go-eCharger API Specification
English Version

Version	Date	Author	Description
1.0	2018-02-14	Peter Pötzi	Initial version
1.2	2018-05-09	Peter Pötzi	Typos corrected
1.3	2018-06-27	Peter Pötzi	fix dws type
1.4	2018-07-16	Peter Pötzi	Explain format
1.5	2019-04-05	Peter Pötzi	Add scheduler, awattar zone, load balancing, custom mqtt

Index

1. Connection **2**
Rate Limiting 2

2. API: status **4**
Request Path 4
Return Format 4
Parameter 5

3. Commands **13**
Set parameter 13
Path 13

4. Return Values **14**
Local WiFi / Hotspot 14

Cloud: MQTT 14
Cloud: REST Api 14
5. Cloud REST Api Workflow **15**
6. Cloud MQTT Workflow **17**
Connection 17
Actions 17
7. Custom MQTT Server **19**

1. Connection

The go-eCharger offers two WLAN interfaces, one of which always serves as a mobile hotspot and another that can connect to an existing WLAN network to establish an Internet connection.

The following connections are offered for the API:

Connection	**Path**
`WiFi Hotspot`	`http://192.168.4.1/`
`WiFi local network`	`http://x.x.x.x/` The IP address is retrieved from the DHCP server
`Cloud: MQTT`	`wss://i8p7v0.messaging.internetofthings.ibmcloud.com`
`Cloud: REST Api`	`https://api.go-e.co/`

Authentication:

Connection	**Authentication**
`WiFi Hotspot`	None (Hotspot WPA key must be known)
`WiFi local network`	None (device must be in the same WLAN and the HTTP Api must be activated with the go-eCharger app)
`Cloud: MQTT`	MQTT deviceID + token go-eCharger Cloud Token
`Cloud: REST Api`	go-eCharger Cloud Token

Rate Limiting

Verbindung	Limit
`WiFi Hotspot`	None (5 second delay recommended)
`WiFi local network`	None (5 second delay recommended)
`Cloud: MQTT`	**Fair-use Limit:** 50MB per month, about 25'000 requests. In case of planned exceeding, please contact go-e GmbH!

`Cloud: REST Api`	**Hard Limit**: 180 requests per 15 minutes sliding window. (~ 5 seconds per request) and source IP address. In case of planned exceeding, please contact go-e GmbH!
	
	`/api`
	Fair use Limit: 50MB per month, about 500'000 requests. In case of planned exceeding, please contact go-e GmbH!
	
	`/api_status`
	Fair use Limit: 50MB per month, about 25'000 requests. In case of planned exceeding, please contact go-e GmbH!

2. API: status

Returns all relevant parameters as a JSON object.

Example (incomplete):

```
{"version":"B","rbc":"251","rbt":"2208867","car":"1","amp":"10","err":"0","ast"
:"0","alw":"1","stp":"0","cbl":"0","pha":"8","tmp":"30","dws":"0","dwo":"0","ad
i":"1","uby":"0","eto":"120","wst":"3","nrg":[2,0,0,235,0,0,0,0,0,0,0,0,0,0,0,0
],"fwv":"020-rc1","sse":"000000","wss":"goe","wke":"","wen":"1","tof":"101","td
s":"1","lbr":"255","aho":"2","afi":"8","ama":"32","al1":"11","al2":"12","al3":"
15","al4":"24","al5":"31","cid":"255","cch":"65535","cfi":"65280","lse":"0","us
t":"0","wak":"","r1x":"2","dto":"0","nmo":"0","eca":"0","ecr":"0","ecd":"0","ec
4":"0","ec5":"0","ec6":"0","ec7":"0","ec8":"0","ec9":"0","ec1":"0","rca":"","rc
r":"","rcd":"","rc4":"","rc5":"","rc6":"","rc7":"","rc8":"","rc9":"","rc1":"","
rna":"","rnm":"","rne":"","rn4":"","rn5":"","rn6":"","rn7":"","rn8":"","rn9":""
,"rn1":""}
```

Request Path

Connection	Path
`WiFi Hotspot`	`http://192.168.4.1/status`
`WiFi local network`	`http://x.x.x.x/status`
`Cloud: MQTT`	Subscribe to: `iot-2/cmd/status/fmt/json`
`Cloud: REST Api`	`https://api.go-e.co/api_status?token=TOKEN`*`[&wait=0]`* The **`wait`** parameter is optional

Return Format

Connection	Path
`WiFi Hotspot`	`Plain STATUS_OBJECT`
`WiFi local networ`	`Plain STATUS_OBJECT`
`Cloud: MQTT`	`Plain STATUS_OBJECT`
`Cloud: REST Api`	`{"success":true,"age":AGE_IN_MILLISECONDS,"data":STATUS_OBJECT}`

Parameter

In addition to these parameters, other parameters may also be added without prior notice and depending on the type of connection.

