White Beet - User Manual

Revision No.:	2.7
State:	Released
Author:	SEVENSTAX GmbH

Confidentiality:	Confidential
Initial version:	2020-09-14
Last change:	2021-10-08

Table of Contents

Tables

List of Figures

1 Abstract

This manual describes the ‘WHITE beet – SLAC/Bridging’ and the ‘WHITE beet – ISO15118’-Module and gives an overview how to use it. It starts with a brief overview of product definition and then goes to the hardware description. The manual also describes the software configuration, the format of the used commands and the firmware update.

Especially it gives an overview about:

- Hardware connections and switches
- Supported functions
- Supported commands including control protocol

1.1 Abbreviation and Glossary

Table 1: Abbreviations

Abbreviation	***Description***
CP	Control Pilot
CA	Certification Authority
EV	Electro Vehicle
EVSE	Electric Vehicle Supply Equipment
EVCC	Electric Vehicle Communication Controller
FWU	Firmware Update
SECC	Supply Equipment Communication Controller
HCI	Host Controller Interface
HLE	Higher Layers Entities
HPGP	HomePlug GreenPHY
MAC	Media Access Control
PE	Protective Earth
PIB	Parameter Information Block
PLC	Powerline Communication
PP	Proximity Pilot
SLAC	Signal Level Attenuation Characterization
SDP	SECC Discovery Protocol
V2GTP	Vehicle to Grid Transport Protocol

1.2 Referenced documents

#	Document	Author	Rev.
#1	ISO 15118-3	International Electrotechnical Commission	First edition 2015-05-15
#2	IEC 61851-1	International Electrotechnical Commission	2019-12
#3	WHITE_beet_E_datasheet_rev.1.00_20201127.pdf	CODICO	Ver.1.00
#4	WHITE_beet_P_datasheet_rev.1.00_20201127.pdf	CODICO	Ver.1.00

2 Product Description

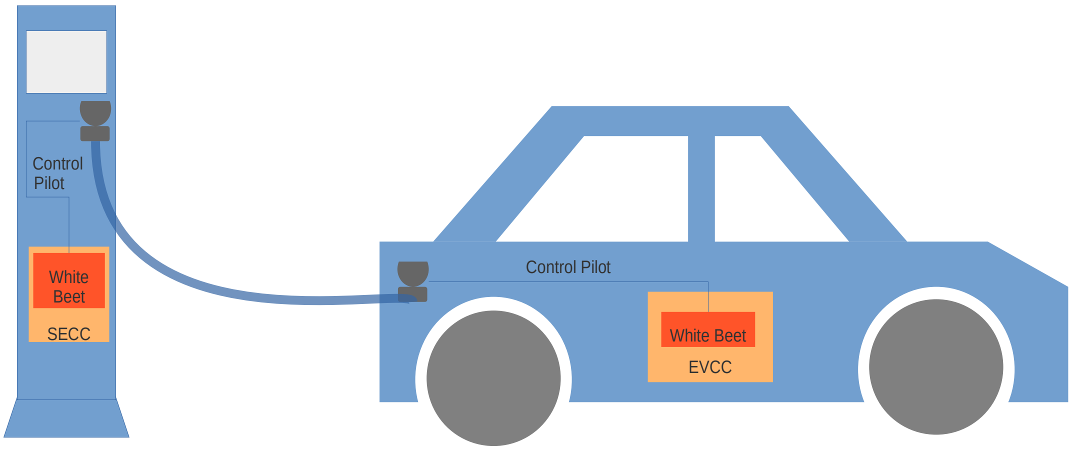


Figure 1: EVSE and EV

For the White beet module there are two software variants ‘SLAC/Bridging’ and ‘ISO15118’, which are identical in their basic functionality. However, variant ‘ISO15118’ additionally contains the functionality that is required for V2G communication according to the ISO/DIN. For both variants, there exist then an EV and an EVSE variant (for ISO15118 currently only EVSE is available).

