipped because robot was still initializing

3.10.16. Task area states

Relevant for the area history inside a task history entry (`get/task_history`) and the area-specific cleaning history (`get/area_history`).

State	**State id**	**Description**
executing	0	Area is being cleaned
done	1	Area cleaning was completed successfully
failed	2	Area cleaning failed for unspecified reasons
extended	3	Area was added as part of reexploration resp. map extension
carpet_unprocessed	4	Area was added during a carpet detection task. Will be changed to carpet_processed after a corresponding set/modify_area or set/delete_area.
carpet_processed	5	Area was added during a carpet detection task, and already either confirmed or rejected.
carpet_extended	6	Area was added during a carpet detection task, while extending the map. Will be changed to carpet_unprocessed if map is saved.
interrupted_failed_relocalization	10	Area cleaning was interrupted because relocalization test failed
interrupted_wet_cleaning_carpet	11	Tried to wet clean a carpet area
interrupted_area_state_not_clean	12	State of area to be cleaned was NOT 'clean'
interrupted_area_not_reachable	13	Failed to reach area, which was therefore skipped
aborted	14	Relevant for `get/area_history`: cleaning of area was aborted (e. g., robot was lifted, user sent a stop-command, battery was low, ...)
interrupted_battery_low	15	Area cleaning was interrupted because battery was low
multiple_map_relocalization	16	Clean-area interrupted by multiple map relocalization
pp_failed_due_to_blocking_area	17	Clean-area failed due to path being blocked by blocking area (or carpet area during wet-clean)
pending	99	Pending areas will be next to clean, in that order

3.10.17. Cleaning strategies

Strategy	**Mode id**	**Description**

normal	1	Clean everything in the target area (default cleaning strategy)
walls_and_corners	2	Concentrate only on cleaning outer boundaries of target area, and spaces near walls and obstacles
deep	3	Clean everything in the target area twice (horizontally and vertically, respectively)
none	4	If this strategy mode is selected for cleaning, the areas will be cleaned with their individual strategy modes (which is set in the strategy_mode field of the area attributes and can be different for each area, see Section 3.10.8)
rigorous	7	Clean everything in the target area twice (horizontally and vertically, respectively), putting extra effort into edge cleaning.

3.10.18. Cleaning methods

Method	Description
none	Use default method. E.g. use wet cleaning if water tank is inserted. Otherwise do dry cleaning. Depends on robot family.
dry	Dry clean.
wet	Wet clean.

3.10.19. Pump volume modes

ID	ID string	Description
0	`none`	Don't change the setting. Not supported for `set/pump_volume_settings`.
1	`low`	
2	`medium`	
3	`high`	
4	`auto`	Can be used in place of `none` if `none` would imply that the parameter should be ignored.
-	`direct`	Not supported anymore.

3.10.20. Sensor types and measurements

Device type	Payload type	Payload	Description

gpio	sensor_input_gpio	`{`
		`"event_id": <int>,`
		`"timestamp": <timestamp>,`
		`"value":    <enum:"active"/"inactive">`
		`}`
			Sensors knowing only the two states on and off, e.g. Bumpers, some dropsensors, dustbin switch.

current_sensor	sensor_input_ current	`{`
		`"current": <milliampere>,`
		`"timestamp": <timestamp>,`
		`}`
			Electrical current

voltage_sensor	sensor_input_ voltage	`{`
		`"voltage": <millivolt>,`
		`"timestamp": <timestamp>,`
		`}`
			Electrical voltage

actuator_pwm	device_command_
	actuator_pwm
		{
		"pwm": <int>
		}
			PWM settings for cleaning gadget motors in percent (0-100)

ir_sensor	sensor_input_
	ir_sensor
		{
		"timestamp":<timestamp>,
		"low_off":<int16>,
		"low_on":<int16>,
		"med_off":<int16>,
		"med_on":<int16>,
		"high_off":<int16>,
		"high_on":<int16>,
		}
			Raw dropsensor adc values)

speed_sensor	sensor_input_
	speed_sensor
		{
		"timestamp":<timestamp>,
		"velocity": <int32, cm/s, 1.22.9>
		}
			Raw wheel speed

3.10.21. Device descriptors

A list of device descriptors typically present on a robot. Note that the precise device descriptors used depend on the precise robot model. The symbol * is a placeholder for an arbitrary string, e.g. bumper_* could include bumper_left and bumper_right or bumper_1 and bumper_2.

Note that a single device descriptor might appear for several different device types since a single device might produce different kinds of measurements (e.g. current and voltage from the battery).

Descriptor	Device type	Remark
drop_*	gpio	
bumper_*	gpio	Typically bumper_left and bumper_right
main_brush	current_sensor	
side_brush_*	current_sensor	Typically side_brush_left and side_brush_right
fan	current_sensor	
battery	current_sensor	
wheel_*	current_sensor	Typically wheel_left and wheel_right
battery	voltage_sensor	
fan	actuator_pwm	
main_brush	actuator_pwm	
side_brush	actuator_pwm	Side brush speeds can only be set for all side brushes at once
sense_*	ir_sensor	Raw infrared values from dropsensors
wheel_left	speed_sensor	Raw speed of left wheel
wheel_right	speed_sensor	Raw speed of right wheel

3.10.22. Data types and meta information

Fixpoint datatypes may be signed or unsigned; the fixpoint format specified in the additional meta information below will contain the sign information.