Explanation Format: All parameters are sent in the JSON object as a string (in quotation marks). Most of these parameters can be converted to an integer format. The data type specified in the format shows the expected size. If the string is not converted to the specified data type, a communication error should be displayed.

Parameter	Format	Explanation
`version`	`String (1)`	**JSON Format.** "B": normal case "C": When end-to-end encryption is enabled
`rbc`	`uint32_t`	**reboot_counter**: Counts the number of boot operations. Sent with end-to-end encryption as protection against replay attacks.
`rbt`	`uint32_t`	**reboot_timer**: Counts the milliseconds since the last boot. Sent with end-to-end encryption as protection against replay attacks. Expires after 49 days, increasing the reboot_counter.
`car`	`uint8_t`	**Status PWM Signaling** 1: charging station ready, no vehicle 2: vehicle loads 3: Waiting for vehicle 4: Charge finished, vehicle still connected
`amp`	`uint8_t`	Ampere value for the PWM signaling in whole ampere of **6-32A**
`err`	`uint8_t`	**error**: 1: RCCB (Residual Current Device) 3: PHASE (phase disturbance) 8: NO_GROUND (earthing detection) 10, default: INTERNAL (other)
`ast`	`uint8_t`	**access_state**: Access control. 0: open 1: RFID / App needed 2: electricity price / automatic
`alw`	`uint8_t`	**allow_charging:** PWM signal may be present

		0: no 1: yes
`stp`	`uint8_t`	**stop_state:** Automatic shutdown 0: deactivated 2: switch off after kWh
`cbl`	`uint8_t`	**Typ2 Cable Ampere encoding** 13-32: Ampere Codierung 0: no cable
`pha`	`uint8_t`	**Phasen** before and after the contactor binary flags: 0b00ABCDEF A ... phase 3, in front of the contactor B ... phase 2 in front of the contactor C ... phase 1 in front of the contactor D ... phase 3 after the contactor E ... phase 2 after the contactor F ... phase 1 after the contactor pha | 0b00001000: Phase 1 is available pha | 0b00111000: Phase1-3 is available
`tmp`	`uint8_t`	**Temperature** of the controller in °C
`dws`	`uint32_t`	**Charged energy** in deca-watt seconds *Example: 100'000 means, 1'000'000 Ws (= 277Wh = 0.277kWh) were charged during this charging process.*
`dwo`	`uint16_t`	**Abschaltwert** in 0.1kWh if **`stp==2`**, for dws parameter *Example: 105 for 10,5kWh* Charging station logic: `if(dwo!=0 && dws/36000>=dwo)alw=0`
`adi`	`uint8_t`	**adapter_in**: Charging box is plugged in with adapter 0: NO_ADAPTER 1: 16A_ADAPTER
`uby`	`uint8_t`	**unlocked_by**: Number of the RFID card that has activated the current charging process
`eto`	`uint32_t`	**energy_total**: Total charged energy in 0.1kWh *Example: 130 means 13kWh charged*
`wst`	`uint8_t`	**wifi_state**: Wi-Fi connection status

		3: connected default: not connected

`nrg`	`array[15]`	Array with values of the current and voltage sensor
		nrg [0]: voltage on L1 in volts
		nrg [1]: voltage on L2 in volts
		nrg [2]: voltage on L3 in volts
		nrg [3]: voltage to N in volts
		nrg [4]: Ampere on L1 in 0.1A (123 equals 12.3A)
		nrg [5]: Ampere on L2 in 0.1A
		nrg [6]: Ampere on L3 in 0.1A
		nrg [7]: power on L1 in 0.1kW (36 equals 3.6kW)
		nrg [8]: power on L2 in 0.1kW
		nrg [9]: power at L3 in 0.1kW
		nrg [10]: power at N in 0.1kW
		nrg [11]: Total power in 0.01kW (360 equals 3.6kW)
		nrg [12]: power factor on L1 in%
		nrg [13]: power factor on L2 in%
		nrg [14]: power factor on L3 in%
		nrg [15]: Power factor on N in%
		
