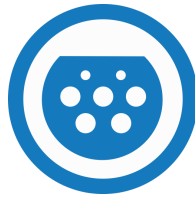


go-eCharger API Specification



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. Verbindung	2
Rate Limiting	2
2. API: status	4
Request Pfad	4
Rückgabeformat	4
Parameter	5
3. Kommandos	13
Parameter setzen	13
Pfad	13
4. Rückgabewerte	14
Lokales WLAN / Hotspot	14
Cloud: MQTT	14
Cloud: REST Api	14

5. Cloud REST Api Workflow	16
6. Cloud MQTT Workflow	17
Verbindung	17
Aktionen	17
7. Custom MQTT Server	19

1. Verbindung

Der go-eCharger bietet zwei WLAN Interfaces, von denen eines immer als mobiler Hotspot dient und ein weiteres das sich mit einem vorhandenen WLAN Netzwerk verbinden kann um eine Internetverbindung herzustellen.

Für die API werden folgende Verbindungen angeboten:

Verbindung	Pfad
WLAN Hotspot	http://192.168.4.1/
WLAN lokales Netzwerk	http://x.x.x.x/ Wobei die IP Adresse vom DHCP Server abgerufen wird
Cloud: MQTT	wss://i8p7v0.messaging.internetofthings.ibmcloud.com
Cloud: REST Api	https://api.go-e.co/

Authentifizierung:

Verbindung	Authentifizierung
WLAN Hotspot	Keine (Hotspot WPA key muss bekannt sein)
WLAN lokales Netzwerk	Keine (Gerät muss im gleichen WLAN sein und die HTTP Api muss mit der go-eCharger App aktiviert werden)
Cloud: MQTT	MQTT deviceId + token go-eCharger Cloud Token
Cloud: REST Api	go-eCharger Cloud Token

Rate Limiting

Verbindung	Limit
------------	-------

WLAN Hotspot	Keines (5 Sekunden Abstand empfohlen)
WLAN lokales Netzwerk	Keines (5 Sekunden Abstand empfohlen)
Cloud: MQTT	Fair-use Limit: 50MB pro Monat, ca. 25'000 Requests. Bei geplanter Überschreitung, bitte go-e GmbH kontaktieren!
Cloud: REST Api	<p>Hard Limit: 180 Requests pro 15 Minuten sliding window. (~ 5 Sekunden pro Request) und Quell-IP Adresse. Bei geplanter Überschreitung, bitte go-e GmbH kontaktieren!</p> <p>/api Fair use Limit: 50MB pro Monat, ca. 500'000 Requests. Bei geplanter Überschreitung, bitte go-e GmbH kontaktieren!</p> <p>/api_status Fair use Limit: 50MB pro Monat, ca. 25'000 Requests. Bei geplanter Überschreitung, bitte go-e GmbH kontaktieren!</p>

2. API: status

Liefert alle relevanten Parameter als JSON Objekt zurück.

Beispiel (unvollständig):

```
{ "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", "adi": "1", "uby": "0", "eto": "120", "wst": "3", "nrg": [2, 0, 0, 235, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], "fwv": "020-rc1", "sse": "000000", "wss": "goe", "wke": "", "wen": "1", "tof": "101", "tds": "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", "ust": "0", "wak": "", "r1x": "2", "dto": "0", "nmo": "0", "eca": "0", "ecr": "0", "ecd": "0", "ec4": "0", "ec5": "0", "ec6": "0", "ec7": "0", "ec8": "0", "ec9": "0", "ec1": "0", "rca": "", "rcr": "", "rcd": "", "rc4": "", "rc5": "", "rc6": "", "rc7": "", "rc8": "", "rc9": "", "rc1": "", "rna": "", "rnm": "", "rne": "", "rn4": "", "rn5": "", "rn6": "", "rn7": "", "rn8": "", "rn9": "", "rn1": "" }
```