Available White beet Modules:

- WHITE-BEET-ES (EVSE) – SLAC/Bridging Module
- WHITE-BEET-PS (PEV) – SLAC/Bridging Module
- WHITE-BEET-EI (EVSE) – Embedded ISO15118 Module

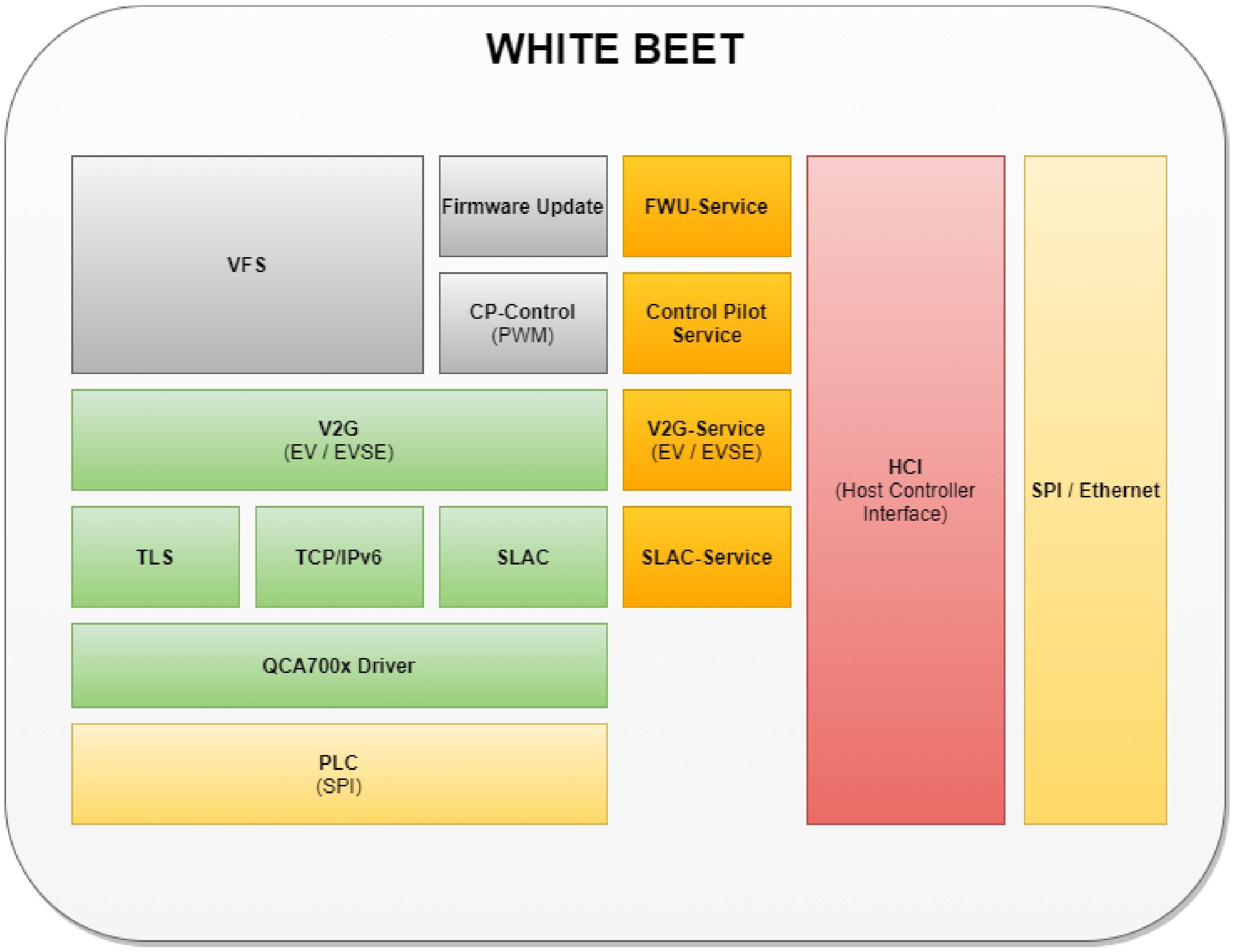


Figure 2: White beet

2.1 SLAC/Bridging - Module

White beet SLAC/Bridging Module was developed to easily equip electric vehicles and their charging stations with HPGP communication based on the ISO/IEC 15118 charging communication standard.

