Charge Control C User Guide

chargebyte GmbH

2024-01-08 14:21:33

Protection Class: **public**

Contents

1 Revision

Revision	Release Date	Valid from Software Version	Changes
8	January 8, 2024	3.5.0	added RCD test feature in sections MQTT, configuration and OCPP
		3.3.0	added a section for customer.json recovery under "Programming"
			OCPP configuration key AuthorizeRemoteTxRequests is now switchable, the description was adapted accordingly
			phase count switching: synchronized minimum allowed switch delay to hardware specification
			extended section about limitations of OCPP implementation
			corrected wire size specification for connectors in chapter 8.17
7	August 29, 2023	3.2.0	mentioned support of exFAT filesystem
			added hint about meter install direction regarding consumption/import and generation/export
			added non-functional RS-485 MCU as error cause to OCPP status notifications
			reworked fake DC HLC session section and updated interfaces/configuration elements
		2.2.0	added new OCPP feature: transmission of the public key of Eichrecht meters via DataTransfer and new configuration keys
			added slightly more detailed description of the OCPP HeartbeatInterval since the implementation now accepts the value of zero from the backend
			added mention/explanation of timestamps in OCPP StatusNotification
6	March 3, 2023	2.1.0	added new section "OCPP Measurand Support" and updated related sections
			fixed OCPP parameter defaults regarding new measurands
			refined description of rcd_monitor/polarity
			refined section current limits for basic AC charging, in particular TOPIC_GLOBAL_DYN_CURRENT_LIMIT
			added OCPP section "FTP and Path Names"
			fixed formatting in the table of peripheral device defaults
			added new column to OCPP configuration which explains when changes apply
			added clarification that customer installed root CA certificates are kept/migrated during firmware update, added warning for older firmwares
			add retain column to MQTT topics of "Fake high level DC charging mode"
			mentioned support and added schematics for Scame 200.23261BS, Scame 200.23261BP, Scame 200.23261BL
		1.1.6	improved description of customer.json key "ports[0]/user_authentication key"
			added behavior of Soft Reset to OCPP limitations
			added new section "Fake high level DC charging mode"
			added proximity errors to OCPP StatusNotification

Revision	Release Date	Valid from Software Version	Changes
			added support for ChargingProfilePurpose TxProfile and adjusted MaxChargingProfilesInstalled
			added persistence errors to OCPP StatusNotification
			extended OCPP TriggerMessage implementation
	September 21, 2022	1.1.0	added a section for board customization with USB under the section "Programming"
			added a section for firmware update customization and signing under the section "Programming"
			added new OCPP configuration option ftpTryTLSUpgrade
			added support for Socomec Countis E03/04 meters
			meter implementation is now disabled by default
	August 24, 2022	1.0.9	documented new option "basic+fake_highlevel_dc" for configuration parameter "charging_type"
			added new OCPP configuration option calibrationLawFormat
	August 16, 2022	1.0.8	adjusted directory name in section "Update via SSH and SFTP" and added hint to remove the update file after installation
			added support for Elecnova DDS1946-2P/2M and DTS1946-4P/4M meters
			added feedback type configuration for ventilation and added "none" option for contactor and ventilation
			updated URI of the Board Support Package Github repository to the chargebyte address
	July 19, 2022	1.0.7	OCPP: documented new TriggerMessage implementations
			mentioned Yocto SDK in chapter Programming, refined section Advices/Requirements for Customer Applications (EIA-485) and added programming example
			added support for Eastron SDM230 meters
			added support for Carlo Gavazzi EM300/ET300 and EM100/ET100 series compatible meters
			changed value of ports[0]/pp/cable_current_limit configuration
	May 20, 2022	1.0.6	documented new MQTT topic TOPIC_GLOBAL_TIME_SET_STATUS
			OCPP: documented new PersistenceError
			OCPP: mentioned current limitation about StatusNotification with status SuspendedEV/SuspendedEVSE
			OCPP: removed 32 bit limitation of transaction IDs
			OCPP: added chapter about local data storage
			fixed customer.json parameter for Phoenix Contact EEM-350-D-MCB
			added support for BZR Bauer/SunSpec compatible meters
			added new OCPP configuration options bootNotificationOnReconnect and StopTransactionSignatureFormat
6	February 15, 2022	1.0.4	added note regarding the maximum size of json values in the customer.json configuration
			added phase count switching chapter

Revision	Release Date	Valid from Software Version	Changes
			added SCAME-200.23260BS to possible locking motors
			updated TOPIC_RFID_AUTHORIZEREQ to TOPIC_RFID_AUTHORIZATION_REQUEST
			added board pin description/clarify signal names for onboard relays
			updated Figures to latest HW revision: Mechanical drawing of Charge Control C Switches on Charge Control CConnectors of Charge Control C
			added support for Iskra WM3M4 & WM3M4C meters
			added new OCPP configuration options allowTxWithInvalidTime and useAsTimeSource
5	November 29, 2021	0.9.13	corrected dimensions of the mechanical drawing
			added a table of supported peripherals in the USB section and added a restriction note for the Huawei E3372 internet dongle
			table Configuration Software Parameter customer.json: new configuration parameters "charging_type", "hlc_protocols"
			added backend qualification against Vector vCharM
			added column for software version information into revision table
			changed order code information
			added section for casync firmware updates and clarified related points
			added definition port[]/failsafe_current_limit of the customer.json configuration
			added section for Charge Point initiated generic DataTransfer
			added explanation for MQTT topic TOPIC_EVSE_BASIC_OFFERED_CURRENT_LIMIT in section Current limits for basic AC charging
			added topic TOPIC_EVSE_BASIC_PHYSICAL_CURRENT_LIMIT
			extended status notification with error cause, impact and how to solve
			added additional parameters for RFID Modbus support
			added support for SMART Technologies ID MCR Legic RFID reader
			clarified Geya recloser model type
			added support for Eastron SDM630 meter
			added support for ABB EV3 meter
			restructured EIA-485 section (i.e. added sections about default Modbus communication parameters and supported electricity meter measurands etc.)
			added support for client-side WebSocket ping
			added hint that dynamic current limits should be published as 'retained'
			changed topics TOPIC_EVSE_BASIC_MAXCURRENTLIMIT, TOPIC_GLOBAL_DYN_CURRENT_LIMIT, TOPIC_EVSE_BASIC_OFFERED_CURRENT_LIMIT, TOPIC_CP_DUTY_CYCLE and TOPIC_CHARGE_PWM_STATUS to floating point representation

Revision	Release Date	Valid from Software Version	Changes
4	January 27, 2021		added use cases for the ventilation control
			reworked charge port explanation in section "MQTT and Mosquitto Documentation" and added "port0/" prefix in the MQTT topic definition section
			updated GetDiagnostics feature and limitation description with newly supported upload protocols
			updated table "CP State Information" and added SHRT_MIN explanation in section "Charge status information" regarding the initialization of the QCA7000
			added ChangeAvailability behavior to OCPP limitations
			added definition of a customer.json config parameter to configure the ventilation control mode and described MQTT topics of the ventilation control
			extended meter error behavior
			added hint regarding alternative mating connector suppliers and different "coding noses"
			added hint where customer specific network configuration files have to be placed
			listed new Phoenix Contact device as supported electrical meter
			added section on additional customer availability
3	September 22, 2020		added additional note for the OCPP ReaderFailure in section OCPP StatusNotification
			added topic definition TOPIC_OCPP_ONLINE in section MQTT Topics
			added Huawei E3372 to supported USB dongle list
			added explanation of the general behavior of the board during firmware update in section Firmware Upgrade
			added explanation for MQTT topic TOPIC_GLOBAL_VERSION_CHARGING_SOFTWARE in section MQTT Topics
			added explanation for MQTT topic TOPIC_GLOBAL_DYN_CURRENT_LIMIT in section Current limits for basic AC charging
			added note that RFID is now optional
			added limitations of SetChargingProfile
			added section RFID authorization
			added section sharing RFID reader (incl. relevant config parameter)
			added section Partitioning to Firmware Upgrade
			updated mains powerline fallback network configuration description
2	May 12, 2020		refined wording in section X8
			added section MQTT Service Discovery
			set hint to export gpios to use in userspace
			table Configuration Parameter leds.json: new LED behavior conditions “failure” and “deactivated”
			table Configuration Hardware Parameter customer.json: new plug lock type “INTRAMCO-603205”
			table Configuration Software Parameter customer.json: new configuration parameter

Revision	Release Date	Valid from Software Version	Changes
			“digital_input_threshold_voltage”, “rfid_stop_transaction”, “force_wake_up”, “wake_up_after_timeout”, “emergency_alarm/polarity”, “user_authentication”; adjusted configuration parameter “evse_id”
			section OCPP Features: added server side WebSocket Ping Pong and FirmwareManagement support
			added section OCPP configuration
			section Basic SECC configuration: removed basic SECC configuration table
			added section which describes software part for the relays, locking motors and the 4 wire PWM fan
			added pairing instructions for mains PLC
			section Digital Input & Output: clarified behavior list belongs to status LED
			reworked SW development sections
			restructured network related stuff to fit Yocto based image
			added instructions how to use USB internet dongle
			removed currently not implemented MQTT topics
			added instructions to export GPIO
			added locking motor logic information
			added mounting section
			added section Mating Connectors
			mentioned new support for more electricity meters
			mentioned new support for RCD recloser
			added section current limits
			mentioned has.to.be certification
			added timings section
			added data transfer section
			listed imitations of OCPP offline operation
			added RFID information
			removed note about RCD & authorization key
1	February 20, 2019		initial release

Thank you very much for your trust. We are happy that you have chosen our Charge Control platform to operate your eMobility charging solution. This User Guide will help you to understand all features of our product and configure them properly to fit your and your customer’s requirements best.

2 Safety Notes

IMPORTANT: Read the following safety instruction carefully and clearly prior to the assembly and use of the device. Please keep these safety instructions for future reference.

- ⚠The installation and assembly may only be carried out by a qualified electrician!
- ⚠This device, which is supplied with mains power, has to be secured by means of a max. B6A circuit breaker. In case of a multi-phase connection, such a circuit breaker has to be provided for each connected outer conductor. These circuit breakers are to be installed directly next to each other.
- **WARNING!** This device is connected to mains power and hazardous voltages which are not covered. Hazardous voltages must be covered inside the charging station to prevent electrical shocks.
- **Attention!** Make sure that the device is not exposed to heat sources which may lead to overheating. Charge Control C can be damaged in case of overheating.
- **Attention!** The device may only be connected in the range of overvoltage category 3 or lower. Operating Charge Control C in a higher category can damage the device.
- **Attention!** Ensure adequate ventilation at the site of installation. Charge Control C can be damaged in case of overheating.
- **Attention!** Do not operate the device in supply networks which do not comply with the specifications on the type plate. Operating Charge Control C in networks not compatible with the specifications on the type plate can damage the device.
- **Attention!** The device may only be installed in dry areas. Exposing Charge Control C to wetness can damage the device.
- This device is designed for installation on DIN rails which provide fire protection as per DIN EN 60950-1.

3 Device Overview

3.1 Product Features

- future-proof technology: ARM Cortex-A7 @ 800 MHz, DDR3, eMMC
- up to 6x digital general-purpose inputs
- up to 6x digital general-purpose outputs
- 4-Wire pulse width modulation (PWM) fan interface
- 10/100 Mbit/s Ethernet
- USB
- rotary switch coded maximum charging current
- debug LEDs
- up to 2x EIA-485
- CAN
- 2x Motor Driver
- 1-Wire Interface
- mains switching relays with sense feedback (only 1 Charging socket supported)
- ISO 15118 compliant control and proximity pilot interface
- HomePlug Green PHY™ on mains
- HomePlug Green PHY™ on control pilot
- filtered mains output
- OCPP 1.6J
- MQTT

Availability of the interfaces depends on the actual variant - see the product datasheet for more details.

3.2 Product Description

Charge Control C is an IEC 61851 and ISO 15118 (CCS) compliant charging controller, born to beat in every kind of electric AC vehicle charging station. It is capable of the ISO 15118 control and proximity pilot signals as well as the PWM charging signal, conform to IEC 61851. It can directly control actuators like locking motors, contactors and ventilation. With its amount of general-purpose IO, it can be connected to a variety of periphery. It comes with the standard interfaces to be connected to electric meters, RFID devices and different kinds of actuators and sensors. Charge Control C is currently available in three different hardware variants (Charge Control C 100, 200 and 300) that are suitable for different complexities of charging stations.

3.3 OCPP features

Charge Control C supports OCPP according to the OCPP 1.6J Specification (JSON over WebSocket) with server and client side WebSocket Ping Pong. Currently to use all OCPP features a supported RFID reader and a meter must be connected.

From OCPP's Smart Charging profile the Charge Control C implements the commands `ClearChargingProfile` and `SetChargingProfile` only for `ChargePointMaxProfile & TxProfile`. In order to reduce eMMC wearout the `ChargePointMaxProfile` will be stored persistently after some delay. `TxProfile` will be stored in RAM.

In case a running transaction is interrupted (e.g. by a power loss on the charging station), this transaction will be terminated after reconnecting to the Central System. For the rare case that the meter has a malfunction in this situation, the charge point will use a saved meter reading which is at least from the start of the transaction.

The OCPP command `GetDiagnostics` currently supports the following upload protocols: FTP, FTPS, HTTP, HTTPS (using method POST).

Supported OCPP messages:

Charge point initiated
Authorize
BootNotification
DiagnosticStatusNotification
FirmwareStatusNotififcation
HeartBeat
MeterValues
StartTransaction
StatusNotification
StopTransaction
Central system initiated
ChangeAvailability
ChangeConfiguration
ClearCache
ClearChargingProfile
DataTransfer
GetConfiguration
GetDiagnostics
GetLocalListVersion
RemoteStartTransaction
RemoteStopTransaction
Reset
SendLocalList
SetChargingProfile
UnlockConnector
UpdateFirmware

Table 1 Supported OCPP messages

3.3.1 Limitations of the OCPP implementation

Currently the following limitations apply:

- Currently for `SendLocalList` only the updateType `Full` is implemented.
- Only a limited set of measurands listed in the OCPP 1.6 standard is supported. See section "OCPP Measurand Support" for details.
- Only one connector is supported currently.
- The `parentId` handling is not supported yet.
- Empty idTags are not supported and will be rejected.
- Charging profiles in a `RemoteStartTransaction` are ignored.
- `SetChargingProfile` only supports two charging profiles with the chargingRateUnit = `A` , chargingProfileKind = `Relative` and chargingProfilePurpose = `ChargePointMaxProfile` or `TxProfile`
- `SetChargingProfile` does not support charging schedules or stacking.
- `SetChargingProfile` does not support chargingProfilePurpose = `TxDefaultProfile`

- `ChangeAvailability` schedules the change to Inoperative in case an EV is present on the connector (instead of a transaction is running), but any attempt to start a new transaction will be rejected.
- Parameters `startTime` and `stopTime` in `GetDiagnostics` are ignored.
- HTTP redirection during WebSocket handshake is not supported.
- `StatusNotification`s currently report charge point status `SuspendedEV` and `SuspendedEVSE` as `Charging`.
- `TriggerMessage` is implemented for `BootNotification`, `Heartbeat` and `StatusNotification`, yet not for the other MessageTrigger types. Therefore, this feature profile is not reported via `SupportedFeatureProfiles`.
- `Reset` of type `Soft` behaves like a `Hard Reset`.
- In case a online transaction is rejected as `Invalid` by the backend, then there no `Finishing StatusNotification` will be sent.
- There is no guarantee that the application works beyond year 2038.
- Since the ISO 8601 standard is quite comprehensive, the implementation focuses on the most commonly used representation and only accepts timestamps of the following format: `YYYY-MM-DDTHH:mm:ss.SSSZ` (everything beyond the dot is optional)
- The `transactionId` provided by the CSMS via `StartTransaction` must always be unique (even the id tag is not accepted)

3.3.2 Limitations of the OCPP offline operation

- If a reboot occurs while offline, the intermediate meter values which were measured while offline are not retained. Only the start and end readings are forwarded to the Central System when back online.
- The Charge Point cannot charge offline until it has registered successfully with its configured Central System URI at least once.
- If the Charge Control C does not have a valid system time, offline charging is possible, but the offline transactions are not forwarded to the Central System when back online, due to the invalidity of their timestamps. This also applies to meter values acquired during that time.
- A firmware update started while offline will be delayed until a connection to the Central System is re-established.
- No StatusNotification about already resolved fault states while offline will be forwarded to the Central System when back online.
- Locally authorized Transaction persistence is limited to 336 entries.

3.3.3 OCPP connection behavior

OCPP 1.6J uses WebSockets (over HTTP/HTTPS) to connect client and server. These provide persistent bi-directional connections.

Should the connection fail or get disconnected, the Charge Point tries to reconnect to the Central System. In case the Central System URI has been changed either through configuration parameter or MQTT topic, the local Transaction database will be erased. Previously handled offline Transactions will not be recoverable.

To observe the state of idle connections, the Charge Point supports the use of WebSocket pings. In addition to the OCPP Heartbeat message, pings and their corresponding pong answers can ensure that an existing connection is stable. If enabled, the Charge Point sends WebSocket pings in the given intervals. Should a corresponding pong in reply to the ping not arrive within the configured timeout, the Charge Point goes into offline mode and closes the WebSocket connection, before trying to reconnect.

The ping interval and timeout can be configured using customer.json as shown below, and via ChangeConfiguration. By default, WebSocket ping is disabled (interval '0'). Note that WebSocket ping and pong impose additional network traffic on the communication channel to the Central System, so the interval should be chosen wisely. Also note that a too small timeout when using high-latency connections may lead to premature disconnects.

Dedicated configuration in customer.json:

Parameter	Description	Type	Default
ocpp/WebSocketPingInterval	The interval, in seconds, between WebSocket pings. The client ping feature is disabled if this key is not available or is set to '0'.	Integer (non-negative)	0
ocpp/webSocketPingTimeout	The timeout, in seconds, after which the connection is considered offline if a WebSocket pong in response to a ping is not received.	Integer (positive)	120

Table 2 WebSocket ping configuration in customer.json

3.3.4 OCPP StatusNotification

In order to limit the rate of StatusNotifications, there is a minimum delay of 10 seconds between two StatusNotifications. StatusNotifications will include a timestamp in case the charge point has a valid time source. If present, the timestamp does not mark the time the event occurred but only the time the message has been sent. In case multiple errors exist, only the error with the highest prio is reported to the backend. Charge Control C implements the following error codes for a StatusNotification:

ChargePointErrorCode	vendorErrorCode	Prio
PowerSwitchFailure		1 (highest)
ConnectorLockFailure	PlugLockMotorCapNotCharged	2
ConnectorLockFailure	PlugLockMotorUnexpectedOpen	3
ConnectorLockFailure	PlugLockMotorUnexpectedClose	4
ConnectorLockFailure	PlugLockCannotLock	5
ConnectorLockFailure	PlugLockCannotUnlock	6
PowerMeterFailure	CommunicationError	7
PowerMeterFailure	InternalError	8
PowerMeterFailure	IdleMeterError	9
PowerMeterFailure	ActiveMeterError	10
ReaderFailure		11
GroundFailure	SpuriousRcdError	12
GroundFailure	RcdTestError	12
GroundFailure	RcdGroundFailure	12
OtherError	EmergencyShutdown	13
OtherError	RcdRecloserFailure	14
OtherError	PersistenceOpenFailed	15
OtherError	PersistenceUnlinkDenied	15
OtherError	PersistenceReadOnly	15
OtherError	PersistenceInvalidType	15
OtherError	PersistenceCountError	15
OtherError	PersistenceClearError	15
OtherError	PersistenceNoInsertRowId	15
OtherError	PersistencePrepareFailed	15
OtherError	PersistenceUpdateFailed	15
OtherError	PersistenceDropFailed	15
OtherError	PersistenceTransactionFailed	15
OtherError	PersistenceCreateFailed	15

ChargePointErrorCode	vendorErrorCode	Prio
OtherError	PersistenceInsertFailed	15
OtherError	PersistenceDeleteFailed	15
OtherError	PersistenceTooManyTransactions	15
OtherError	PersistenceFinalizeFailed	15
LocalListConflict		16 (lowest)
OtherError	ProximityNoCable	
OtherError	ProximityInvalid	
OtherError	ProximityChanged	

Table 3 OCPP StatusNotification

3.3.4.1 PowerSwitchFailure

Description	contactor feedback doesn't match expected state
Affected ConnectorId	>= 1
MQTT condition	contactor/error = 1
Error cause	• contactor feedback misconfigured • contactor broken/welded
Impact	• charging interrupted / not possible • plug lock unlocked • CP duty cycle is set to 0% • may still hazardous voltage on cable/connector
How to solve remotely	• verify the configured value of "ports[0]/contactor/feedback_type" in customer.json with EVSE
How to solve on-site	• check contactor feedback • replace contactor

3.3.4.2 ConnectorLockFailure (vendorErrorCode = PlugLockMotorCapNotCharged)

Description	capacitor for plug lock motor isn't charged within expected time
Affected ConnectorId	>= 1
MQTT condition	plug_lock/permanent_failure = PlugLockMotorCapNotCharged
Error cause	• capacitor on Charge Control board is broken • power supply output voltage is not okay
Impact	• plug lock doesn't work as expected • CP duty cycle is set to 100% • charging not possible
How to solve on-site	• check EVSE plug lock motor & wires • check power supply output voltage to be at least 12 V

	• replace Charge Control board

3.3.4.3 ConnectorLockFailure (vendorErrorCode = PlugLockMotorUnexpectedOpen)

Description	**plug lock feedback does return open while it should be close**
Affected ConnectorId	>= 1
MQTT condition	plug_lock/permanent_failure = PlugLockMotorUnexpectedOpen
Error cause	• wrong motor configured • mechanical penetration of outlet/inserted EV plug
Impact	• CP duty cycle is set to 100% • charging is interrupted
How to solve remotely	• verify the configured value of "ports[0]/plug_lock/type" in customer.json with used socket / motor
How to solve on-site	• check EVSE plug lock & wires

3.3.4.4 ConnectorLockFailure (vendorErrorCode = PlugLockMotorUnexpectedClose)

Description	**plug lock feedback does return close while it should be open**
Affected ConnectorId	>= 1
MQTT condition	plug_lock/permanent_failure = PlugLockMotorUnexpectedClose
Error cause	• wrong motor configured • mechanical penetration of outlet/inserted EV plug
Impact	• EVSE plug lock doesn't work as expected • charging not possible
How to solve remotely	• verify the configured value of "ports[0]/plug_lock/type" in customer.json with used socket / motor
How to solve on-site	• check EV plug • check EVSE plug lock & wires

3.3.4.5 ConnectorLockFailure (vendorErrorCode = PlugLockCannotLock)

Description	**plug lock feedback doesn't return lock state after driving**
Affected ConnectorId	>= 1
MQTT condition	plug_lock/permanent_failure = PlugLockCannotLock
Error cause	• wrong motor configured • damaged EV plug prevent locking • EVSE plug lock damaged or stuck
Impact	• EVSE plug lock doesn't work as expected • CP duty cycle is set to 100% • charging not possible
How to solve remotely	• try to send OCPP UnlockConnector

	• verify the configured value of "ports[0]/plug_lock/type" in customer.json with used socket / motor
How to solve on-site	• check EV plug • check EVSE plug lock & wires

3.3.4.6 ConnectorLockFailure (vendorErrorCode = PlugLockCannotUnlock)

Description	**plug lock feedback doesn't return lock state after driving**
Affected ConnectorId	>= 1
MQTT condition	plug_lock/permanent_failure = PlugLockCannotUnlock
Error cause	• wrong motor configured • damaged plug prevent unlocking • EVSE plug lock damaged or stuck
Impact	• EVSE plug lock doesn't work as expected • unplug not possible
How to solve remotely	• try to send OCPP UnlockConnector • verify the configured value of "ports[0]/plug_lock/type" in customer.json with used socket / motor
How to solve on-site	• check EV plug • check EVSE plug lock & wires

3.3.4.7 PowerMeterFailure (vendorErrorCode = CommunicationError)

Description	**power meter doesn't reply to requests**
Affected ConnectorId	>= 1
MQTT condition	metering/meter/available = 0
Error cause	• meter misconfigured • incorrect RS-485 bus termination • mixing incompatible protocols on the same bus • power meter broken • connection to power meter interrupted • interferences on bus
Impact	• charging not possible • OCPP transaction stop delayed
How to solve remotely	• verify all configured values below "ports[0]/meter" in customer.json with used meter
How to solve on-site	• check RS-485 bus termination • separate incompatible protocols on different busses • check power meter • check power meter connection

3.3.4.8 PowerMeterFailure (vendorErrorCode = InternalError)

Description	**internal error during meter handling**
Affected ConnectorId	>= 1
Error cause	• power meter broken • power meter firmware bug
Impact	• charging not possible
How to solve remotely	• try to send OCPP Reset • check configuration
How to solve on-site	• check power meter • check power meter connection

3.3.4.9 PowerMeterFailure (vendorErrorCode = IdleMeterError)

Description	**power meter error during idle state**
Affected ConnectorId	>= 1
Error cause	• power meter broken • power meter firmware bug • power outtageon meter
Impact	• charging not possible
How to solve remotely	• try to send OCPP Reset • check configuration
How to solve on-site	• check power meter • check power meter connection

3.3.4.10 PowerMeterFailure (vendorErrorCode = ActiveMeterError)

Description	**power meter error during active state**
Affected ConnectorId	>= 1
Error cause	• power meter broken • power meter firmware bug • power outtage meter • RS-485 MCU on CC C not functional
Impact	• charging not possible
How to solve remotely	• try to send OCPP Reset • check configuration
How to solve on-site	• check power meter • check power meter connection • replace Charge Control board

3.3.4.11 ReaderFailure

Description	**RFID reader doesn't reply to requests (applies only when RFID is enabled)**
Affected ConnectorId	>= 1
MQTT condition	rfid/available = 0
Error cause	• RFID reader misconfigured • incorrect RS-485 bus termination • RFID reader broken • connection to RFID reader interrupted • interferences on bus • RS-485 MCU on CC C not functional
Impact	• ongoing charging isn't interrupted • charging session cannot be started/stopped via RFID • new charging session can only be started via RemoteStartTransaction
How to solve remotely	• verify all configured values below "ports[0]/rfid" in customer.json with used RFID reader
How to solve on-site	• check RS-485 bus termination • check RFID reader • check RFID reader connection • replace Charge Control board

3.3.4.12 GroundFailure (vendorErrorCode =SpuriousRcdError)

Description	**RCD tripped when charging station not in charging or testing**
Affected ConnectorId	>= 1
MQTT condition	rcd/error ="SpuriosRcdError"
Error cause	• RCD feedback connection broken • RCD misfunction • unexpected residual direct current • polarity of RCD feedback inverted • RCD misconfigured
Impact	• CP duty cycle is set to 0% • charging not possible • plug lock unlocked and prevented from locking
How to solve remotely	• verify all configured values of "ports[0]/rcd_monitor" in customer.json with EVSE
How to solve on-site	• check EVSE installation & wiring • check RCD • restart EVSE

3.3.4.13 GroundFailure (vendorErrorCode =RcdTestError)

Description	**RCD tripped during RCD test**
Affected ConnectorId	>= 1
MQTT condition	rcd/error = "RcdTestError"
Error cause	• RCD feedback or test connection broken • RCD misfunction during RCD test • polarity of RCD feedback or test inverted • RCD misconfigured
Impact	• CP duty cycle is set to 100% when EV connected • pluglock keep closed until EV is disconnected • charging not possible • CP duty cycle is set to 0% when EV is disconnected • plug lock unlocked and prevented from locking when EV is disconnected
How to solve remotely	• verify all configured values of "ports[0]/rcd_monitor" in customer.json with EVSE
How to solve on-site	• check EVSE installation & wiring • check RCD • replace RCD

3.3.4.14 GroundFailure (vendorErrorCode =RcdGroundFailure)

Description	**RCD tripped during charging session**
Affected ConnectorId	>= 1
MQTT condition	rcd/error = "RcdGroundFailure"
Error cause	• Residual direct current occurred • RCD feedback connection broken • RCD misfunction • RCD misconfigured
Impact	• CP duty cycle is set to 100% • plug lock keep closed until EV is disconnected • charging is interrupted / not possible • plug lock unlocked when EV is disconnected
How to solve remotely	• verify all configured values of "ports[0]/rcd_monitor" in customer.json with EVSE
How to solve on-site	• Disconnect EV • check RCD • check EVSE installation & wiring

3.3.4.15 OtherError (vendorErrorCode = EmergencyShutdown)

Description	**Emergency shutdown has been asserted**
Affected ConnectorId	>= 0
MQTT condition	emergency_shutdown = 1
Error cause	• user asserted emergency shutdown • polarity of emergency switch inverted
Impact	• CP duty cycle is set to 0% • charging is interrupted / not possible • plug lock unlocked
How to solve remotely	• verify the configured value of "ports[0]/emergency_alarm" in customer.json with EVSE
How to solve on-site	• check EVSE installation & wiring

3.3.4.16 OtherError (vendorErrorCode = RcdRecloserFailure)

Description	**RCD recloser doesn't reply to requests**
Affected ConnectorId	>= 1
MQTT condition	rcd/recloser/error = 1
Error cause	• RCD recloser misconfigured • incorrect RS-485 bus termination • mixing incompatible protocols on the same bus • RCD recloser broken • connection to RCD recloser interrupted • interferences on bus • RS-485 MCU on CC C not functional
Impact	• ongoing charging isn't interrupted
How to solve remotely	• verify all configured values of "ports[0]/recloser" in customer.json with used RCD recloser
How to solve on-site	• check RS-485 bus termination • separate incompatible protocols on different busses • check RCD recloser • check RCD recloser connection • replace Charge Control board

3.3.4.17 OtherError (vendorErrorCode = PersistenceOpenFailed)

Description	**Local database subsystem reported an error**
Affected ConnectorId	>= 0
Error cause	• filesystem on internal eMMC storage was corrupted
Impact	• charging is not possible

How to solve remotely	• gather diagnostics data, contact vendor and forward collected diagnostics for remote analysis • try to send OCPP Reset
How to solve on-site	• check/repair internal filesystem on eMMC • manually delete database file /var/lib/ocppd/transactions.db and reboot

3.3.4.18 OtherError (vendorErrorCode = PersistenceUnlinkDenied)

Description	**Local database subsystem reported an error**
Affected ConnectorId	>= 0
Error cause	• filesystem on internal eMMC storage was corrupted
Impact	• charging is not possible
How to solve remotely	• gather diagnostics data, contact vendor and forward collected diagnostics for remote analysis • try to send OCPP Reset
How to solve on-site	• check/repair internal filesystem on eMMC • manually delete database file /var/lib/ocppd/transactions.db and reboot

3.3.4.19 OtherError (vendorErrorCode = PersistenceReadOnly)

Description	**Local database subsystem reported an error**
Affected ConnectorId	>= 0
Error cause	• filesystem on internal eMMC storage was corrupted
Impact	• charging is not possible
How to solve remotely	• gather diagnostics data, contact vendor and forward collected diagnostics for remote analysis • try to send OCPP Reset
How to solve on-site	• check/repair internal filesystem on eMMC • manually delete database file /var/lib/ocppd/transactions.db and reboot

3.3.4.20 OtherError (vendorErrorCode = PersistenceInvalidType)

Description	**Local database subsystem reported an error**
Affected ConnectorId	>= 0
Error cause	• internal database was corrupted
Impact	• charging is not possible
How to solve remotely	• gather diagnostics data, contact vendor and forward collected diagnostics for remote analysis
How to solve on-site	• check/repair internal filesystem on eMMC • manually delete database file /var/lib/ocppd/transactions.db and reboot

3.3.4.21 OtherError (vendorErrorCode = PersistenceCountError)

Description	**Local database subsystem reported an error**
Affected ConnectorId	>= 0
Error cause	• internal database was corrupted • filesystem on internal eMMC storage was corrupted
Impact	• charging is not possible
How to solve remotely	• gather diagnostics data, contact vendor and forward collected diagnostics for remote analysis • try to send OCPP Reset
How to solve on-site	• check/repair internal filesystem on eMMC • check free space on /srv filesystem • manually delete database file /var/lib/ocppd/transactions.db and reboot

3.3.4.22 OtherError (vendorErrorCode = PersistenceClearError)

Description	**Local database subsystem reported an error**
Affected ConnectorId	>= 0
Error cause	• internal database was corrupted • filesystem on internal eMMC storage was corrupted
Impact	• charging is not possible
How to solve remotely	• gather diagnostics data, contact vendor and forward collected diagnostics for remote analysis • try to send OCPP Reset
How to solve on-site	• check/repair internal filesystem on eMMC • check free space on /srv filesystem • manually delete database file /var/lib/ocppd/transactions.db and reboot

3.3.4.23 OtherError (vendorErrorCode = PersistenceNoInsertRowId)

Description	**Local database subsystem reported an error**
Affected ConnectorId	>= 0
Error cause	• internal database was corrupted • filesystem on internal eMMC storage was corrupted • free space on internal eMMC storage exhausted
Impact	• charging is interrupted / not possible • on-going transactions might be lost • transactions not yet transferred/commited to to backend might be lost
How to solve remotely	• gather diagnostics data, contact vendor and forward collected diagnostics for remote analysis
How to solve on-site	• check/repair internal filesystem on eMMC • check free space on /srv filesystem

	• manually delete database file /var/lib/ocppd/transactions.db and reboot

3.3.4.24 OtherError (vendorErrorCode = PersistencePrepareFailed)

Description	**Local database subsystem reported an error**
Affected ConnectorId	>= 0
Error cause	• internal database was corrupted • filesystem on internal eMMC storage was corrupted
Impact	• charging is interrupted / not possible • on-going transactions might be lost • transactions not yet transferred/commited to to backend might be lost
How to solve remotely	• gather diagnostics data, contact vendor and forward collected diagnostics for remote analysis
How to solve on-site	• check/repair internal filesystem on eMMC • check free space on /srv filesystem • manually delete database file /var/lib/ocppd/transactions.db and reboot

3.3.4.25 OtherError (vendorErrorCode = PersistenceUpdateFailed)

Description	**Local database subsystem reported an error**
Affected ConnectorId	>= 0
Error cause	• internal database was corrupted • filesystem on internal eMMC storage was corrupted • free space on internal eMMC storage exhausted • transaction id provided by backend not unique
Impact	• charging is interrupted / not possible • on-going transactions might be lost • transactions not yet transferred/commited to to backend might be lost
How to solve remotely	• gather diagnostics data, contact vendor and forward collected diagnostics for remote analysis • try to send OCPP Reset
How to solve on-site	• check/repair internal filesystem on eMMC • check free space on /srv filesystem • manually delete database file /var/lib/ocppd/transactions.db and reboot

3.3.4.26 OtherError (vendorErrorCode = PersistenceDropFailed)

Description	**Local database subsystem reported an error**
Affected ConnectorId	>= 0
Error cause	• internal database was corrupted • filesystem on internal eMMC storage was corrupted

Impact	• charging is not possible
How to solve remotely	• gather diagnostics data, contact vendor and forward collected diagnostics for remote analysis • try to send OCPP Reset
How to solve on-site	• check/repair internal filesystem on eMMC • check free space on /srv filesystem • manually delete database file /var/lib/ocppd/transactions.db and reboot