		App logic:
		`if(Math.floor(pha/8) ==1 &&`
		  `parseInt(nrg[3])>parseInt(nrg[0])){`
		      `nrg[0]=nrg[3]`
		      `nrg[7]=nrg[10]`
		      `nrg[12]=nrg[15]`
		`}`

`fwv`	`String`	**Firmware Version** *Example: "020-rc1"*
`sse`	`String`	**Serial number** number formatted as %06d Example: "000001"
`wss`	`String`	WiFi **SSID** *Example: "My home network"*
`wke`	`String`	WiFi **Key** *Example: "*********" for fwv after 020* *Example: "password" for fwv before 020*
`wen`	`uint8_t`	**wifi_enabled**: Wi-Fi enabled

		0: deactivated 1: activated
`tof`	uint8_t	**time_offset**: Time zone in hours for internal battery-powered clock +100 Example: 101 is GMT + 1
`tds`	uint8_t	**Daylight saving time offset** (Summer time) in hours *Example: 1 for Central Europe*
`lbr`	uint8_t	**LED brightness** from 0-255 0: LED off 255: LED brightness maximum
`aho`	uint8_t	Minimum **number** of hours in which to load with "electricity price - automatic" *Example: 2 ("Car is full enough after 2 hours")*
`afi`	uint8_t	Hour (**time**) in which with "electricity price - automatically" the charge must have lasted at least aho hours. *Example: 7 ("Done until 7:00, so before at least 2 hours loaded")*
`azo`	uint8_t	Awattar price zone 0: Austria 1: Germany
`ama`	uint8_t	Absolute max. Ampere: Maximum value for ampere setting *Example: 20 (can not be set to more than 20A in the app)*
`al1`	uint8_t	Ampere Level 1 for push button on the device. 6-32: Ampere level activated 0: level deactivated (is skipped)
`al2`	uint8_t	Ampere Level 2 for push button on the device. Must be either 0 or> al1
`al3`	uint8_t	Ampere Level 3 for push button on the device. Must be either 0 or> al2
`al4`	uint8_t	Ampere Level 4 for push button on the device. Must be either 0 or> al3
`al5`	uint8_t	Ampere Level 5 for push button on the device. Must be either 0 or> al4

`cid`	`uint24_t`	Color idle: **color value for standby** (no car plugged in) as a number *Example: parseInt ("# 00FFFF"): 65535 (blue / green, default)*
`cch`	`uint24_t`	Color charging: **color value for charging active**, as a number *Example: parseInt ("# 0000FF"): 255 (blue, default)*
`cfi`	`uint24_t`	Color idle: **color value completed for charging**, as a number *Example: parseInt ("# 00FF00"): 65280 (green, default)*
`lse`	`uint8_t`	**led_save_energy**: Turn off the LED automatically after 10 seconds 0: Energy saving function deactivated 1: Energy saving function activated
`ust`	`uint8_t`	**unlock_state**: Cable lock adjustment 0: lock as long as the car is plugged in 1: Automatically unlock after charging 2: Always leave the cable locked
`wak`	`String`	WiFi **Hotspot Password** *Example: "abdef0123456"*
`r1x`	`uint8_t`	**Flags** 0b1: HTTP Api in the WLAN network activated (0: no, 1: yes) 0b10: End-to-end encryption enabled (0: no, 1: yes)

`dto`	`uint8_t`	**Remaining time** in milliseconds remaining on activation by electricity prices
		App logic:
		`if(json.car==1)message = "Zuerst Auto anstecken"`
		`else message = "Restzeit: …"`

`nmo`	`uint8_t`	**Norway mode** activated 0: deactivated (earthing detection activated) 1: activated (no earthing detection, intended only for IT grids)

`eca`
`ecr`
`ecd`
`ec4`
`ec5`
`ec6`
`ec7`
`ec8`
`ec9`
	`uint32_t`	Charged **energy per RFID card** from 1-10
		