Data type	Description
uint8	unsigned 8-bit integer
int8	signed 8-bit integer
uint16	unsigned 16-bit integer
int16	signed 16-bit integer
uint32	unsigned 32-bit integer

int32	signed 32-bit integer
uint64	unsigned 64-bit integer
int64	signed 64-bit integer
fract8	8-bit fixpoint number
fract16	16-bit fixpoint number
fract32	32-bit fixpoint number
float	single-precision floating point number
double	double-precision floating point number
string	String

The following meta information may appear in addition to a data type:

Information	**Description**
fixpoint_format	fixpoint format of a fixpoint number in the format sign bits.pre-comma bits.post-comma bits
min	minimum allowed value for a numeric primitive
max	maximum allowed value for a numeric primitive

4. Error Handling

When the server does not reply with 2xx (i.e. success), the response will be either empty (in case of an unknown error event) or contain a standard error message.

A standard error message will be formatted like this:

```
{
  "error_code": <code>,
  "error_tag" : "<Tag>",
  "error_msg" : "<Message>"
}
```

4.1. Possible Error Codes

error_code	error_tag
101	unknown_request
102	parameter_error
103	value_unknown
104	not_implemented
105	data_timeout
106	request_deprecated
107	request_not_successful

A. Examples

A.1. Get the correct Feature Map

Request the feature map from the robot// `http://<ip-of-robot>:<port>/get/feature_map` The robot will answer with status 200 OK and the following content:

```
{
  "map": {
  "lines": [
  {
    "x1": 100,
    "y1": 100,
    "x2": 200,
    "y2": 100
  },
  {
    "x1": 200,
    "y1": 100,
    "x2": 200,
    "y2": 200
  },
  {
    "x1": 200,
    "y1": 200,
    "x2": 100,
    "y2": 200
  },
  {
    "x1": 100,
    "y1": 200,
    "x2": 100,
    "y2": 100
  }
  ]
  }
}
```

A.2. Get the current robot Status

`http://<ip-of-robot>/get/status`

The robot will answer with status 200 OK and the following content:

```
{
  "voltage": 16384,
  "mode": "exploring",
  "cleaning_parameter_set": 0,
  "battery_level": 79,
  "charging": "disconnected",
  "time": {
    "year": 2014,
    "month": 4,
    "day": 11,
    "hour": 17,
    "min": 42
  }
}
```

The robot is in exploration mode, the battery level is 79%, and the voltage is 16384/1024 = 16 V (in FXP 1.5.10).

A.3. Send robot to some location

`http://<ip-of-robot>/set/target_point?x1=150&y1=150`
If the request is valid, the robot will answer with status 200 OK and following response:

```
{
"cmd_id": 1
}
```

Note that the ordering of the parameters is important. The following will not work in the current implementation:
`http://<ip-of-robot>/set/target_point?y1=150&x1=150`
It might answer with an error code (400) and send an error response:

```
{
  "error_code": 102,
  "error_tag" : "parameter_error",
  "error_msg" : "Unexpected Parameter y1"
}
```

The robot will then switch into `target _point` mode (this will be the mode returned by `get/status`). Upon completion of the command, a request to `get/command_result` might yield the following outcome:

```
{
  "commands": [
  {
    "cmd_id" : 1,
    "status" : "executing"
  }
  ]
}
```

If you send then a `set/stop` command which answers with:

```
{
"cmd_id": 2
}
```

and wait until the robot stops, `get/command_result` will show:

```
{
  "commands": [
  {
    "cmd_id" : 1,
    "status" : "aborted"
    "error_code": 0
  },
  {
   "cmd_id" : 2,
    "status" : "done"
    "error_code": 0
  }
  ]
}
```

A.4. Add scheduled task

We can send a http request to the robot (remember to use the right port):

```
http://<ip-of-robot>/set/add_scheduled_task?cleaning_mode=0&cleaning_parameter_set=0&
year=2018&month=10&day=10&hour=10&min=10&repeated=01&map_id=0&param1=10&param2=10"
```

In this case we set a new scheduled task to be executed on 10.10.2018 at 10:10.
If the request is valid, the robot will answer with status 200 OK and following response:

```
{
"cmd_id": 1
}
```

Note that the ordering of the parameters is important. If we send, for example, an incomplete request:

```
http://<ip-of-robot>/set/add_scheduled_task?cleaning_mode=0&cleaning_parameter_set=0&
year=2018&month=10&day=10"
```

It might answer with an error code (400) and send an error response:

```
{
"error_code": 102,
"error_tag" : "parameter_error",
"error_msg" : ""
}
```

Please follow strictly the rules about mandatory parameters.

A.5. Erroneous command result

If an http operation returns an error, the response on `get/command_result` will show:

```
{
  "commands": [
  {
    "cmd_id":1,
    "status": "error"
    "error_code": 0
  }]
}
```

Example: Add an area to non-existing map (with map_id=123).
Command `set/add_area?map_id=123&x1=100&y1=200&x2=100&y2=200&x3=100&y3=200` returns:

```
{
"cmd_id": 1
}
```

After calling `get/command_result`, we obtain:

```
{
  "commands": [
  {
    "cmd_id":1,
    "status": "error"
    "error_code": 0
  }]
}
```