3.3.4.27 OtherError (vendorErrorCode = PersistenceTransactionFailed)

Description	**Local database subsystem reported an error**
Affected ConnectorId	>= 0
Error cause	• internal database was corrupted • filesystem on internal eMMC storage was corrupted • free space on internal eMMC storage exhausted
Impact	• charging is not possible
How to solve remotely	• gather diagnostics data, contact vendor and forward collected diagnostics for remote analysis • try to send OCPP Reset
How to solve on-site	• check/repair internal filesystem on eMMC • check free space on /srv filesystem • manually delete database file /var/lib/ocppd/transactions.db and reboot

3.3.4.28 OtherError (vendorErrorCode = PersistenceCreateFailed)

Description	**Local database subsystem reported an error**
Affected ConnectorId	>= 0
Error cause	• internal database was corrupted • filesystem on internal eMMC storage was corrupted • free space on internal eMMC storage exhausted
Impact	• charging is not possible
How to solve remotely	• gather diagnostics data, contact vendor and forward collected diagnostics for remote analysis • try to send OCPP Reset
How to solve on-site	• check/repair internal filesystem on eMMC • check free space on /srv filesystem • manually delete database file /var/lib/ocppd/transactions.db and reboot

3.3.4.29 OtherError (vendorErrorCode = PersistenceInsertFailed)

Description	**Local database subsystem reported an error**
Affected ConnectorId	>= 0

Error cause	• internal database was corrupted • filesystem on internal eMMC storage was corrupted • free space on internal eMMC storage exhausted • transaction id provided by backend not unique
Impact	• charging is interrupted / not possible • on-going transactions might be lost • transactions not yet transferred/commited to to backend might be lost
How to solve remotely	• gather diagnostics data, contact vendor and forward collected diagnostics for remote analysis • try to send OCPP Reset
How to solve on-site	• check/repair internal filesystem on eMMC • check free space on /srv filesystem • manually delete database file /var/lib/ocppd/transactions.db and reboot

3.3.4.30 OtherError (vendorErrorCode = PersistenceDeleteFailed)

Description	**Local database subsystem reported an error**
Affected ConnectorId	>= 0
Error cause	• internal database was corrupted • filesystem on internal eMMC storage was corrupted
Impact	• charging is not possible
How to solve remotely	• gather diagnostics data, contact vendor and forward collected diagnostics for remote analysis • try to send OCPP Reset
How to solve on-site	• check/repair internal filesystem on eMMC • manually delete database file /var/lib/ocppd/transactions.db and reboot

3.3.4.31 OtherError (vendorErrorCode = PersistenceTooManyTransactions)

Description	**Maximum of offline transactions reached**
Affected ConnectorId	>= 0
Error cause	• maximum of offline transactions reached
Impact	• charging is not possible
How to solve remotely	• try to send OCPP Reset
How to solve on-site	• check/repair internet connection

3.3.4.32 OtherError (vendorErrorCode = PersistenceFinalizeFailed)

Description	**Local database subsystem reported an error**
Affected ConnectorId	>= 0
Error cause	• internal database was corrupted

	• filesystem on internal eMMC storage was corrupted
Impact	• charging is interrupted / not possible • on-going transactions might be lost • transactions not yet transferred/commited to to backend might be lost
How to solve remotely	• gather diagnostics data, contact vendor and forward collected diagnostics for remote analysis • try to send OCPP Reset
How to solve on-site	• check/repair internal filesystem on eMMC • manually delete database file /var/lib/ocppd/transactions.db and reboot

3.3.4.33 LocalListConflict

Description	**Authorize.Conf mismatch Local auth list**
Affected ConnectorId	== 0
Error cause	• Local auth list out of sync
Impact	• charging not possible
How to solve remotely	• update OCPP local auth list

3.3.4.34 ProximityNoCable

Description	**Proximity pilot not available**
Affected ConnectorId	>=1
MQTT condition	session/proximity_error = ProximityNoCable
Error cause	• EV plug damaged • EVSE socket damaged • EVSE wiring damaged • board malfunction due electronic issues
Impact	• CP duty cyle is set to 100% • if in charging: ○ charging is interrupted ○ plug lock remains locked • if not in charging: ○ charging not possible ○ plug lock remains unlocked
How to solve remotely	• inform user about possible charging cable issue and try different cable
How to solve on-site	• check EVSE installation & wiring & board • check EV charging cable • check EVSE plug lock & wires

3.3.4.35 ProximityInvalid

Description	**Invalid proximity pilot detected**
Affected ConnectorId	>=1
MQTT condition	session/proximity_error = ProximityInvalid
Error cause	• EV plug damaged • EVSE socket damaged • EVSE wiring damaged • board malfunction due electronic issues
Impact	• CP duty cyle is set to 100% • if in charging: ○ charging is interrupted ○ plug lock remains locked • if not in charging: ○ charging not possible ○ plug lock remains unlocked
How to solve remotely	• inform user about possible charging cable issue and try different cable
How to solve on-site	• check EVSE installation & wiring & board • check EV charging cable • check EVSE plug lock & wires

3.3.4.36 ProximityChanged

Description	**proximity pilot changed during charge**
Affected ConnectorId	>=1
MQTT condition	session/proximity_error = ProximityChanged
Error cause	• EV plug damaged • EVSE socket damaged • EVSE wiring damaged • board malfunction due electronic issues
Impact	• charging continue with reduced current limit until EV disconnected
How to solve remotely	• inform user about possible charging cable issue and try different cable
How to solve on-site	• check EVSE installation & wiring & board • check EV charging cable • check EVSE plug lock & wires

3.3.5 Availability

In addition to the control of Availability via OCPP, the availability of the charging station can be controlled with an MQTT topic. This external control is noted as "customer availability" in the table below. Note that, if the backend has set the charging station to "inoperative", the customer availability cannot force it to "operative".

Backend availability	Customer availability	Actual (resulting) availability
inoperative	operative	inoperative
inoperative	inoperative	inoperative
operative	operative	operative
operative	inoperative	inoperative

Table 4 Availability matrix

The following are the MQTT topics used to externally observe and control the availability.

Topic	Subscribe/Publish	Type	Retain	Unit	Remarks
port0/ci/availability/target	publishable	String	Yes	-	Contains one of the following strings to signal the additional customer availability: `operative`, `inoperative`.
port0/availability/actual	subscribable-only	String	Yes		Contains one of the following strings to signal the overall, resulting availability: `operative`, `inoperative`.

Table 5 Availability topics

3.3.6 OCPP Measurand Support

The OCPP 1.6 standard defines a number of measurands which could be reported to the central system using `MeterValues.req` and/or `StopTransaction.req` messages.

The standard also allows some optional properties to describe in detail at which point the corresponding measurand was obtained.

The Charge Control C implementation for example does not use the property `location`, which means that the default location "Outlet" as defined by the standard should be used for interpretation.

Each measurand can also be tagged with an optional property phase. Here the Charge Control C implementation differentiates between two use-cases: it runs in a single-phase or in a three-phase charging station. In a single-phase system the phase property is never used, that means that only the overall measurands are provided and available. In a three-phase system, also the phase-specific measurands might be available. However, this depends on the actually used electricity meter, its capabilities and the state of the current implementation.

Implemented measurands are listed in the following table.

Measurand	Description
Voltage	In a three-phase system, the measurands are available with the `phase` property set to `Lx-N`, e.g. as `Voltage.L1-N`. In a three-phase system, no overall `Voltage` is available. In a single-phase system, only the overall `Voltage` is available.
Frequency	Only the overall `Frequency` is supported for both, single- and three-phase systems.
Current.Import	In a three-phase system, the measurands are available with the `phase` property set to `Lx`, e.g. as `Current.Import.L1`. In a three-phase system, an overall `Current.Import` is only available in case the electricity meter provides such a reading - the charging stack does not calculate it itself if not provided by the meter. In a single-phase system, only the overall `Current` is available.
Power.Active.Import	In a three-phase system, the measurands are available with the `phase` property set to `Lx`, e.g. as `Power.Active.Import.L1`. In a three-phase system, an overall `Power.Active.Import` is usually available since nearly each electricity meter provide such a reading. In a single-phase system, only the overall `Power.Active.Import` is available.
Energy.Active.Import.Register	For both, single- and three-phase systems, only the overall `Energy.Active.Import.Register` is supported since most three-phase meters do not implement dedicated phase-specific counter registers.
Current.Offered	In a three-phase system, the measurands are available with the `phase` property set to `Lx`, e.g. as `Current.Offered.L1`. The offered current depends on the currently active current limits due to e.g. cabling, load balancing etc.

Measurand	Description
	In a three-phase system, an overall `Current.Offered` is calculated by the charging stack as the sum of the individual offered currents on each phase. Note, that this sum might vary depending on the actual system state, e.g. when phase count switching is enabled and in effect. In a single-phase system, only the overall `Current.Offered` is available.
Power.Offered	For both, single- and three-phase systems, only an overall `Power.Offered` is calcuated by the charging stack as the product of the multiplication of `Current.Offered` (i.e. the overall total current) and the configured nominal voltage (see customer.json).

3.3.7 OCPP configuration

Charge Control C supports the following configuration parameters. Except for `HeartbeatInterval`, all writeable parameters are stored in the customer.json.

As noted in the column "Changes apply", it depends on the specific parameter when it takes effect. Immediately means at this point, that the new value is used as soon as the charging stack uses it during the usual workflow.

Example: "useAsTimeSource" is effectly used when the next OCPP message "BootNotification" or "Heartbeat" is exchanged, but it does not mean, that setting this value results automatically in an adjusted clock on the charging station. The behavior of the other parameters is similar.

Parameter	Access	Default	Changes apply	Vendor specific
AllowOfflineTxForUnknownId	R/W	false	Immediately	
allowTxWithInvalidTime	R/W	false	Immediately	X
AuthorizationCacheEnabled	R/W	true	After reboot	
AuthorizeRemoteTxRequests	R/W	false	Immediately	
bootNotificationOnReconnect	R/W	false	Immediately	X
calibrationLawFormat	R/W	ocmf,plain	Next transaction	X
CentralSystemURI	R/W		After reboot	X
ChargeProfileMaxStackLevel	R	0	-	
ChargingScheduleAllowedChargingRateUnit	R	Current	-	
ChargingScheduleMaxPeriods	R	1	-	
ClockAlignedDataInterval	R/W	0	After reboot	
ConnectionTimeOut	R/W	60	Immediately	
ConnectorPhaseRotation	R	unknown	-	
ftpTryTLSUpgrade	R	false	-	X
GetConfigurationMaxKeys	R	10	-	
HeartbeatInterval	R/W	600	Immediately	

Parameter	Access	Default	Changes apply	Vendor specific
LocalAuthListEnabled	R/W	true	After reboot	
LocalAuthListMaxLength	R	10000	-	
LocalAuthorizeOffline	R/W	false	Immediately	
LocalPreAuthorize	R/W	true	Immediately	
MaxChargingProfilesInstalled	R	2	-	
MeterValuesAlignedData	R/W	[ ]	Next transaction	
MeterValuesAlignedDataMaxLength	R	27	-	
MeterValuesSampledData	R/W	[Energy.Active.Import.Register]	Next transaction	
MeterValuesSampledDataMaxLength	R	27	-	
MeterValueSampleInterval	R/W	30	Next transaction	
Meter1PublicKey	R	- unset -	-	X
NumberOfConnectors	R	1	-	
ResetRetries	R	0	-	
SendLocalListMaxLength	R	10000	-	
sendMeterPublicKeyOnBootNotification	R/W	true	Immediately	X
StopTransactionOnEVSideDisconnect	R	true	-	
StopTransactionOnInvalidId	R/W	true	Immediately	
StopTransactionSignatureFormat	R	MR	-	
StopTxnAlignedData	R/W	[ ]	Next transaction	
StopTxnAlignedDataMaxLength	R	27	-	
StopTxnSampledData	R/W	[ ]	Next transaction	
StopTxnSampledDataMaxLength	R	27	-	
SupportedFeatureProfiles	R	Core, FirmwareManagement, LocalAuthListManagement	-	
SupportedFileTransferProtocols	R	FTP, FTPS, HTTP, HTTPS	-	
TransactionMessageAttempts	R/W	3	After reboot	
TransactionMessageRetryInterval	R/W	30	After reboot	
UnlockConnectorOnEVSideDisconnect	R	true	-	
useAsTimeSource	R/W	true	Immediately	X

Parameter	**Access**	**Default**	**Changes apply**	**Vendor specific**
vendorId	R	com.in-tech.smartcharging	-	X
WebSocketPingInterval	R/W	0	Immediately	
webSocketPingTimeout	R/W	120	Immediately	X

Table 6 OCPP configuration

Description of HeartbeatInterval

As mentioned, the `HeartbeatInterval` configuration value is special. It is not stored in the `customer.json` since it is always sent with the `BootNotification.conf` message. However, OCPP 1.6 allows to send a value of zero. Quote from the specification:

> *If that interval value is zero, the Charge Point chooses a waiting interval on its own, in a way that avoids flooding the Central System with requests.*

The Charge Control C implementation chooses the value of 10 minutes in this case.

Description of Meter1PublicKey

In case of an electrical meter with Eichrecht support, the configuration key `Meter1PublicKey` contains the ASN.1 encoded public key of the meter as hex-encoded string if the meter supports reading it. Otherwise, i.e. in case the meter does not support reading the public key, it contains an empty string (zero length). If the electrical meter has no support for Eichrecht, then this configuration key is unset.

Example content for Eichrecht case (wrapped for better readability):

```
3059301306072a8648ce3d020106082a8648ce3d030107034200043865f715c2
8598e21598e37da6e5db4e97c7501d9db9228ddbc8ae9a5aee705572d9da78e2
31378dcc1435dfce5e9b30cc5b8f3a016773205791cbf7be78e23f
```

Example openssl dump (long hex-string left out for better readability):

```
$ echo "<hexstring see above>" | xxd -r -p | openssl asn1parse -in - -
inform DER -dump -i
    0:d=0  hl=2 l=  89 cons: SEQUENCE
    2:d=1  hl=2 l=  19 cons:  SEQUENCE
    4:d=2  hl=2 l=   7 prim:   OBJECT            :id-ecPublicKey
   13:d=2  hl=2 l=   8 prim:   OBJECT            :prime256v1
   23:d=1  hl=2 l=  66 prim:  BIT STRING
      0000 - 00 04 38 65 f7 15 c2 85-98 e2 15 98 e3 7d a6 e5
..8e.........}..
      0010 - db 4e 97 c7 50 1d 9d b9-22 8d db c8 ae 9a 5a ee
.N..P..."....Z.
      0020 - 70 55 72 d9 da 78 e2 31-37 8d cc 14 35 df ce 5e
pUr..x.17...5..^
      0030 - 9b 30 cc 5b 8f 3a 01 67-73 20 57 91 cb f7 be 78   .0.[.:.gs
W....x
      0040 - e2 3f                                              .?
```

3.3.8 Generic OCPP DataTransfer

The Generic OCPP DataTransfer interface provides a transparent channel from the MQTT interface to the Central System and vice versa. It operates on both directions, both with Central System initiated DataTransfer and Charge Point initiated DataTransfer.

3.3.8.1 Central System initiated

	MQTT Topic payloads for general synchronous interface from Central System
Direction	Central System → Charge Point
Request	
MessageTypeNumber (Call)	2
UniqueID	string of up to 36 characters
Action	DataTransfer
Payload Object	<vendorId>, <messageId>, <data>
Confirm	
MessageTypeNumber (CallResult)	3
UniqueID	string of 36 characters
Payload Object	<status>, <data>

Table 7 MQTT Topic payloads for general synchronous interface from Central System

Note: It is recommended to encode the data with Base64.

Name	**MQTT topic**
complete request	ocpp/data_transfer_from_cs/request
complete confirm	ocpp/data_transfer_from_cs/confirm

Table 8 MQTT topics

Dedicated configuration in customer.json:

Parameter	**Description**	**Type**	**Default**
ocpp/csDataTransfers[]/vendorId	specifies the vendorId of a dataTransfer from the central system	String	
ocpp/csDataTransfers[]/messageId	specifies the optional messageId of a dataTransfer from the central system	String	
ocpp/csDataTransfers[]/confirmTimeout	specifies the timeout of dataTransfer confirm on MQTT interface in seconds	Integer (1 .. 60)	1

Table 9 Dedicated configuration in customer.json

In order to use the generic DataTransfer interface the customer application needs to:

- subscribe to the request topic
- process JSON data from the request topic
- publish (no retain) the whole DataTransfer confirm as JSON to the confirm MQTT topic
    - UniqueID in confirm must be identical to the request

3.3.8.2 Charge Point initiated

	MQTT Topic payloads for general synchronous interface from Charge Point
Direction	Charge Point → Central System
Request	
MessageTypeNumber (Call)	2
UniqueID	string of up to 36 characters
Action	DataTransfer
Payload Object	<vendorId>, <messageId>, <data>
Confirm	
MessageTypeNumber (CallResult / CallError)	3 / 4
UniqueID	string of 36 characters
Payload Object	<status>, <data>

Table 10 MQTT Topic payload for general synchronous interface from Charge Point

Name	MQTT topic
register service	ocpp/data_transfer_to_cs/service_id
complete request	ocpp/data_transfer_to_cs/<service_id>/request
complete confirm	ocpp/data_transfer_to_cs/<service_id>/confirm

Table 11 MQTT topics

Limitation: service_id must not contain: slashes, spaces, plus, hash, dollar sign

In order to use the generic DataTransfer interface the customer application needs to:

- register its service by publishing a service_id to the register topic (e.g. PID or similar unique ID)
- subscribe to the confirm topic
- publish (no retain) the whole request as JSON to the request MQTT topic
- process JSON data from the confirm topic

3.3.8.3 MQTT formatting

The payload on the MQTT interface is formatted as JSON messages, with a formatting corresponding exactly to OCPP messages as defined in *Open Charge Point Protocol JSON 1.6, OCPP-J 1.6 Specification*.

Note: The given MQTT Request/Confirm formats are usable for Central System → Charge Point and vice versa.

MQTT Request example:

```
[2,"73a0d9a1-12d1-4223-ad16-90928d30de1e","DataTransfer",{"vendorId":"com.
vendor","messageId":"fooBar","data":"base64encodedData"}]
```

MQTT Confirm example:

```
[3,"73a0d9a1-12d1-4223-ad16-90928d30de1e",{"status":"Accepted","data":"bas
e64encodedData"}]
```

On Charge Point initiated DataTransfer, MQTT Confirm contains the exact CallResult as returned by the Central System. If the Central System returns a CallError, this is also forwarded as such. If the MQTT Request cannot be processed locally and therefore not forwarded to the Central System, a CallError indicating the issue is also generated and returned.

3.3.9 Specific OCPP DataTransfer

Charge Control C supports the following DataTransfer messages:

	Get the current state of the RCD recloser
Direction	Central System → Charge Point
Request	
vendorId	com.in-tech.smartcharging
messageId	rcdRecloserGetState
data	<connectorId>
Response (good case)	
status	Accepted
data	<recloserState>
Response (bad case)	
status	Rejected
data	<recloserReason>
	Change the current state of the RCD recloser
Direction	Central System → Charge Point
Request	
vendorId	com.in-tech.smartcharging
messageId	rcdRecloserChangeState

data	<connectorId>, <recloserState>
Response (good case)	
Status	Accepted
data	<recloserState>
Response (bad case)	
status	Rejected
data	<recloserReason>

Table 12 DataTransfer message

Definition:

<recloserState>= open / close

<recloserReason>= notPresent / notAvailable / invalidRequestData

<conncectorId>= 1

3.3.10 OCPP DataTransfer for Public Key of Eichrecht Meters

The OCPP daemon supports the custom DataTransfer as specified by has.to.be in the be.energized knowledge base.

When an Eichrecht capable meter is detected during startup, and the configuration key `sendMeterPublicKeyOnBootNotification` is set to `true`, then the OCPP daemon sends this custom DataTransfer message after every BootNotification.

Any failure response of the Central System is ignored, i.e. the transmission is not retried automatically. Only after the next BootNotification, the transmission is repeated.

	Transmission of Public Key of Eichrecht Meter to Central System
Direction	Charge Point → Central System
Request	
vendorId	generalConfiguration
messageId	setMeterConfiguration
data	<JSON Object containing the Public Key>
Response	
status	<ignored>
data	<ignored>

Table 13 Public Key DataTransfer Message

3.3.11 Local Data Storage

The OCPP daemon needs to store some data on the internal eMMC storage during normal operation. This data is located in different files on the filesystem below `/var/lib/ocppd`. The files are kept during firmware update and - if necessary - the databases are migrated to newer schema after firmware update.

Filename	**File type**	**Usage**
ocpp_data.json	plain text / JSON	Common data, connector status etc.
auth.db	SQLite database	authorization cache and local authorization list
transactions.db	SQLite database	local cache for transaction meta data and meter values

In case of trouble, these files could be deleted manually - the files are re-created after a reboot of the Charge Control C. However, this might result in losing data: for example, transactions which are not yet transferred/committed to the backend might get lost.

3.3.12 FTP and Path Names

When using FTP URLs for e.g. uploading diagnostic data or downloading firmware update files, then a special feature of the FTP protocol must be taken into account: the given path in the URL is relative to the directory the FTP client enters.

For example, to upload the diagnostic file into the directory `mywallbox` located inside the home directory at the ftp site, an FTP URL in the form `ftp://ftp.example.com/mywallbox/` must be used.

If you want to upload the file to a directory `myotherwallbox` located in the root directory of that same site, you need to specify the absolute path name, i.e. with an extra slash at the beginning of the path name, e.g. `ftp://ftp.example.com//myotherwallbox/`

Note, that it also depends on the configuration of the FTP server itself whether the FTP client sees "upper" directories relative to the directory at which it enters.

3.4 Timings

The following table describes the fixed timings of the charging stack:

Description	Value	Start point
Minimum time until a BootNotification is sent (meter found)	40 s	Boot finished
Maximum time until a BootNotification is sent (no meter found)	60 s	Boot finished
Time until fallback to Offline mode	120 s	Boot finished
Delay before storing a charging profile persistently	60 s	OCPP SetChargingProfile received
Minimum delay between two OCPP StatusNotification	10 s	OCPP StatusNotification sent
Timeout until an OCPP change configuration request fails	1 s	OCPP ChangeConfiguration received
Timeout until an OCPP unlock connector request fails	20 s	OCPP UnlockConnector received
Timeout until an OCPP RCD recloser change request fails	10 s	OCPP DataTransfer received
Minimum time between opening and closing contactor	6 s	Contactor opened

Table 14 Fixed timings of the charging stack

4 HMI

4.1 LEDs

Charge Control C has three LEDs populated.

- LED1 - green
- LED2 - yellow
- LED3 - red

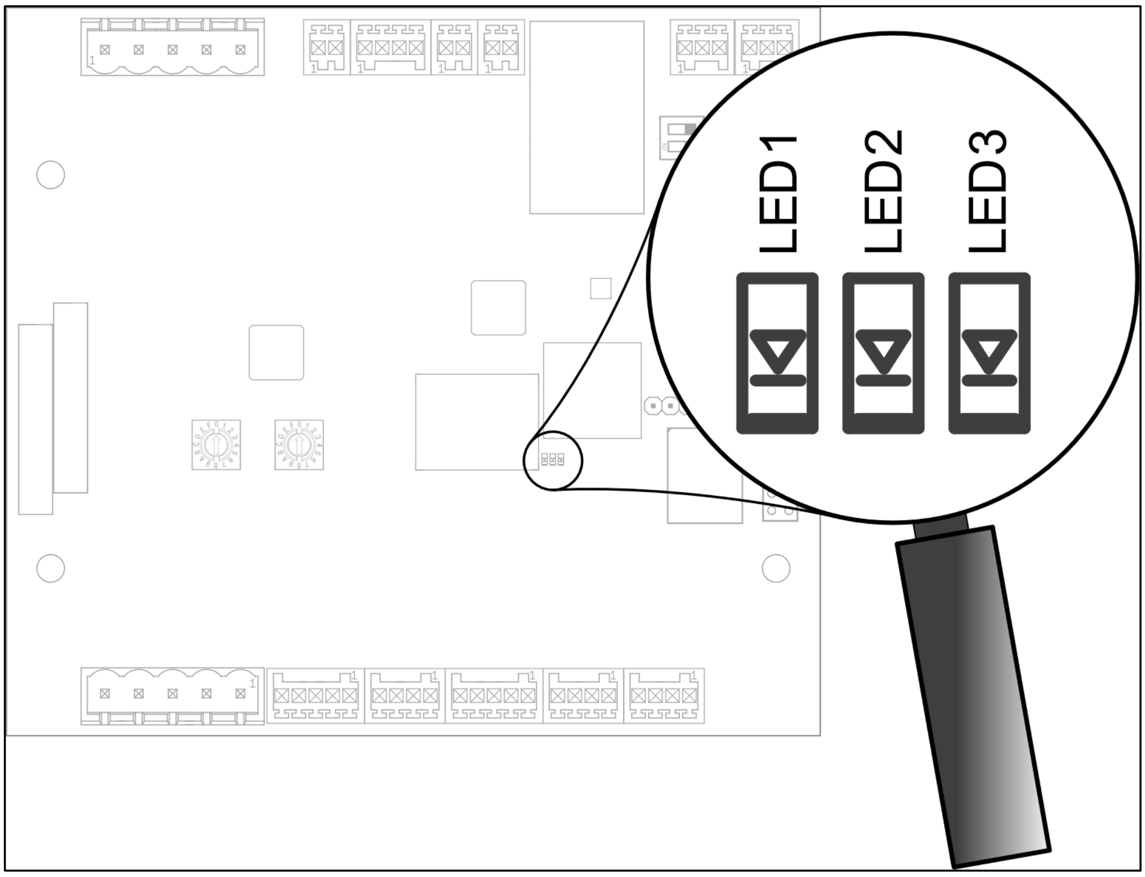


Figure 1 LEDs on Charge Control C

4.1.1 LED1 (green)

Default behavior is:

- blinking: booting
- permanently on: boot is finished and charging software is operational

4.1.2 LED2 (yellow)

Default behavior is:

- permanently on: USB Stick was plugged in and is being searched for update images
- blinking (250ms on / 250ms off): update in progress

4.1.3 LED3 (red)

Default behavior is:

- Linux Heartbeat (pulsing depending on load)

4.2 Switches

Charge Control C comes with three switches as shown in Figure Switches on Charge Control C.

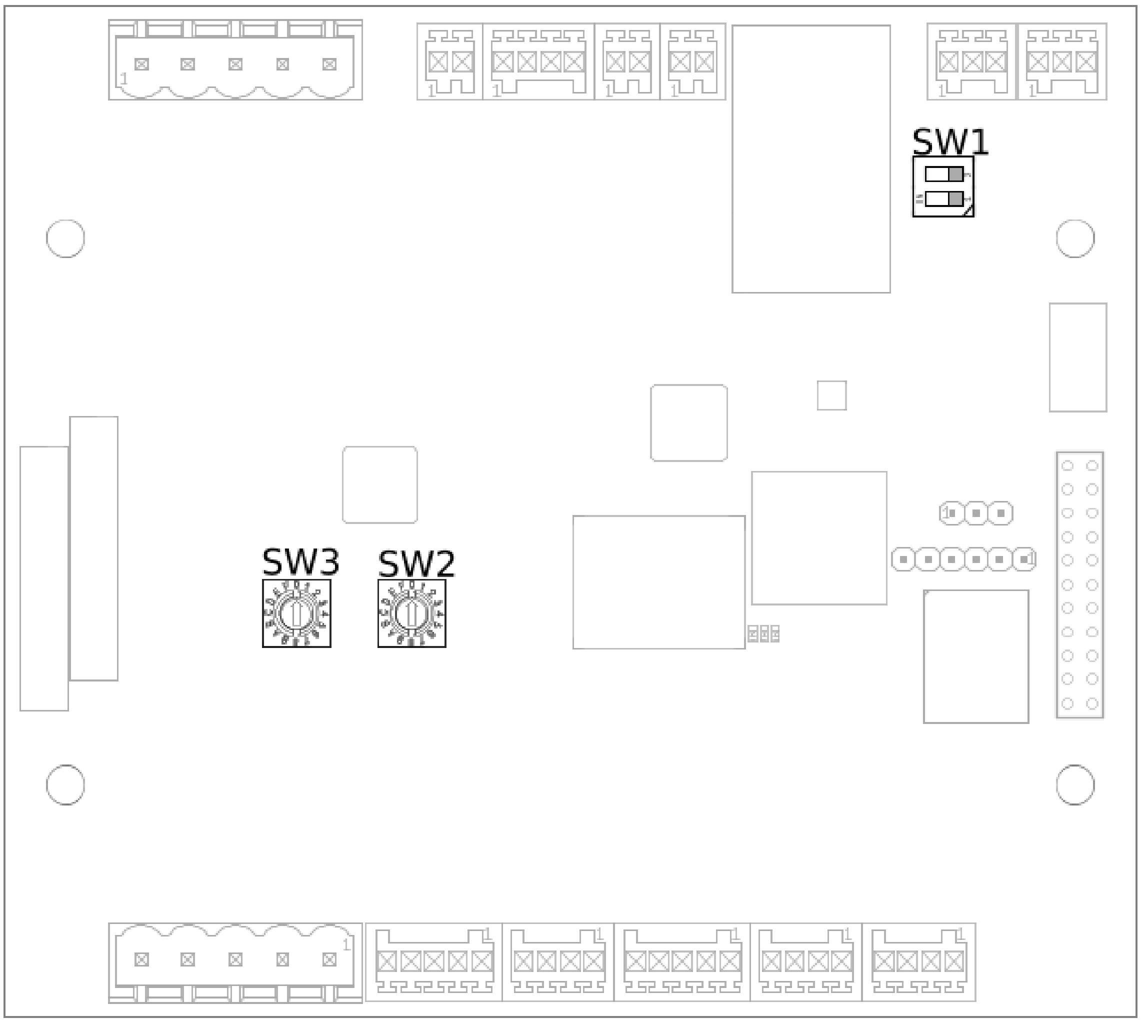


Figure 2 Switches on Charge Control C

4.2.1 SW1 - EIA-485 Termination

SW1 enables or disables the termination resistor of the EIA-485 1 available on X7.

Pos 1&2	Termination
On	On
Off	Off

Table 15 SW1 - EIA-485 Termination

4.2.2 SW2 - Rotary Coded Switch

The rotary coded switch is intended to provide a setup possibility for field service technicians or similar personnel. The switch is read in software - the default use is to set up the maximum current that the charge controller may allow for charging, and the number of phases used. The use of the switch can be changed in software if desired.

The currently implemented current limits are:

SW2 position	current limit in A	number of phases

0	6	1
1	10	1
2	13	1
3	16	1
4	20	1
5	32	1
6	40	1
7	63	1
8	6	3
9	10	3
A	13	3
B	16	3
C	20	3
D	32	3
E	40	3
F	63	3

Table 16 Current limits

CAUTION!
Electrical shock hazard! The switch is usually locking the position of the selector at a valid position, but it is possible to leave the selector in an invalid position between two states. Changing the selection should only be done in power-off state!

4.2.3 SW3

SW3 is reserved for future use and is not populated at the moment.

5 Mechanical Dimensions

The mechanical dimensions and mounting holes of this product are dimensioned in Figure mechanical drawing.

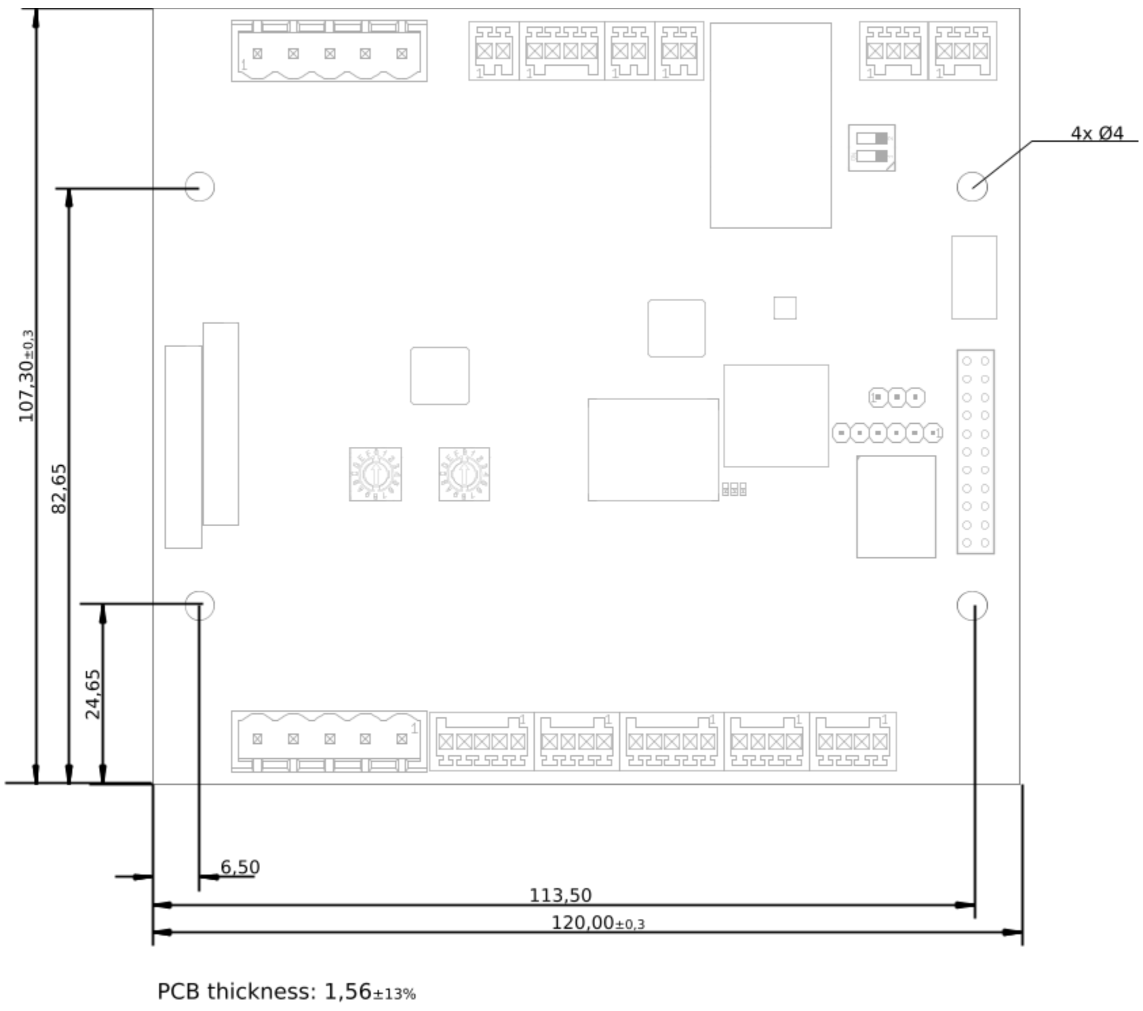


Figure 3 Mechanical drawing of Charge Control C

6 Mounting

- Mounting position is irrelevant as long as operating parameters are met.
- Every mounting hole has a copper restrict area to support mounting via enclosure domes and screws. Screws and domes should not exceed a diameter of 7.8 mm.
- Tightening torque should not exceed 4 Nm.