		Example: eca == 1400: 140kWh charged on card 1
		Example: ec7 == 1400: 140kWh charged on board 7
		Example: ec1 == 1400: 140kWh charged on card 10

ec1		
rca rcr rcd rc4 rc5 rc6 rc7 rc8 rc9 rc1	String	**RFID Card ID** from 1-10 as a string Format and length: variable, depending on the version
rna rnm rne rn4 rn5 rn6 rn7 rn8 rn9 rn1	String	**RFID Card Name** from 1-10 Maximum length: 10 characters
tme	String	**Current time,** formatted as ddmmyyhhmm 0104191236 corresponds to 01.04.2019 12:36
sch	String	**Scheduler settings** (base64 encoded) Functions for encode and decode are here: https://gist.github.com/peterpoetzi/6cd2fad2a915a2498776912c5a a137a8 The settings can be set in this way: r21=Math.floor(encode(1)) r31=Math.floor(encode(2)) r41=Math.floor(encode(3)) *Direct setting of sch = is not supported*
sdp	uint8_t	**Scheduler double press**: Activates charge after double pressing the button if the load has just been interrupted by the scheduler 0: Function disabled 1: Allow charge immediately
upd	uint8_t	**Update available** (only available if connected via go-e server)

		0: no update available 1: Update available
`cdi`	`uint8_t`	**Cloud disabled** 0: cloud enabled 1: cloud disabled
`loe`	`uint8_t`	**Load balancing enabled** 0: load balancing disabled 1: Load balancing activated via cloud
`lot`	`uint8_t`	**Load balancing group total ampere**
`lom`	`uint8_t`	**Load balancing minimum amperage**
`lop`	`uint8_t`	**Lastmanagement priority**
`log`	`String`	**Lastmanagement group ID**
`lon`	`uint8_t`	**Lastmanagement:expected number of charging stations** (currently not supported)
`lof`	`uint8_t`	**Load balancing fallback amperage**
`loa`	`uint8_t`	**Load balancing Ampere** (current permitted charging current) is automatically controlled by the load balancing)
`lch`	`uint32_t`	**Load balancing: seconds since the last current flow while the car is still plugged in** 0 when charging is in progress
`mce`	`uint8_t`	**MQTT custom enabled** Connect to your own MQTT server 0: Function disabled 1: Function activated
`mcs`	`String(63)`	**MQTT custom Server** Hostname without protocol specification (z.B. test.mosquitto.org)
`mcp`	`uint16_t`	**MQTT custom Port** i.e. 1883
`mcu`	`String(16)`	**MQTT custom Username**
`mck`	`String(16)`	**MQTT custom key** For MQTT authentication

`mcc`	`uint8_t`	**MQTT custom connected** 0: not connected 1: connected

3. Commands

The following parameters can only be read:

```
version rbc rbt car err cbl pha tmp dws adi uby eto wst nrg fwv sse eca ecr
ecd ec4 ec5 ec6 ec7 ec8 ec9 ec1 rca rcr rcd rc4 rc5 rc6 rc7 rc8 rc9 rc1
```

The following parameters can be set:

```
amp ast alw stp dwo wss wke wen tof tds lbr aho afi ama al1 al2 al3 al4 al5
cid cch cfi lse ust wak r1x dto nmo rna rnm rne rn4 rn5 rn6 rn7 rn8 rn9 rn1
```

Set parameter

For all parameters that can be set, the format is for the command:

Method	Payload
`SET`	`[param]=[value]` *Example: amp=16* *Example: `wss=my home network`*

Path

Connection	Path
`WiFi Hotspot`	`http://192.168.4.1/mqtt?payload=`
`WiFi local network`	`http://x.x.x.x/mqtt?payload=`
`Cloud: MQTT`	Publish to topic : `iot-2/evt/pub/fmt/json` Payload: {"secret":"TOKEN","topic":"req","msg":MESSAGE}
`Cloud: REST Api`	`https://api.go-e.co/api?token=TOKEN&payload=MESSAGE`

4. Return Values

Local WiFi / Hotspot

Connection	Response
`WiFi Hotspot`	Complete status JSON object with already changed value
`WiFi local network`	Complete status JSON object with already changed value

For every status request and every command, the status JSON object is returned. An unsuccessful command can be recognized by the fact that the value in the status object has not changed.