The module takes over the time-critical SLAC negotiation and the layer2-bridging between the Host Controller Interface and the PLC, so that the host controller take care of the overlying protocols only. After successful SLAC negotiation all incoming MAC frames will be forwarded to the PLC interface and vice versa.

The module additionally comes with support for controlling the basic signalling following IEC 61851-1 (#2).

2.1.1 Features

- Bridging Mode between Host Controller Interface and PLC
- Integrated SLAC for EV and EVSE side
- Integrated Control Pilot Interface (PWM Generation and State detection)
- API for SLAC configuration and control (start/stop)
- API for Control Pilot control (Host Controller Interface API)

2.2 ISO15118 - Module

In addition to the features from variant ‘SLAC/Bridging’, the variant ‘ISO15118’ also includes support for High Level Communication based on the ISO/IEC 15118 (#1) charging communication standard.

The module takes over the V2G communication including finding the remote station using SDP and the time-critical SLAC negotiation. Furthermore, the module offers the possibility to enable basic communication according to IEC on the Control Pilot.

An application that runs on a host controller only has to take care of the essential things. For this purpose the module provides a host control API.

A simplified charging sequence can be found in chapter 2.8

2.2.1 Features

- Integrated V2G Stack (EIM) for EVSE side (including SECC Discovery Protocol)
- Integrated SLAC for EV and EVSE side
- Integrated Control Pilot Interface (PWM Generation and State detection)
- API for V2G configuration and control (Host Controller Interface API)
- API for SLAC configuration and control (Host Controller Interface API)
- API for Control Pilot control (Host Controller Interface API)

2.3 Host Controller Interface (HCI)

The WHITE beet module provides an interface for configuration and control which is called ‘Host Controller Interface’ (HCI). All supported functions, which are marked as services and modules, can be used via the HCI.

Below you will find a list of them:

Table 2: HCI Services and Modules

#	Service Module	Comment	Chapter
1	System Module	System configuration and status	11
2	Network Configuration Module	Control and status of network configuration	12
3	Control Pilot Service	Used for controlling CP (PWM and states)	14
4	SLAC-Service	Used for executing SLAC matching process	15
5	Vehicle to Grid Service	Used for V2G communication (only ISO15118 Module)	17
6	GPIO	GPIO control and status	18
7	Firmware Update	Firmware Update Modul	13

To use one of the above services, the HCI API can be used via one of the available interfaces (2.4).

The HCI API commands have to be transmitted in the format as described in chapter 9. The exact description of the individual commands can be found in the chapters of the individual services (please have a look to table 2 to find the corresponding chapters). There you can also find various examples.

2.4 Interfaces

The WHITE beet module offers several interfaces for host controller communication. Information about the use of the available interfaces is described in Chapter 9.

Table 3: Interfaces

#	Interface	Chapter
1	Ethernet	9.1
2	SPI	9.2

2.5 Filesytem Ressources

This section gives an overview of the file system with all relevant files for the user. These files can be modified or exchanged only by a signed firmware update. For this purpose a FWU file must be created as described in chapter 7.

Table 4: Relevant files inclusive paths (needed for Firmware Update generation)

Configuration	File path	Max. Size
STM32 MAC-Address	fs/dev/mac.bin	4 KB
QCA7005 Firmware (OLD)	fs/fw/qca700x/fw1/firmware.bin	512 KB
QCA7005 Firmware (NEW)	fs/fw/qca700x/fw2/firmware.bin	512 KB
QCA7005 EV Configuration (for Firmware OLD)	fs/config/qca/fw1/ev.pib	16 KB
QCA7005 EVSE Configuration (for Firmware OLD)	fs/config/qca/fw1/evse.pib	16 KB
QCA7005 EV Configuration (for Firmware NEW)	fs/config/qca/fw2/ev.pib	16 KB
QCA7005 EVSE Configuration (for Firmware NEW)	fs/config/qca/fw2/evse.pib	16 KB
Customer CA Certificate	fs/cert/fwu/appl/config.crt	16 KB
Factory Configuration	fs/setup/FactoryCfg.bin	16 KB
Device Configuration	fs/setup/DeviceCfg.bin	16 KB
User Configuration	fs/setup/UserCfg.bin	16 KB