7 Interfaces

7.1 Ethernet

This device supports 10/100 Mbit/s Ethernet. In the Linux operating system it is available as network interface eth0. Starting with Yocto-based firmware releases, this interface is part of a bridge interface br0, see following sections for details.

Board Interface	**Linux Interface**
Ethernet	eth0

Table 17 Ethernet

7.2 USB

USB support is composed of a USB OTG core controller. It is compliant with the USB 2.0 specification.

USB is mainly used for USB internet dongles, firmware updates and for commissioning purposes.

Currently supported peripherals:

Manufacture	**Model**	**Device type**	**Restrictions**
Huawei	E3531	Internet dongle (3G, GPRS/HSDPA/UMTS)	--
Huawei	E3372	Internet dongle (4G, LTE)	Only supported with a separate USB power supply via e.g. an USB 2.0 hub with external power supply.

Table 18 Supported USB devices

7.3 EIA-485

7.3.1 Overview

In order to connect Charge Control C to a backend or an internal peripheral (e.g. smart meters, display and RFID readers), the board supports up to two EIA-485 interfaces.

While the charging stack ships with included support for some peripheral devices, the "link to backend" functionality is not implemented by default.

The baud rate of each EIA-485 interface is configurable up to 115200 bps.

Board Interface		**EIA-485 #1 isolated (X7)**	**EIA-485 #2 (X8)**
Linux Interface		/dev/ttymxc0	/dev/ttymxc4
Termination		yes, 120 Ohm deactivatable via SW1	yes, 120 Ohm permanently activated
Failsafe Biasing[1]		PCB board revision ≤ V0R3[2]: no	yes
		PCB board revision >V0R3[2]: yes	
Intended Usage	**Charge Control C 100**	link to backend / internal peripheral	-not available-
	Charge Control C 200	link to backend / internal peripheral	-not available-
	Charge Control C 300	link to backend	internal peripheral

Table 19 Board Interface

[1]: 390 Ohm Pull-up & 390 Ohm Pull-down resistors permanently activated

[2]: PCB board revision string can be found on the left side of the board near the relays

7.3.2 Supported Peripheral Devices

The factory shipped charging stack supports several peripheral devices out-of-the-box. For each type of peripheral, the charging stack support is provided in the form of a dedicated daemon, i.e. "rfidd", "meteringd", "recloserd".

Since Charge Control C can be freely programmed, it is possible that customers add additional device support on their own, writing a customer specific daemon which then replaces the functionality of the factory shipped daemon.

Currently supported internal peripherals using Modbus:

- Electricity meter
    - ABB EV3 012-100
    - BZR Bauer BSM-WS36x-Hxx-1xxx-000x (as SunSpec compatible device)
    - Carlo Gavazzi EM300/ET300/EM100/ET100 series
    - DZG DVH4013
    - Eastron SDM72D-M
    - Eastron SDM230
    - Eastron SDM630 v2
    - Elecnova DDS1946-2P/2M
    - Elecnova DTS1946-4P/4M
    - Klefr 693x/694x
    - Iskra WM3M4/WM3M4C
    - Phoenix Contact EEM-350-D-MCB
    - Socomec Countis E03/04
    - SunSpec compatible meters (meter model 203)
- Recloser devices
    - Geya GRD9M/L-S
- RFID reader
    - StrongLink SL032 (with customized Modbus protocol)

Currently supported internal peripherals using proprietary protocols:

- RFID reader
    - StrongLink SL032 (public available, proprietary protocol)
    - SMART Technologies ID RFID Einbaumodul MCR LEGIC

<u>Note</u>: It should be avoided to use different protocols on the same connector.

The following table documents the default communication parameters for Modbus peripherals used by the charging stack unless configured explicitly. Usually, these defaults are derived from the meter's default settings to allow Plug & Play. But especially in cases where a meter implementation supports several models, it must be cross-checked that the connected meter's (default) settings matches - adapt the configuration of the meter and/or change the charging stack configuration to make it work.

Peripheral Device	**Baud rate**	**Parity**	**Modbus Address**	**Note**
ABB EV3	9600	even	1	Parity cannot be changed on meter devices, so customer needs to configure it in customer.json.
BZR Bauer BSM-WS36x-Hxx-1xxx-000x	19200	even	42	This meter responds very slowly (up to 10s) to each Modbus query. So it is recommended to use it as single device on a dedicated RS-485 port only. Please also consider this when using real-time load balancing.
Carlo Gavazzi EM300/ET300/EM100/ET100 series	9600	none	1	
DZG DVH4013	9600	none	Modbus Address scan is performed	DZG devices ships with parity set to "even" by default, so customer needs to configure it in customer.json.
Eastron SDM72D-M	9600	none	1	Only parity "even" is documented as default in device manuals.
Eastron SDM230	9600	none	1	This Eastron model is shipped with factory defaults set to baud rate 2400 and settings 8E1, so customer usually needs to change baud rate and parity values in customer.json.
Eastron SDM630 v2	9600	none	1	No documented defaults in device manuals.
Elecnova DDS1946-2P/2M	9600	none	1	No documented defaults in device manuals. The factory Modbus address is usually derived from the serial number.
Elecnova DTS1946-4P/4M	9600	none	1	No documented defaults in device manuals. The factory Modbus address is usually derived from the serial number.
Klefr 693x/694x	9600	none	1	
Iskra WM3M4/WM3M4C	115200	none	33	
Phoenix Contact EEM-350-D-MCB	9600	none	1	
Geya GRD9M/L-S	9600	none	3	
Socomec Countis E03/04	38400	none	5	The factory default settings for Modbus communication interface are not documented in the datasheet.
StrongLink SL032 (with customized Modbus protocol)	9600	none	17	

7.3.3 Supported Electricity Meter Features/Measurands

While the supported electricity meters all use Modbus as communication protocol, there are differences in the supported measurands/features by the meters.

Meter Model	Reading via Modbus...				
	Meter Serial Number	Eichrecht Support	Active Power (Overall Consumption)	Active Energy (Overall Consumption)	Voltage/Current/Power per Phase
ABB EV3	yes	no	yes	yes	yes/yes/yes
BZR Bauer BSM-WS36x-Hxx-1xxx-000x	yes	planned	yes	yes	yes/yes/yes
Carlo Gavazzi EM300/ET300/EM100/ET100 series	yes	no	yes	yes	yes/yes/yes
DZG DVH4013	yes	no	yes	yes	yes/yes/no
Eastron SDM72D-M (HW revision: v1)	no	no	yes	yes	no/no/no
Eastron SDM72D-M (HW revision: v2)	yes	no	yes	yes	yes/yes/yes
Eastron SDM230	yes	no	yes	yes	yes/yes/yes
Eastron SDM630 v2	no	no	yes	yes	yes/yes/yes
Elecnova DDS1946-2P/2M	no	no	yes	yes	yes/yes/yes
Elecnova DTS1946-4P/4M	no	no	yes	yes	yes/yes/yes
Klefr 693x/694x	yes	no	yes	yes	yes/yes/yes
Iskra WM3M4/WM3M4C	yes	WM3M4C only	yes	yes	yes/yes/yes
Phoenix Contact EEM-350-D-MCB	yes	no	yes	yes	yes/yes/yes
Socomec Countis E03/04	yes	no	yes	yes	yes/yes/yes

For SunSpec compatible meters it depends on the specific device and model which register values are filled by the meter's firmware. For the single phase meter models, the overall consumption/energy corresponds to the available single phase which is internally handled as L1 phase.

Some meters also support bidirectional counting, so please double check, whether the meter is installed in correct direction. The current charging stack does not yet support bidirectional charging, so it's always looking for import (aka "consumption") registers. Depending on the actual meter protocol implementation, exported current/power reported by the meter might be filtered out and forwarded as zero value in internal MQTT API.

7.3.4 General Assumptions

An EIA-485 bus is not considered a plug and play bus. It is assumed that peripheral devices are connected before powering the charging station, or at least power up simultaneously with the Charge Control C board.

7.3.5 Advices/Requirements for Customer Applications

Since the implementation of peripheral device support on charging stack side consists of multiple daemons, all daemons must coordinate their access to the same UART interface when communicating with devices on the same EIA-485 bus.

So any access to the EIA-485 serial UART device (e.g. /dev/ttymxc0) must be protected by `TIOCEXCL`/`TIOCNXCL` ioctl calls. These ioctls are functionally only ensured for non-root users. For this reason, all daemons accessing serial ports must be run as user "daemon" and group "`dialout`". While factory shipped start scripts take care of this, customers' start scripts should mimic this behavior. Special care must also be taken during development, e.g. when manually starting and testing daemons via SSH to run with appropriate changed uid/guid.

Another point to consider is, that serial accesses of each daemon should be limited to a short time (<1 s). Otherwise, the functionality of other daemons using the same serial port is limited.

On Charge Control C platform, customers writing their own serial port application need be aware about the hardware echo while sending data to the serial port. It is up to the (customer) software to discard this local echo since it is not possible to disable the echo on hardware-side.

Customers who want to access Modbus peripherals are advised to also use libmodbus, an open-source C library for this protocol, as factory shipped daemons do. The libmodbus variant deployed on Charge Control C was already extended with functionality to discard the local echo on library side. Moreover, the TIOCEXCL functionality was added. While the port locking is done transparently, the echo discarding functionality must be enabled by customers' applications by calling `modbus_rtu_set_suppress_echo`.

A customer implementation in C might look like the following example. It detects whether libmodbus provides the `modbus_rtu_set_suppress_echo` call and is thus usable in native PC environments with standard libmodbus as shipped by Linux distributions, but also uses the feature when available/running on a Charge Control C device.

```
...
#include <modbus/modbus.h>
int modbus_rtu_set_suppress_echo(modbus_t *ctx, bool on)
__attribute__((weak));

...

int main(int argc, char *argv[])
{
    /* the libmodbus context */
    modbus_t *mbctx;

    /* set local_echo to true if you run on a platform which has local
hardware echo */
    bool local_echo = true;

    ...

    mbctx = modbus_new_rtu(...);

    if (modbus_rtu_set_suppress_echo) {
        printf("Info: %senabling local echo suppression", local_echo ? ""
: "not ");
        modbus_rtu_set_suppress_echo(mbctx, local_echo);
    } else {
        if (local_echo) {
            printf("Error: local echo suppression support requested, but
no support in libmodbus.");
            exit(1);
        } else {
            printf("Info: libmodbus without local echo suppression
support");
        }
    }

    ...
}
```

Code Block 1 Using libmodbus' Local Echo Suppression Feature from a C program

Our modified source code of libmodbus can be found on Github.

We also included a libmodbus recipe in our Yocto BSP distribution layer which uses our libmodbus repository as source when building/including libmodbus in the firmware image.

7.3.6 Metering Daemon

7.3.6.1 Start-up Behavior

When configured for "dzg" in customer.json and no meter serial number is provided, then during power up of the charging station a single scan of the Modbus address range 01 - 99 is performed by the metering daemon. In worst case this takes up to nearly 60 seconds. While the scan is active, no other access to the EIA-485 bus is possible.

For other meter protocols, no such bus scan is implemented, and the factory defaults of the electricity meter are assumed (i.e. Modbus address). This can be overwritten via customer.json by configuring a Modbus address explicitly.

7.3.6.2 Behavior in case of unavailability

During power up, no further scan attempt is made in case no meter was found during scan or when no meter was found under the expected Modbus address.

However, during normal operation, the metering daemon is safe against temporary disconnects or unavailability.

For example, the Klefr meters make a CRC check after power cycling during which the meters do not respond to Modbus queries and thus cause temporary unavailabilities. In order to workaround this behavior and to prevent reporting power meter failures during charging, a specific timeout of 10 seconds is implemented for these meters, while for all other meter types the timeout is set to 2 seconds before reporting an error.

7.3.7 RFID / Recloser Daemons

In contrast to the metering daemon, when these daemons do not find their peripheral devices during booting up, they continue to try to communicate with the peripherals.

7.4 Mains PLC

This device supports 10 Mbit/s HomePlug Green PHY™ power line communication on mains. This interface is available (if present) as eth2. Please note, that for security reasons this interface does not ship from factory with the Network Management Key (NMK) set to "HomePlugAV" like traditional powerline devices did for a long time to ease installation. During the manufacturing process, a random NMK is generated for each device and installed as factory default setting. This prevents attackers from accessing the device over mains powerline with a well-known NMK.

Board Interface	Linux Interface
Mains PLC	eth2

Table 20 Mains PLC

7.4.1 Pairing

When your HomePlug compatible companion is already setup and working, you are ready to join the powerline network.

For this you need to pair the Charge Control C with your HomePlug compatible companion.

There are currently two different ways of pairing PLC devices with Charge Control C.

Putting into service the Powerline connection by means of the push button method

The push button pairing method is the most famous method. Charge Control C has no push button to activate this method, but the push button can be simulated with on-board tools.

1. Press the powerline security button on the companion (e.g. wallplug adapter) to start the pairing process.
2. Run the following command on Charge Control C - this emulates pressing the pairing button of the evaluation board:

```
root@tarragon:~ $ plctool -B join -i eth2
eth2 00:B0:52:00:00:01 Join Network
eth2 00:01:87:FF:FF:2B Joining ...
root@tarragon:~ $
```

3. You should see the remote powerline adapter after a short while:

```
root@tarragon:~ $ plcstat -t -i eth2
 P/L NET TEI ------ MAC ------ ------ BDA ------  TX  RX CHIPSET FIRMWARE
 LOC STA 002 00:01:87:FF:FF:2B 00:01:87:FF:FF:FE n/a n/a QCA7000 MAC-QCA7000-1.1.3.1531-00-20150204-CS
 REM CCO 001 00:0B:3B:AA:86:55 E0:CB:4E:ED:1F:53 009 009 INT6400 INT6000-MAC-4-1-4102-00-3679-20090724-FINAL-B
root@tarragon:~ $
```

4. You have successfully created a powerline connection.

Putting into service the Powerline connection using software

You can also add the device by means of DAK (Device Access Key, often also called device password or security ID) to an existing powerline network or couple it with a powerline Ethernet adapter. The DAK is indicated in the device labels 2D DataMatrix code of the Charge Control C. It consists of 4 x 4 letters, separated by hyphens.

1. Note this DAK and install the device in the power grid.
2. After the device has been put into service, you can add the device to the existing powerline network using the software of your powerline companion (e.g. FRITZ!Powerline or Devolo Cockpit for powerline ethernet adapter as powerline companion).
3. In doing so, the DAK is to be entered.
4. Please refer to the documentation of your powerline companion for further information about this process.

7.5 Control Pilot / Proximity Pilot

For ISO 15118 / DIN 70121 compliant communication between EVSE and PEV, Charge Control C supports CP (control pilot) and PP (proximity pilot) signaling including Green PHY communication. This Green PHY communication is available on interface eth1.

Since for EVSE/EV communication only IPv6 SLAAC is required, there is no further configuration (IPv4 etc.) necessary for this Linux interface.

Board Interface	Linux Interface
Control Pilot PLC	eth1

Table 21 Control Pilot / Proximity Pilot

Note: The Charge Control boards use a Qualcomm Atheros QCA7000 chip for Green PHY communication on CP line. The shipped QCA7000 firmware configuration contains a default set of prescalers which influence the CP signal level („loudness"). It is recommended to re-check these settings in customer's specific setup and environment and tune them accordingly if necessary.

7.6 Locking motors

Charge Control C provides connectors for 2 locking motors. Currently only the plug lock motor is handled by the Charging stack.

The locking motor outputs (M- and M+) are controlled via GPIO (IN1 and IN2) with a logical behavior.

$M+ = IN1 \wedge \neg IN2$

$M- = IN2 \wedge \neg IN1$

Additionally, for the X9 interface IN1 must be inverted

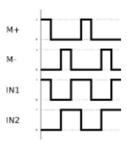


Figure 4 Locking Motor Logic

7.6.1 Plug lock motor X9

Board Interface	Polarity	Linux Interface
IN1	active low	/sys/class/gpio/gpio71
IN2	active high	/sys/class/gpio/gpio72
FAULT	active low	/sys/class/gpio/gpio19
SENSE ADC	-	/sys/bus/iio/devices/iio:deviceX/in_voltage0_raw

Table 22 Plug lock motor X9

7.6.2 Cover lock motor X10

Board Interface	Polarity	Linux Interface
IN1	active high	/sys/class/gpio/gpio73
IN2	active high	/sys/class/gpio/gpio136
FAULT	active low	/sys/class/gpio/gpio25
SENSE ADC	-	/sys/bus/iio/devices/iio:deviceX/in_voltage1_raw

Table 23 Cover lock motor X10

Note: Since the enumeration of the IIO device depends on all connected devices, it is not guaranteed that the ADC is always iio:device0. Therefore, the name of the IIO device should always be checked (name must be 2198000.adc).

7.7 Relays

Board Interface	Signal Name	Linux Interface	MQTT topic
R1	NO_1	/sys/class/gpio/gpio76	port0/contactor/state/target
S1	SENSE_1	/sys/class/gpio/gpio131	port0/contactor/state/actual
R2	NO_2	/sys/class/gpio/gpio77	port0/ventilation/state/target
S2	SENSE_2	/sys/class/gpio/gpio130	port0/ventilation/state/actual

Table 24 Relays

The MQTT topics reflect the usual charging stack configuration where relais 1 is used to switch the (primary) contactor and relais 2 switches an external ventilation device.

7.8 1-Wire

This is a generic 1-Wire interface. It is realised with an I2C to 1-wire bridge. The bridge is handled by the DS2484 1-Wire Linux driver and provides the interface /sys/bus/w1/.

’Application Note 7 - Charge Control C - Thermal Management’ shows an example of how to use the 1-Wire bridge. Since Charge Control C can be freely programmed, it is possible to add device support on your own.

Board Interface	Linux Interface
1-Wire	/sys/bus/w1/

Table 25 1-Wire

7.9 Digital Input & Output

7.9.1 Digital Input

Charge Control C supports up to six digital inputs. All digital inputs have one common adjustable reference level from 0 V until +12 V.

The reference voltage can be set through period time and duty cycle of a PWM in nano seconds [ns].

A 100% duty cycle corresponds to around 12 V reference voltage.

The required duty cycle to a given period time and reference voltage can be calculated with the following formula:

```
duty_cycle[ns]=reference_voltage[V]*period[ns]/12V
```

E.g. reference voltage should be 6 V while reference voltage generator is driven with a 25 kHz PWM signal:

- duty_cycle = 6 V * (1/25000 Hz * 10ˆ9) / 12 V = 20000 ns

Vice versa the set reference voltage can be calculated:

```
reference_voltage[V]=duty_cycle/period*12V
```

Board Interface	Linux Interface	Preconfigured function	Polarity
DIG_IN_1	/sys/class/gpio/gpio121	emergency switch	active high
DIG_IN_2	/sys/class/gpio/gpio122	RCD feedback	active high
DIG_IN_3	/sys/class/gpio/gpio124	authorization key switch	active high
DIG_IN_4	/sys/class/gpio/gpio123		
DIG_IN_5	/sys/class/gpio/gpio116		
DIG_IN_6	/sys/class/gpio/gpio119		

Table 26 Digital Input

To use the digital inputs in Linux userspace, the corresponding GPIO may need to be exported and set to correct direction.

e.g.:

```
echo "119" > /sys/class/gpio/export
echo "in"  > /sys/class/gpio/gpio119/direction
```

Board Interface	Linux Interface
reference voltage duty cycle	/sys/class/pwm/pwmchip1/pwm0/duty_cycle
reference voltage period time	/sys/class/pwm/pwmchip1/pwm0/period

Table 27 Board and Linux Interface

7.9.2 Digital Output

Charge Control C supports up to six digital outputs.

The digital outputs are real push-pull drivers. Up to 100 mA can be drawn from a single output.

Board Interface	Linux Interface	Preconfigured function	Polarity
PUSH_PULL_OUT_1	/sys/class/gpio/gpio84	status LED	active high
PUSH_PULL_OUT_2	/sys/class/gpio/gpio85		
PUSH_PULL_OUT_3	/sys/class/gpio/gpio86		
PUSH_PULL_OUT_4	/sys/class/gpio/gpio87	RCD test trigger	active high
PUSH_PULL_OUT_5	/sys/class/gpio/gpio88		
PUSH_PULL_OUT_6	/sys/class/gpio/gpio89		

Table 28 Digital Output

Status LED behavior:

- solid high = ready
- 1000 ms high. 1000 ms low = charging
- 100 ms high, 100 ms low = error

To use the digital outputs in Linux userspace, the corresponding GPIO may need to be exported and set to correct direction.

e.g.:

```
echo "89"  > /sys/class/gpio/export
echo "out" > /sys/class/gpio/gpio89/direction
```

7.10 4-wire fan

Charge Control C uses common tools like the hwmon/ thermal framework of the Linux Kernel.

As per default, Charge Control C only uses the thermal sensor on the i.MX6ULL SoC and tries to regulate the temperature via the X3 fan connector. The responsible program is the shell script "fancontrol" (taken from lmsensors) which is started automatically during boot on Charge Control C 300.

8 Board Connections

Charge Control C has 14 connectors (X1...X14) and three pinheaders (JP1...JP3) as shown in Figure connectors of Charge Control C.

The pinheaders are for configuring (JP1), debugging (JP2) and expanding (JP3) purposes.

The connectors are used to establish the connection to the external EVSE periphery.

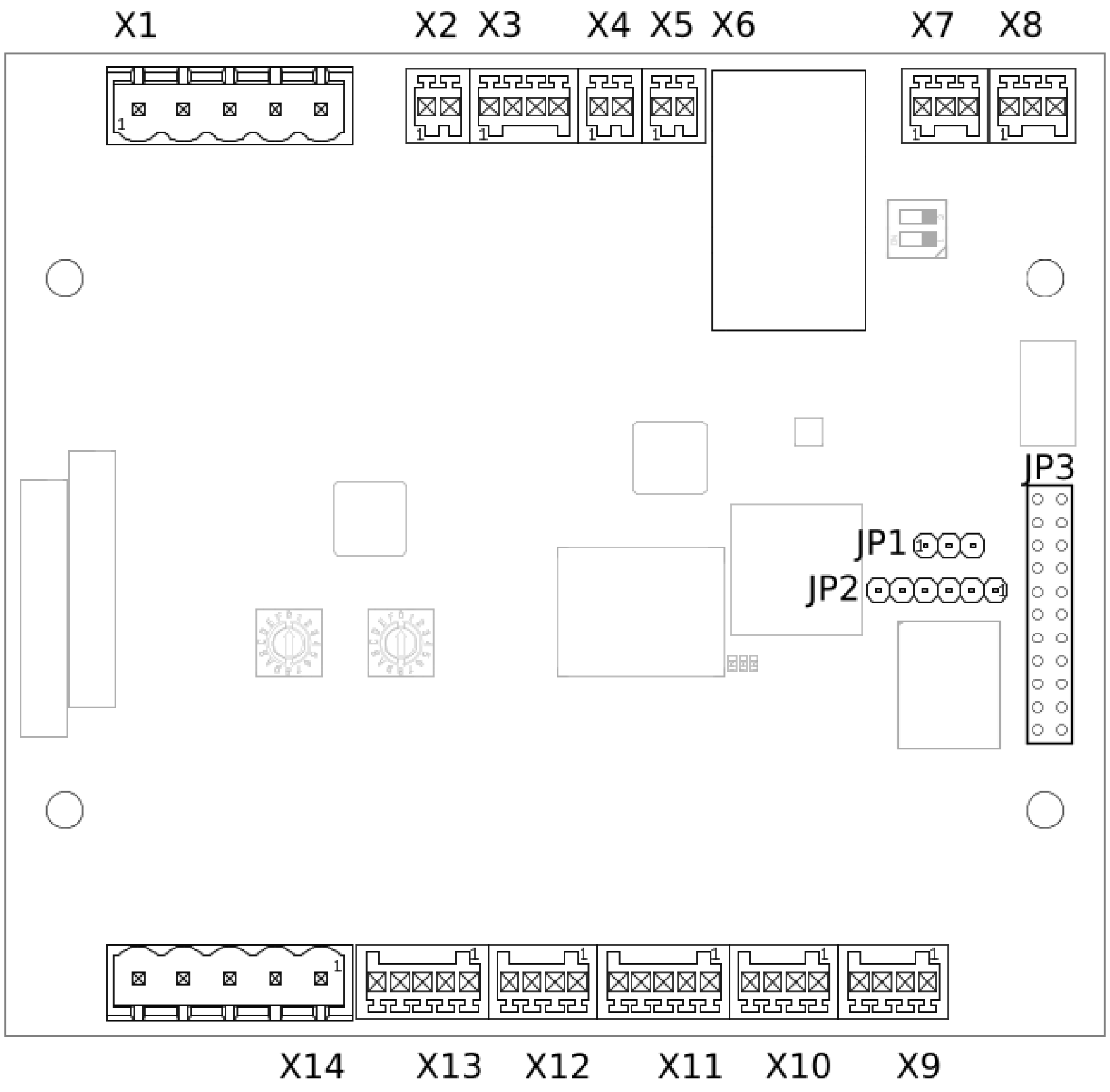


Figure 5 Connectors of Charge Control C

Please refer to the according section of the datasheet for electrical input and output values.

8.1 X1 - mains

The connector is used to connect the mains voltage to it. It provides a filtered mains output.

Connecting the AC/DC-power-supply to this output port helps improving PLC signal integrity while using noisy power supplies.

Up to 250 mA can be drawn from this port.

Pin#	Signal	Note
1	L_FILTERED	filtered mains output L

2	N_FILTERED	filtered mains output N
3	L	mains input L
4	N	mains input N
5	PE	protective earth, also board GND reference level

Table 29 X1 - mains

8.2 X2 - DC in

This product needs DC supply voltage input.

Pin#	Name
1	+12V
2	GND

Table 30 X2 - DC in

8.3 X3 - fan

Charge Control C provides an output for 4-Wire pulse width modulation (PWM) controlled fans.

Pin#	Signal
1	CONTROL
2	SENSE
3	+12V
4	GND

Table 31 X3 - fan

ATTENTION! Most fans have a pullup on the tach signal resulting in signals exceeding the absolute maximum rating of the fan interface. This could potentially destroy the whole device.

8.4 X4 - 1-Wire

This product provides a 1-Wire master interface where 1-Wire downstream slave devices (such as temperature sensors) can be connected.

Pin#	Signal
1	1W_IO
2	GND

Table 32 X4 - 1-Wire

8.5 X5 - Control and Proximity pilot

The connector is used for connecting to EV. It provides the signals for control pilot, proximity pilot as well as Green-PHY powerline communication.

Pin#	Signal
1	Control pilot
2	Proximity pilot

Table 33 X5 - Control and Proximity pilot

8.6 X6 - Ethernet - USB

X6 is a stacked Ethernet and USB connector.

8.6.1 Ethernet

The Ethernet port supports 10/100 MBit/s and has embedded link and activity LED indicators.

8.6.2 USB

Charge Control C usually acts as USB host at this port. Up to 500 mA can be drawn from this port. It also can be used for provisioning purposes.

8.7 X7 - EIA-485 1

The first EIA-485 (RS-485) of Charge Control C is a galvanically isolated one.

Pin#	Signal

1	B
2	A
3	REF

Table 34 X7 - EIA-485 1/2

8.8 X8 - EIA-485 2 / CAN

This connector is used to connect to the i.MX6ULL using CAN or EIA-485 (RS-485). Whether X8 is a CAN or EIA-485 interface is an assembly option of the board. Please see ordering information to select the appropriate variant.

Both interfaces are referenced to GND.

Pin#	Signal	
	CAN	**RS-485**
1	H	B
2	L	A
3	GND	

Table 35 X8 - EIA-485 2/2 - CAN

8.9 X9 / X10 - Locking Motor

X9 and X10 have the same pinout.

Pin#	Signal
1	M-
2	M+
3	SENSE
4	GND

Table 36 X9 / X10 - locking motor

Only X9 supports motor lock failsafe opening in case of power loss.

There are locking motors available with different internal feedback circuity. Attach them according to the following images.

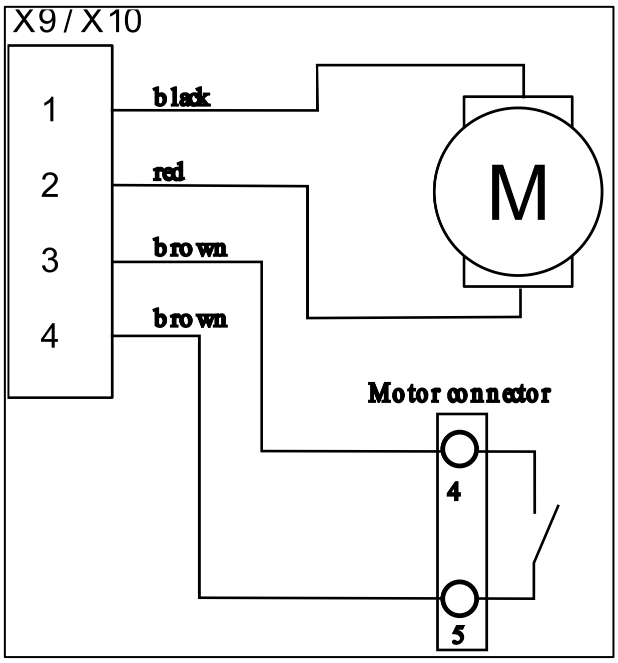


Figure 6 Scame 200.23260BS

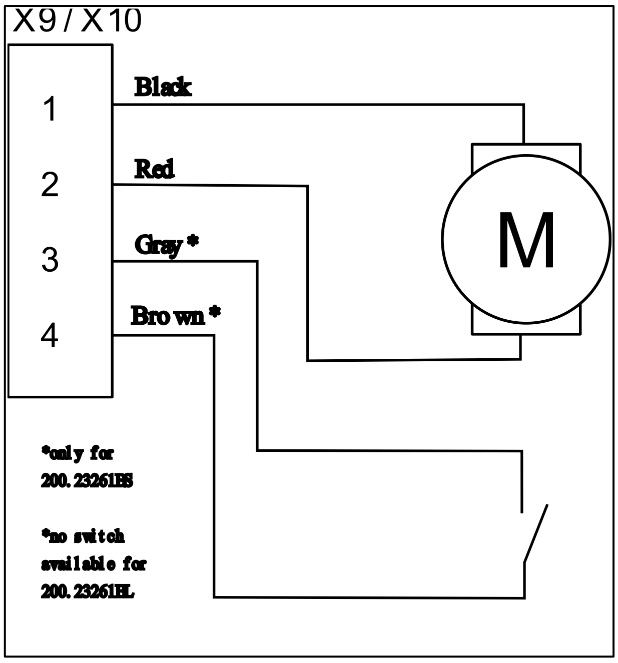


Figure 7 Scame 200.23261BS/BL

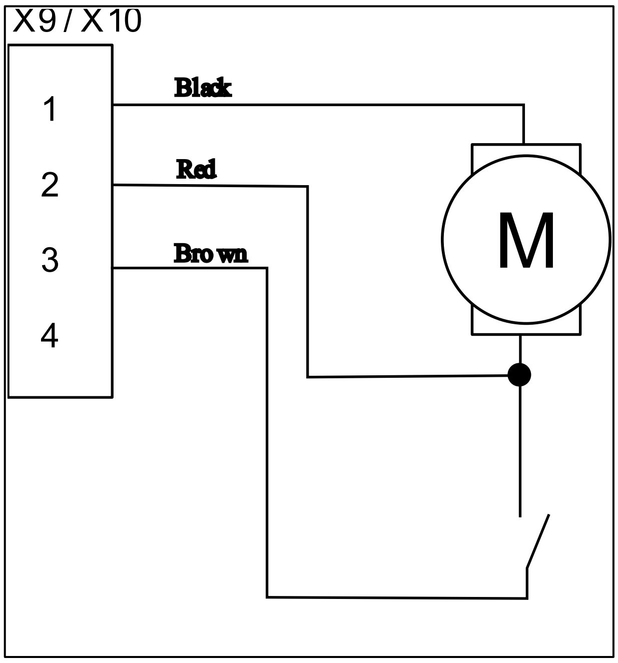


Figure 8 Scame 200.23261BP

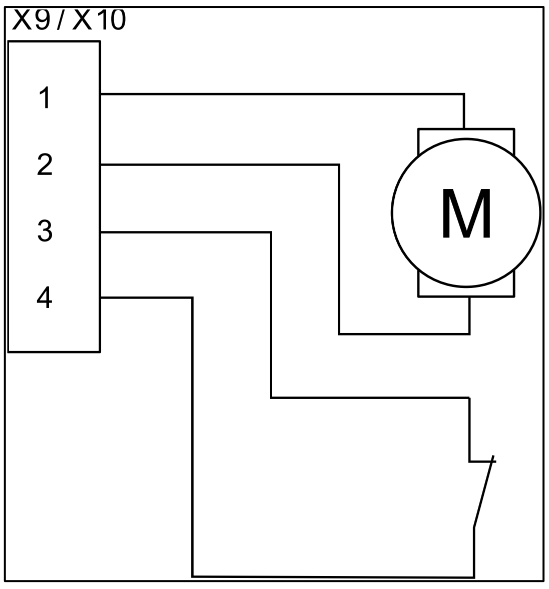


Figure 9 Typical Küster 02S, Phoenix Motor

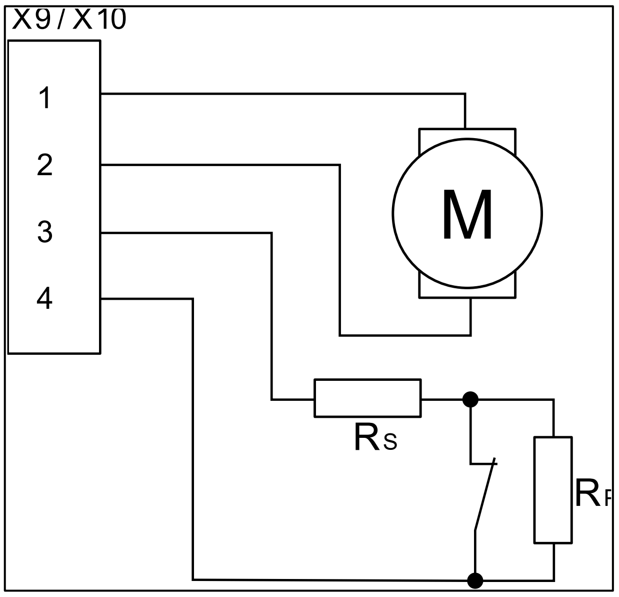


Figure 10 Typical Küster 04S Motor, Rs=1kΩ, Rp=10kΩ

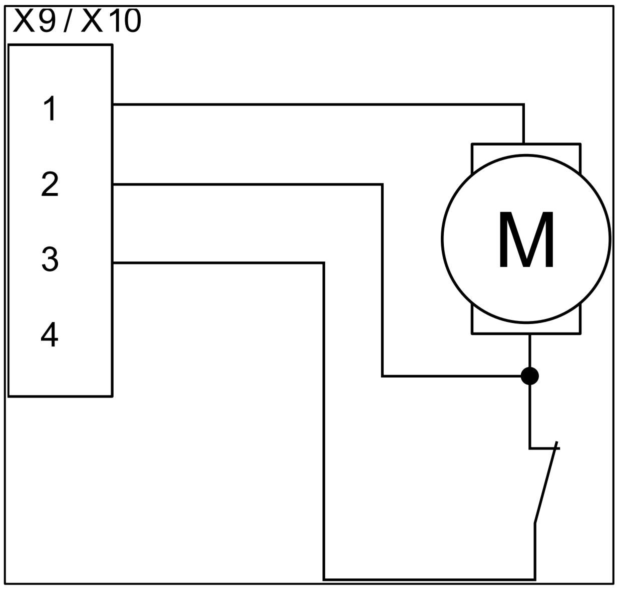


Figure 11 Typical Hella, Bals, Menekes & Walther Werke Motor

8.10 X11 - Digital In

This port supports digital inputs with digital adjustable reference level of up to +12 V.