Request Pfad

Verbindung	Pfad
WLAN Hotspot	http://192.168.4.1/status
WLAN lokales Netzwerk	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] Der <code>wait</code> Parameter ist optional

Rückgabeformat

Verbindung	Pfad
WLAN Hotspot	Plain STATUS_OBJECT
WLAN lokales Netzwerk	Plain STATUS_OBJECT
Cloud: MQTT	Plain STATUS_OBJECT
Cloud: REST Api	<code>{"success":true,"age":AGE_IN_MILLISECONDS,"data":STATUS_OBJECT}</code>

Parameter

Zusätzlich zu diesen Parametern können auch ohne vorherige Ankündigung und je nach Verbindungsart weitere Parameter dazu kommen.

Erklärung Format: alle Parameter werden im JSON Objekt als String gesendet (in Anführungszeichen). Die meisten dieser Parameter können in ein integer Format konvertiert werden. Der bei Format angegebene Datentyp zeigt die zu erwartende Größe. Sollte der String nicht in den angegebenen Datentyp konvertiert werden, soll ein Kommunikationsfehler angezeigt werden.

Parameter	Format	Erklärung
version	String (1)	JSON Format. "B": Normalfall "C": Wenn Ende-zu-Ende Verschlüsselung aktiviert
rbc	uint32_t	reboot_counter: Zählt die Anzahl der Bootvorgänge. Wird mit der Ende-zu-Ende Verschlüsselung als Schutz gegen Replay Attacken gesendet.
rbt	uint32_t	reboot_timer: Zählt die Millisekunden seit dem letzten Bootvorgang. Wird mit der Ende-zu-Ende Verschlüsselung als Schutz gegen Replay Attacken gesendet. Läuft nach 49 Tage über und erhöht dabei den reboot_counter.
car	uint8_t	Status PWM Signalisierung 1: Ladestation bereit, kein Fahrzeug 2: Fahrzeug lädt 3: Warte auf Fahrzeug 4: Ladung beendet, Fahrzeug noch verbunden
amp	uint8_t	Ampere Wert für die PWM Signalisierung in ganzen Ampere von 6-32A
err	uint8_t	error: 1: RCCB (Fehlerstromschutzschalter) 3: PHASE (Phasenstörung) 8: NO_GROUND (Erdungserkennung) 10, default: INTERNAL (sonstiges)
ast	uint8_t	access_state: Zugangskontrolle. 0: Offen 1: RFID / App benötigt

		2: Strompreis / automatisch
alw	uint8_t	allow_charging: PWM Signal darf anliegen 0: nein 1: ja
stp	uint8_t	stop_state: Automatische Abschaltung 0: deaktiviert 2: nach kWh abschalten
cbl	uint8_t	Typ2 Kabel Ampere codierung 13-32: Ampere Codierung 0: kein Kabel
pha	uint8_t	Phasen vor und nach dem Schütz binary flags: 0b00ABCDEF A... phase 3, vor dem Schütz B... phase 2 vor dem Schütz C... phase 1 vor dem Schütz D... phase 3 nach dem Schütz E... phase 2 nach dem Schütz F... phase 1 nach dem Schütz pha 0b00001000: Phase 1 ist vorhanden pha 0b00111000: Phase1-3 ist vorhanden
tmp	uint8_t	Temperatur des Controllers in °C
dws	uint32_t	Geladene Energiemenge in Deka-Watt-Sekunden <i>Beispiel: 100'000 bedeutet, 1'000'000 Ws (=277Wh = 0,277kWh) wurden in diesem Ladevorgang geladen.</i>
dwo	uint16_t	Abschaltwert in 0.1kWh wenn stp==2 , für dws Parameter <i>Beispiel: 105 für 10,5kWh</i> Ladebox-Logik: <code>if(dwo!=0 && dws/36000>=dwo)alw=0</code>
adi	uint8_t	adapter_in: Ladebox ist mit Adapter angesteckt 0: NO_ADAPTER 1: 16A_ADAPTER
uby	uint8_t	unlocked_by: Nummer der RFID Karte, die den jetzigen Ladevorgang freigeschalten hat
eto	uint32_t	energy_total: Gesamt geladene Energiemenge in 0.1kWh