2.6 GPIOs

The module also offers GPIOs, which can be controlled via the HCI and the status can also be queried. Below is a table that describes the mapping of the GPIOs to the pins:

Table 5: Available GPIOs for HCI-Interface

WHITE beet Pin	WB-CARRIER-Pin	HCI GPIO Service Pin Number
20	J4.1	20
19	J4.3	21
73	J4.4	22
74	J4.6	23
17	J4.7	24
79	J4.8	25
16	J4.9	26
14	J4.13	27

The commands for the HCI can be found in Chapter 18.

2.7 Preconditions

To use the WHITE beet module (SLAC/Bridging / ISO15118) you need either your own hardware on which the module is mounted or the evaluation board ‘WHITE beet carrier board’ (3) which already contains one of the WHITE beet modules.

2.8 Simplified Charging Sequence

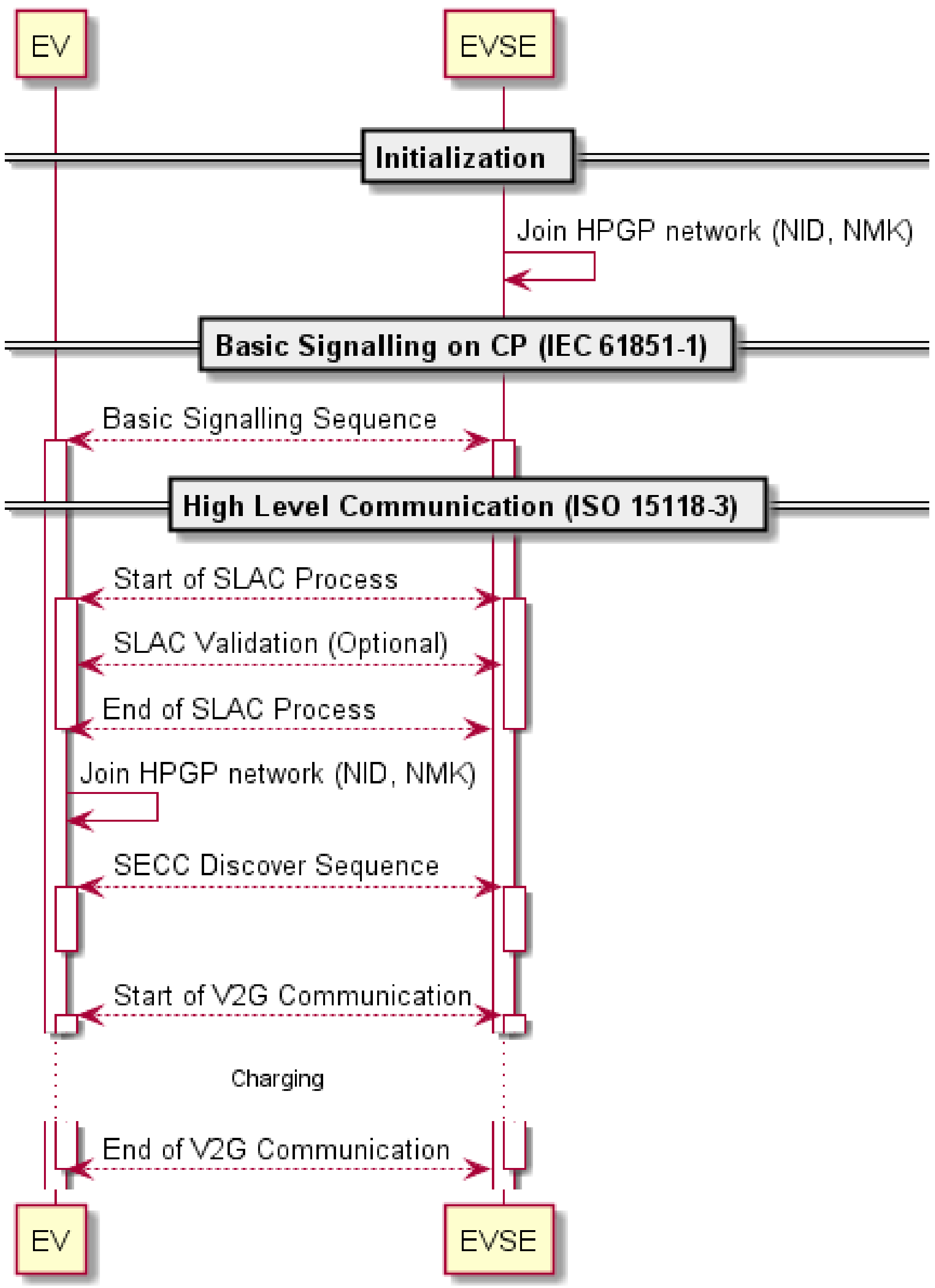

EV
EVSE
Initialization
Join HPGP network (NID, NMK)
Basic Signalling on CP (IEC 61851-1)
Basic Signalling Sequence
High Level Communication (ISO 15118-3)
Start of SLAC Process
SLAC Validation (Optional)
End of SLAC Process
Join HPGP network (NID, NMK)
SECC Discover Sequence
Start of V2G Communication
Charging
End of V2G Communication
EV
EVSE

Figure 3: Simplified ISO15118 process (with IEC61851)

2.8.1 Initialization

Both sides start with the initializing phase which have to be completed before EV and EVSE are connected. The EVSE will create a HPGP network which will be later used for high level communication (after SLAC).

More detailed information about the process can be found in chapter 19.3 for EVSE side and chapter 19.4 for EV side. Information about the structure of the used HCI-Commands in Chapter 15.

2.8.2 Basic Signalling