Pin#	Signal
1	DIG_IN_1
2	DIG_IN_2
3	DIG_IN_3
4	DIG_IN_4
5	GND

Table 37 X11 - digital in

8.11 X12 - Digital In and Out

This port supports two digital inputs with digital adjustable reference level of up to +12 V and two digital outputs. The outputs are real push-pull drivers. Up to 100 mA can be drawn from a single output.

Pin#	Signal
1	DIG_IN_5
2	DIG_IN_6
3	PUSH_PULL_OUT_6
4	PUSH_PULL_OUT_5

Table 38 X12 - digital in and out

8.12 X13 - Digital Out

This port supports digital outputs with real push-pull drivers. Up to 100 mA can be drawn from a single output.

Pin#	Signal
1	PUSH_PULL_OUT_4
2	PUSH_PULL_OUT_3
3	PUSH_PULL_OUT_2
4	PUSH_PULL_OUT_1
5	GND

Table 39 X13 - digital out

8.13 X14 - Relays

Two normally open (NO) relays are populated on Charge Control C. They are able to handle mains voltage level. One sense input for every switched load is supported.

Pin#	Signal	Description
1	COM_L	mains L input
2	NO_1	relay #1 switched L output
3	SENSE_1	relay #1 sense input
4	NO_2	relay #2 switched L output
5	SENSE_2	relay #2 sense input

Table 40 X14 - relays

Both sense inputs need reference to Neutral (N). Leave it open for "inactive" feedback. Tie it to Neutral (N) for "active" feedback.

8.14 JP1 - Bootmode Jumper

Jumper Position	Bootmode
1-2	USB serial downloader
2-3, or removed	eMMC internal boot

Table 41 JP1 - bootmode jumper

8.15 JP2 - Debug UART

JP1	Signal
1	GND
2	not connected
3	not connected
4	RX of i.MX6ULL
5	TX of i.MX6ULL
6	not connected

Table 42 JP2 - debug UART

This pinout is compatible with a variety of USB/RS232 adapters. Preferably you should use the FTDI cable "TTL-232R-3V3" or similar. Do not use long wires to connect the debug UART.

ATTENTION! Do not use generic RS232 adapters, as they usually have 12 V voltages for their logic signals. The pins here are only 3.3 V tolerant. You may damage the debug UART with incompatible adapters.

8.16 JP3 - Expansion Port

JP3 is a connector for additional expansion boards. Please contact sales team for details.

8.17 Mating Connectors

Header Designator	Pin Count	Matching Terminal Block	Rated Wiring Solid Wire		Rated Wiring Stranded Wire	
			Metric	AWG	Metric	AWG
X1, X14	5	Metz Connect SP06505VBNC	0.08 mm² - 2.5 mm²	AWG 28 - AWG 12	0.08 mm² - 2.5 mm²	AWG 28 - AWG 12
X2, X4, X5	2	Würth Elektronik 691381000002	0.08 mm² - 0.5 mm²	AWG 28 - AWG 20	0.08 mm² - 0.5 mm²	AWG 28 - AWG 20
X8	3	Würth Elektronik 691381000003				
X3, X9, X10, X12	4	Würth Elektronik 691381000004				
X11, X13	5	Würth Elektronik 691381000005				

Table 43 Mating Connectors

Note: Terminal blocks of alternative suppliers might have a different count or position of the "coding noses" and therefore might not fit.

9 Use Cases

In the following sections typical use cases for the Charge Control C family are shown.

Signals and components drawn with light gray color are optional signals and components and are not required for successful charging behavior.

9.1 IEC 61851 AC charging

A basic AC charging station including IEC 61851 conform proximity detection, control pilot function and PWM charging signal can be built up with every member of the Charge Control C family.

Different kinds of IEC 61851 compliant charging stations are shown below.

9.1.1 One phase charging

A typical Charge Control C - 100 based EV charging station is shown by Figure Use Case Charge Control C - 100 - 1 Phase Charging and consists of:

- Charge Control C - 100
- 12 V - AC/DC converter
- Type-2 cable
- Mains contactor
- RCD Type B with feedback signal
- Status LED
- Circuit breaker
- Emergency switch

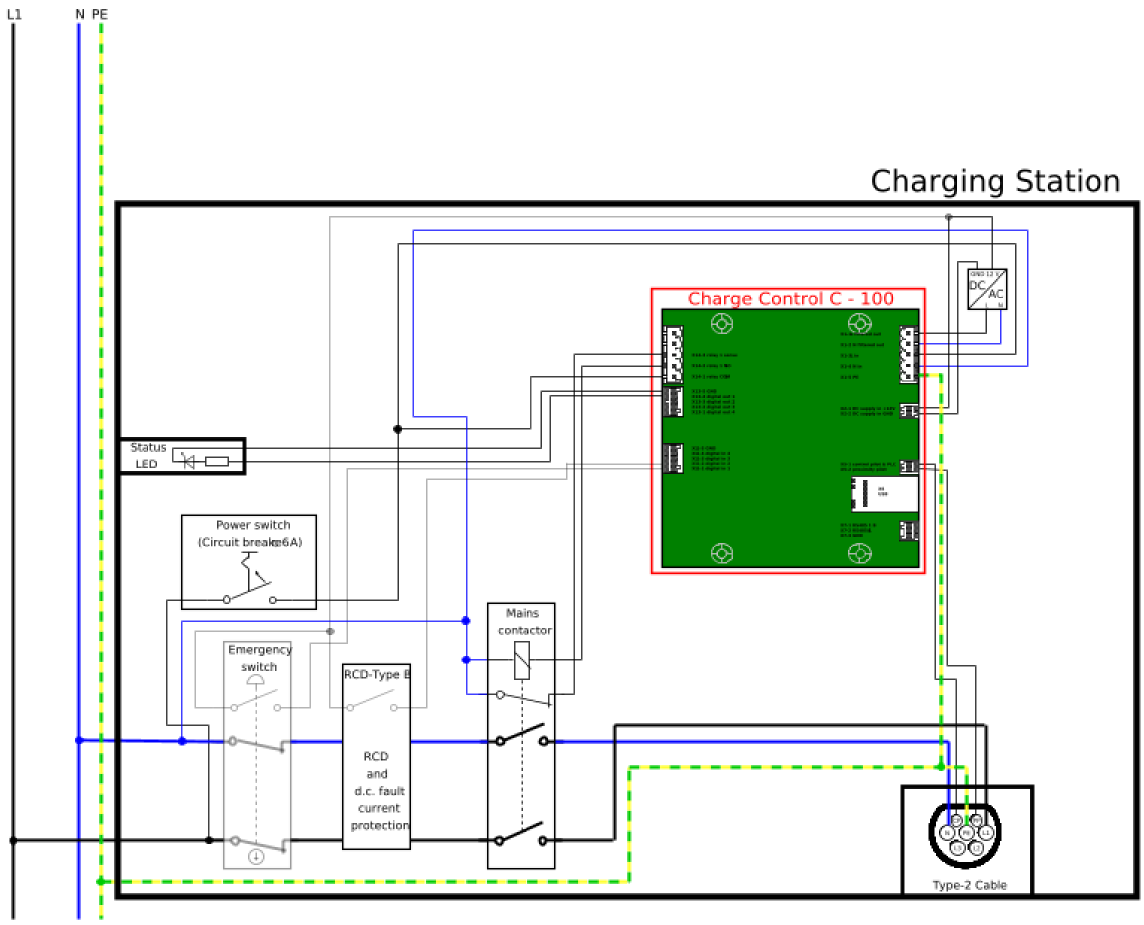


Figure 12 Use case Charge Control C - 100 - 1 Phase Charging

9.1.2 Three phase charging

The realization of 3 phase EV charging stations are possible with all variants of Charge Control C. Figure Use Case Charge Control C - 100 - 3 Phase Charging shows Figure Use Case Charge Control C - 100 - 1 Phase Charging but in 3 phase configuration.

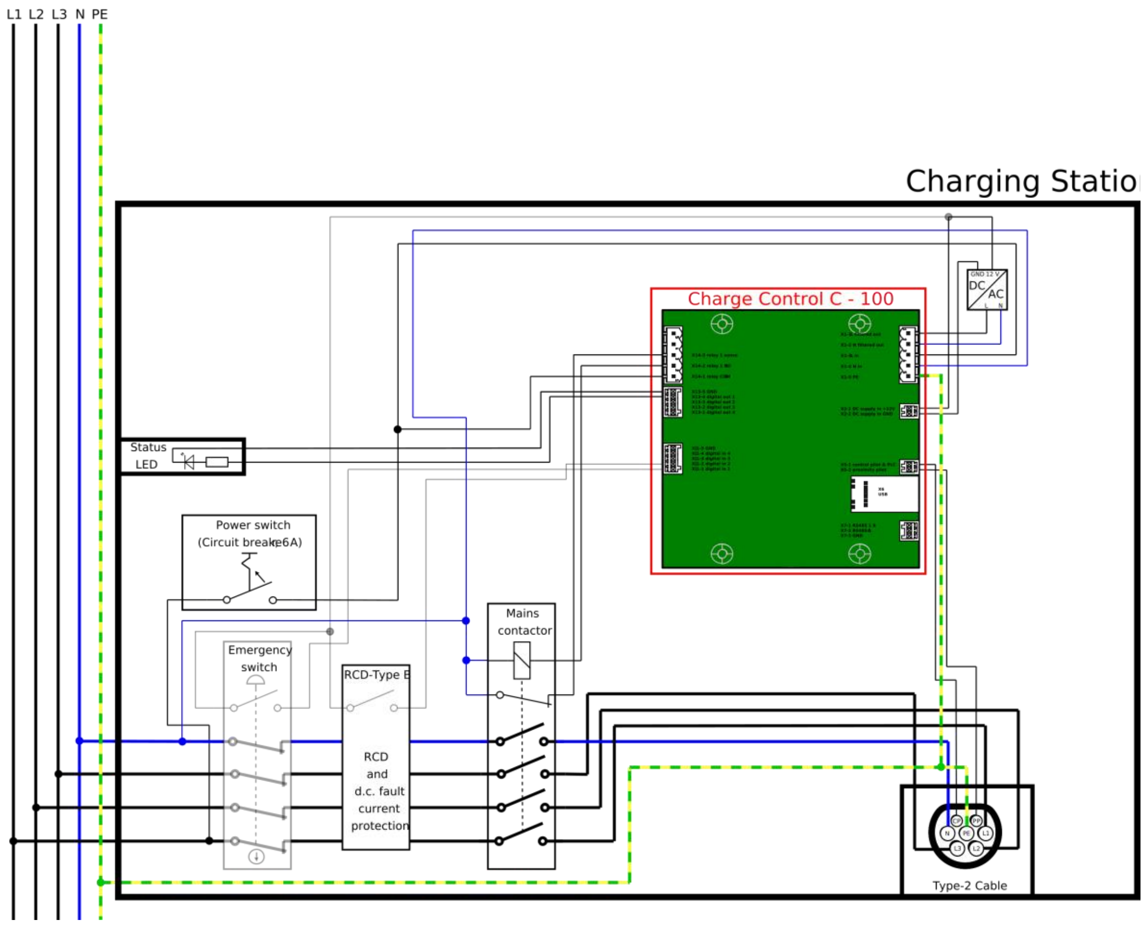


Figure 13 Use case Charge Control C - 100 - 3 Phase Charging

9.2 ISO 15118 AC charging

AC charging stations with ISO 15118 control and proximity pilot signals as well as the PWM charging signal, conform to IEC 61851, can be built up with Charge Control C 200/300.

Different kind of ISO 15118 compliant charging station are shown below.

9.2.1 Type-2 cable connection

A typical Charge Control C - 200 based EV charging station is shown by Use case Charge Control C - 200 - with fixed charging cable and consists of:

- Charge Control C - 200
- 12 V - AC/DC converter
- Type-2 cable
- Mains contactor
- RCD Type B with feedback signal
- Status LED
- Circuit breaker
- Emergency switch
- Ventilation for the charging area
- Smart meter (RS-485)
- Temperature sensors (1-Wire)

- RFID reader (RS-485)
- Display (RS-485)
- Backend (Connected via Ethernet)

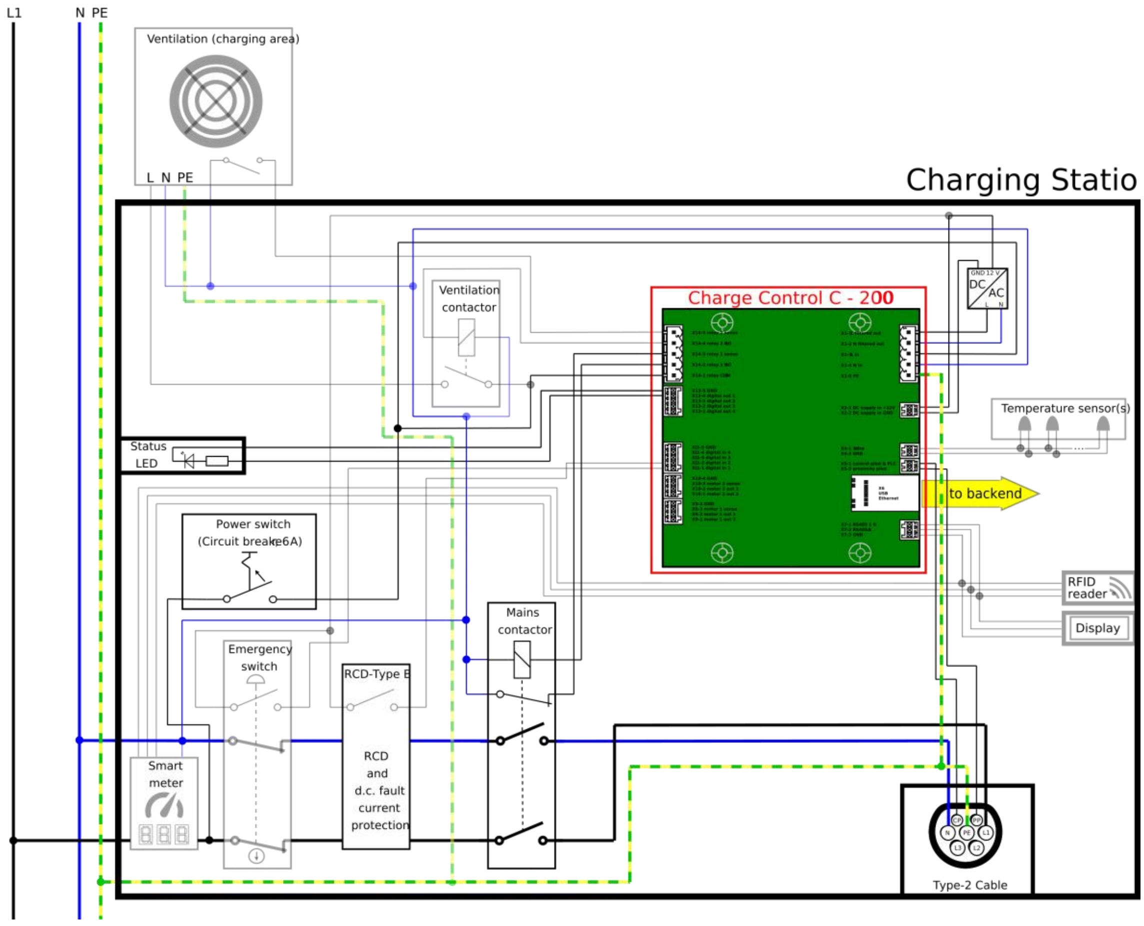


Figure 14 Use case Charge Control C - 200 - with fixed charging cable

9.2.2 Type-2 inlet connection

Charge Control C is capable of driving Type-2 Inlet locking motors. Therefore, it is possible to build charging stations with Type-2 inlet instead of fixed Type-2 cable. Figure Use case Charge Control C - 200 - with pluggable charging cable shows Figure Use case Charge Control C - 200 - with fixed charging cable-configuration but with Type-2 inlet and interlock instead of fixed Type-2 cable.

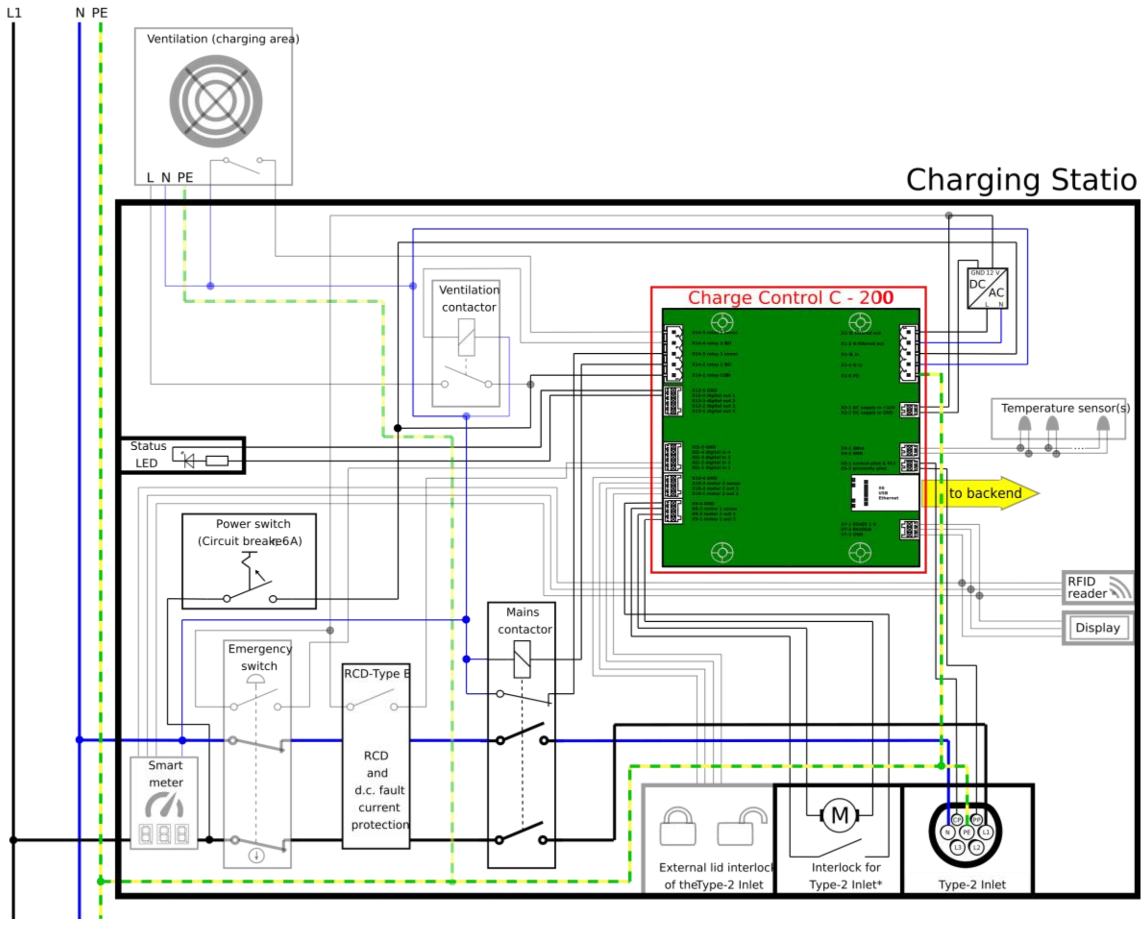


Figure 15 Use case Charge Control C - 200 - with pluggable charging cable

9.2.3 Three phase charging

Only Charge Control C - 300 supports the full set of available connectors.

A typical Charge Control C - 300 based EV charging station consists of:

- Charge Control C - 300
- 12 V - AC/DC converter
- Type-2 cable
- Mains contactor
- RCD Type B with feedback signal
- Status LED
- Circuit breaker
- Emergency switch
- Ventilation for the charging area
- RS-485 devices (Smart meter, RFID reader, Display) distributed over two separate RS-485 interfaces
- Temperature sensors (1-Wire)
- Backend (Connected via Ethernet)
- Enclosure ventilation

- supports more input and output ports for additional I/O-devices

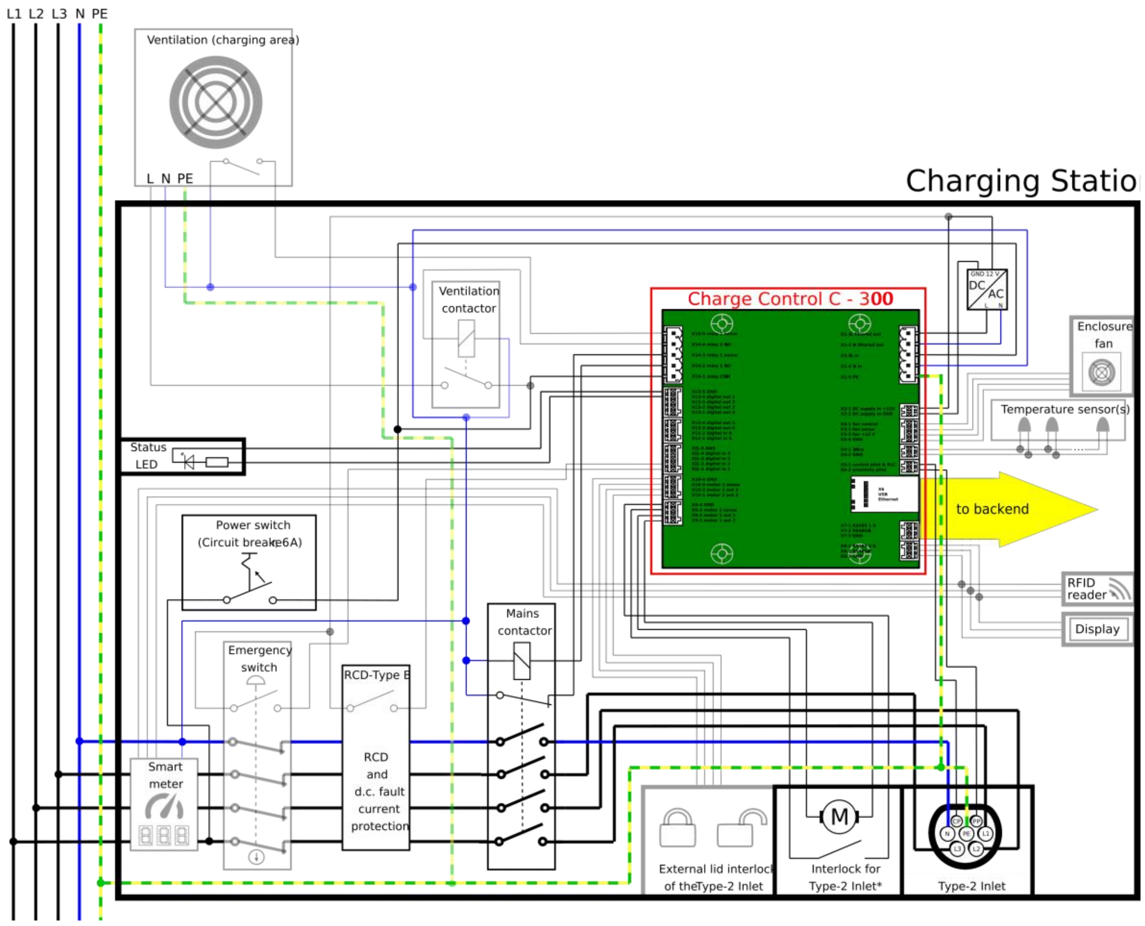


Figure 16 Use case Charge Control C - 300 - in 3 phase configuration and with pluggable charging cable

10 Programming

Charge Control C is shipped with pre-flashed firmware including the Charge Control charging stack. However, it is possible for customers to add new programs and customer software and/or modify none-charging stack configuration files. The shipped firmware was created with Yocto, a project to create custom Linux distributions for embedded devices. Please have a look at the project's website at https://www.yoctoproject.org/ to get familiar with it.

To create a Linux system for Charge Control C, please follow our Board Support Package documentation in our public Github repository at: https://github.com/chargebyte/chargebyte-bsp

Some general notes and recommendation for custom software development:

- Develop your customer software on your local PC Linux environment. Here you can use compiler, debugger etc. you are familiar with. Since Charge Control stack's API is provided via MQTT, you can simply setup one "developer" Charge Control C device and access the stack's API via Ethernet network. If everything works as expected in this setup, then switch to cross-compiling for the target system.
- Use autotools or cmake and pkg-config in your customer software projects as build environment. Prefer such mature and proven tools since these are widely supported and understood, and cross-compiling with these tools is usually easy. Also Yocto itself has rich support for those widely used tools.
- If you start your project from scratch, have a look at libraries which are already required by Charging Stack and/or Linux distribution. Re-use these libraries to keep the overall firmware footprint small. The benefit is when updating the boards, it will take less time when transferring the firmware update image and flashing it to internal storage.
- You have to decide how your own software components will finally interact with the Charge Control stack. For example when writing a daemon for a custom EIA-485 peripheral device: does it need to run in parallel to the factory shipped daemons (e.g. meteringd, rfidd)? Are factory daemons just disabled per configuration file (customer.json), or are such daemons even removed completely from your custom firmware image? In the first case, please also have a look at section Advices/Requirements for Customer Applications.

10.1 Firmware Update Customization and Signing

After making sure that the customized software is working on the board, there might be the requirement to pack this with our pre-flashed firmware to create your own firmware update file which should be signed. This facilitates the production process. The following steps illustrate how to create your own signed firmware update image. For this you would need a native Linux machine or a Linux virtual machine which includes the tool "RAUC" (https://rauc.readthedocs.io/en/latest/index.html). RAUC is the framework we use for performing our firmware update. During the steps, the tool will be used to extract our firmware image, re-pack it including your customized software and sign the new firmware update image. Note that you need your own Public Key Infrastructure (PKI) to sign firmware update images later.

1. Download chargebyte's digital certificate found on chargebyte's website which is used to validate firmware update images distributed by chargebyte GmbH.
2. Download the latest firmware update image found on chargebyte's website.
3. Download and install RAUC tool for the host environment; follow the guide here: https://github.com/rauc/rauc/#host-build-prerequisites It is also possible, that your Linux distribution already ships with pre-compiled packages which just need to be installed with your package management system. In this case, you can skip this step.

```
$ sudo apt-get install build-essential automake libtool libdbus-1-
   dev libglib2.0-dev libcurl3-dev libssl-dev libjson-glib-dev
$ git clone https://github.com/rauc/rauc
$ cd rauc
$ ./autogen.sh
$ ./configure --prefix=/usr
$ make
$ sudo make install
$ cd ..
```

4. Create your own PKI if you do not have one already (see https://rauc.readthedocs.io/en/latest/advanced.html#security).
5. Extract the root filesystem image shipped by chargebyte from the firmware update image file. Note that the directory "bundle-staging" will be created, and the content of the firmware update image file will be extracted into it.

```
$ rauc extract --keyring=<chargebyte_certificate>.crt
   <shipped_firmware>.image bundle-staging
```

6. Mount the ext4 root filesystem image as a loop device.

```
$ sudo mkdir -p /tmp/rootfs
$ sudo mount bundle-staging/core-image-minimal-tarragon.ext4
   /tmp/rootfs -o loop
```

7. Modify and extend the root file system image with your customized software by changing the files/directories below the mountpoint.
   a. Install your PKI certificate which is used later during firmware updates to verify your firmware update file. For this step you need to copy your PKI certificate, created in step 4, and place it under /`tmp/rootfs/etc/rauc/`. Then, replace the `/tmp/rootfs/etc/rauc/keyring.pem` symlink with your PKI certificate.

   ```
   $ cp <your_CA_certficate>.crt /tmp/rootfs/etc/rauc/
   $ cd /tmp/rootfs/etc/rauc/
   $ ln -sf <your_CA_certficate>.crt
      /tmp/rootfs/etc/rauc/keyring.pem
   ```

   b. Copy your additional files, software components etc. from your developer board into this corresponding directory below `/tmp/rootfs` .
8. Unmount the loop device with

```
$ sudo umount /tmp/rootfs
```

9. Make sure that the customized filesystem is in a clean state. This is important since otherwise, the installation process and/or the production process would fail.

```
$ fsck.ext4 -f bundle-staging/core-image-minimal-tarragon.ext4
```

10. This step only needs to be done, if you want chargebyte to intially flash your own firmware during the manufacturing process of the Charge Control C boards. Create MD5 hashsum of the file with the following command. This hash is used during production process to ensure that the image file is not altered but installed correctly into flash. Please send us this hash.

```
$ md5sum bundle-staging/core-image-minimal-tarragon.ext4
```

11. Pack your modified root filesystem image into a firmware update file. The firmware update file must be signed with your PKI using the RAUC tool.

```
$ rauc bundle --keyring=<your_CA_certficate>.crt --
   key=<your_key>.key --cert=<your_certificate>.crt bundle-staging
   <your_firmware_update>.image
```

12. Test the firmware update image file. On the test board, transfer your PKI to the board via SFTP to `/etc/rauc` folder so that the board accepts your firmware updates. The symbolic link refers originally to our chargebyte certificates, so you probably would need to change this. After this, the board should be rebooted.

```
$ cd /etc/rauc
$ ln -sf <your_CA_certficate>.crt /etc/rauc/keyring.pem
$ reboot
```

13. Transfer your newly created firmware update image <your_firmware_update>.image via SFTP to `/srv` folder on your test board.

14. Install the image via SSH or debug UART with the command `rauc install /srv/<your_firmware_update>.image`. Wait until the update is installed and reboot the test board.

10.2 Board customization with USB

For an easier customization of the boards e.g., modification of configuration files or updating to a new customized firmware, a USB flash drive can be used. This is a two-step process where in the second step a script, which you need to write, would be triggered automatically to perform the customization you want to the board. However, for this script to run, a first step of replacing our `/etc/rauc/keyring.pem` with your CA certificate must be performed. This step is also done automatically when you insert a USB flash drive in the USB port of the board. Below is a description of how this feature behaves, and the requirements that must be fulfilled for a successful process.

10.2.1 Replacement of /etc/rauc/keyring.pem

1. Send us your pubilc CA certificate so that we can sign i.e., cross-sign it for you with our certificates.

2. On a USB flash drive, place your cross-signed CA certificate together with the corresponding signature file. The pair has to be named as follows: `fwupdate-keyring.pem` & `fwupdate-keyring.pem.p7s`. You would get this pair from us as a result of step 1.

3. Insert the USB flash drive in the USB port of the board. The following behavior will be triggered automatically:

    a. The onboard yellow diagnostics LED is turned on to indicate the detected USB flash drive.

    b. It is checked whether the pair `fwupdate-keyring.pem` and `fwupdate-keyring.pem.p7s` exist on the USB flash drive.

    c. It is checked whether `fwupdate-keyring.pem` is contains at least one X.509 certificate.

    d. It is checked whether the signature of `fwupdate-keyring.pem` is valid against the current certificate found in `/etc/rauc/keyring.pem`. This is originally a symbolic link referring to our CA certificates. Therefore, step 1 is needed.

e. If the verification has succeeded, the current `/etc/rauc/keyring.pem` will be deleted, and the content of `fwupdate-keyring.pem` is saved as new `/etc/rauc/keyring.pem`. From this point in time, this new certificate is active for all later uses, i.e. further customizations but also regular firmware updates.

10.2.2 Performing customization through autorun.sh

This assumes that the first step has successfully ended, and the file `/etc/raun/keyring.pem` now contains your CA certificate. However, this will also work later at any time, as long as the CA certificate matches to the script signature.

1. Write a script that contains the customization you want to do. This has to be named `autorun.sh`.
2. Sign it by your key and certificate. The resulting signature file must be named `autorun.sh.p7s`.
3. Place the pair on a USB flash drive and insert it in the USB port of the board. The following behavior will be triggered automatically:
    a. It is checked whether the pair `autorun.sh` and `autorun.sh.p7s` exist on the USB flash drive.
    b. It is checked whether the signature of `autorun.sh` is valid against the current certificate found in `/etc/rauc/keyring.pem` . Remember, that this usually now contains your CA certificate.
    c. If the verification has succeeded, the current working directory is changed to the mount point where the USB flash drive was mounted, and the `autorun.sh` script is executed by means of `/bin/sh`. The changed working directory should make it easier to reference files you want to install from the USB flash drive.

10.2.3 Notes

- If you call `reboot` within your `autorun.sh` script, this will not end the execution of the script immediately, but the upcoming lines of the script will continue to be executed until the reboot command is propagated through the system.
- Every time you insert the USB flash drive or perform a reboot, we prevent re-execution of your customized `autorun.sh` script if it has not been changed. This does not depend on the success of your script, i.e., the return value of the script. This is done by saving the MD5 checksum of `autorun.sh` found on the USB flash drive in the file /`var/cache/usb-autorun/executed.list`. You can control this behavior in your `autorun.sh` by e.g., removing the `executed.list` file completely or just deleting single MD5 checksums from it.
- Please keep in mind, that certificate verifications are done without time checking since it cannot be ensured that the device has a valid date/time at the moment when you want to use this feature.
- To sign `autorun.sh` and get `autorun.sh.p7s` as a signature file, the following commands can be used - typically on your Linux developer host system:

```
openssl smime -sign -outform DER -binary -inkey <your-releasemanager-
key>.key -signer <your-releasemanager-certificate>.crt -in autorun.sh -out
autorun.sh.p7s
```

- To verify `autorun.sh` against your CA certificate, the following command can be used:

```
openssl smime -verify -no_check_time -inform DER -CAfile
<your_CA_certificate>.crt -content autorun.sh -in autorun.sh.p7s
```

10.3 Recovery of customer.json

It may occur that the configuration file under `/etc/secc/customer.json` gets damaged either by being corrupted or deleted. To make the `customer.json` more reliable, a startup script was added in firmware 3.3.0 which checks for the existence of `/etc/secc/customer.json` and that it is not an empty file. If the file does not exist or is empty, the script recovers it from a previously created backup file created by configd under `/var/backups/customer.json` , or -in worst case- from the factory-default version under `/etc/secc/customer.json.shipped`.