		<i>Beispiel: 130 bedeutet 13kWh geladen</i>
wst	uint8_t	wifi_state: WLAN Verbindungsstatus 3: verbunden default: nicht verbunden
nrg	array[15]	Array mit Werten des Strom- und Spannungssensors nrg[0]: Spannung auf L1 in Volt nrg[1]: Spannung auf L2 in Volt nrg[2]: Spannung auf L3 in Volt nrg[3]: Spannung auf N in Volt nrg[4]: Ampere auf L1 in 0.1A (<i>123 entspricht 12,3A</i>) nrg[5]: Ampere auf L2 in 0.1A nrg[6]: Ampere auf L3 in 0.1A nrg[7]: Leistung auf L1 in 0.1kW (<i>36 entspricht 3,6kW</i>) nrg[8]: Leistung auf L2 in 0.1kW nrg[9]: Leistung auf L3 in 0.1kW nrg[10]: Leistung auf N in 0.1kW nrg[11]: Leistung gesamt in 0.01kW (<i>360 entspricht 3,6kW</i>) nrg[12]: Leistungsfaktor auf L1 in % nrg[13]: Leistungsfaktor auf L2 in % nrg[14]: Leistungsfaktor auf L3 in % nrg[15]: Leistungsfaktor auf N in % App Logik:: 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 <i>Beispiel: "020-rc1"</i>
sse	String	Seriennummer als %06d formatierte Zahl <i>Beispiel: "000001"</i>
wss	String	WLAN SSID <i>Beispiel: "Mein Heimnetzwerk"</i>
wke	String	WLAN Key <i>Beispiel: "*****" für fwv ab 020</i>

		<i>Beispiel: "passwort" für fww vor 020</i>
wen	uint8_t	wifi_enabled: WLAN aktiviert 0: deaktiviert 1: aktiviert
tof	uint8_t	time_offset: Zeitzone in Stunden für interne batteriegestützte Uhr +100 <i>Beispiel: 101 entspricht GMT+1</i>
tds	uint8_t	Daylight saving time offset (Sommerzeit) in Stunden <i>Beispiel: 1 für Mitteleuropa</i>
lbr	uint8_t	LED Helligkeit von 0-255 0: LED aus 255: LED Helligkeit maximal
aho	uint8_t	Minimale Anzahl von Stunden in der mit "Strompreis - automatisch" geladen werden muss <i>Beispiel: 2 ("Auto ist nach 2 Stunden voll genug")</i>
afi	uint8_t	Stunde (Uhrzeit) in der mit "Strompreis - automatisch" die Ladung mindestens aho Stunden gedauert haben muss. <i>Beispiel: 7 ("Fertig bis 7:00, also davor mindestens 2 Stunden geladen")</i>
azo	uint8_t	Awattar Preiszone 0: Österreich 1: Deutschland
ama	uint8_t	Absolute max. Ampere: Maximalwert für Ampere Einstellung <i>Beispiel: 20 (Einstellung auf mehr als 20A in der App nicht möglich)</i>
a11	uint8_t	Ampere Level 1 für Druckknopf am Gerät. 6-32: Ampere Stufe aktiviert 0: Stufe deaktiviert (wird übersprungen)
a12	uint8_t	Ampere Level 2 für Druckknopf am Gerät. Muss entweder 0 oder > a11 sein
a13	uint8_t	Ampere Level 3 für Druckknopf am Gerät. Muss entweder 0 oder > a12 sein
a14	uint8_t	Ampere Level 4 für Druckknopf am Gerät. Muss entweder 0 oder > a13 sein