After the initialization sequence has been completed and the EV was connected to the EVSE the basic signalling process starts on both sides. The EVSE will start PWM generation and the EV uses the resistors to change the state depending on the internal state.

More detailed information about the process can be found in chapter 19.2 for EVSE side. Information about the structure of the used HCI-Commands is in chapter 14.

2.8.3 SLAC

If the result from basic signalling is that both sides wants to use high level communication then the EV starts sending SLAC messages to proceed the full SLAC matching sequence according to ISO15118-3. After successful matching the EV will join to the HPGP network which was received in the SLAC matching response from EVSE.

More detailed information about the process can be found in chapter 19.3 for EVSE side and chapter 19.4 for EV side. Information about the structure of the used HCI-Commands is in Chapter 15.

2.8.4 SDP

In the next step when both sides are in the same network and a link was detected on EV side, the vehicle will send a SDP request to get the IP and port from the EVSE.

Further information can be found in chapter 19.8.

More detailed information about the process can be found in chapter 19.8 for EVSE side. Information about the structure of the used HCI-Commands is in Chapter 17.

2.8.5 V2G

If getting of IP and port from EVSE was successful the EV will start with the V2G communication and begin charging if there was no error detected on both sides.

More detailed information about the process can be found in chapter 19.8 for EVSE side. Information about the structure of the used HCI-Commands in Chapter 17.