Cloud: MQTT

Connection	Return-Topic
`Cloud: MQTT`	`iot-2/cmd/status/fmt/json`

There is no synchronous feedback on a publish to topic iot-2 / evt / pub / fmt / json. However, the charging box will try to publish the status object within one second.

Cloud: REST Api

Responses for **`/api`**

Condition	Response
`Token not specified`	`{"success":false,"error":"no token"}`
`Payload not specified`	`{"success":false,"error":"no payload"}`
`Token not found in database`	`{"success":false,"error":"wrong token"}`
`Rate limit exception`	`{"success":false,"error":"rate limiting"}`
`Success`	`{"success":true,"payload":original_payload}`

Return values for /`api_status`

Condition	Response
`Token not specified`	`{"success":false,"error":"no token"}`
`Token not found in database`	`{"success":false,"error":"wrong token"}`
`Rate limit exception`	`{"success":false,"error":"rate limiting"}`
`Status not available`	`{"success":false,"error":"other"}`
`Success`	`{"success":true,"age":AGE_IN_MILLISECONDS,"data":STATUS_OBJECT}`

Response time for /`api_status`

Condition	Response time
`Last Status <10 seconds old`	`~ 300 Millisekunden`

`Last Status >10 seconds old`	**If `wait=1`:**
	`~ 300 bis ~3500 milliseconds`
	
	If `wait=0`:
	`~ 300 milliseconds`
	
	Explanation: If `wait=1` (**default**) API Server sends ping to load box and waits up to 3 seconds for a new status object. If no new status arrives after 3 seconds, the last received status will be sent.

`Status nicht abrufbar`	`< 1000 milliseconds`

5. Cloud REST Api Workflow

Examples:

Action	Set charging current to 16A

URL	`https://api.go-e.co/api?payload=amp=16&token=__________`
Response	`{"success":true,"payload":"amp=16"}`

Action	**Deactivate charging**
URL	`https://api.go-e.co/api?payload=alw=0&token=__________`
Response	`{"success":true,"payload":"alw=0"}`

Action	**Activate charging**
URL	`https://api.go-e.co/api?payload=alw=1&token=__________`
Response	`{"success":true,"payload":"alw=1"}`

Action	**Get Status**
URL	`https://api.go-e.co/api_status?token=__________&wait=0`
Response (simplified)	`{"success":true,"age":1234,"data":{"version":"B",[...], "car":"1","amp":"16","err":"0",[...]}}`

6. Cloud MQTT Workflow

Connection

Server	`wss://i8p7v0.messaging.internetofthings.ibmcloud.com`
Username	`use-token-auth`
Password	**`MQTT_AUTH`**
Device-ID	`d:i8p7v0:app:`**`DEVICE_ID`**
Subscribe to:	`iot-2/cmd/status/fmt/json`

In order to receive the authentication data, please send a request to the go-e GmbH.

Actions

Action	**Activate subscription for charging box.** Subscription expires after 35 seconds and must be reactivated(recommended: 30 second interval)
Topic	`iot-2/evt/sub/fmt/json`
Payload	`{"secret":"TOKEN","apv":"CLIENT_VERSION"}` `CLIENT_VERSION`: A self-selected string that identifies the client

Action	**Send command**
Topic	`iot-2/evt/pub/fmt/json`

Payload	`{"secret":"TOKEN","topic":"req","msg":"CMD"}`
	
	Example:
	`{"secret":"TOKEN","topic":"req","msg":"amp=16"}`

Action	**Send ping.** The charging box sends the status object every 5 seconds as long as it is active. After 60 seconds without an incoming command, the status is no longer sent. If you want to wake up the

	charging box, or to query the status continuously, you have to ping before the expiration of the 60 seconds.
Topic	`iot-2/evt/pub/fmt/json`
Payload	`{"secret":"TOKEN","topic":"req","msg":"ping"}`

7. Custom MQTT Server

From firmware version 030 on it is possible to use a separate MQTT server in addition to the go-e cloud.

Commands are accepted via this topic:
`go-eCharger/000000/cmd/req`
Where 000000 must be replaced by the respective serial number.

The status object is output every 5 seconds via the following topic:
`go-eCharger/000000/status`

It is not necessary to activate the sending, the go-eCharger continuously sends data to/ status.