a15	uint8_t	Ampere Level 5 für Druckknopf am Gerät. Muss entweder 0 oder > a14 sein
cid	uint24_t	Color idle: Farbwert für Standby (kein Auto angesteckt) als Zahl <i>Beispiel: parseInt("#00FFFF"): 65535 (blau/grün, Standard)</i>
cch	uint24_t	Color charging: Farbwert für Ladevorgang aktiv , als Zahl <i>Beispiel: parseInt("#0000FF"):255 (blau, Standard)</i>
cfi	uint24_t	Color idle: Farbwert für Ladevorgang abgeschlossen , als Zahl <i>Beispiel: parseInt("#00FF00"): 65280(grün, Standard)</i>
lse	uint8_t	led_save_energy : LED automatisch nach 10 Sekunden abschalten 0: <i>Energiesparfunktion deaktiviert</i> 1: <i>Energiesparfunktion aktiviert</i>
ust	uint8_t	unlock_state : Kabelverriegelung Einstellung 0: Verriegeln solange Auto angesteckt 1: Nach Ladevorgang automatisch entriegeln 2: Kabel immer verriegelt lassen
wak	String	WLAN Hotspot Password <i>Beispiel: "abcdef0123456"</i>
r1x	uint8_t	Flags 0b1: HTTP Api im WLAN Netzwerk aktiviert (0: nein, 1:ja) 0b10: Ende-zu-Ende Verschlüsselung aktiviert (0: nein, 1:ja)
dto	uint8_t	Restzeit in Millisekunden verbleibend auf Aktivierung durch Strompreise App-logik: if(json.car==1)message = "Zuerst Auto anstecken" else message = "Restzeit: ..."
nmo	uint8_t	Norwegen-Modus aktiviert 0: deaktiviert (Erdungserkennung aktiviert) 1: aktiviert (keine Erdungserkennung, nur für IT-Netze gedacht)
eca ecr ecd ec4 ec5 ec6	uint32_t	Geladene Energiemenge pro RFID Karte von 1-10 <i>Beispiel: eca==1400: 140kWh auf Karte 1 geladen</i> <i>Beispiel: ec7==1400: 140kWh auf Karte 7 geladen</i> <i>Beispiel: ec1==1400: 140kWh auf Karte 10 geladen</i>

ec7 ec8 ec9 ec1		
rca rcr rcd rc4 rc5 rc6 rc7 rc8 rc9 rc1	String	RFID Karte ID von 1-10 als String Format und Länge: variabel, je nach Version
rna rnm rne rn4 rn5 rn6 rn7 rn8 rn9 rn1	String	RFID Karte Name von 1-10 Maximallänge: 10 Zeichen
tme	String	Aktuelle Uhrzeit , formatiert als ddmmyyhhmm 0104191236 entspricht 01.04.2019 12:36
sch	String	Scheduler einstellungen (base64 encodiert) Funktionen zum encodieren und decodieren gibt es hier: https://gist.github.com/peterpoetzi/6cd2fad2a915a2498776912c5aa137a8 Die Einstellungen können so gesetzt werden: r21 =Math.floor(encode(1)) r31 =Math.floor(encode(2)) r41 =Math.floor(encode(3)) <i>Ein direktes Setzen von sch= wird nicht unterstützt</i>
sdp	uint8_t	Scheduler double press: Aktiviert Ladung nach doppeltem Drücken des Button, wenn die Ladung gerade durch den Scheduler unterbrochen wurde