3 Evaluation Board (WB-CARRIER-BOARD)

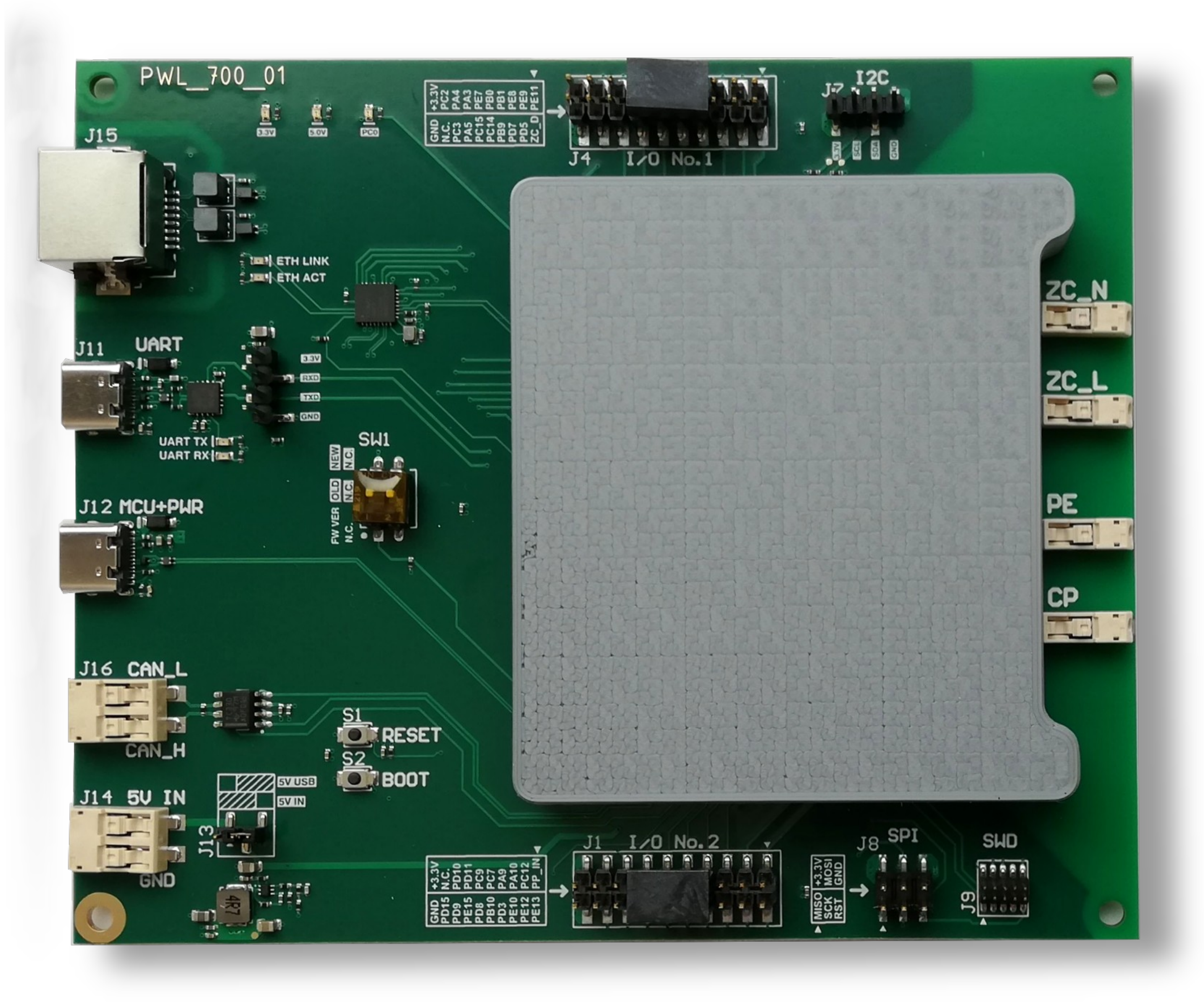


Figure 4: WB-CARRIER-BOARD

The WB-CARRIER-BOARD is an evaluation and development board. It contains the WHITE beet module with an STM32F7 microcontroller, the firmware (SLAC/Bridging or ISO15118) and one jumper for configuration purpose. Depending on the version (PEV, EVSE or 'Home Control'), there are minor differences in the assembly. Furthermore the board is still available with different software (SLAC/Brdging and ISO15118).

Available CARRIER-BOARDs:

- WB-CARRIER-BOARD-ES (EVSE) – SLAC/Bridging
- WB-CARRIER-BOARD-PS (PEV) – SLAC/Bridging
- WB-CARRIER-BOARD-EI (EVSE) – Embedded ISO15118

The board can be powered either by using a USB-C cable (on **J12** - USB Port 'MCU-PWR)') or the connector **J14** (5V). To use power supply via USB, jumper **JP13** must be set to '5 V USB'.

More detailed information on the boards can be found in the corresponding data sheets (#3, #4). There you will find a description of the pins, connectors, switches and buttons.