11 Firmware upgrade

The following sections describe different ways to install a firmware upgrade on your Charge Control product. Please ensure that the power supply is stable during a firmware upgrade. In case of an unsuccessful update or of power loss during the installation of the update, the Charge Control device performs a rollback to the previous stable version of the charging firmware. For more information about the rollback mechanism, see the section Rollback mechanism. The update is finished when the board is rebooted and the green LED1 switches from blinking to steady-on. The board can now be safely switched off by turning off the power supply.

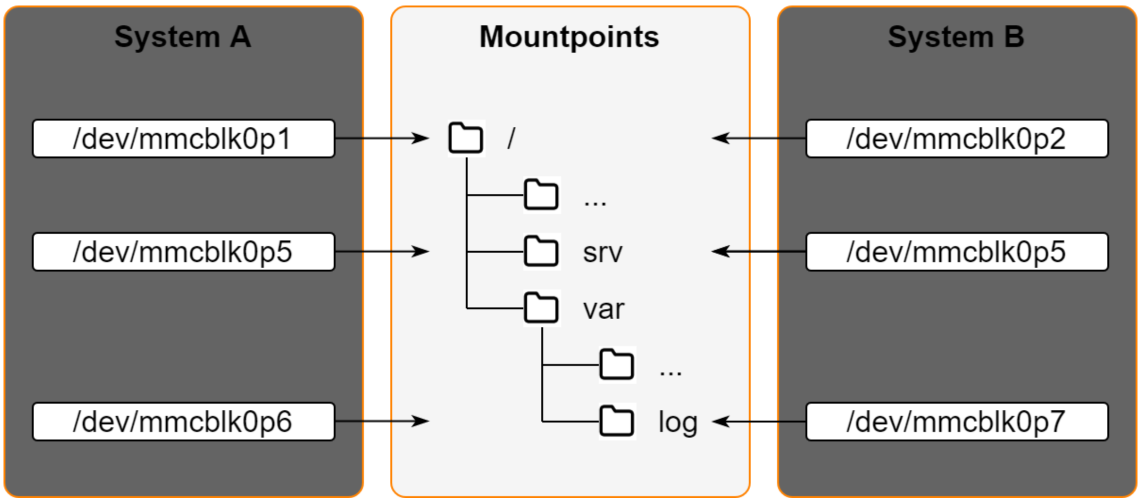
The currently installed firmware can be checked with the MQTT topic "`ci/global/version/charging_software`", whose value is determined at boot and retained. This topic can also be used as a general indication that the charging software is running and operational.

11.1 Partitioning

The internal eMMC storage of a Charge Control device is divided into several partitions. The main aim is to have two independent systems available, i.e. system A and system B. This allows to run firmware updates in background while performing normal charging operation and then switch to the updated system with a fast restart of the device. This also allows to support a rollback mechanism in case of failures during firmware updates. In other words, during a firmware update, the active root file system switches from A to B or vice versa, leaving the other as rollback.

Partition	Size	Description
/dev/mmcblk0p1	1 GB	Root file system A
/dev/mmcblk0p2	1 GB	Root file system B
/dev/mmcblk0p3		Extended Partition Container
/dev/mmcblk0p5	1 GB	Data Partition (/srv). This partition can be accessed by both root file systems and will be not changed during update process.
/dev/mmcblk0p6	128 MB	Logging file system A (/var/log)
/dev/mmcblk0p7	128 MB	Logging file system B (/var/log)

eMMC Partitioning



Filesystem Mountpoints

11.2 Update via USB

Preparation of the USB update

1. Download the firmware update image file onto your workstation. The file size is about 30 MB in current standard configuration.
2. Plug a USB flash drive into your workstation.
3. Format the USB flash drive as EXT2/3/4, FAT16/32, exFAT or NTFS.
4. Copy the firmware update image file (*.image) onto the USB flash drive's root directory.

Notes:

- The exFAT filesystem is supported since firmware version 3.1.0.
- Do not place multiple *.image files for Charge Control onto the root folder of the USB flash drive, since it is not guaranteed in which order the files are tried and applied.

Updating the Charge Control Firmware

1. Connect the board to the power supply.
2. Wait until the board is booted.
3. Connect to the board via SSH or Debug UART to backup all your own implementation and configuration files.
4. Plug in the USB flash drive with the Firmware Update Image file in the USB port of the board.
5. Observe the LED update indications:
   - If the USB is plugged, the yellow LED (LED2 of the board) is turned on statically.
   - If the update process has started, the yellow LED is blinking (250ms on/250ms off).
   - In case no update file was compatible, the yellow LED is turned off.
   - If the firmware update is successful, the device is rebooted and LED is now turned off.
   - After the device is rebooted, the USB flash drive is detected again and thus the yellow LED is also turned on again.
   - But now the new firmware notices that the firmware update is already installed and the yellow LED is turned off again (this can take some time).
6. Wait until the whole firmware update and reboot process is finished - it takes up to 5 minutes.
7. When the firmware update process is finished and the yellow LED is turned off again, the USB flash drive can be unplugged.

11.3 Update via SSH and SFTP

1. Connect to the board via SSH (e.g. PuTTY).
2. Backup all your own implementation and configuration files if necessary.
3. Transfer the update image file via SFTP to the board and store it in the directory `/srv` with e.g. filename `my-update.image`.Note: On Windows systems you can use WinSCP or Filezilla for example.
4. Run the following command via SSH console: `rauc install /srv/my-update.image`.

5. The update process should start and report progress and success via console messages.
6. Reboot into the new system by running the following command via SSH console: `reboot`.
7. Re-login into the new system and delete the file `/srv/my-update.image`.

11.4 Update via SSH or Serial Console and HTTP or FTP

1. Connect to the board via SSH (e.g. PuTTY) or serial terminal.
2. Backup all your own implementation and configuration files if necessary.
3. Place the update image file on an HTTP or FTP server which is reachable via network by your Charge Control device.
4. Note the URL of the download. In case authentication is required, you must provide the credentials in the URL, e.g. `http://username:passwd@my-site.com/update.image`.
5. Run the following command via SSH console: `rauc install <url>`, where you replace the URL with your actual URL.
6. The update process should start and report progress and success via console messages.
7. After success, reboot into the new system by running the following command via SSH console: `reboot`.

11.5 Update via MQTT API

It is possible to trigger a firmware update via MQTT API. This requires that the update image file is accessible via network download from an HTTP or FTP server. Then this download URL can be published via MQTT API and the Charge Control firmware will retrieve the download and install it.

Note that in case authentication is required, you must provide the credentials in the URL, e.g. `http://username:passwd@my-site.com/update.image`.

The Charge Control firmware observes and provides the following MQTT topics for its firmware update process:

Topic	Subscribe/Publish	Type	Unit	Remarks
firmware/update/target/url	publishable	String	-	Contains the URL to download the update file from. This topic must be set last as changes will start the update process. All other topics will be latched at this time. URLs ending with .raucb are handled as casync firmware updates, see discussion below, all other URL endings are supposed to be traditional full-file firmware updates.

Topic	Subscribe/Publish	Type	Unit	Remarks
firmware/update/target/max_trials	publishable	Integer	-	If this integer is set, then the download will be automatically retried in case of download failure. Values < 1, empty string or unset value will be silently used as 1. For casync firmware updates, the download is integral part of the installation, that means that the whole installation is retried.
firmware/update/target/interval	publishable	Integer	seconds	If this integer is given, the interval to wait between multiple download retries (if applicable). Values < 0, empty string or unset value will be silently used as 0. Note, that this interval is a high-level view of the update procedure, i.e. internal timeouts of a few seconds and minutes - for example during establishing a connection to the download server - are not converted here.
firmware/update/state/actual	subscribeable-only	String	-	Contains one of the following strings to signal the overall firmware update state: `idle`, `installing`, `install-succeeded`, `install-failed`, `downloading`, `download-succeeded`, `download-failed`. See table below.
firmware/update/state/progress	subscribeable-only	Integer	percent	Reports the (estimated) progress of the current operation in percent. Do not expect this topic to contain specific values, it is provided for convenience / debug purposes only (e.g. ease web GUI).

Topic	Subscribe/Publish	Type	Unit	Remarks
firmware/update/result/state	subscribeable-only	String	-	Contains the result of the last firmware update operation: `succeeded`, `failed` or empty string (in case no update was triggered before (after boot)).
firmware/update/result/message	subscribeable-only	String	-	Contains an English error message in case the last update operation failed, or empty string. In case max_trials is set to a value > 1, this topic might be published after an unsuccessful installation attempt - in contrast to "firmware/update/result/state" which is only published after all attempts failed (or directly after the first successful attempt).

MQTT topics for firmware update process

Update State in firmware/update/state/actual	Description
idle	The charging station is not performing firmware update related tasks. This is the usual state after boot.
downloading	The request to install a new firmware was received and the file is now transferred to the device over network.
download-succeeded	The firmware update file was transferred successfully. For casync firmware updates, the download phase is also reported - however, the real download is integral part of the installation and thus not finished yet when this state is reported.
download-failed	The firmware update file transfer failed. A possible reason may be retrieved with `firmware/update/result/message` topic. For casync firmware updates, this state is only reported for HTTP(S) URLs when a simple access test to the URL of the Charging Stack failed (HTTP HEAD request).
installing	The transferred file was passed to update framework and is now signature checked and installed.
install-succeeded	The firmware update file was successfully installed by the framework (no reboot happened yet).
install-failed	The firmware update installation failed. A possible reason may be retrieved with `firmware/update/result/message` topic.

MQTT topics for firmware update process

Please note that a reboot of the device does not happen automatically, since the firmware update installation is run in the background and does not disturb ongoing charging sessions. Your controlling software should observe the overall situation and decide when the time for a reboot is best and then trigger the reboot.

11.6 Update via OCPP

The firmware update via OCPP commands `UpdateFirmware` and `FirmwareStatusNotification` is implemented. Such an update via OCPP automatically triggers a reboot of the charge point after successfully installing the new firmware and as soon as an active charging session is finished. The central system is notified about the successful installation after this reboot of the charge point. Note that the messages `FirmwareStatusNotification` are only sent to the central system in case the update has been triggered via `UpdateFirmware` (and e.g. not during a USB update).

Implementation details:

- The following download protocols are supported: HTTP, HTTPS, FTP, FTPS
- Any attempt to reset the charge point via OCPP during installation of firmware will be rejected.
- Any OCPP request for a new firmware update during a running firmware installation will be confirmed, but ignored otherwise.
- It is recommended to use a reasonable value (e.g. 3) for the OCPP field "retries".
- In case a firmware update has already been scheduled, the old request will be overwritten by the new one (last writer wins).

11.7 Support for casync based Firmware Updates

As mentioned in the previous sections, the firmware update mechanism uses rauc (https://rauc.io) as update framework. The traditional method of updating a device is, that a big firmware update image file is transferred to the device and then installed. However, this approach is sometimes not ideal, for example when during different firmware releases only few parts changed. To address such scenarios, rauc itself supports casync based bundles. For more details, please refer to rauc's own documentation: https://rauc.readthedocs.io/en/latest/advanced.html#rauc-casync-support. It is obvious that such kind of firmware updates require an established Internet connection (or a locally available HTTP/FTP server which can be reached by the charging station) - but the concept does not work for USB pen drive firmware updates.

The casync support in the rauc framework is included since Charge Control firmware version 0.10.0. Older versions do not include the casync executable, thus these versions are not able to handle casync firmware update files. On casync-enabled Charge Control firmware versions, it is possible to pass a URL referring to a casync firmware update image directly to the rauc framework via `rauc install <URL>`. Please remember that rauc will use the passed URL to construct a base URL for accessing the smaller chunks. For example the URL `http://example.com/update.raucb` would lead to the assumption that all chunks can be accessed via the base URL `http://example.com/update.castr/`. Note, that the chunk store is a whole directory with many small files inside, not a simple file - the file extension like directory naming might suggest this.

To fully support casync based firmware updates also in the Charge Control stack APIs, that means via OCPP (where applicable) and via MQTT API, some points must be taken into consideration. Firmware update files shipped by chargebyte traditionally use the file extension `.image`. Such images are downloaded first by the Charge Control stack and then forwarded as a local file to the rauc framework. This approach allows to have more control over the download itself, e.g. to inform the OCPP backend about the current progress or download failures etc. However, this approach fails for casync based firmware updates, since rauc (and casync under the hood) must know the original URL to access the chunk store (as mentioned above). To decide which "down-passing to rauc approach" must be used by Charge Control stack and this without downloading and inspecting the file before a simple naming convention is assumed by the Charge Control stack: URLs ending with .raucb are assumed to be casync firmware update files, while all other endings are expected to be traditional ones. This is in alignment to rauc's internal expectation to construct the base URL for the chunks where the suffix `.raucb` is replaced with `.castr` (as mentioned above). It is recommended for customers to follow this naming convention.

11.8 Rollback mechanism

11.8.1 Introduction

The internal storage of Charge Control devices is subdivided into several partitions. This is used to form a redundant setup with A and B system in which one system is active and running while the other one is inactive. Thus it is possible to update the inactive system in background without interrupting the main operation of the system. After a firmware update was performed and the system has booted into the new firmware, the previously active system still exists - however, inactive now. It will be used for later firmware updates, but immediately after a firmware update was performed, it serves as fallback option. This is possible since it can be safely assumed, that this previously active system is still in a recent state or at least not in a worse condition than before then firmware update.

The process of using this previous system as fallback after a firmware update, is called rollback mechanism. Several software components must work together to provide this feature. The main purpose is to ensure that after a firmware update was installed, all software components are running smoothly, even with the restored/kept configuration files of the previous system.

11.8.2 How it works

As mentioned, the filesystem architecture of Charge Control devices consists of several partitions. During manufacturing process, two of these partitions are flashed with the same firmware version of the charging software - and would be both bootable per se. After initial starting of a board, only the first partition (rootfs A) is activated and is used by the boot loader to complete the boot process. When this boot process passed the point at which it considers itself being in a known-to-be-good state, it disables the other partition (rootfs B).

The rollback mechanism is only activated while performing a software update of Charge Control devices. After starting of the update process the update image file will be installed to the other, currently deactivated partition. The currently booted rootfs filesystem stays unaffected while performing the update. As soon as the update is successfully installed, the currently booted partition is deactivated. It depends on the used update method when a reboot is actually performed, e.g. during update from USB pen drive this reboot happens immediately. Now the bootloader is responsible to boot the partition with the new software image. While the board is booting, the green led (LED 1 on Charge Control C and LED 3 on Charge Control M) is blinking and it stops blinking when the update process is completed. Then, after successfully starting the charging software, the rollback mechanism is deactivated and the currently installed software version is notified with MQTT topic `ci/global/version/charging_software`. The charging software is now running and operational.

But in case something went wrong during the update, e.g. some software component does not come up or behave as expected, then the charging software does not mark this boot process as successful. The included watchdog functionality will detect such a misbehavior and trigger a reboot of the whole device. It is tried up to 3 times to get the new firmware running. After the last trial failed, the bootloader will switch back to boot the previous system again.

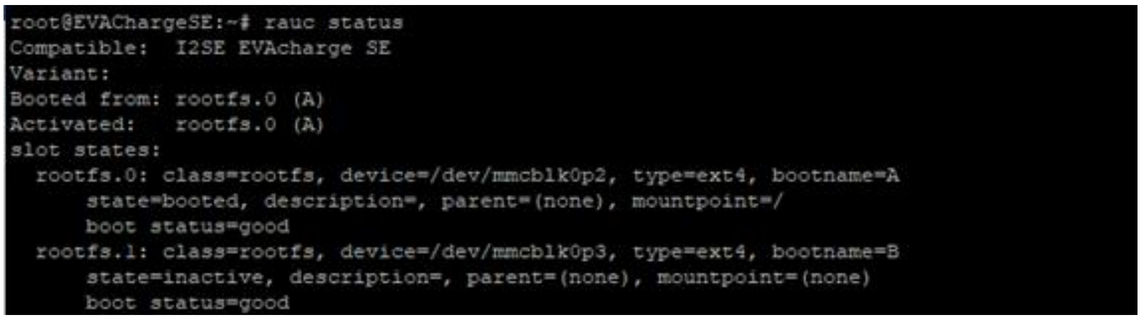
As noted, a successfully booted new firmware will disable the rollback mechanism. On the other hand, in case the previous system is booted again - and it can be assumed that this system boots also successfully - this previous system will also disable the rollback mechanism. At the end, the rollback mechanism is automatically disabled after a firmware update was installed or the rollback was performed. In any case, the device runs with working firmware which is the base for normal operation and also later firmware updates.

However, it is worth to remember that the partitions are not synchronized. Thus if one rootfs is customized, by e.g. integrating custom software applications, the other partition stays unaffected. Only the configuration files, stored in `/etc/secc` on EVSE side or `/etc/evcc` on EV side, as well as custom network device configurations, are migrated during installation of an update. This point must especially be considered before performing a standard software update from chargebyte. In the case of customized file system contents, it is recommended to create a customized firmware image update which is based on the standard update image. For more information regarding the creation of own firmware image updates please contact the chargebyte support.

11.8.3 Performing the rollback mechanism manually

In some situations, as in case to rescue data of the other partition after a software update, it might be necessary to perform a manual rollback to the other partition.

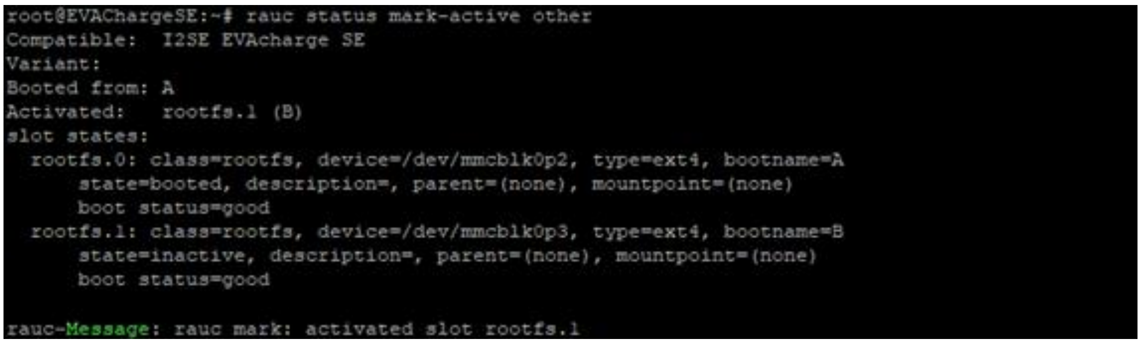
The Charge Control device uses the rauc update framework for updating the device with a new firmware version and managing the boot partitions. Before using rauc command line tool it is necessary to establish a connection via SSH or Debug UART. After the connection is established the status of the by rauc managed partition can be retrieved by using the `rauc status` console command. The following figure shows the console output of the `rauc status` command. Here the board - a Charge Control M board in this example, but the same priniciples also apply to other products of the Charge Control family - was booted from the first partition “rootfs.0” (A).



```
root@EVAChargeSE:~# rauc status
Compatible:  I2SE EVAcharge SE
Variant:
Booted from: rootfs.0 (A)
Activated:   rootfs.0 (A)
slot states:
  rootfs.0: class=rootfs, device=/dev/mmcblk0p2, type=ext4, bootname=A
      state=booted, description=, parent=(none), mountpoint=/
      boot status=good
  rootfs.1: class=rootfs, device=/dev/mmcblk0p3, type=ext4, bootname=B
      state=inactive, description=, parent=(none), mountpoint=(none)
      boot status=good
```

Console output of the "rauc status" command

The rollback to the other partition can now be performed by using the command `rauc status mark-active other`. The console output should now look like the following figure.



```
root@EVAChargeSE:~# rauc status mark-active other
Compatible:  I2SE EVAcharge SE
Variant:
Booted from: A
Activated:   rootfs.1 (B)
slot states:
  rootfs.0: class=rootfs, device=/dev/mmcblk0p2, type=ext4, bootname=A
      state=booted, description=, parent=(none), mountpoint=(none)
      boot status=good
  rootfs.1: class=rootfs, device=/dev/mmcblk0p3, type=ext4, bootname=B
      state=inactive, description=, parent=(none), mountpoint=(none)
      boot status=good

rauc-Message: rauc mark: activated slot rootfs.1
```

Console output of the "rauc status mark-active other" command

After rebooting of the board with command `reboot` the board should be successfully booted on the other partition. To revert to the initial partition, just repeat this process.

11.8.4 Development tools

During development it may be useful to access/mount the inactive partition. Then it is required to first determine the inactive partition using rauc command line tool. To simplify things, a helper shell script is included in the firmware which is called `mount-other-rootfs`. It takes one command line argument as parameter, that is, a target directory used as mountpoint for the inactive root filesystem.

Example: `mount-other-rootfs /mnt`

Note, that this helper script does not modify any rauc status information regarding this slot.

12 Charging Stack Initialization

The Charging Stack initializes itself after device boot.

Service files for the stack daemons are located in:

```
/lib/systemd/system/*.service
```

Service files are referenced with symlinks from:

```
/etc/systemd/system/multi-user.target.wants/
```

If required, it is possible to manage individual stack components with `systemctl`.

13 Device Access

There are different possibilities to access the device for configuration purposes.

The username- password combination required for login is:

Username	Password
root	zebematado

Table 44 Device Access

This is a generic password, so it is required to be changed by the customer!

13.1 Debug UART

Use the following settings to connect to the debug UART:

Setting	Value
Baud rate	115200
Data bits	8
Stop bits	1
Parity	None
Flow control	None

Table 45 Settings to connect to the debug UART

13.2 SSH

Charge Control C is shipped with SSH (Secure Shell) service running on the bridge interface, i.e. Ethernet and mains powerline interface (only Charge Control 300). It allows you to connect to Charge Control C securely and perform Linux command-line operations. The SSH service is listening on the well-known port number for SSH: TCP port 22.

13.3 Website

Charge Control C is running a web server to provide a web frontend to configure the device and control various aspects of the charging stack. The web server is listening on the standard TCP Port 80. The web frontend can be accessed via the Ethernet interface and/or mains powerline interface (only Charge Control 300). To access it, simply put the device’s IPv4 or IPv6 address into your browser’s address bar. Since the Charge Control C devices ship with DHCP enabled, you might need to access your router’s web frontend first to determine the IP address given to your Charge Control C board. Alternatively, you might use the static fallback IP as documented in the section Network Configuration.

14 Configuration

The Charge Control stack is built as an application on top of a Linux system. Several software components interact with each other and rely on various configuration files.

The Charge Control configuration files are described in the following section. Devices are shipped with a default configuration as shown in Configuration Hardware Parameter customer.json and Configuration Software Parameter customer.json. Several configuration files which are present on regular Linux systems also influence stack behavior.

chargebyte is not liable for a standard compliant charger configuration.

’Application Note 9 - Charge Control C - Digital Input/Output configuration’ gives a detailed overview about how to configure digital input and output of the product.

14.1 Charging Stack Configuration Files

configuration file	description	preserved during update
/etc/secc/customer.json	defines most of the charging stack behavior	yes
/etc/secc/leds.json	defines the LED behavior for the digital outputs	yes
/etc/secc/rotaryencd.json	defines the rotary encoder switch meaning	yes

Table 46 Charging stack configuration files

- These JSON files use comments to describe the keys’ functions and their possible values. (Comments are a non-standard extension to JSON)
- Please ensure that, when editing these files, correct JSON syntax (plus the comments) is adhered to.
- In case of modifying of the customer.json file, please consider the maximum size of the values of the JSON keys. Otherwise, some values might be imported incorrectly or truncated. E.g. in case the maximum size of "ocpp/chargePointVendor" is exceeded, it will be transmitted truncated over OCPP.
- The port[0] syntax refers to the first JSON object in the port[] array of the configuration file, configuring the first charge port of the station.

Hardware Configuration (found in file /etc/secc/customer.json)				
Parameter	Description	Type	Default	OCPP
ports[0]/pluggable	Must be set by the charging station manufacturer to indicate to the charging stack whether a fixed charging cable is attached to the charging station, or whether the charging station has a socket and the car driver connects his private cable. In case of "true", i.e. a socket is installed, the charging station must also connect the Proximity Pilot pin of the socket because this is used to determine the maximum current limit which is allowed by the cable.	Boolean (true = socket, false = fixed cable)	true	yes
ports[0]/pp/cable_current_limit	In case of a fixed cable, the charging station manufacturer must configure the rating of the used charging cable in this parameter. It must contain the maximum current in Ampere which is allowed by the cable vendor. Enter the single value, i.e. the maximum current per phase - not the summed limit for all phases. In case of "pluggable = true", this configured limit is ignored because the value is automatically determined by reading the Proximity Pilot.	Integer	6	yes
ports[0]/evse_current_limit	This parameter defines the charging station's internal maximum cable rating in Ampere. It must be configured by the charging station manufacturer and contain a single value, i.e. the maximum current per phase - not the summed limit for all phases.	Integer	6	yes

Hardware Configuration (found in file /etc/secc/customer.json)				
ports[0]/failsafe_current_limit	With this parameter the charging station manufacturer can configure a maximum current limit after initial start-up of the charging station. This value affects the computation of the offered overall current limit only as long as a current limit was not transmitted via OCPP or via a dynamic current limit (MQTT). The aim of this parameter is, that a load management application can make safe assumptions about the start-up behavior of the charging station during reboot phase and until communication is restored again. If this parameter is configured to "-1", this failsafe current limit is not applied.	Integer	-1	yes
ports[0]/contactor/feedback_type	defines the logic behind the contactor feedback (Relay 1). Note: Because of safety reasons it is strongly recommended to use a contractor with feedback pin. If no feedback is configured the charging software is not able to check whether the contactor is truly opened or closed.	String ("nc" = normally close, "no" = normally open, "none" = no feedback)	"no"	yes
ports[0]/ventilation/enable	enables external ventilation control (Relay 2)	Boolean	false	yes
ports[0]/ventilation/control	defines the ventilation control mode. The ventilation can be controlled internally by the charging software or externally over customer software via MQTT topics. Precondition: Config parameter for external ventilation control must be enabled.	String ("internal", "external")	"internal"	yes

Hardware Configuration (found in file /etc/secc/customer.json)				
ports[0]/ventilation/feedback_type	defines the logic behind the contactor feedback (Relay 2). Note: Because of safety reasons it is strongly recommended to use a contractor with feedback pin. If no feedback is configured the charging software is not able to check whether the contactor is truly opened or closed.	String ("nc" = normally close, "no" = normally open, "none" = no feedback)	"no"	yes
ports[0]/emergency_alarm/enable	enables emergency alarm monitoring	Boolean	false	yes
ports[0]/emergency_alarm/gpio	defines GPIO line connected to emergency contact	Integer	121	yes
ports[0]/emergency_alarm/polarity	defines GPIO polarity connected to emergency contact	String ("active_low", "active_high")	"active_high"	yes
ports[0]/rcd_monitor/enable	enables RCD monitoring	Boolean	false	yes
ports[0]/rcd_monitor/gpio	defines GPIO line connected to the RCD feedback	Integer	122	yes
ports[0]/rcd_monitor/polarity	defines GPIO polarity connected to the RCD feedback during normal operation, i.e. when no fault is present Example: When the attached RCM/RCD provides 0 V to the board's digital input pin in normal mode and it provides e.g. 12 V when a fault condition is signaled, then "active_low" must be used here.	String ("active_low", "active_high")	"active_high"	yes
ports[0]/rcd_monitor/test_gpio	defines GPIO line connected to the RCD test trigger pin. When RCD monitoring is disabled, RCD test feature is not available. When this key is set to '-1', the RCD test feature is disabled, while RCD monitoring is still enabled.	Integer	87	yes

Hardware Configuration (found in file /etc/secc/customer.json)				
	When this key is commented or missing, the default value is used.			
ports[0]/rcd_monitor/test_gpio_polarity	defines GPIO polarity connected to the RCD test trigger during normal operation, i.e. when no fault is present. When this key is commented or missing, the default value is used. This key is only used when RCD test feature is enabled.	String ("active_low", "active_high")	"active_high"	yes
ports[0]/rcd_monitor/test_trigger_time	defines the duration in milliseconds how long the test GPIO is driven active to start a self-test of the RCD. (see RCD monitoring and testing) When this key is commented or missing, the default value is used. This key is only used when RCD test feature is enabled.	Integer (1 - 5000)	815	yes
ports[0]/rcd_monitor/test_check_tripped_time	defines a delay in milliseconds when the implementation looks at the feedback signal after the RCM test was started. The duration here counts from the start of the RCM test request, i.e. the time when the test trigger output pin changed from inactive to active. (see RCD monitoring and testing) When this key is commented or missing, the default value is used. This key is only used when RCD test feature is enabled.	Integer (1 - 5000)	810	yes

Hardware Configuration (found in file /etc/secc/customer.json)				
ports[0]/rcd_monitor/test_check_normal_time	defines a delay in milliseconds when the implementation looks finally at the feedback signal to ensure that the pin is in its original state again. This duration is measured from the time when the test trigger output pin changed back to usual state. (see RCD monitoring and testing) When this key is commented or missing, the default value is used. This key is only used when RCD test feature is enabled.	Integer (1 - 5000)	410	yes
ports[0]/plug_lock/type	selects the plug lock motor type	String ("KUESTER-02S", "KUESTER-04S", "EV-T2M3S-E-LOCK12V", "EV-T2M3SM-E-LOCK12V", "HELLA-MICRO-ACTUATOR-1", "WALTHER-WERKE-9798999009", "INTRAMCO-603205", "SCAME-200.23260BS", "SCAME-200.23261BS","SCAME-200.23261BP", "SCAME-200.23261BL")	"EV-T2M3S-E-LOCK12V"	yes
ports[0]/meter/enable	enables metering support (required for OCPP)	Boolean	false	yes
ports[0]/meter/port	defines UART interface to the meter	String	"/dev/ttymxc0"	yes

Hardware Configuration (found in file /etc/secc/customer.json)				
ports[0]/meter/protocol	Specifies the meter's Modbus protocol to use. Special note for Eastron support: When just using "eastron", the implementation tries to auto-detect the connected meter. However, this only works for newer models, that means for models which already have a newer meter firmware which fills the Meter Code register. If you have meters with older firmware, you can force the usage of a specific meter model by giving the corresponding specific model type in this string.	String ("dzg", "eastron", "eastron,sdm230", "eastron,sdm630", "eastron,sdm72dm-v1", "eastron,sdm72dm-v2", "elecnova,dds1946", "elecnova,dts1946", "gavazzi", "klefr", "eem-350-d-mcb", "abb", "iskra", "socomec", "sunspec")	"dzg"	yes
ports[0]/meter/baudrate	baud rate of Modbus protocol	Integer	9600	yes
ports[0]/meter/parity	parity of Modbus protocol	String ("none", "odd", "even")	"even"	yes
ports[0]/meter/address	force usage of a given Modbus address of the meter	Integer (1 - 247)	depends on protocol	yes
ports[0]/recloser/enable	enables RCD recloser support	Boolean	false	yes
ports[0]/recloser/port	defines UART interface to the recloser device	String	"/dev/ttymxc0"	yes
ports[0]/recloser/protocol	specifies the recloser's Modbus protocol	String ("geya")	"geya"	yes
ports[0]/recloser/baudrate	baud rate of Modbus protocol	Integer	9600	yes
ports[0]/recloser/parity	parity of Modbus protocol	String ("none", "odd", "even")	"none"	yes
ports[0]/recloser/address	force usage of a given Modbus address of the recloser	Integer (1 - 247)	depends on protocol	yes
ports[0]/rfid/enable	enables RFID support	Boolean	false	yes
ports[0]/rfid/port	defines UART interface to the RFID reader	String	"/dev/ttymxc0"	yes
ports[0]/rfid/protocol	specifies the RFID protocol	String ("Stronglink", "stronglink-modbus", "smarttec-mcr-legic", "mqtt")	"Stronglink"	yes

Hardware Configuration (found in file /etc/secc/customer.json)				
ports[0]/rfid/baudrate	baud rate of RFID reader protocol	Integer	9600	yes
ports[0]/rfid/parity	parity of RFID reader protocol	String ("none", "odd", "even")	"none"	yes
ports[0]/rfid/address	force usage of a given Modbus address of the RFID reader	Integer (1 - 247)	unset, i.e. protocol dependent default applies	yes
ports[0]/rfid/remote_ports	This item can be used to configure remote destinations for the RFID reader implementation when a single charging station case encloses several Charge Control C boards for individual charging sockets. This "remote_ports" key is an JSON array, which consists of JSON objects whereas each object defines one remote destination. Each of these JSON objects must define the key "uri", otherwise the object is ignored. The current implementation supports up to three remote objects, surplus objects are silently ignored. Note, that even array items without "uri" key counts in this regards.	Array of Objects	one example item with commented out "uri" key, which in turn renders this item inoperable	yes
ports[0]/rfid/remote_ports[]/uri	This parameter takes an URI which defines for this remote destination, how the RFID reader implementation should connect to the remote MQTT broker. Example: mqtt://192.168.0.1:9001 The current implementations assumes, that charging port 0 is used at the destination MQTT broker.	String	none	yes
io/digital_input_threshold_voltage	Specifies the threshold voltage for the digital inputs in mV	Integer (0 - 12000)	6000	yes

Table 47 Configuration of hardware parameters in customer.json file

Software Configuration (found in file /etc/secc/customer.json)				
Parameter	Description	Type	Default	OCPP
ports[0]/hlc_protocols	Specifies the supported high level communication protocols	String ("urn:iso:15118:2:2013")	"urn:iso:15118:2:2013"	yes
ports[0]/always_accept_cp_state_d	Indicates whether an EV which requests CP state D (ventilation) should always be accepted	Boolean	false	yes
ports[0]/force_wake_up	Indicates whether a legacy EV should be woken up after successful authorization by imposing O V on the CP	Boolean	false	yes
ports[0]/wake_up_after_timeout	Indicates whether a legacy EV should be woken up after a timeout of 30 s by imposing O V on the CP	Boolean	false	yes
ports[0]/charging_type	Specifies the allowed charging types during a charging session. For IEC 61851-only charging the parameter must be configured to "basic". For ISO 15118-only charging the parameter must be configured to "highlevel". A combination of both charging types is currently not supported. If "basic+fake_highlevel_dc" is configured, a faked HLC session is started to retrieve MAC address and V2G DC parameters and continues with a basic charging session.	String ("basic”, "highlevel", "basic+fake_highlevel_dc")	"basic"	yes
ports[0]/user_authentication	Specifies the authentication method of the user	String ("free”, "ocpp", "mqtt", "key-switch")	"free"	yes

Software Configuration (found in file /etc/secc/customer.json)				
	Possible values: "free”: free charging, no user authentication required "ocpp": user authentication over OCPP "key-switch": user authentication over a key-switch "mqtt": user authentication via MQTT topic TOPIC_AUTHORIZATION_STATUS			
ports[0]/highlevel_authentication_mode	Specifies the authentication mode for the high level charging session	String ("eim")	"eim"	yes
ports[0]/tls_security	Controls how to handle encrypted communication to the EV during high level charging	String ("prohibit”, "allow”, "force”)	"prohibit”	yes
ports[0]/evse_id	Provides the EVSEID of the actual charger	String[37]	”DE*INT* ECH 1234567890”	yes
ports[0]/meter/publish_settings/timer	Defines the MQTT publish interval of metering data in seconds	Integer	30	yes