		0: Funktion deaktiviert 1: Ladung sofort erlauben
upd	uint8_t	Update available (nur verfügbar bei Verbindung über go-e Server) 0: kein Update verfügbar 1: Update verfügbar
cdi	uint8_t	Cloud disabled 0: cloud enabled 1: cloud disabled
loe	uint8_t	Lastmanagement enabled 0: Lastmanagement deaktiviert 1: Lastmanagement über Cloud aktiviert
lot	uint8_t	Lastmanagement Gruppe Total Ampere
lom	uint8_t	Lastmanagement minimale Amperezahl
lop	uint8_t	Lastmanagement Priorität
log	String	Lastmanagement Gruppen ID
lon	uint8_t	Lastmanagement: erwartete Anzahl von Ladestationen (derzeit nicht unterstützt)
lof	uint8_t	Lastmanagement Fallback Amperezahl
loa	uint8_t	Lastmanagement Ampere (derzeitiger erlaubter Ladestrom) wird vom Lastmanagement automatisch gesteuert
lch	uint32_t	Lastmanagement: Sekunden seit letzten Stromfluss bei noch angestecktem Auto Bzw. 0 wenn Ladevorgang gerade läuft
mce	uint8_t	MQTT custom enabled Verbindung mit eigenen MQTT Server herstellen 0: Funktion deaktiviert 1: Funktion aktiviert
mcs	String(63)	MQTT custom Server Hostname ohne Protokollangabe (z.B. test.mosquitto.org)
mcp	uint16_t	MQTT custom Port z.B. 1883

mcu	String(16)	MQTT custom Username
mck	String(16)	MQTT custom key Für MQTT Authentifizierung
mcc	uint8_t	MQTT custom connected 0: nicht verbunden 1: verbunden

3. Kommandos

Folgende Parameter können nur gelesen werden:

```
version rbc rbt car err cb1 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
```

Folgende Parameter können gesetzt werden:

```
amp ast alw stp dwo wss wke wen tof tds lbr aho afi ama a11 a12 a13 a14 a15  
cid cch cfi lse ust wak r1x dto nmo rna rnm rne rn4 rn5 rn6 rn7 rn8 rn9 rn1
```

Parameter setzen

Bei allen Parametern, die gesetzt werden können, ist das format für das Kommando:

Method	Payload
SET	[param]=[value] <i>Beispiel: amp=16</i> <i>Beispiel: wss=mein heimnetzwerk</i>

Pfad

Verbindung	Pfad
WLAN Hotspot	http://192.168.4.1/mqtt?payload=
WLAN lokales Netzwerk	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. Rückgabewerte

Lokales WLAN / Hotspot

Verbindung	Rückgabe
WLAN Hotspot	Komplettes status JSON Objekt mit bereits geänderten Wert
WLAN lokales Netzwerk	Komplettes status JSON Objekt mit bereits geänderten Wert

Bei jedem Status Request und jedem Kommando wird das Status JSON-Objekt zurückgegeben. Ein nicht erfolgreiches Kommando erkennt man daran dass sich der Wert im Status Objekt nicht geändert hat.

Cloud: MQTT

Verbindung	Rückgabe-Topic
Cloud: MQTT	iot-2/cmd/status/fmt/json

Es gibt keine synchrone Rückmeldung auf ein publish to topic iot-2/evt/pub/fmt/json. Die Ladebox wird jedoch versuchen innerhalb einer Sekunde das Status-Objekt zu publishen.

Cloud: REST Api

Rückgabewerte für /api

Bedingung	Rückgabe
Token nicht angegeben	<code>{"success":false,"error":"no token"}</code>
Payload nicht angegeben	<code>{"success":false,"error":"no payload"}</code>
Token nicht in Datenbank gefunden	<code>{"success":false,"error":"wrong token"}</code>
Rate limit exception	<code>{"success":false,"error":"rate limiting"}</code>

Success	<code>{"success":true,"payload":original_payload}</code>
----------------	--

Rückgabewerte für /api_status

Bedingung	Rückgabe
Token nicht angegeben	<code>{"success":false,"error":"no token"}</code>
Token nicht in Datenbank gefunden	<code>{"success":false,"error":"wrong token"}</code>
Rate limit exception	<code>{"success":false,"error":"rate limiting"}</code>
Status nicht abrufbar	<code>{"success":false,"error":"other"}</code>
Success	<code>{"success":true,"age":AGE_IN_MILLISECONDS,"data":STATUS_OBJECT}</code>