4 Start-up Guide (WB-CARRIER-BOARD)

Commissioning steps:

- Connect the WB-CARRIER-BOARD to a Host Controller (e.g. PC) using one of the available interfaces (Table 3).
- Configure the WB-CARRIER-BOARD by using the Jumpers (please have a look to the WHITE beet documentation).
- Connect the WB-CARRIER-BOARD to a communication remote station via connectors **CP** and **PE**.
- Power-up the board (using **J12** or **J14**)

Expected behavior after switching on:

- LED **PC0** starts flashing every second.
- WHITE-BEET Module sends debug information on the UART (**J10** and **J12** – 115200/8N1)

After the board has been connected and put into operation, configuration and control can be started. Information on this topic can be found in chapters 5 and 9.

5 WHITE beet - Module Configuration

In order to use the module outside of the laboratory, it must be configured first. For this purpose, depending on the configuration, different files must be produced and/or configured. In the next chapters you can find information how the configuration files can be created, changed and which file format is expected.

Later these files can be uploaded to the module using the secure firmware update tool. For this purpose, the corresponding files must be packed into the firmware package using the StxFwGen-Tool. The output of the StxFwGen-Tool (FWU-File) can then be uploaded to the module using the FW-Upload-Tool.

Details on how a corresponding firmware is generated can be found in Chapter 7.1.

Details to the firmware upload is described in chapter 7.2.

The WHITE beet module needs the following configuration options:

Table 6: Configuration parameters for the firmware update

Configuration	***Configuration is mandatory***	***Description***
STM32 Configuration	no	Binary file that contains the WHITE beet module configuration (e.g. System parameters). The format is described in chapter 5.1.
STM32 MAC-Address	yes	Binary file that contains the MAC addresses (the format is described in chapter 5.2)
QCA7005 Configuration	yes	PIB File that contains the MAC address and the SLAC configuration, among other things More information about this in chapter 5.3.
Customer CA Certificate	yes	CA Certificate which is used to check the firmware update More information about this in chapter 5.4.

5.1 Configuration via StxCfgGen-Tool

The default configuration of the board can be set with the help of the StxCfgGen-Tool. The default configuration is the configuration that is established after a factory reset. But note that the factory reset does not change the configuration of the PLC chip (PIB file), only that of the STM32.

The following settings can be configured with the help of the tool:

Table 7: Configurable WHITE beet Module Parameters

#	Modul	Parameter	Description	Default-Value
1	SYSTEM	save_mode	Save Mode: 0: Manual 1: Auto	auto
2	SYSTEM	id_manufacturer	Manufacturer ID - Optional parameter which is used by the Firmware Update.	Sevenstax GmbH
3	SYSTEM	id_product	Product ID - Optional parameter which is used by the Firmware Update.	Empty

An individual configuration file can be created using the StxCfgGen tool. This configuration file contains all the configurable parameters and will be loaded on startup. The device will use these settings until parameters were changed during runtime. If parameters were changed during runtime and the device is using the automatic save option, all changed parameters are saved in the user configuration file. The user configuration file will overload the parameters from the configuration file at startup. A reset to factory default values will restore the configuration.

The following figure shows a configuration file template:

Table 8: Configuration file template for WHITE beet module

```
 User instructions
# =================
# - Sections should not be missing, even if no options indicated.
# - The sequence of the individual sections is not important.
# - All or some options of any single section may be missing. Commenting options out is sufficient.
# - The sequence of the individual options of a section is not important.

[SYSTEM]
# save_mode; 0 = manual; 1 = automatic
save_mode = 0
id_manufacturer = CUSTOMER_MANUFACTURER
id_product = CUSTOMER_PRODUCT
```

5.1.1 Build own Configuration file

The StxCfgGen-Tool exists in two version. On the one hand as EXE file for Windows and on the other hand there is a Python variant, which ca be used on all systems where Python is available and the requirements of chapter 20.1 are met. This section provides information how to use it.

'StxCfgGen' supports the following parameters:

StxCfgGen.exe [-h] -i CONFIGFILE

Table 9: Supported StxCfgGen tool parameters

Parameter	***Description***
-h	Help
-i	Input file

<u>Example for generating a White Beet configuration:</u>

StxCfgGen.exe -i customer.cfg

The following three steps are necessary to update the module configuration:

- create individual configuration file with all relevant settings.
- Generate configuration file
- Upload the configuration file via FWU file to the WHITE beet module.