Software Configuration (found in file /etc/secc/customer.json)				
fake_highlevel_dc_vendors	Vendor specific configuration of the `fake_highlevel_dc` charging mode, see `ports[0]/charging_type` above. Keys in this object are the names of vendor sub-objects which allow to fine-tune the fake HLC session and the switching to the usual basic charging. See configuration details for the example vendor "MyCar" below. The factory shipped default configuration contains settings for vendors Tesla and Volkswagen which can be used as a blue print for customer specific additional settings.	Object	Predefined keys for "Tesla" and "Volkswagen"	yes

Software Configuration (found in file /etc/secc/customer.json)				
fake_highlevel_dc_vendors/MyCar/oui	This configuration list contains the vendor OUIs (Organizationally Unique Identifier) and/or MAC address which are checked in order against the actual MAC address of the connected EV. MAC addresses themselves consist of 6 octets. The first 3 octets of the MAC address are the OUI which usually stands for a given vendor. The remaining 3 octets are asssigned individually to each EV. Often vendors have multiple OUIs in parallel since the amount of number space in the last 3 octets is limited. This is why here a configuration of a list is possible. When an EV is connected, all these OUIs are search for a match and this determines which vendor sub-object is used for the fine-configuration of the fake DC session. If no match is found, compiled-in default values are used. The list can contain arbitrary long strings which are compared against the start of the MAC address obtained during the charging session. It is also possible to configure the complete 48 bit long MAC address.	Array of Strings	vendor specific	yes

Software Configuration (found in file /etc/secc/customer.json)				
	Two-digit hex numbers must be separated by a colon. For example for Tesla, the list could look like: "98:ED:5C", "4C:FC:AA", "54:F8:F0", "0C:29:8F", "DC:44:27:1" Here this vendor has three usual 24-bit OUIs and one 28-bit OUI (MA-M).			
fake_highlevel_dc_vendors/MyCar/hlc_terminates_after	Configuration of time when a faked DC HLC session should be terminated for a given vendor. Possible values are: "MAC" - after receiving of the EV MAC address "SoC" - after receiving of the State of Charge value	String ("MAC", "SoC")	"SoC"	yes
fake_highlevel_dc_vendors/MyCar/switch_method	Configuration of the switch method between the faked DC HLC session and basic AC PWM charging for a given vendor. Please contact support for detailed information. Possible values are: 1-4	Integer	1	yes
ocpp/enable	Enables the OCPP client	Boolean	false	no
ocpp/uri	Identifies the charge point specific URI of the backend's WebSocket service (must begin with lowercase ws:// or wss://)	String		no
ocpp/verifyCert	Indicates whether to verify the Secure WebSocket server's TLS certificate (only valid for secure connections, not recommended to disable)	Boolean	true	no

Software Configuration (found in file /etc/secc/customer.json)				
ocpp/rfidStopTransaction	Indicates whether a re-authorization from RFID ends a charging session	Boolean	true	yes
ocpp/rfidRequiresEvPresent	Indicates whether an EV must be present before RFID authentication	Boolean	false	yes
ocpp/chargePointModel	Identifies the model of the Charge Point	String[20]		no
ocpp/chargePointVendor	Identifies the vendor of the Charge Point	String[20]		no
ocpp/chargePointSerialNumber	Identifies the serial number of the Charge Point	String[25]		no
ocpp/ftpTryTLSUpgrade	Whether to try upgrading to SSL/TLS when using FTP for downloading a firmware update file or during an OCPP getDiagnostics handling, if the corresponding FTP server offers it.	Boolean	false	yes
ocpp/AllowOfflineTxForUnknownId	Whether to allow transactions for unknown id tags while being offline from CS	Boolean	false	yes
ocpp/allowTxWithInvalidTime	Indicates whether a valid time is required for a transaction to be started.	Boolean	false	yes
ocpp/AuthorizationCacheEnabled	Indicates whether to use the authorization cache	Boolean	true	yes

Software Configuration (found in file /etc/secc/customer.json)				
ocpp/AuthorizeRemoteTxRequests	If AuthorizeRemoteTxRequests is true, the idTag received with the RemoteStartTransaction.req will be checked against the Local Authorization List, Authorization Cache and/or with an Authorize.req against the backend. A transaction will only be started after authorization was obtained. If AuthorizeRemoteTxRequests is false, then a transaction for the given idTag is started immediately and a StartTransaction.req is sent to the backend. The backend will then check the authorization status of the idTag when processing this StartTransaction.req.	Boolean	false	yes
ocpp/calibrationLawFormat	Configures the output format for signed meter values.	String ("ocmf,plain", "ocmf,hex")	"ocmf,plain"	yes
ocpp/ClockAlignedDataInterval	Specifies (in seconds) the size of the clock-aligned data interval for a whole day	Integer (0, 10 - 86400)	0	yes
ocpp/ConnectionTimeOut	Specifies (in seconds) until when the EV must be connected after a successful authentication	Integer (0, 10 -)	60	yes
ocpp/LocalAuthListEnabled	Indicates whether the local authorization list is enabled	Boolean	true	yes
ocpp/LocalPreAuthorize	Indicates whether to skip OCPP authorize request before start a transaction	Boolean	true	yes

Software Configuration (found in file /etc/secc/customer.json)				
ocpp/LocalAuthorizeOffline	Indicates whether to allow transactions for locally-authorized while being offline	Boolean	true	yes
ocpp/MeterValueSampleInterval	Indicates (in seconds) the sampling interval for metering data	Integer (0, 10 - 86400)	60	yes
ocpp/MeterValuesSampledData	Specifies the periodically sampled measurands to be included in a MeterValues.req	Array of Strings ("Energy.Active. Import.Register”, "Power.Offered"), see section "OCPP Measurand Support" for details	["Energy.Active. Import.Register”]	yes
ocpp/StopTransactionOnInvalidId	Indicates whether to send a StopTransaction for a non-accepted StartTransaction	Boolean	true	yes
ocpp/StopTxnSampledData	Specifies the sampled measurands to be included in the StopTransaction.req	Array of Strings ("Energy.Active. Import.Register”), see section "OCPP Measurand Support" for details	[ ]	yes
ocpp/TransactionMessageAttempts	Specifies how often a message should be repeated in case CS fails to process it	Integer	3	yes
ocpp/TransactionMessageRetryInterval	Specifes the duration (in seconds) until a message is repeated to the CS	Integer	30	yes
ocpp/useAsTimeSource	Indicates whether to use the time of day given by the Central System as system time	Boolean	true	yes

Software Configuration (found in file /etc/secc/customer.json)				
ocpp/bootNotificationOnReconnect	Some Central Systems identify the charge point via BootNotification instead of an URI. Enabling this option enforces a BootNotification after a Websocket reconnect in order to help these Central Systems. Since this is non-conform behavior to OCPP 1.6J, this option should be only enabled if required by the backend.	Boolean	false	yes

Table 48 Configuration Software Parameter customer.json

Hardware Configuration (found in file /etc/secc/leds.json)			
Parameter	Description	Type	Default
leds[0]/gpios	defines the digital output GPIOs which are connected to the LEDs (1st for red, 2nd for green, 3rd for blue)	Array of Integer [3]	[84]
software configuration			
leds[0]/behaviors[]/condition	specifies under which condition this behavior takes effect	String ("init", "updating", "auth_pending", "auth_accepted", "auth_rejected", "ready", "deactivated", "reserved", "connected", "charge_request", "ventilation_request", "charge_1", "ventilation_1", "error", "failure")	
leds[0]/behaviors[]/mode	specifies the LED mode of this behavior (blink fast = 100 ms on / 100 ms off, blink slow = 1 s on / 1 s off)	String ("solid", "blink_slow", "blink_fast")	
leds[0]/behaviors[]/color	specifies the LED color of this behavior (black = off)	String ("black", "white", "red", "green", "blue", "cyan", "yellow", "magenta")	
leds[0]/behaviors[]/duration	specifies the duration of this behavior in 1/100 ms (0 = permanent)	Integer	0

Table 49 Configuration Parameter leds.json

Configuration (found in file /etc/secc/rotaryencd.json)		
Parameter	Description	Type
rotary_switch_1[0]/phase_count	defines the phase count for setting 0 on rotary switch SW2	Integer (1 , 3)
rotary_switch_1[0]/current_limit	defines the grid current limit in Ampere for setting 0 on rotary switch SW2	Integer (1 .. 10000)
rotary_switch_1[1]/phase_count	defines the phase count for setting 1 on rotary switch SW2	Integer (1 , 3)
rotary_switch_1[1]/current_limit	defines the grid current limit in Ampere for setting 1 on rotary switch SW2	Integer (1 .. 10000)
...	...	...
rotary_switch_1[15]/phase_count	defines the phase count for setting F on rotary switch SW2	Integer (1 , 3)
rotary_switch_1[15]/current_limit	defines the grid current limit in Ampere for setting F on rotary switch SW2	Integer (1 .. 10000)

Table 50 Configuration Parameter rotaryencd.json

14.2 Network Configuration

The default Charge Control C network configuration creates a virtual Ethernet bridge `br0` consisting of the wired Ethernet interface `eth0` and -if available- the USB dongle for mobile broadband communication (see next section).

This bridge interface ships with DHCP enabled by default, plus a static fallback IPv4 address in the AutoIP network range (see RFC3927) to ease access in direct connections with Microsoft Windows™ PCs. The MAC address of the bridge interface corresponds to the MAC address of the wired Ethernet interface, is board-specific and is contained in the 2D barcode within the device label.

For network configuration the systemd's networkd is used as background service. For details, please refer to https://www.freedesktop.org/software/systemd/man/systemd-networkd.html. The platform specific factory default configuration files can be found in the firmware's root filesystem below `/lib/systemd/network`.

Customer can control network settings within a limited range: it is possible to switch between DHCP and a static IPv4 configuration for the bridge interface via Charge Control's stack configuration file `customer.json`, see following table. Network setups beyond the mentioned scenarios must be individually configured by customer. This is possible by turning the Charge Control's network configuration generation off. Then it is up to customers' firmware and/or additional configuration files to fully control network devices/settings etc. Customers are recommended to use the factory default files in `/lib/systemd/network` as templates and replace the files in this location when configuration should be modified as part of customer specific firmware builds, i.e. customer wants to create a firmware with modified factory default settings. When only a runtime change is intended for individual boards, then such board and customer specific network configuration files must be placed in `/etc/systemd/network` directory. This directory is also preserved during the firmware update process.

Network Configuration (found in file /etc/secc/customer.json)				
Parameter	Description	Type	Default	OCPP
network/skip_configuration	Set to true if network settings should not be managed by Charge Control stack. Use this when a special (unsupported) setup is required and configuration files for systemd-networkd are shipped/generated by customer software. Note that all other JSON types or if this option is missing, this is considered as false and the configuration is generated on a best-effort base.	Boolean	false	yes
network/ipv4/dhcp	If set to true, a DHCP client is enabled on the interface. When set to false, you most likely want to provide a static address (see below). If this config option is missing or is not a JSON boolean, then it is considered true.	Boolean	true	yes
network/ipv4/address	When DHCP is not enabled, a static IPv4 address should be specified here with network prefix len appended (aka CIDR notation). Example: 192.168.178.2/24	String	none	yes

network/ipv4/gateway	The IPv4 address of the standard gateway. Example: 192.168.178.254	String or Array of Strings	none	yes
network/ipv4/dns	The IPv4 address of a DNS server. Example: 192.168.178.254	String or Array of Strings	none	yes
network/ntp	If a dedicated NTP server should be used (e.g. when no NTP server is provided via DHCPv4) and/or the platform's default NTP servers are not reachable (e.g. when traffic is filtered). It should normally not be necessary to specify this setting since system defaults are carefully chosen to support a wide range of setups.	String or Array of Strings	none	yes

Table 51 Network Configuration Parameters in customer.json

Note: When upgrading from a Debian-based firmware to a Yocto-based one, the IP address obtained via DHCP may change due to the use of a different DHCP client.

The mains powerline interface `eth2` is always configured as DHCP client. In case DHCP fails on this interface, there is no fallback to a link-local address since this would result in a network address collision with the wired Ethernet interface.

While the Control Pilot interface could also be configured using systemd-networkd overrides, we do not recommend changing the factory settings nor should it be necessary at all.

Board Interface	Linux Interface
Virtual Bridge	br0

Table 52 Board and Linux Interface

MAC address[1]	00:01:87:XX:XX:XX
Fallback IPv4 address	169.254.12.53
IPv6 address[1]	fe80::xyxx:xxff:fexx:xxxx

Table 53 MAC and IPv6 address

[1]: MAC address and IPv6 address are device specific

MAC to IPv6 calculation rules:

IPv6 address is calculated out of the device specific MAC address.

MAC address	XX:XX:XX:XX:XX:XX
IPv6 Link Local address	fe80::xyxx:xxff:fexx:xxxx

Table 54 MAC to IPv6 calculation rules

Where y = X XOR 2. Furthermore 'ff:fe' is inserted and 'fe80::' prepended.

'y = X XOR 2' means inverting the 2nd bit from the right.

Example:

MAC address	00:01:87:12:34:56
IPv6 address	fe80::201:87ff:fe12:3456

Table 55 MAC and IPv6 address example

14.2.1 USB internet dongles

To easily connect Charge Control devices to the internet, USB internet dongles may be used. A list of supported USB internet dongles can be found in section USB.

These USB dongles provide full mobile internet router functionality and thus can be shared by multiple Charge Control devices. In technical terms, these USB dongles provide a virtual Ethernet interface with DHCP server, default gateway etc. Charge Control C firmware automatically includes this virtual Ethernet interface in the bridge br0 when the USB internet dongle is connected to the device; and when still configured for DHCP (factory default), it acquires IP address, DNS and router settings from the USB internet dongle automatically.

Since the virtual Ethernet interface is part of the bridge, devices connected via wired Ethernet can also use and share this internet uplink. These devices also only need to acquire their IP configuration via DHCP. This way it is also possible for technical staff to access the USB dongle firmware's web frontend, e.g. for configuring roaming options or similar.

Another typical scenario is to attach the USB internet dongle to one Charge Control device, and then attach additional Charge Control devices and/or other network devices to the Ethernet port. A standard Ethernet network switch can be used if more than one additional device needs to be connected.

The APN settings required for the mobile internet connection can be configured using the Charge Control's customer.json configuration file. If configured, the Charge Control firmware checks the current USB dongle settings and updates them if required. This is particularly useful to deploy e.g. APN configuration changes via OCPP. When multiple Charge Control devices share one mobile internet connection, then all devices check the APN settings. However, only the Charge Control device which is directly connected to the USB internet dongle will update the settings, all other devices will only warn about the (possibly) outdated configuration.

Additionally, a firmware component tries to obtain ICCID and IMSI settings from the USB internet dongle's firmware. This component is enabled as soon as the APN settings contain an APN name and when the USB mobile internet stick is not directly connected to the Charge Control device - it is then assumed to be an "enslaved" Charge Control device which uses a shared uplink connection. When available, the ICCID and IMSI information is then included as parameters in OCPP communication.

Summary of the requirements for using such a USB internet dongle as upstream internet connection:

1. Supported USB internet dongle
2. SIM card
3. APN configuration settings (if the default settings of the SIM card do not apply - please contact your mobile operator for details)

Steps to do during deployment:

1. Ensure that your Charge Control device is configured with correct APN settings in customer.json, see table below.
2. Insert the SIM card into the USB internet dongle.
3. Connect the USB internet dongle to the Charge Control board.
   Hint: A short USB extension cord might help in case the USB internet dongle is too big. Please also note, that the Charge Control's USB port can only provide limited power. Modern USB internet dongles may draw more power than allowed and thus will fail to work stable. The usage of a self-powered USB hub between Charge Control and USB internet dongle is recommended in this case.
4. Connect a notebook to Charge Control's wired Ethernet interface and access the USB internet dongle's web frontend with your browser (e.g. 192.168.8.1 for Huawei E3531)
   a. Remove the SIM card's pin protection
   b. Enable auto reconnect feature
   c. Enable roaming if necessary

Mobile Uplink Configuration (found in file /etc/secc/customer.json)				
Parameter	Description	Type	Default	OCPP
uplink/apn_name	APN to use. Contact your mobile internet provide for this setting. This setting is required to enable mobile connection related behavior, e.g. querying IMSI/ICCID from USB mobile device. If not required or desired (also see notes above), use empty string or do not give at all.	String	<empty>	yes
uplink/apn_username	APN username to use. If not required, use JSON null value or do not give at all.	String	Null	yes
uplink/apn_password	APN password to use. If not required, use JSON null value or do not give at all.	String	Null	yes

Table 56 Mobile Uplink Configuration Parameters in customer.json

14.3 OCPP Root Certificate Authority Keys / Certificates

OCPP communication with a backend provider might be encrypted using a TLS connection. The backend service provider usually provides configuration details, i.e. whether TLS is required or not, indicated by a URI using the `wss`: scheme. If TLS is required, the OCPP server presents an X.509 certificate to the device, which may be signed by a "well-known" Root Certificate Authority, or not. Well-known Root Certificate Authority in this context means that its root certificate is known by common browsers. On Linux systems, such certificates are usually packaged in a package named "ca-certficates" or similar. Charge Control firmware also includes such a collection of certificates. And since Charge Control's OCPP implementation uses standard TLS libraries, the usual Linux system-wide Root Certificate Authority files apply to OCPP's TLS connections, too.

If the OCPP backend provider does not use such a well-known Root CA, the Root CA's certificate file must be additionally installed on the device. Such X.509 certificates must be stored as PEM encoded files with .crt extension in /usr/local/share/ca-certificates. After placing the file(s) at this location, `update-ca-certificates` must be invoked on the device to update the Root CA bundle file, install required symlinks, etc. See the man page `man 8 update-ca-certificates` on a standard Desktop Linux to get familiar with the approach and to learn about details.

During a firmware update, these customer Root CA certificates are migrated to the updated system partition as well and thus are also available after the updated system started.

Warning: This migration feature was not included in firmware updates until version 1.1.5 or 2.0.0 - only later firmware updates now handle this. Thus for older firmware versions and updates, in case the backend connection requires such a customer certificate, this will lead to an offline charging station as the updated firmware does not trust the certificate anymore.

15 MQTT and Mosquitto Documentation

In the default configuration it is not necessary to interact with MQTT at all.

15.1 MQTT Interface and Configuration

The charging interface uses the MQTT protocol to exchange the charging information between the different charge control software clients. The topic_customer_hlc_ac.h defines all topics of the charging interface. The MQTT broker is available over the internal and external (Ethernet) interface. To implement an own MQTT client it is necessary to connect to the MQTT broker. The default configuration of the MQTT broker is:

- MQTT_HOSTNAME: "localhost" (internal) or IPv4/IPv6 address of the board (external)
- MQTT_PORT: 1883
- MQTT_CHARGE PORT: "port0"

If it is necessary to connect to the MQTT broker from an external device, consider using the charging service discovery as describe in the MQTT service discovery section.

After establishing an MQTT connection to the local message broker it is possible to subscribe and publish to topics of the charging interface. The interface uses QoS level 0 for all MQTT messages. The published messages do not need to be retained.

Charge Control C offers the possibility to provide several charging ports at only one charging station. Some topics definitions in this document use the charge port prefix “port0/” to open the way to distinguish between these different charge ports. This feature will be integrated in a future release of the charging software. As long as the feature is not supported by the charging software only “port0“ is accepted by the MQTT interface of the charging software.

Most of the MQTT message content is defined as "SimpleType” and will/must be published as a simple string-encoded number. For example, the content of topic TOPIC_EVSE_INIT_SESSIONID can be published as "1234567890". MQTT messages with content type "ComplexType” are using JSON objects. In chapter Physical value type is an example for a complex type definition.

15.2 MQTT Service Discovery

In case it is necessary to interact with the MQTT broker in the Charge Control device from an external device, it might be a problem to know the IP address of the device, e.g. when DHCP client is enabled and the assigned IP address thus might change. For this, the Charge Control implements a network service to discover it on the LAN. The used approach is DNS-SD, also known as Zeroconf and/or Bonjour protocol.

The DNS-SD announcements of a Charge Control device in the network uses the following service parameters:

- Name: This is a product and device dependent string which is only intended to be human-readable, e.g. to be shown in DNS-SD browsers etc. Do not use it for automatic processing by software. Usually, this string will be similar to e.g. "Charge Control C [00:01:87:01:02:03]”. Note: The MAC address presented in this string is always the MAC address of the wired Ethernet interface (even if the request was received from the mains PLC interface if present).
- Service Type:
    - Name: _charge-control
    - Protocol: _tcp
    - Subtype: <none>
- Port: This field will contain the port number on which the MQTT server listens on, usually 1883.
- Additional Text Records:

- Serial Number:
    - Key: serial
    - Value: serial number of the Charge Control board as ASCII digits
    - Example: serial=123456789

To look up all Charge Control devices on the network, just search for the mentioned service type, i.e. "_chargecontrol._tcp". Keep in mind that not even one single Charge Control device might be present on the network, so that your client might implement other means to choose the desired device. Also note, that later specifications could also specify additional text records, e.g. when a username and password is required to connect to the MQTT broker, or when the MQTT broker requires an encrypted connection etc.

15.3 Charge status information

A charging session consists of consecutive charging phases. This charging phases are represented by the MQTT charge status topics. The charge status topics can be marked as "started" or "finished". After receiving one of these topics, it is necessary to handle the phase specific charging information. This charging information is based on bot, content from the EVCC, which is published to the customer interface, and EVSE content, which needs to be published to the customer interface. So, the first step in each phase is to analyze the received EVCC information and then react according to the standard, with the help of the EVSE MQTT messages. The name of the MQTT message indicates which messages need to be handled. The figure below shows the structure of the charge status specific topics.

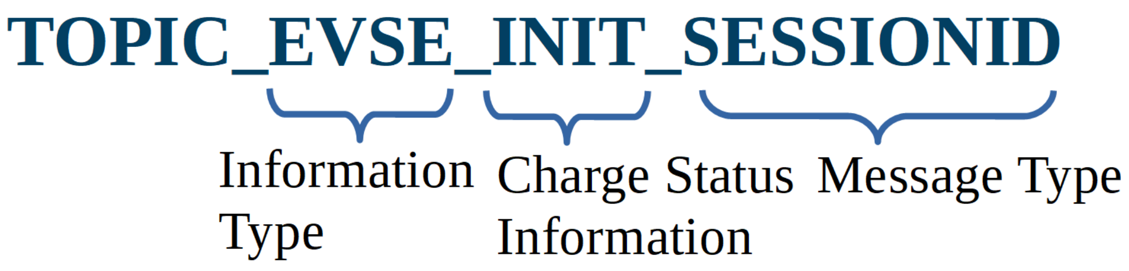


Figure 17 Structure of topics

The information type can be "EV" or "EVSE". On EVSE side only topics with "EVSE" are intended to be published to the customer interface. All topics with "EV” will be published from the charging interface. The charge status information indicates the current charging phase. The EVSE topic TOPIC_EVSE_INIT_SESSIONID needs to be published after receiving the TOPIC_CHARGE_INIT_STATUS with "started" topic. Customers who want to develop a charger need to subscribe to topics which start with TOPIC_CHARGE and TOPIC_EV and publish topics which start with TOPIC_EVSE. Customers who want to develop an EV need to subscribe to topics which start with TOPIC_CHARGE and TOPIC_EVSE and publish topics which start with TOPIC_EV. This status topics represent the following charging phases of ISO 15118 protocol:

1. The initialization phase begins at plug connect till the high level message ServicePaymentSelection.
2. The authentication phase begins at the CertificateInstallation message (ISO 15118-PNC mode only) until the end of the Authentication message.
3. The charge parameter phase lasts as long as the ChargeParameterDiscovery message is exchanged.
4. The charge phase lasts as long as the ChargingStatus message is exchanged. Note: The PowerDelivery message before AND after the charge phase is contained here.

The flow chart below shows a normal charging sequence with the charging phases of ISO 15118. In most cases the "finished" flags can be ignored because the phases will be processed sequential.

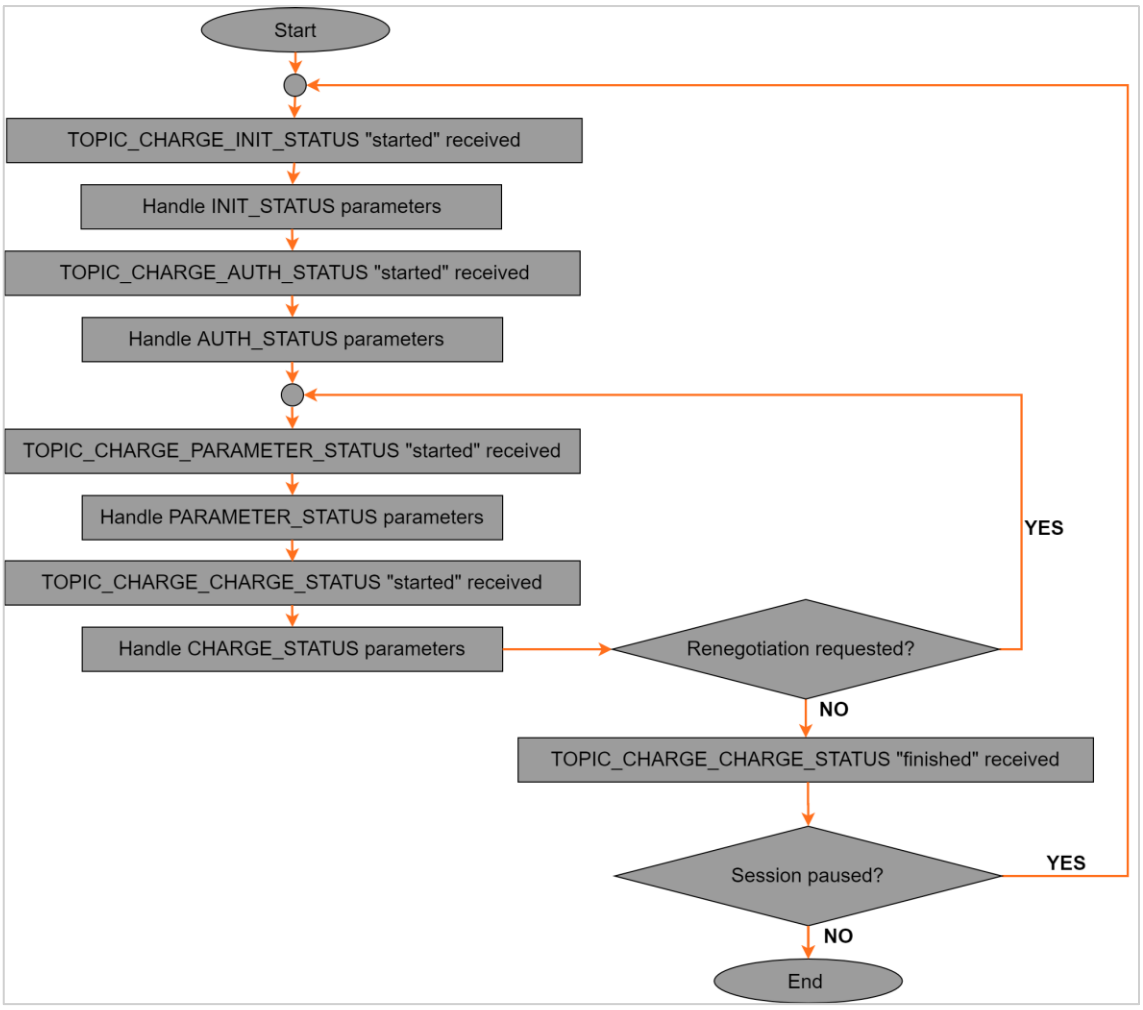

Start
TOPIC_CHARGE_INIT_STATUS "started" received
Handle INIT_STATUS parameters
TOPIC_CHARGE_AUTH_STATUS "started" received
Handle AUTH_STATUS parameters
TOPIC_CHARGE_PARAMETER_STATUS "started" received
Handle PARAMETER_STATUS parameters
TOPIC_CHARGE_CHARGE_STATUS "started" received
Handle CHARGE_STATUS parameters
Renegotiation requested?
YES
NO
TOPIC_CHARGE_CHARGE_STATUS "finished" received
Session paused?
YES
NO
End

Figure 18 Charge flow

In addition to the phase specific status topics, it is possible to observe the current connection status. The topics TOPIC_CHARGE_CP_STATUS and TOPIC_CHARGE_PWM_STATUS can be used to observe the physical state of the CP pin. The combination of both values provides important information about the current connection state.

The table below shows the possible combinations:

CP State	PWM-Status	CP State Information
A (12V)	100%	A1: EV unplugged
A (12V)	5%	A2: EV unplugged with PWM
B (9V)	100%	B1: EV connected, charging not possible
B (9V)	5%	B2: EV connected, high level communication possible
B (9V)	8-97%	B2: EV connected, only PWM possible
C (6V)	5%	C2: High level charging
C (6V)	8-97%	C2: Basic charging
F (-12V)		Unavailability of the charging station
E (0V)		Power outage or short of the control pilot to PE

Table 57 CP State Information

In case of high level communication, the QCA7000 of the board needs a few seconds to initialize itself. After reconnection of an EV the PWM switches from 100% to 5% as soon as the QCA7000 is ready to process SLAC messages of the connected EV.

Besides the CP State information, the current TCP status (indicated by topic TOPIC_CHARGE_TCP_STATUS) is another essential information about the current connection status. After the EVCC uses the SECC Discovery Protocol (SDP) to get the IP address and port number of the SECC, the EVCC can establish a TCP connection to the SECC. The TCP status switches from ’0’ (not connected) to ’1’ (connection established). This is one of the first topics after the plug of the EV is connected. In the cases of EVCC timer and error handling, it is possible that the EV switches to CP State ’B’ and disconnects the TCP connection immediately within a charging session, so the status topics shall be observed throughout the whole charging session.

15.4 EVSEProcessing

The EVSEProcessing parameter can be used to delay a specific charging phase. The charging phases "AUTH" and "PARAMETER" use this parameter. The pre-defined default value of this parameter is set to ’1’ (Ongoing) for each phase. When the processing of the received EVCC data is finished and the charging flow can be continued, this parameter needs to be explicitly set to ’0’ (Finished). If within the customer configuration file (See chapter Basic SECC configuration) the value for "free charging" is set to "true", the value for EVSEProcessing of the "AUTH" phase is set to ’0’ (Finished) automatically. When the identification of the physical limits (like EVSEMaxCurrentLimit) of the EVSE is finished, the value for EVSEProcessing shall be set to ’0’ (finished) for the "PARAMETER" phase.

To avoid skipping of data, it is recommended to publish the EVSE-Processing with "Finished" after providing the last phase specific parameter.

If the values for the physical limits are preconfigured via the customer configuration file (See chapter Basic SECC configuration) or the "SW2 - Rotary Coded Switch” (See chapter SW2 - Rotary Coded Switch) then the TOPIC_EVSE_PARAMETER_EVSEPROCESSING can be sent right after TOPIC_CHARGE_INIT_STATUS with value "started" was received.

15.5 MQTT Topics

15.5.1 Topics indicating the actual status of the ongoing charge phase

Topic Name	Topic	Type	Value	Comment
TOPIC_CHARGE_INIT_PROTOCOL	"port0/ci/charge/init/protocol"	SimpleType	string	Topic which indicates which charge protocol is used in the actual charge session. The value of this topic will either be "ISO15118" or "IEC61851".
TOPIC_CHARGE_INIT_STATUS	"port0/ci/charge/init/status"	SimpleType	string	Topic which indicates the beginning or the end of the initialization phase of a high level charge compliant to ISO15118. This includes the protocols SLAC, SDP, TCP and V2GTP from message SupportedAppProtocol until PaymentDetails. The value of this topic will either be "started" or "finished".
TOPIC_CHARGE_AUTH_STATUS	"port0/ci/charge/auth/status"	SimpleType	string	Topic which indicates the beginning or the end of the authentication phase of a high level charge compliant to ISO15118. This includes the V2GTP message Cerificate* and Authentication. The value of this topic will either be "started" or "finished".
TOPIC_CHARGE_PARAMETER_STATUS	"port0/ci/charge/parameter/status"	SimpleType	string	Topic which indicates the beginning or the end of the charge parameter discovery phase of a high level charge compliant to ISO15118. This includes the V2GTP message ChargeParameterDiscovery. The value of this topic will either be "started" or "finished".
TOPIC_CHARGE_CHARGE_STATUS	"port0/ci/charge/charge/status"	SimpleType	string	Topic which indicates the beginning or the end of the charge phase of a high level charge compliant to ISO15118. This includes the V2GTP messages PowerDelivery, ChargingStatus and MeteringReceipt. The value of this topic will either be "started" or "finished".

Topic Name	Topic	Type	Value	Comment
TOPIC_CHARGE_CP_STATUS	"port0/ci/charge/cp/status"	SimpleType	string	Topic which indicates the actual CP state. The value of this topic will be an enumeration of the possible CP states according IEC61851 which are "A", "B", "C", "D", "E" or "F".
TOPIC_CHARGE_PWM_STATUS	"port0/ci/charge/pwm/status"	SimpleType	string	Topic which indicates the actual PWM duty cycle in %. Its string represents a floating point number with two decimal places and a "." as decimal separator, e.g. "53.33". The value of this topic will be the actual value between "0.00" and "100.00”.
TOPIC_CHARGE_TCP_STATUS	"port0/ci/charge/tcp/status"	SimpleType	string	Topic which indicates the actual status of the TCP connection between an EV and an EVSE. The value of this topic will either be "1" if the connection was established and "0" otherwise.
TOPIC_CHARGE_PLUG_STATUS	"port0/ci/charge/plug/status"	SimpleType	string	Topic which indicates the actual status of the Plug lock on customer's side. The value of this topic will either be "locked" or "unlocked" if the lock state can be detected. "unknown" when the lock state cannot be detected.
TOPIC_CHARGE_CONTACTOR_STATUS	"port0/ci/charge/contactor/status"	SimpleType	string	Topic which indicates the actual status of the contactor on customer's side. The value of this topic will either be "closed" or "opened" if the contactor state can be detected. "unknown" when the contactor state cannot be detected.

Table 58 Topics indicating the actual status of the ongoing charge phase

15.5.2 Global topics for port independent charging software information

Topic Name	Topic	Type	Value	Comment
TOPIC_GLOBAL_VERSION_CHARGING_SOFTWARE	"ci/global/version/charging_software"	SimpleType	string	Topic which provides the version of the charging software. The version will be published as string (e.g. "0.7.0") after successfully booting of the board. This topic will be published with MQTT retain flag, thus the client will receive the topic immediately after subscribing.
TOPIC_GLOBAL_TIME_SET_STATUS	"ci/global/time_set/status"	SimpleType	bool	Topic which helps to detect whether the system time was set. This topic will be published retained with a value of '1' after a component of the charging stack sets the system time or in case the system time was successfully set via NTP. Whether this topic is published with a value of '0' before, i.e. during bootup, is not defined; or in other words, until the time was set once, don't rely on that this topic is published at all since this behavior might change later.