Antwortzeit für /api_status

Bedingung	Antwortzeit
Letzter Status <10 Sekunden alt	~ 300 Millisekunden
Letzter Status >10 Sekunden alt	<p>Wenn wait=1: ~ 300 bis ~3500 Millisekunden</p> <p>Wenn wait=0: ~ 300 Millisekunden</p> <p>Erklärung: Wenn wait=1 (default) API Server sendet Ping an Ladebox und wartet bis zu 3 Sekunden auf ein neues Status Objekt. Falls nach 3 Sekunden kein neuer Status kommt, wird der zuletzt empfangene Status gesendet.</p>
Status nicht abrufbar	< 1000 Millisekunden

5. Cloud REST Api Workflow

Beispiele:

Aktion	Ladestrom auf 16A stellen
URL	https://api.go-e.co/api?payload=amp=16&token=_____
Rückgabe	<pre>{"success":true,"payload":"amp=16"}</pre>

Aktion	Ladung deaktivieren
URL	https://api.go-e.co/api?payload=alw=0&token=_____
Rückgabe	<pre>{"success":true,"payload":"alw=0"}</pre>

Aktion	Ladung aktivieren
URL	https://api.go-e.co/api?payload=alw=1&token=_____
Rückgabe	<pre>{"success":true,"payload":"alw=1"}</pre>

Aktion	Status abfragen
URL	https://api.go-e.co/api_status?token=_____&wait=0
Rückgabe (gekürzt)	<pre>{"success":true,"age":1234,"data":{"version":"B",[...], "car":"1","amp":"16","err":"0",[...]}}</pre>

6. Cloud MQTT Workflow

Verbindung

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

Um die Authentifizierungsdaten zu erhalten, fragen stellen Sie bitte eine Anfrage an die go-e GmbH.

Aktionen

Aktion	Subscription für Ladebox aktivieren. Subscription verfällt nach 35 Sekunden und muss davor jeweils neu gesendet werden (empfohlen: 30 Sekunden Intervall)
Topic	iot-2/evt/sub/fmt/json
Payload	<pre>{"secret":"TOKEN","apv":"CLIENT_VERSION"}</pre> CLIENT_VERSION: Ein selbst gewählter String der den Client identifiziert

Aktion	Kommando senden
Topic	iot-2/evt/pub/fmt/json
Payload	<pre>{"secret":"TOKEN","topic":"req","msg":"CMD"}</pre> Beispiel: <pre>{"secret":"TOKEN","topic":"req","msg":"amp=16"}</pre>

Aktion	Ping senden. Die Ladebox sendet solange sie aktiv ist alle 5 Sekunden das Status-Objekt. Nach 60 Sekunden ohne eingehendes
--------	---

	Kommando wird der Status nicht mehr gesendet. Wenn man die Ladebox aufwecken will, oder den Status kontinuierlich abfragen will, muss man vor dem Ablauf der 60 Sekunden einen Ping senden.
Topic	iot-2/evt/pub/fmt/json
Payload	<code>{"secret":"TOKEN","topic":"req","msg":"ping"}</code>

7. Custom MQTT Server

Ab Firmware Version 030 ist es möglich einen eigenen MQTT Server zusätzlich zur go-e Cloud zu verwenden.

Kommandos werden über dieses Topic entgegengenommen:

go-eCharger/000000/cmd/req

Wobei 000000 durch die jeweilige Seriennummer ersetzt werden muss.

Das Status Objekt wird alle 5 Sekunden über folgendes Topic ausgegeben:

go-eCharger/000000/status

Es ist nicht notwendig das Senden speziell zu aktivieren, der go-eCharger sendet durchgehend Daten auf /status.