Topic Name	Topic	Type	Value	Comment
TOPIC_OCPP_ONLINE	"ocpp/online"	SimpleType	bool	Topic which provides the connection status to the OCPP backend. The payload is set to "1" (connected) if the connection to an OCPP backend is established, otherwise "0" (not connected)

Table 59 Global topics for port independent configuration of the charging software

15.5.3 EVSE specific V2G parameters of the initialization phase

Topic Name	Topic	Type	Value	Comment
TOPIC_EVSE_INIT_SESSIONID	"port0/ci/evse/init/sessionid"	SimpleType	long integer	Topic which provides the SessionID of the actual charging session. Will be or should be published at the very beginning of the high level charge, at least after the TCP status changes to "1”.

Topic Name	Topic	Type	Value	Comment
TOPIC_EVSE_INIT_EVSEID	"port0/ci/evse/init/evseid"	SimpleType	integer	Topic which provides the EVSEID of the actual charger. Will be or should be published at the very beginning of the high level charge, at least after the TCP status changes to "1".

Topic Name	Topic	Type	Value	Comment
TOPIC_EVSE_INIT_DATETIMENOW	"port0/ci/evse/init/datetimenow"	SimpleType	integer	Topic which provides the actual date and time of the charger in milliseconds from epoch. Will be or should be published at the very beginning of the high level charge, at least after the TCP status changes to "1".

Topic Name	Topic	Type	Value	Comment
TOPIC_EVSE_INIT_PAYMENTOPTIONS	"port0/ci/evse/init/paymentoptions"	ComplexType	paymentOptionJsonObject { "PaymentOption0": int, "PaymentOption1": int }	Topic which provides the payment options of the actual charger. Will be or should be published at the very beginning of the high level charge, at least after the TCP status changes to "1".

Topic Name	Topic	Type	Value	Comment
TOPIC_EVSE_INIT_CHARGESERVICE	"port0/ci/evse/init/chargeservice"	ComplexType	chargeServiceJsonObject { "ServiceID": int, "ServiceName": "Name of the Service", "ServiceCategory": int, "ServiceScope": "Name of the ServiceScope", "FreeService": bool, ,"SupportedEnergyTransferType0": int "SupportedEnergyTransferType1": int "SupportedEnergyTransferType2": int "SupportedEnergyTransferType3": int "SupportedEnergyTransferType4": int }	Topic which provides the charge services of the actual charger. Will be or should be published at the very beginning of the high level charge, at least after the TCP status changes to "1".

Topic Name	Topic	Type	Value	Comment
TOPIC_EVSE_INIT_SERVICELIST	"port0/ci/evse/init/servicelist"	ComplexType		Topic which provides the additional services of the charger beside the charge service. Will be or should be published at the very beginning of the high level charge, at least after the TCP status changes to "1".

Topic Name	Topic	Type	Value	Comment
TOPIC_EVSE_INIT_SERVICEPARAMETERLIST	"port0/ci/evse/init/serviceparameterlist"	ComplexType		Topic which provides the parameter of additional services of the charger beside the charge service. This will be important when an EV requests details to the additional service in the ServiceDetail request. Will be or should be published at the very beginning of the high level charge, at least after the TCP status changes to "1".

Table 60 EVSE specific V2G parameters of the initialization phase

15.5.4 EVSE specific V2G parameters of the authentication phase

Topic Name	Topic	Type	Value	Comment
TOPIC_EVSE_AUTH_GENCHALLENGE	"port0/ci/evse/auth/genchallenge"	ComplexType		Topic which provides the Gen-Challenge during the Authentication phase of the charge. Will be or should be published at the very beginning of the high level charge, at least after the TCP status changes to "1".
TOPIC_EVSE_AUTH_EVSEPROCESSING	"port0/ci/evse/auth/evseprocessing"	SimpleType	following enumeration table "EVSEProcessingType" ISO15118-2	Topic which provides the actual status of the authentication of the user at the charger. Will be or should be published from the beginning of the auth phase of the charge which can be indicated using TOPIC_CHARGE_AUTH_STATUS.
TOPIC_EVSE_AUTH_ERROR	"port0/ci/evse/auth/error"	SimpleType	bool 1 = error, 0 = no error	Topic which indicates if an error occurred while the charger tries to authenticate the user. Will be or should be published from the beginning of the auth phase of the charge which can be indicated using TOPIC_CHARGE_AUTH_STATUS.

Table 61 EVSE specific V2G parameters of the authentication phase

15.5.5 EVSE specific V2G parameters of the parameter discovery phase

Topic Name	Topic	Type	Value	Comment
TOPIC_EVSE_PARAMETER_EVSEPROCESSING	"port0/ci/evse/parameter/evseprocessing"	SimpleType	following enumeration table "EVSEProcessingType" ISO15118-2	Topic which provides the actual status of the parameter discovery of the charger. Will be or should be published from the beginning of the parameter phase of the charge which can be indicated using TOPIC_CHARGE_PARAMETER_STATUS.
TOPIC_EVSE_PARAMETER_SASCHEDULELIST	"port0/ci/evse/parameter/saschedulelist"	Complex Type	sascheduleListJson Object: { "SAScheduleTupleID" : int, "Start" : [int, ...], "Duration" : int, "PMax" : [int, ...], "PMaxMultiplier" : [int, ...] }	Topic which provides the charging schedule list of the charger. Will be or should be published from the beginning of the parameter phase of the charge which can be indicated using TOPIC_CHARGE_PARAMETER_STATUS.
TOPIC_EVSE_PARAMETER_SALESTARIFF	"port0/ci/evse/parameter/salestariff"	Complex Type		Topic which provides the SalesTariffs of the energy provider. Will be or should be published from the beginning of the parameter phase of the charge which can be indicated using TOPIC_CHARGE_PARAMETER_STATUS.

Topic Name	Topic	Type	Value	Comment
TOPIC_EVSE_PARAMETER_MAXCURRENTLIMIT	"port0/ci/evse/parameter/maxcurrentlimit"	Complex Type	maxCurrentLimitJsonObject f "Multiplier" : byte, "Value" : short g	Topic which provides the chargers maximum current limit in the parameter discovery phase of the charge. Will be or should be published from the beginning of the parameter phase of the charge which can be indicated using TOPIC_CHARGE_PARAMETER_STATUS.
TOPIC_EVSE_PARAMETER_NOTIFICATIONMAXDELAY	"port0/ci/evse/parameter/notificationmaxdelay"	SimpleType	integer	Topic which provides the chargers notification max delay in the parameter discovery phase of the charge. Will be or should be published from the beginning of the parameter phase of the charge which can be indicated using TOPIC_CHARGE_PARAMETER_STATUS.
TOPIC_EVSE_PARAMETER_NOTIFICATION	"port0/ci/evse/parameter/notification"	SimpleType	following enumeration table "EVSENotificationType" ISO15118-2	Topic which provides the chargers notification in the parameter discovery phase of the charge. Will be or should be published from the beginning of the parameter phase of the charge which can be indicated using TOPIC_CHARGE_PARAMETER_STATUS.

Topic Name	Topic	Type	Value	Comment
TOPIC_EVSE_PARAMETER_NOMINALVOLTAGE	"port0/ci/evse/parameter/nominalvoltage"	Complex Type	nominalVoltageJsonObject { "Multiplier" : byte, "Value" : short }	Topic which provides the chargers nominal line voltage in the parameter discovery phase of the charge. Will be or should be published from the beginning of the parameter phase of the charge which can be indicated using TOPIC_CHARGE_PARAMETER_STATUS.
TOPIC_EVSE_PARAMETER_RCD	"port0/ci/evse/parameter/rcd"	SimpleType	boolean "0" or "1"	Topic which provides the chargers status of the Residual Current Device in the parameter discovery phase of the charge. Will be or should be published from the beginning of the parameter phase of the charge which can be indicated using TOPIC_CHARGE_PARAMETER_STATUS.

Table 62 EVSE specific V2G parameters of the parameter discovery phase

15.5.6 EVSE specific V2G parameters of the charge phase

Topic Name	Topic	Type	Value	Comment
TOPIC_EVSE_CHARGE_NOTIFICATIONMAXDELAY	"port0/ci/evse/charge/notificationmaxdelay"	SimpleType	integer	Topic which provides the chargers notification max delay in the charge phase of the charge. Will be or should be published from the beginning of the parameter phase of the charge which can be indicated using TOPIC_CHARGE_CHARGE_STATUS.

Topic Name	Topic	Type	Value	Comment
TOPIC_EVSE_CHARGE_NOTIFICATION	"port0/ci/evse/charge/notification”	SimpleType	following enumeration table "EVSENotification-Type” ISO15118-2	Topic which provides the chargers notification in the charge phase of the charge. Will be or should be published from the beginning of the parameter phase of the charge which can be indicated using TOPIC_CHARGE_CHARGE_STATUS.
TOPIC_EVSE_CHARGE_METERINFO	"port0/ci/evse/charge/meterinfo"	ComplexType		Topic which provides the chargers meter info in the charge phase of the charge. Will be or should be published from the beginning of the parameter phase of the charge which can be indicated using TOPIC_CHARGE_CHARGE_STATUS.
TOPIC_EVSE_CHARGE_RECEIPTREQUIRED	"port0/ci/evse/charge/receiptrequired"	SimpleType		Topic which indicates if the charger requires a receipt of the meter info in the charge phase of the charge. Will be or should be published from the beginning of the parameter phase of the charge which can be indicated using TOPIC_CHARGE_CHARGE_STATUS.

Table 63 EVSE specific V2G parameters of the charge phase

15.5.7 EV specific V2G parameters of the initialisation phase

Topic Name	Topic	Type	Value	Comment
TOPIC_EV_INIT_EVCCID	"port0/ci/ev/init/evccid"	SimpleType	string	Topic which provides EVCCID which is mostly the MAC address of the EV´s comunication device. Will be or should be published from the beginning of the parameter phase of the charge which can be indicated using TOPIC_CHARGE_INIT_STATUS.
TOPIC_EV_INIT_SERVICESCOPE	"port0/ci/ev/init/servicescope"	SimpleType	string	Topic which provides the scope of the EV´s service discovery. Will be or should be published from the beginning of the parameter phase of the charge which can be indicated using TOPIC_CHARGE_INIT_STATUS.
TOPIC_EV_INIT_SERVICECATEGORY	"port0/ci/ev/init/servicecategory"	SimpleType	following enumeration table "serviceCategory-Type" ISO15118-2	Topic which provides the scope of the EV´s service discovery. Will be or should be published from the beginning of the parameter phase of the charge which can be indicated using TOPIC_CHARGE_INIT_STATUS.

Topic Name	Topic	Type	Value	Comment
TOPIC_EV_INIT_SELECTEDSERVICEID	"port0/ci/ev/init/selectedserviceid"	SimpleType	integer	Topic which provides the EV´s selected service ID. Will be or should be published from the beginning of the parameter phase of the charge which can be indicated using TOPIC_CHARGE_INIT_STATUS.
TOPIC_EV_INIT_SELECTEDPAYMENTOPTION	"port0/ci/ev/init/selectedpaymentoption"	SimpleType	following enumeration table "paymentOption-Type" ISO15118-2	Topic which provides the EV´s selected payment option. Will be or should be published from the beginning of the parameter phase of the charge which can be indicated using TOPIC_CHARGE_INIT_STATUS.

Table 64 EV specific V2G parameters of the initialisation phase

15.5.8 EV specific V2G parameters of the authentication phase

Topic Name	Topic	Type	Value	Comment
TOPIC_EV_AUTH_EMAID	"port0/ci/ev/auth/emaid"	SimpleType	string	Topic which provides the identifier of the charging contract at the very beginning of the authentication phase which can be indicated using TOPIC_CHARGE_AUTH_STATUS.
TOPIC_EV_AUTH_CONTRACTSIGNATURECERTCHAIN	"port0/ci/ev/auth/contractsignaturecertificatechain"	ComplexType		Topic which provides the contract certificate and optional sub certificates at the very beginning of the authentication phase which can be indicated using TOPIC_CHARGE_AUTH_STATUS.

Topic Name	Topic	Type	Value	Comment
TOPIC_EV_AUTH_CONTRACTID	"port0/ci/ev/auth/contractid"	SimpleType	string	Topic which provides the identifier of the charging contract at the very beginning of the authentication phase which can be indicated using TOPIC_CHARGE_AUTH_STATUS.
TOPIC_EV_AUTH_GENCHALLENGE	"port0/ci/ev/auth/genchallenge"	SimpleType	string	Topic which provides the challenge sent by the SECC which can be indicated using TOPIC_EVSE_AUTH_GENCHALLENGE.

Table 65 EV specific V2G parameters of the authentication phase

15.5.9 EV specific V2G parameters of the parameter phase

Topic Name	Topic	Type	Value	Comment
TOPIC_EV_PARAMETER_MAXSASCHEDULETUPLES	"port0/ci/ev/parameter/maxsascheduletuples"	SimpleType	integer	Topic which indicates the maximal number of entries in the SAScheduleTuple the EVSE shall provide. The EVSE can transmit up to the maximum number of entries defined in the parameter.
TOPIC_EV_PARAMETER_REQUESTEDENERGYTYPE	"port0/ci/ev/parameter/requestedenergytype"	SimpleType	following enumeration table "energyTransferType" ISO15118-2	Topic which provides the EV´s requested energy transfer type. Will be or should be published from the beginning of the parameter phase of the charge which can be indicated using TOPIC_CHARGE_PARAMETER_STATUS.

Topic Name	Topic	Type	Value	Comment
TOPIC_EV_PARAMETER_DEPARTURETIME	"port0/ci/ev/parameter/departuretime”	SimpleType	long integer	Topic which provides the intended departure time of the EV in seconds from the point in time when sending the according message. Will be or should be published from the beginning of the parameter phase of the charge which can be indicated using TOPIC_CHARGE_PARAMETER_STATUS.
TOPIC_EV_PARAMETER_EAMOUNT	"port0/ci/ev/parameter/eamount”	Complex Type	eAmountJsonObject {"Multiplier” : byte, "Value" : short }	Topic which provides the estimated amount of energy the EV will consume in the upcoming charge. Will be or should be published from the beginning of the parameter phase of the charge which can be indicated using TOPIC_CHARGE_PARAMETER_STATUS.
TOPIC_EV_PARAMETER_MAXVOLTAGELIMIT	"port0/ci/ev/parameter/maxvoltagelimit"	Complex Type	maxVoltageLimitJsonObject { "Multiplier" : byte, "Value” : short }	Topic which provides the maximum voltage limit supported by the EV in the parameter discovery phase of the charge. Will be or should be published from the beginning of the parameter phase of the charge which can be indicated using TOPIC_CHARGE_PARAMETER_STATUS.

Topic Name	Topic	Type	Value	Comment
TOPIC_EV_PARAMETER_MAXCURRENTLIMIT	"port0/ci/ev/parameter/maxcurrentlimit"	Complex Type	maxCurrentLimitJson Object { "Multiplier" : byte, "Value" : short }	Topic which provides the EVs maximum current limit supported by the EV in the parameter discovery phase of the charge. Will be or should be published from the beginning of the parameter phase of the charge which can be indicated using TOPIC_CHARGE_PARAMETER_STATUS.
TOPIC_EV_PARAMETER_MINCURRENTLIMIT	"port0/ci/ev/parameter/mincurrentlimit"	Complex Type	minCurrentLimitJson Object { "Multiplier" : byte, "Value": short}	Topic which provides the EVs minimum current limit supported by the EV in the parameter discovery phase of the charge. Will be or should be published from the beginning of the parameter phase of the charge which can be indicated using TOPIC_CHARGE_PARAMETER_STATUS.

Table 66 EV specific V2G parameters of the parameter phase

15.5.10 EV specific V2G parameters of the charge phase

Topic Name	Topic	Type	Value	Comment
TOPIC_EV_CHARGE_READYTOCHARGESTATE	"port0/ci/ev/charge/readytochargestate"	SimpleType	following enumeration table "chargeProgressType" ISO15118-2	Topic which provides if the EV is ready to charge. Will be or should be published from the END of the parameter phase of the charge which can be indicated using TOPIC_CHARGE_PARAMETER_STATUS OR it will end the pre charge phase which can be indicated using TOPIC_CHARGE_PRECHARGE_STATUS.

Topic Name	Topic	Type	Value	Comment
TOPIC_EV_CHARGE_CHARGINGPROFILETUPLEID	"port0/ci/ev/charge/chargingprofiletupleid”	SimpleType	string	Topic which provides the selected SAScheduleTupleID from the TOPIC_EVSE_PARAMETER_SASCHEDULELIST. Will be or should be published from the beginning of the charge phase of the charge which can be indicated using TOPIC_ CHARGE_CHARGE_STATUS.
TOPIC_EV_CHARGE_CHARGINGPROFILEENTRY	"port0/ci/ev/charge/chargingprofileentry"	Complex Type	chargingProfileJson Object: { "Start” : [int,...], "PMax” : [int,...], "PMaxMultiplier” : [int,...], "NumberOfPhases” : [int,...] }	Topic which provides information about the selected charging profile from the TOPIC_EVSE_PARAMETER_SASCHEDULELIST. Will be or should be published from the beginning of the charge phase of the charge which can be indicated using TOPIC_CHARGE_CHARGE_STATUS.

Table 67 EV specific V2G parameters of the charge phase

15.5.11 EV specific V2G parameters of the session stop request

Topic Name	Topic	Type	Value	Comment
TOPIC_EV_STOP_PROGRESS	"port0/ci/ev/stop/progress”	SimpleType	following enumeration table "chargingSessionType” ISO15118-2	Topic which provides the EV´s information if the charging process shall either be terminated or paused. Will be published if the SessionStopRequest is received on EVSE side.

Table 68 EV specific V2G parameters of the session stop request

15.6 Programming example for subscribing and publishing topics

```
void handleStatusTopics(struct mosquitto *mosq, void *userdata, const struct mosquitto_message *message)
{
    int result;
    if (strcmp(message->topic, TOPIC_CHARGE_INIT_STATUS) == 0) {
        if (strcmp((char *) message->payload, "started") == 0) {
            char * sessionId = generateSessionId();
            result = mosquitto_publish(mosq, NULL, TOPIC_EVSE_INIT_SESSIONID, sizeof (sessionId), sessionId, qosLevel,
retain);

            char * evseId = getEVSEId();
            result = mosquitto_publish(mosq, NULL, TOPIC_EVSE_INIT_EVSEID, sizeof (evseId), evseId, qosLevel, retain);

            char * datetimenow = getActualTime().toCharArray();
            result = mosquitto_publish(mosq, NULL, TOPIC_EVSE_INIT_DATETIMENOW, sizeof (datetimenow), datetimenow,
qosLevel, retain);
            // ... publish every further topic which can be published here
        }
    }
    else if (strcmp(message->topic, TOPIC_CHARGE_AUTH_STATUS) == 0) {
        if (strcmp((char *) message->payload, "started") == 0) {
            char authenticated = (char) authenticateUser();
            if (EVSEProcessing.Finished == authenticated) { // some enumerated value ???
                result = mosquitto_publish(mosq, NULL, TOPIC_EVSE_AUTH_EVSEPROCESSING, 1, &authenticated, qosLevel,
retain);
            }
            // error case
            else if (Authentication.Error == authenticated) {
                char error = '1';
                result = mosquitto_publish(mosq, NULL, TOPIC_EVSE_AUTH_ERROR, 1, &error, qosLevel, retain);
            }
        }
    }
    else if (strcmp(message->topic, TOPIC_CHARGE_PARAMETER_STATUS) == 0) {
        if (strcmp((char *) message->payload, "started") == 0) {
            char * nominalVoltage = createNominalVoltageJSonObject();
            result = mosquitto_publish(mosq, NULL, TOPIC_EVSE_PARAMETER_NOMINALVOLTAGE, sizeof (nominalVoltage),
nominalVoltage, qosLevel, retain);
            // and so on ...
        }
    }
    else if (strcmp(message->topic, TOPIC_CHARGE_CHARGE_STATUS) == 0) {
        if (strcmp((char *) message->payload, "started") == 0) {
            startCharging(); // close contactors before
        }
    }
}

void handleCharge(struct mosquitto *mosq, void *userdata, const struct mosquitto_message *message) {
    if (strcmp(message->topic, TOPIC_EV_CHARGE_READYTOCHARGESTATE) == 0) {
        if (strcmp((char *) message->payload, "Start") == 0) {
            closeContactors();
        }
    }
    // ...
}

......
```

15.7 Physical value type

The physical value type is used to determine the parameters of the power electronic. This type is defined as JSON object and consists of three name/value pairs.

```
physical_value_json_object{
  "Multiplier":byte,
  "Value":short,
  "Unit":byte //optional
}
```

The table below shows the physical value type definition.

Key	Type	Values
Multiplier	byte	-3, -2, -1, 0, +1, +2, +3
Unit	byte	0 (hours), 1 (minutes), 2 (seconds), 3 (ampere) 4 (volt), 5 (watt), 6 (watt-hours)
Value	short	-32.768 to +32.767

Table 69 Physical value type definition

When electrical parameters are signalized, their units do not need to be transmitted because those are predefined in the ISO15118-2 table 68. So for current and voltage the units will be omitted from the JSON objects.

15.8 Basic SECC configuration

The basic SECC configuration is stored in path /etc/secc/ under the JSON file "customer.json". The JSON object "grid" provides all power-electronic specific parameters. The configuration will be used to initialize the charging parameters and will be automatically loaded after the EV plug is connected. The topic TOPIC_CHARGE_INIT_STATUS with "started" signalizes that the initialization is finished and the MQTT interface is ready to receive messages. Most of the default parameters can be overwritten with the content of the respective MQTT message. If a developer doesn´t need to provide other values, the according topics do not need to be sent.

15.9 Stop charging and error shutdown on EVSE side

To stop the charge the EVSE has the possibility to send topic TOPIC_EVSE_*_NOTIFICATION with payload "StopCharging" ('1') in some phases of the charge. The "EVSENotification" topic is not provided in every charging phase. Within the authentication phase the TOPIC_EVSE_AUTH_ERROR topic can be used to abort a charging session. Note, there can be EVs which don´t support the stop by ignoring the "EVSENotification". In this case we suggest to abort the charge by using TOPIC_EVSE_AUTH_ERROR. Contactors should be opened immediately after sending this topic. For external broker communication over the Ethernet interface it is recommended to use the "Last Will and Testament" (LWT) mechanism of MQTT to handle a situation if the connection is closed unexpectedly.

15.10 Stop charging and error shutdown indication on EV side

A charging session can be interrupted by the EV in any charging phase. The common way is that the EV changes the CP State from 'C' to 'B' within a running charging phase (See chapter Charge status information). The charging stack observes the CP State throughout the entire charging session and closes the TCP connection if an unexpected CP State has been detected. In addition, the EV can set a failed error code in one of the possible error code topics. The error codes are described in the ISO15118/DIN70121 standards. The default value, when the EV has no error detected, is '0' ("No Error"). The error codes are intended for informational purposes only, and they shall not influence the EVSE charging process.

15.11 Renegotiation process

In ISO15118 it is possible to renegotiate the charging schedule between EV and EVSE. Both EV and EVSE can trigger the renegotiation process and can interrupt the charging progress. The EVSE can request the renegotiation process over the phase specific message type "NOTIFICATION”. This parameter must be set to "Renegotiation” (’3’). Either of an EVSE request or own request the EV signalizes the start of renegotiation process over the topic TOPIC_EV_CHARGE_READYTOCHARGESTATE with "Renegotiate" (’2’). The figure "charge flow” of chapter Charge status information shows the typical message sequence of the renegotiation process. If the renegotiation process has started, the charging phase will be interrupted and the EV switches to CP State B. After receiving of the topic TOPIC_CHARGE_PARAMETER_STATUS with "started” the charging schedule must be updated over topic TOPIC_EVSE_PARAMETER_SASCHEDULELIST. The physical values shall be set to a valid state before starting the next phase. If the EVSE is ready to continue the charging progress, phase "PARAMETER" must be set to "finished” (’0’) with the EVSE-Processing parameter (See chapter EVSEProcessing).

List of topics which will be reset if the renegotiation process has started:

1. TOPIC_EVSE_PARAMETER_EVSEPROCESSING (Ongoing ’1’)
2. TOPIC_EVSE_PARAMETER_SASCHEDULELIST (Empty list)

15.12 Pausing and resuming of a charging session

In ISO15118 it is possible that the EV can pause a charging session and later resume it. The figure "charge flow" of chapter Charge status information shows the typical message sequence if the EV uses the pause mechanism. This feature is currently not implemented.

15.13 Internal error behavior & safe state

The Charging stack always monitors the internal hardware abstraction (e.g. contactor or locking motor). In case it doesn’t react within the defined times, the stack goes into an error mode and tries to achieve a safe state. This state is defined as following:

Topic Name	Topic	state
TOPIC_CONTACTOR_STATE_TARGET	"port0/contactor/state/target"	0
TOPIC_VENTILATION_STATE_TARGET	"port0/ventilation/state/target"	0
TOPIC_CHARGING	"port0/charging"	0
TOPIC_PLUG_LOCK	"port0/plug_lock/state/target"	0
TOPIC_CP_DUTY_CYCLE	"port0/cp/duty_cycle"	0.00 (unrecoverable) or 100.00 (recoverable)

Table 70 Internal error behaviour & safe state

Note: The duty cycle in safe state depends on whether the error case is considered as unrecoverable (e.g. RCD error, contactor error) or not (e.g. plug lock error).

Additionally in case of a permanent plug lock failure, the Charging stack tries to recover from this situation by driving back to the last valid state and then to the desired state.

15.14 Emergency alarm

In order to terminate the power supply for safety reasons the Charging stack supports an emergency alarm. This feature must be configured in the `customer.json` accordingly. In case the emergency switch has been asserted, the Charging stack tries to achieve the safe state (unrecoverable) as fast as possible.

15.15 RCD monitoring and testing

In order to detect and react to residual direct currents, the Charging stack supports RCD monitoring and testing. This feature must be configured in the customer.json (Charging Stack Configuration Files).

Both features, RCD monitoring and testing, are optional features. The RCD monitoring can be used without testing, but to be in line with the standard IEC62955, RCD testing must be enabled when using RCD's.

15.15.1 Hardware configuration

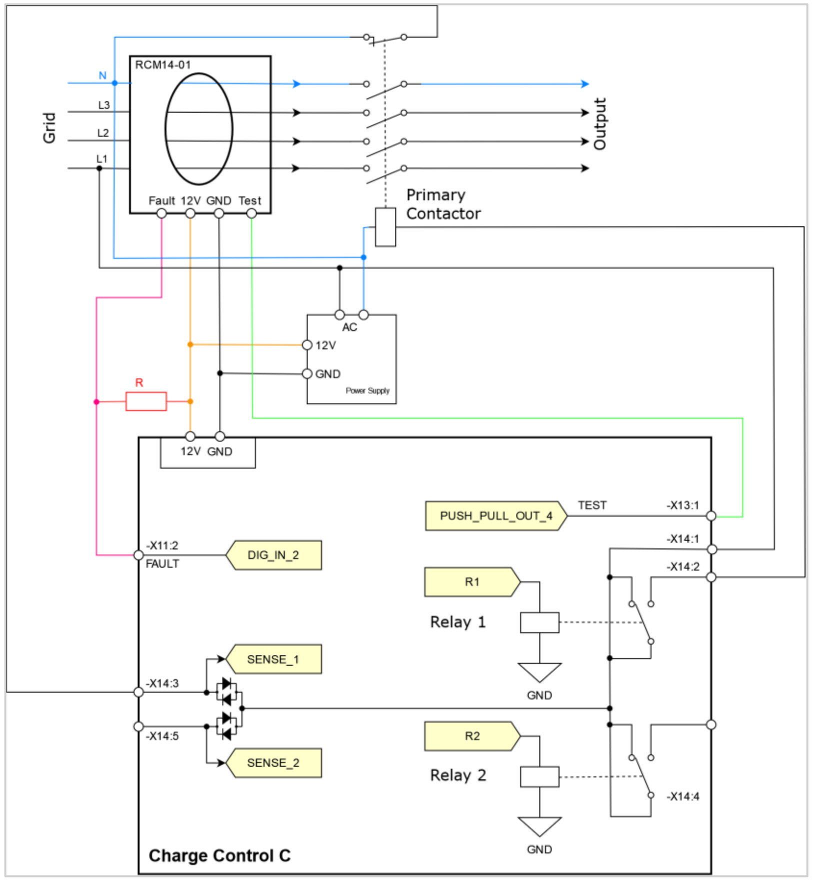


Figure 19 RCD connection

Parameter in customer.json	Default	Remark
ports[0]/rcd_monitor/gpio	122 (= DIG_IN_2)	GPIO for RCD feedback
ports[0]/rcd_monitor/polarity	active high	GPIO polarity for RCD feedback
ports[0]/rcd_monitor/test_gpio	87 (= PUSH_PULL_OUT_4)	GPIO for RCD test trigger
ports[0]/rcd_monitor/test_gpio_polarity	active high	GPIO polarity for RCD test trigger

Note: The feedback GPIO pin must not be in floating state. To ensure this, connect a pullup or pulldown resistor to this GPIO pin, depending on the RCD device. This information can be taken from RCD datasheet.

15.15.2 RCD monitoring behavior

There are basically three errors. The error will be sent to the backend via OCPP (see MQTT topics)

	Description	Reaction
SpuriousRcdError	This error occurs when RCD feedback signal reports an error, and no EV is connected. When this error occurs the error source has to be searched in Charging station and equipment.	• leads to a non-recoverable error state (no charging possible) • the error must be cancelled manually • OCPP status notification is sent to backend with error code (see OCPP StatusNotification)
RcdTestError	This error occurs when RCD test failed. The error source has to be searched at the RCD device or wiring.	• leads to an non-recoverable error state (no charging possible) • the error must be cancelled manually • OCPP status notification is sent to backend with error code (see OCPP StatusNotification)
RcdGroundFailure	This error occurs when RCD feedback signal reports an error during an EV is connected. When error is still available after EV is disconnected, the error is going to be a SpuriousRcdError. When error disappears, after EV is disconnected, the error source has to be searched at the EV.	This error is a recoverable error. The error is only present when EV is connected. When error is still available after disconnection, the error leads to a non-recoverable state (no charging possible). OCPP status notification is sent to backend with error code (see OCPP StatusNotification) Otherwise, no error is available anymore and a new charging session can be started. An empty string is sent to the backend via OCPP.

15.15.3 RCD test behavior

The automatic RCD test will be executed at:

- every start of the charging station

- before each charging session when EV is going to be connected
- after 24h have passed

The following graph shows the moments when an automatic RCD test is executing and the reaction to the RCD test result.

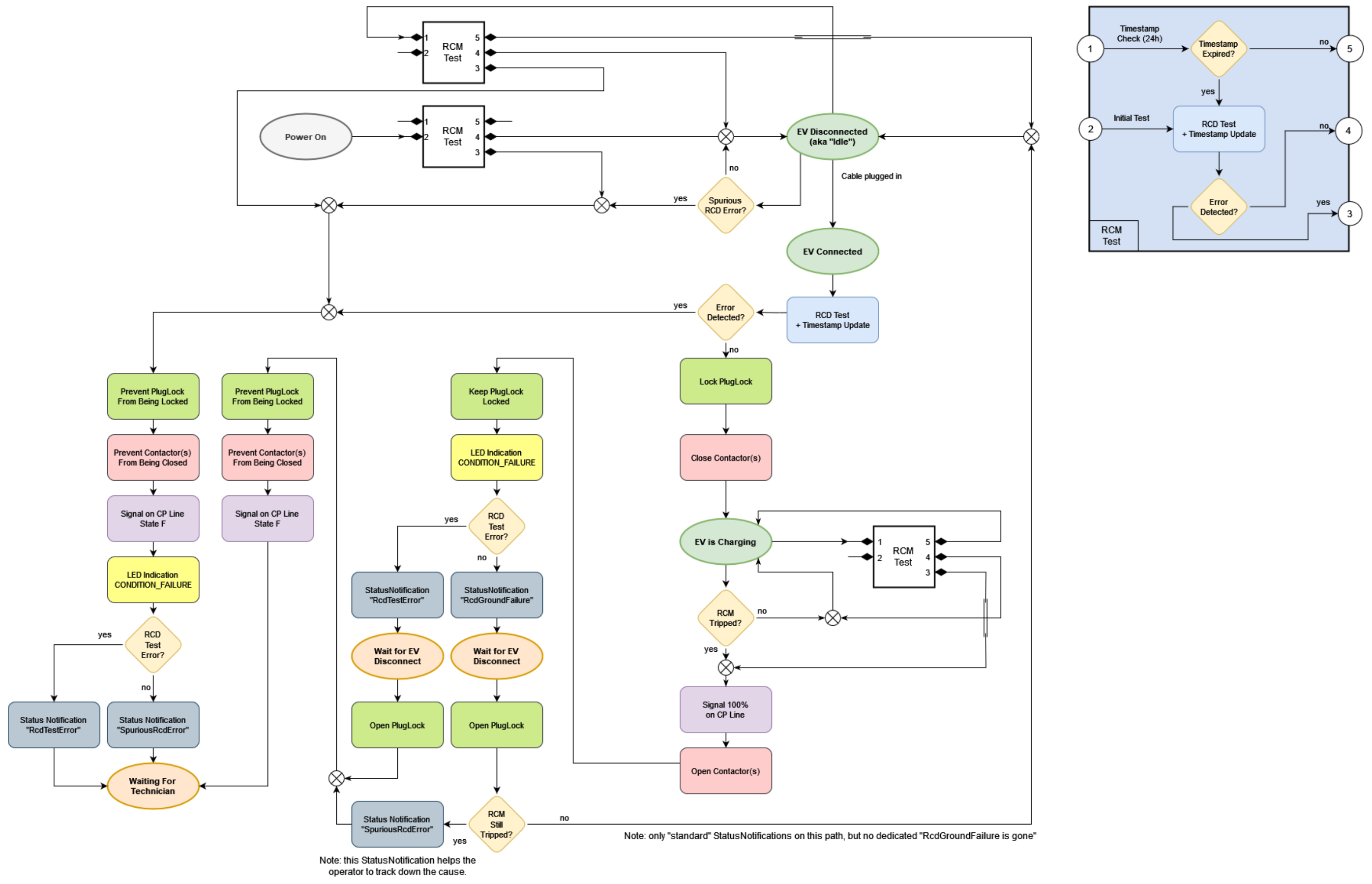

RCM Test
1
2
3
4
5
Power On
EV Disconnected (aka "Idle")
Cable plugged in
no
yes
Spurious RCD Error?
EV Connected
RCD Test + Timestamp Update
Error Detected?
Lock PlugLock
Close Contactor(s)
EV is Charging
RCM Tripped?
Signal 100% on CP Line
Open Contactor(s)
Prevent PlugLock From Being Locked
Prevent Contactor(s) From Being Closed
Signal on CP Line State F
LED Indication CONDITION_FAILURE
RCD Test Error?
Status Notification "RcdTestError"
Status Notification "SpuriousRcdError"
Waiting For Technician
Keep PlugLock Locked
StatusNotification "RcdTestError"
StatusNotification "RcdGroundFailure"
Wait for EV Disconnect
Open PlugLock
RCM Still Tripped?
Status Notification "SpuriousRcdError"
Note: this StatusNotification helps the operator to track down the cause.
Note: only "standard" StatusNotifications on this path, but no dedicated "RcdGroundFailure is gone"
Timestamp Check (24h)
Timestamp Expired?
Initial Test
RCD Test + Timestamp Update
Error Detected?
RCM Test

Figure 20 RCD test overview

15.15.4 RCD test timings

Due different functionality of RCD sensors, the timings are configurable in the customer.json (Charging Stack Configuration Files). The respective times must be determined by the customer. Datasheets can be checked for timing diagrams.

For example: The Western Automation RCM 14-01 needs to hold the trigger output pin high until the RCD feedback is checked. The Bender RCD-121 just needs a short 50ms impulse to trigger a test.

The following graph shows the configurable timings and relationship for an RCD test.

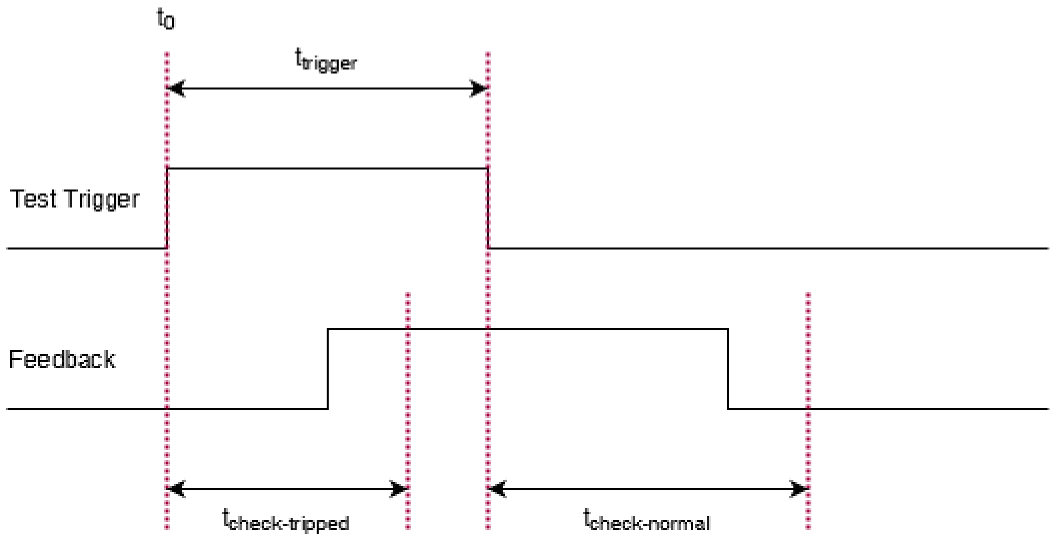


Figure 21 RCD test timing

Parameter	Parameter in customer.json	Constraints
$t_{trigger}$	ports[0]/rcd_monitor/test_trigger_time	0 ms < $t_{trigger}$ < 5000 ms
$t_{check\text{-}tripped}$	ports[0]/rcd_monitor/test_check_tripped_time	0 ms < $t_{check\text{-}tripped}$ < 5000 ms
$t_{check\text{-}normal}$	ports[0]/rcd_monitor/test_check_normal_time	0 ms < $t_{check\text{-}normal}$ < 5000 ms
		$t_{check\text{-}tripped} < t_{trigger} + t_{check\text{-}normal}$ (this constraint ensures the order of 1st "tripped" and back to "normal" later)

Table 71 Parameter description

	Western Automation RCM14-01 in ms	Bender RCD-121 in ms
$t_{trigger}$	815	50
$t_{check-tripped}$	810	750
$t_{check-normal}$	410	2000

Table 72 Examples for RCD devices

15.15.5 MQTT topics

Topic Name	Topic	Type	Value	Comment
TOPIC_RCD_STATE_ACTUAL	"port0/rcd/state/actual"	SimpleType	bool	Actual RCD state (1 = not tripped, 0 = tripped) This topic is only going to state "tripped" if a residual current occurs or a spurious RCD error comes up. During a test this topic is going to be ignored and in state "not tripped".
TOPIC_RCD_FEEDBACK_AVAILABLE	"port0/rcd/feedback/available"	SimpleType	bool	Flag to indicate when RCD feedback is configured (0 = not available, 1 = available)
TOPIC_RCD_TEST_AVAILABLE	"port0/rcd/test/available"	SimpleType	bool	Flag to indicate when RCD test is configured (0 = not available, 1 = available)
TOPIC_RCD_TEST_PERFORM	"port0/rcd/test/perform"	SimpleType	-	Internal signal to trigger an RCD test (no retained). The end of the test is signalized via TOPIC_RCD_ERROR.
TOPIC_RCD_TEST_TIMESTAMP	"port0/rcd/test/timestamp"	SimpleType	string	This topic tracks when the latest RCM test finished. It can be used to monitor the regular testing of the RCM, e.g. every 24h. Timestamp in RFC 3339 format, e.g. "2023-06-23T06:54:55.123Z"
TOPIC_RCD_ERROR	"port0/rcd/error"	SimpleType	string	This topic indicates the type of RCD error to inform the ocppd and/or customer implementation about the detected error type: 1. In case of a residual RCD error detected during a charging session, this topic holds the string "RcdGroundFailure".

Topic Name	Topic	Type	Value	Comment
				2. In case of a spurious RCD error during a non-charging state this topic holds the string "SpuriousRcdError", and topic "port0/rcd/state/actual" is going to state "tripped" 3. In case of an error caused by a test, this topic holds the string "RcdTestError" (topic "port0/rcd/state/actual" keeps state "not tripped"), and when a test succeeded without error, then this topic is published as empty string. 4. When "RcdGroundFailure" disappears, this topic holds an empty string.

Table 73 MQTT topics for RCD handling

15.16 Current limits for basic AC charging

The charging stack internally determines the minimum of all current limits. All these values are considered in Ampere. The special value of -1 should be considered as no current limit. The following current limits apply equally to single- and three-phase systems, i.e. for a three-phase system a current limit means that the current is allowed on each phase.

Topic Name	Topic	Type	Value	Comment
TOPIC_CABLE_CURRENT_LIMIT	"port0/cable_current_limit"	SimpleType	integer	Internal topic which indicates the supported current by the attached cable. This is published by Charge Control C and should not be overwritten. The value is determined by evaluating the Proximity Pilot signal. In case the charging station is equipped with a fixed cable, i.e. the parameter "portX/pluggable" in configuration file "customer.json" is set to "false", then the cable rating must be configured also in that configuration file via parameter "portX/pp/cable_current_limit". This MQTT topic is published as "-1" in this case, and the value is ignored by the charging stack - the value from the configuration file is used directly.
TOPIC_EVSE_GRID_CURRENT_LIMIT_ACTUAL	"port0/ci/evse/basic/grid_current_limit/actual"	SimpleType	integer	Topic which indicates the maximum current rating of the grid cabling and fusing the charging station is connected to. This is published and evaluated once by Charge Control C and should not be overwritten.
TOPIC_EVSE_EVSE_CURRENT_LIMIT_ACTUAL	"port0/ci/evse/basic/evse_current_limit/actual"	SimpleType	integer	Topic which indicates the maximum current rating of EVSE's internal cabling. This is published and evaluated once by Charge Control C and should not be overwritten.

Topic Name	Topic	Type	Value	Comment
TOPIC_EVSE_BASIC_MAXCURRENTLIMIT	"ci/evse/basic/maxcurrentlimit"	SimpleType	float	Topic which indicates the maximum current of the whole charge point used for dynamic load management. It can be represented as an integer, or as a floating point number with one decimal place and a "." as decimal separator, e.g. "20.2". This value is intended for OCPP smart charging. A value below 6 effectively sets the limit to 0 and forces a CP duty cycle of 100%, but keeps the current contactor state.
TOPIC_GLOBAL_DYN_CURRENT_LIMIT	"ci/global/dyn_current_limit/+"	SimpleType	float	Topic which can be used to configure the maximum current of the whole charge point used for dynamic load management by customer applications. It can be represented as an integer, or as a floating point number with one decimal place and a "." as decimal separator, e.g. "20.2". The names used in place of the '+' wildcard character allow to publish up to 5 different current limits to the Charge Control C. These names are free to choose, but should be kept as short as possible. The topic names and the values are not stored persistently, so after a reboot they all get lost. A value below 6 effectively sets the limit to 0 and forces a CP duty cycle of 100%, but keeps the current contactor state. See the example below to publish this topic.

Topic Name	Topic	Type	Value	Comment
				As notes above, the value published here applies to each individual phase in a three-phase system. The charging stack does not spread it itself to the available phases. The customer application is in charge to track the actual available phase count, e.g. single-phase vs. three-phase system and/or phase count switching feature is enabled and in effect. It is recommended to publish this topic as "retained" so that e.g. other MQTT clients are aware of it when connecting to the broker after the limit was published.
TOPIC_EVSE_BASIC_OFFERED_CURRENT_LIMIT	"port0/ci/evse/basic/offered_current_limit"	SimpleType	float	Topic which indicates the overall offered current limit of the EVSE. It represents a floating point number with one decimal place and a "." as decimal separator, e.g. "20.2". This topic is published each time a current limit is configured via topic or configuration file. This topic can also be used as feedback signal of a dynamic current request sent via topic TOPIC_GLOBAL_DYN_CURRENT_LIMIT.

Topic Name	Topic	Type	Value	Comment
TOPIC_EVSE_BASIC_PHYSICAL_CURRENT_LIMIT	"port0/ci/evse/basic/physical_current_limit"	SimpleType	integer	Topic which indicates the maximum possible current which can be offered by the EVSE when no dynamic limit applies. It is determined by the charging stack by using the minimum of the values grid_current_limit, evse_current_limit and cable_current_limit - but regardless of any dynamic limits set by customer. This topic is published each time when any of these limits change, e.g. when cable is plugged in and the station does not have a fixed cable.

Table 74 Current limits for basic AC charging

15.16.1 Example to publish dynamic current limits over MQTT

Publish up to 5 different limits. Notice how the 6th published limit is ignored:

```
ci/global/dyn_current_limit/internal_example_limit 6
ci/global/dyn_current_limit/custom_ocpp_limit 13
ci/global/dyn_current_limit/special_limit 20
ci/global/dyn_current_limit/custom_limit_1 32
ci/global/dyn_current_limit/custom_limit_2 63
ci/global/dyn_current_limit/ignored_limit 1

Configured current limit: 6 A (value of internal_example_limit,
because it is the lowest value of the five relevant topics, the sixth
topic is ignored)
```

Once published the limits can be overwritten at runtime by retransmitting the topic:

```
ci/global/dyn_current_limit/internal_example_limit 20

Configured current limit: 13 A (value of custom_ocpp_limit is now the
lowest value of the five relevant topics)
```

Note: Please consider that the current limit is determined as a minimum over all current limits. Thus, the value of the current limit can only be adjusted over topic TOPIC_GLOBAL_DYN_CURRENT_LIMIT if it fits with the other limitations.

15.17 RFID authorization

Topic Name	Topic	Type	Value	Comment
TOPIC_RFID_AUTHORIZATION_REQUEST	"port0/rfid/authorization_request"	ComplexType	array of strings	Internal topic which contains the RFID standard used and the UID of the accompanying RFID tag. The latest version of the charging stack expects that this topic is **not** retained. The RFID standard element is a string. Supported standards are "ISO14443" and "ISO15693". In case the standard information is not available in the implementation, "UNKNOWN" is used. The UID is also a string. The format is hex representation, little endian, zero filled and without any delimiter. Element 0: RFID standard Element 1: RFID tag Example:

				["ISO14443","1F2D 3A4F5506C7"]

Table 75 RFID authorization

Note: The old topic port0/rfid/authorizereq for signalling the RFID tag has been marked **deprecated**. To ensure backwards compability the old RFID topic is still available but will be removed soon.

15.18 Sharing one RFID reader between multiple Charge Control C

Charge Control charging stack provides the ability to share a single RFID reader between multiple Charge Control C. This requires that the Charge Control C without the RFID reader is available under a fixed address. Also the Charge Control C with the RFID reader must be able to connect to the MQTT broker of the other board.

Parameter	Value	Note
ocpp/rfidStopTransaction	false	This value is required.
ocpp/rfidRequiresEvPresent	true	This value is required.
ports[0]/user_authentication	ocpp	
ports[0]/rfid/enable	true	
ports[0]/rfid/protocol	Stronglink	
ports[0]/rfid/remote_ports[]/uri	mqtt://192.168.1.5	Only first 3 entries will be handled.

Table 76 Example settings for Charge Control C with RFID reader (MQTT Master):

Parameter	Value	Note
ocpp/rfidStopTransaction	false	This value is required
ocpp/rfidRequiresEvPresent	true	This value is required
ports[0]/user_authentication	ocpp	
ports[0]/rfid/enable	true	
ports[0]/rfid/protocol	mqtt	

Table 77 Example settings for Charge Control C without RFID reader (MQTT Slave):

Note: It is possible to share an RFID reader and a USB internet dongle. This requires at least a static IP configuration of the MQTT Slave, otherwise those boards would race for the dynamic IP addresses from the dongle.

15.18.1 Shared RFID authorization behavior

Sharing one RFID reader requires a special authorization behavior:

- no EV connected → RFID tag ignored
- one non-charging EV connected → RFID tag accepted
- one non-charging EV connected, one charging EV connected → RFID tag accepted
- two non-charging EV connected since up to 60 seconds → RFID tag accepted, last one will be selected
- two non-charging EV connected since more than 60 seconds → RFID tag ignored
- two charging EV → RFID tag ignored

15.19 Ventilation Control

Charge Control C allows to connect an external ventilation on relay 2 of the board. The Charge Control C configuration offers two options for the ventilation control. The ventilation can be controlled "internally" by the charging software or "externally" by the customer's own software via MQTT topics.

The table below shows the relevant MQTT topics for "external" ventilation control.

Topic Name	Topic	Type	Comment
TOPIC_VENTILATION_AVAILABLE	"port0/ventilation/available"	bool	This topic published once by charging software during startup when internal ventilation is enabled via configuration file. The payload is set to "1" if the ventilation control is enabled, otherwise "0". This topic must be published by the customer if "external" ventilation is enabled in the configuration.

Topic Name	Topic	Type	Comment
TOPIC_VENTILATION_STATE_TARGET	"port0/ventilation/state/target"	bool	This topic is used to control the connected ventilation fan. This topic must be published by the customer software if the external control mode is enabled via configuration, otherwise it is published by the charging stack itself. The payload must be set to "1" if the ventilation should be started (relay 2 should be closed) and "0" if the ventilation should be stopped (relay 2 should be opened).

Topic Name	Topic	Type	Comment
TOPIC_VENTILATION_STATE_ACTUAL	"port0/ventilation/state/actual"	bool	This topic provides the actual ventilation state as read from the related GPIO (see table Relays). This topic will be published by the charging software. The payload is set "1" if the ventilation is started (relay 2 is closed) and "0" if the ventilation is stopped (relay 2 is opened). This topic can be used to check whether the relay has actually switched. Note: If no feedback is configured in the customer.json this topic is only published as confirmation that the target value was received successfully.

Table 78 MQTT topics for ventilation

The next table gives an overview of different use cases for the ventilation control and shows the necessary configurations to configure them.

Use case	Description	customer.json configuration file		MQTT topic		
		ventilation/enable	**ventilation/control**	**port0/ventilation/available**	**port0/ventilation/state/target**	**port0/ventilation/state/actual**
1	Customer doesn't want to use ventilation, and also doesn't want to use the relais at all	false	internal	"0" is published by charging stack	not published by charging stack	published by charging stack
2	Customer doesn't want to use ventilation, but wants to use the relais for custom purpose	false	external	"0" must be published by external/customer software, but should not be of interest at all	not published by charging stack, can be used by customer/external software to switch the relay for customer purpose	published by charging stack
3	Customer wants to use ventilation (controlled by charging stack)	true	internal	"1" is published by charging stack	published by charging stack as soon as MQTT topic "available" is published	published by charging stack

Table 79 Use cases for the ventilation control

The next table shows the preconditions of the config parameter and MQTT topics to allow charging with requested ventilation by the EV (CP State D charging).

customer.json configuration file always_accept_cp_state_d	customer.json configuration file ventilation/enable	customer.json configuration file ventilation/control	MQTT topic port0/ventilation/available	MQTT topic port0/ventilation/state/actual	Remarks
true	don't care	don't care	don't care	don't care	The EVs request for ventilation is always accepted.
false	false	internal	don't care	don't care	Charging is not allowed. This is indicated with duty cycle of 100%. The plug lock (if available) is/remains closed.
		external	don't care	is published by charging stack	
	true	internal	is published by charging stack	is published by charging stack	
		external	must be published by external/customer software	must be published by external/customer software depending on the actual request (port0/ventilation/state/target) by charging stack	Charging is only possible when external/customer software announces that ventilation is actually available.

Table 80 Requirements for CP state D charging

15.20 Phase Count Switching between 3- and 1-phase Charging

Legal notice: This feature is considered as experimental. chargebyte assumes no liability for damage to electric vehicles caused by phase count switching.

Instead of using relay 2 on the board for a fan, it is also possible to use it for phase count switching during a basic charging session (PWM controlled). In this case, two switching devices need to be connected to relay 1 and 2. The phase count switching must be triggered via MQTT. Please note that this feature only works with Charge Control C 200-300 and will interrupt an ongoing charging session if there is any. If the switching is triggered during an active charging session the stack will pause and try to resume the charging process.

The following requirements must be fulfilled to use this feature:

- Charge Control C 200 or 300 must be used (Charge Control C 100 is only equipped with one relay)

- rotary encoder switch must be set to 3 phases
- charging type in customer.json must be configured to "basic"
- ventilation control for relay 2 must be disabled (see above)

Currently there are the following limitations:

- there is no kind of wake-up function in case the EV doesn't react on the CP duty cycle change after the switch delay
- it's not possible to use this feature via OCPP smart charging

The diagram below shows the corresponding hardware setup for this feature. This example wiring is using additional NC feedback contacts of the primary and secondary contactors, however, both feedback types can be configured individually in the customer.json configuration file. In this example wiring, a primary contactor which is only rated for 230 V AC could be used, since the same phase is used for controlling the relay and which is switched through to the EV. However, the secondary contactor must still be rated for 400 V AC since two different phases are switched.

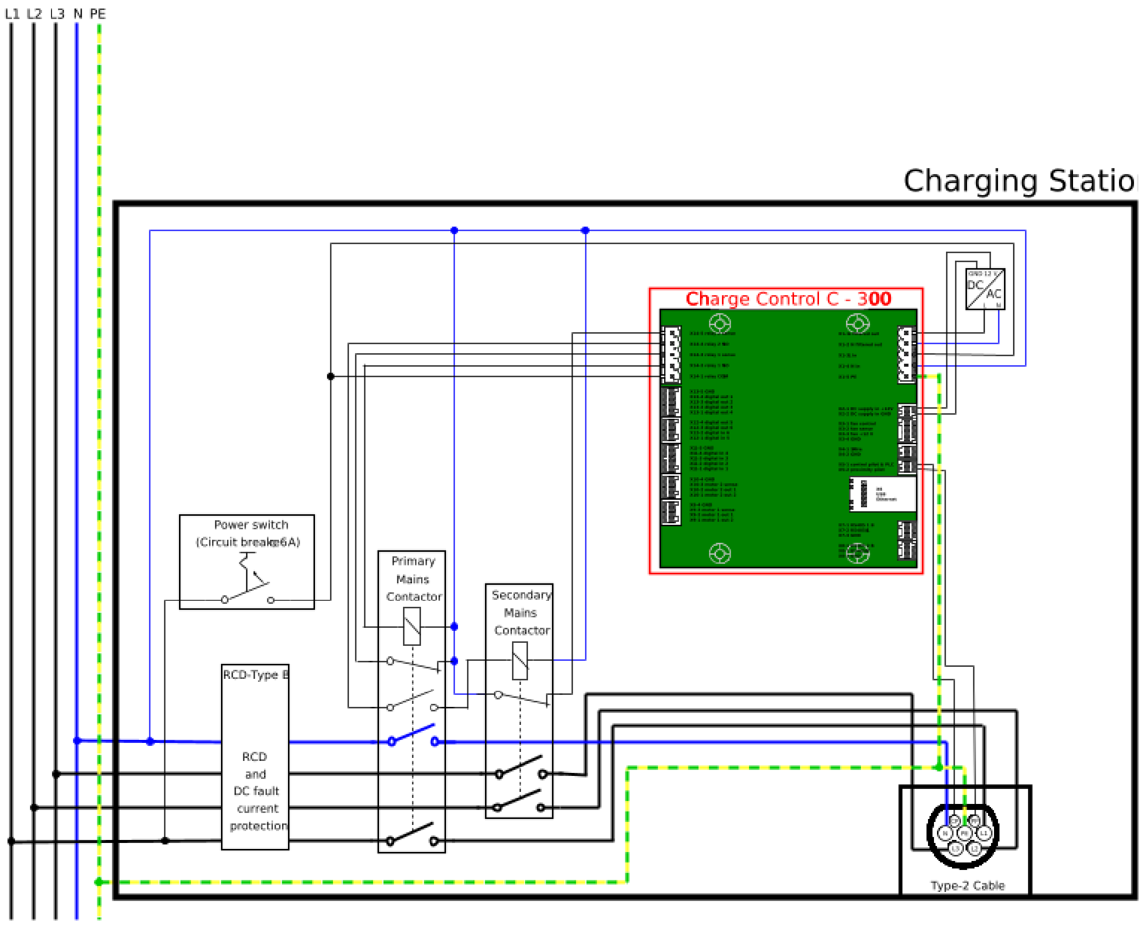


Figure 22 Wiring Diagram for Phase Count Switching

Note, that the order for switching on/off both relays is fixed:

- providing power to the EV: secondary contactor is closed first (if the requested count of phases require it), then primary contactor is closed
- revoking power from the EV: primary contactor is opened first, then secondary contactor is opened (if it was closed)

The table below shows the relevant MQTT topics for "phase count switching".

Topic Name	Topic	Type	Range	Comment
TOPIC_EVSE_PHASE_TARGET	"port0/ci/evse/phase/target"	integer	1 or 3	Target count of phases to switch to. Invalid values are ignored.
TOPIC_EVSE_PHASE_ACTUAL	"port0/ci/evse/phase/actual"	integer	0, 1 , 3	Actual phase count.
TOPIC_EVSE_PHASE_SWITCH_DELAY	"port0/ci/evse/phase/switch_delay"	integer	10-300	Delay in seconds between switching off the first contactor and re-enabling the CP duty cycle, Invalid values are ignored.

Table 81 MQTT Topics for Phase Count Switching

The table below describes the relevant configuration parameters.

Configuration (found in file /etc/secc/customer.json)			
Parameter	Description	Type	Default
ports[0]/switch3to1phase/enable	Enables or disables the phase count switching feature. Enabling this feature prohibits use of ventilation.	boolean	false
ports[0]/switch3to1phase/feedback_type	Defines the logic behind the feedback of the secondary contactor (nc = normally close, no = normally open).	string	no
ports[0]/switch3to1phase/switch_delay	Defines the length in seconds of a safety/guard interval which is enforced during switching the phase count. After a phase count change is requested, the charging stack will first switch off all phases, then wait this amount of time, and finally present the new phase count to the EV. This value can also be changed during runtime via MQTT. Possible values are: 10 - 300	integer	20

Table 82 Configuration Parameter for Phase Count Switching

15.21 Fake High-Level DC Charging Mode

Charge Control C supports a feature to retrieve the MAC address and SOC (State of Charge) value of a high-level communication (HLC) session and to switch automatically to a basic AC PWM charging session afterwards. This mode is called "fake_highlevel_dc" and can be activated in the customer.json file over the key "ports[0]/charging_type" and option "basic+fake_highlevel_dc".

When the feature is activated the charging software indicates to the EV that HLC is required. The charging software waits at least 20 seconds for the EV to start the SLAC process. In case a HLC charging session could be established, at least the MAC address is published via MQTT topic TOPIC_V2G_MAC. The MAC address can be used by customers to implement an authentication method for a basic AC PWM charging session. For this use-case the customer.json key "ports[0]/user_authentication" must be configured to "mqtt" and the MQTT topic TOPIC_AUTHORIZATION_STATUS needs to be published with payload '1' when the EV MAC address is authorized for charging. If the customer wants to use this feature in combination with an OCPP backend, then the key "ports[0]/user_authentication" must be configured to "ocpp", the remaining OCPP client configuration must be valid and the OCPP client must be enabled. Then the MAC address is passed to the OCPP backend according to the Autocharge standard (sends "VID:<mac address>" as authorization token).

Please note that the MAC address is not reliably a static unique address and can be switched by the EV dynamically ("randomized") after each charging session for privacy reasons. Especially in this case, the usability with OCPP-based authentication is limited.

The SOC value can only be provided by the charging software when the EV starts a DIN 70121 or ISO 15118 DC charging session. The SOC is published only once at the beginning of the HLC session over MQTT topic TOPIC_EV_PARAMETER_RESSSOC. While this topic is sent out not-retained, a customer implementation cannot detect reliably when a new EV is connected and whether its SOC could be read successfully. To address this, the MQTT topic TOPIC_EV_SOC_ACTUAL was introduced which is sent out retained, see below for details.

During an on-going charging session, it is possible to try a re-init of the charging session. This is not done automatically by the charging stack, but can be triggered by customer's software via MQTT topic TOPIC_CHARGE_REINIT_SESSION. Please note, that this may not work with all EVs. If supported and successfully carried out, then the MQTT topics TOPIC_EV_PARAMETER_RESSSOC, TOPIC_EV_SOC_ACTUAL and TOPIC_EV_SOC_TIMESTAMP are updated with the latest obtained values.

In an ISO 15118 AC charging session, i.e. in case the EV only supports ISO 15118 AC or prefers ISO 15118 AC over DIN 70121, only the MAC address can be retrieved.

In case the EV is not reacting to the HLC trigger the charging software switches automatically, after a 20 seconds SLAC timeout, to a basic AC PWM charging session.

Topic Name	Topic	Type	Retain	Range	Comment
TOPIC_EV_PARAMETER_RESSSOC	"port0/ci/ev/parameter/resssoc"	integer	no	0-100	Topic which provides the EV´s state of charge (SOC) of the battery (RESS - Rechargeable Energy Storage System) in percent. Will be published at the beginning of a ISO15118 HLC charging session.

Topic Name	Topic	Type	Retain	Range	Comment
TOPIC_V2G_MAC	"port0/session/v2g_mac"	string	no	-	Topic which provides the hex encoded MAC address (uppercase) of the SLAC process. Example: "00:12:34:56:78:9A"
TOPIC_AUTHORIZATION_STATUS	"port0/session/authorization_status"	integer	yes	0-1	Topic which must be published by the customer to authorize an user to charge in case of "ports[0]/user_authentication" is set to "mqtt". Valid payloads:0 = charging not allowed,1 = charging allowed In case of a session based authorization the topic should be published as not retained. When OCPP is selected as authorization method, then the internal OCPP client takes care of this topic - customer software must not publish it in this case.

Topic Name	Topic	Type	Retain	Range	Comment
TOPIC_EV_SOC_ACTUAL	"port0/ci/ev/soc/actual"	integer	yes	-2-100	Topic which provides the EV´s state of charge (SoC) in percent, or the following special values: -1 when EV is not connected, OR when EV is connected, but HLC communication did not yet made progress to charge parameter phase and thus the values is not yet known OR the current setup does not allow the process to reach the charge parameter phase (e.g. Tesla is connected) OR only basic charging is configured (see charging_type) -2: the charge parameter phase is passed, but the value is still unknown (e.g. error...) Implementation notes: It is reasonable to pre-initialize a variable which holds the received topic value with -1 because this signals that the SoC is not yet, or still unknown, or that the charging stack is configured so that the SoC cannot be determined at all.

Topic Name	Topic	Type	Retain	Range	Comment
					The iso15118d will publish the topic with value -1 during bootup. The iso15118d will publish this topic after the charge parameter phase passed, either with a valid SoC value in percent or with -2. The iso15118d will publish a value of -1 when the EV is unplugged. The iso15118d will publish the topic with updated SoC value (or with -2) in case of a re-init request.
TOPIC_EV_SOC_TIMESTAMP	"port0/ci/ev/soc/timestamp"	string	yes	Timestamp in RFC 3339 format, e.g. "2023-06-23T06:54:55.123Z"	The timestamp when TOPIC_EV_SOC_ACTUAL was updated last. In case of a valid SoC value in TOPIC_EV_SOC_ACTUAL, this should be near enough to the actual readout time.

Topic Name	Topic	Type	Retain	Range	Comment
TOPIC_CHARGE_REINIT_SESSION	"port0/ci/charge/reinit_session"	signal	no	-	Trigger a re-init of a fake DC HLC session, i.e. the current PWM charging session is terminated and the EV is instructed to go into a HLC phase again so that a readout of the SOC is possible. After this, a switch back to a new PWM charging session is done. Please note, that not all EVs support such a switching between HLC and PWM sessions.

Table 83 MQTT Topics for fake high level DC charging mode

15.22 Best practice

1. Check if the EV is plugged and new TCP connection is established (CP State 'B' received over topic TOPIC_CHARGE_CP_STATUS and TOPIC_CHARGE_TCP_STATUS with "connected" ('1'))
2. Observe the CP States and the TCP connection status throughout the whole charging session (See chapter Charge status information)
3. Use the status topics to indicate the current charging phase (See chapter Charge status information)
4. Analyze the subscribed MQTT EV topics and handle the EVSE topics according to the standard (ISO15118/DIN70121)
   a. Initialization phase (TOPIC_CHARGE_INIT_STATUS with "started")
      - Initialize the charging parameters over the MQTT topics before reaching the corresponding phase. All parameters can be initialized after receiving of the topic TOPIC_CHARGE_INIT_STATUS with "started". Depending on the initialization process, the "INIT" parameter must be provided no later than three seconds after "started". Alternatively, it is possible to use the configuration file to initialize the charging parameters (See chapter Basic SECC configuration)
      - Check the selected protocol (TOPIC_CHARGE_INIT_PROTOCOL). The ISO 15118 charging flow supports additional mechanisms like renegotiation of charging parameter (See chapter Renegotiation process) and resume of an old session (See chapter Pausing and resuming of a charging session), and additional parameters to control the charging session.

   b. Authentication phase (TOPIC_CHARGE_AUTH_STATUS with "started")

- If the authentication phase has finished on EVSE side, the EVSE-Processing parameter of the "AUTH" phase needs to be set to "Finished" ('0').
- The authentication method depends on the provided list within the TOPIC_EVSE_INIT_PAYMENTOPTIONS message. Only "ExternalPayment" ('1') is currently supported.

c. Charge parameter phase (TOPIC_CHARGE_PARAMETER_STATUS with "started")

- If the charge parameter phase has finished on EVSE side, the EVSE-Processing parameter of the "PARAMETER" phase needs to be set to "Finished" ('0').
- If EVSE-Processing was set to "Finished" ('0') the connector has to be locked on EV-side.
- In ISO15118 at least one SA-schedule must be sent over topic TOPIC_EVSE_PARAMETER_SASCHEDULELIST. One schedule is already predefined in the configuration file, but this one should be adapted with valid data (See chapter Basic SECC configuration).

d. Charge phase (TOPIC_CHARGE_CHARGE_STATUS with "started")

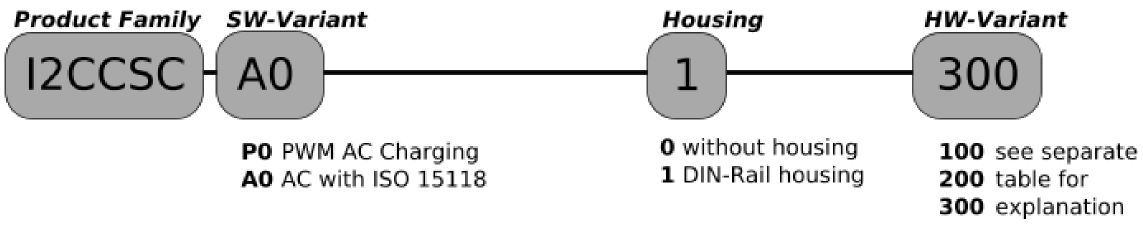
- The EVSE shall follow the requested EV target voltage and target current under the conditions of the handled maximum and minimum limits.
- The topic TOPIC_EV_CHARGE_READYTOCHARGESTATE indicates the current charge progress of the EV. If it is set to "Start" ('0') or "Stop" ('1') the EV requests to start or stop the energy flow, if is set to "Renegotiate" ('2') the EV requests the renegotiation process. (Only relevant for ISO15118. See chapter Renegotiation process)

5. Initiate a shutdown of the power electronic if the EV indicates stop charging (See chapter Stop charging and error shutdown indication on EV side) or for EVSE emergency reasons (See chapter Stop charging and error shutdown on EVSE side).
6. The charging session is terminated by receiving the topic TOPIC_CHARGE_TCP_STATUS with "disconnected" ('0') and TOPIC_CHARGE_CP_STATUS with CP State 'A'.

16 Order Information

Product Code

Product Code



Order Code

Order Code

Available order codes	SW-Variant	OCPP 1.6 included	Housing	HW-Variant
I2CCSC-P00-105	PWM AC Charging	no	no housing	100
I2CCSC-A00-204	AC with ISO15118	yes	no housing	200
I2CCSC-A00-274	AC with ISO15118	yes	no housing	200
I2CCSC-A00-303	AC with ISO15118	yes	no housing	300

17 Device Marking

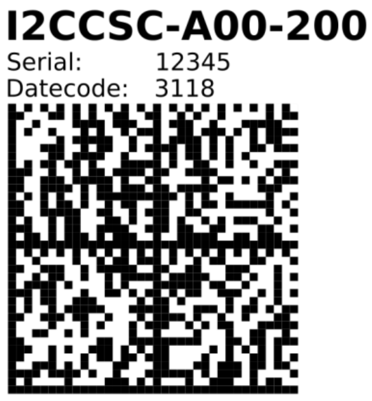
Each device is marked with a label containing the following data:

1. Order Code
2. Serial Number
3. Production Data Code: WWYY
4. 2D DataMatrix code containing the following information as a list of space separated values:
   a. Order Code
   b. MAC address Ethernet[1] (only present for variant 200 and 300)
   c. MAC address CP QCA7000[1] (only present for variant 200 and 300)
   d. MAC address CP QCA7000 Linux interface[1] (only present for variant 200 and 300)
   e. MAC address mains QCA7000[1] (only present for variant 300)
   f. MAC address mains QCA700 Linux interface[1] (only present for variant 300)
   g. DAK mains QCA7000 (only present for variant 300)
   h. Serial Number[2]
   i. Production Data Code

[1]: without colons or other delimiters

[2]: 10 digits, with leading zeros

An example is shown in figure Example Label for Charge Control C.



I2CCSC-A00-200
Serial: 12345
Datecode: 3118

Figure 23 Example Label for Charge Control C

18 Certifications

Figure 24 Compatible with the back end system be.ENERGISED

Figure 25 Works with Vector vCharM

Figure 26 Certified by Gridware

19 Contact

chargebyte GmbH

Bitterfelder Straße 1-5

04129 Leipzig

Germany

Website: https://chargebyte.com