For further information please have a look to chapter 7.1.1 and 7.2.

5.2 BIN-File for MAC address configuration

In order to configure the used MAC addresses from the Whitebeet module, a BIN file must be generated first which contains the MAC addresses to be used from the STM32 microcontroller.

Note: Please note that the device only provides MAC addresses for testing and commissioning upon delivery. The MAC addresses must be replaced by your own for later operation. By default, any Ethernet device needs a globally unique MAC address in the Ethernet. An address pool can be ordered at IEEE under „http://standards.ieee.org/regauth/oui/pilot-ind.html".

The format of the BIN file must look like this:

Table 10: BIN-File Format for MAC address configuration

0	**1**	**2**
Number of MAC addresses	1. MAC Address ETH0	2. MAC Address ETH1

Table 11: BIN-File Format for MAC address configuration (Bytes)

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**

NumMac	MAC ETH0 [0]	MAC ETH0 [1]	MAC ETH0 [2]	MAC ETH0 [3]	MAC ETH0 [4]	MAC ETH0 [5]	MAC ETH1 [0]
MAC ETH1 [1]	MAC ETH1 [2]	MAC ETH1 [3]	MAC ETH1 [4]	MAC ETH1 [5]			

Table 12: Parameter description of MAC address configuration file.

Parameter	Number of bytes	Description
NumMac	1	Number of MAC addresses in file available. Note: Maximum of two MAC addresses supported!
MAC ETH0	6	MAC address of STM32 SPI-Interface for QCA7005 communication.
MAC ETH1	6	MAC address of STM32 ETH-Interface.

The following three steps are necessary to use your own addresses:

- include MAC binary file in the firmware update configuration
- create a firmware update file (FWU-File)
- Upload the FWU file to the Whitebeet module.

For further information please have a look to chapter 7.1.1 and 7.2.

5.3 Creating or changing PIB file

The configuration of the QCA7005 chip is done with the help of PIB files. These contain various parameters such as the MAC address, Network ID&Key, SLAC-Configuration and many other parameters.

So a PIB file should be adapted in any case, because on the one hand an own MAC address must be used and on the other hand this file also has an influence on the electrical properties.

With the help of the Open-PLC-Utils, a PIB file can be downloaded and changed. It is possible to change the MAC address, the SLAC configuration and other parameters.

Further information about usage and the software can be found under the following link:

https://github.com/qca/open-plc-utils

Note: The WHITE beet module comes with a preprogrammed PIB file!

To use an own PIB file the following three steps are necessary:

- include PIB file in the firmware update configuration
- create a firmware update file (FWU-File)
- Upload the FWU file to the WHITE beet module.

For further information please have a look to chapter 7.1.1 and 7.2.

5.4 CA Certificate Configuration

The WHITE beet module can be configured with the help of firmware update files. For this it is necessary to generate own signed firmwares by using a signing certificate.

The supplied WHITE beet module already contains a start CA certificate (A start signing certificate is supplied), but this should be replaced by your own certificate to avoid that the configuration is changed by anyone. Therefore you have to create own certificates from your PKI (one CA certificate and at least one signing certificate).

For creating a firmware update a signing certificate is required, which is checked during the firmware upload by the WHITE beet module using the configured CA certificate. The firmware is only accepted if the issuer of the signing certificate matches the configured CA certificate.

More information about firmware generation is described in chapter 7.1.

Please note that the certificates must be created in the DER format.

Note: The certificates required for the firmware update must be created by a PKI. If you don´t have a PKI yet and you need help with it, please feel free to contact us.

5.5 Configuration of Attenuation Values

In order to be able to execute the SLAC process correctly, a valid configuration of the transmission path is necessary. The transmission path must be configured once for each device, regardless of whether it is EV or EVSE. Various things play a role here, such as the cable used to connect the vehicle and the charging station.

In order to be able to set the attenuation values **AttnRx** and **AttnTx** correctly, a measurement is necessary. The determined values (for all 58 groups of frequencies) can then be set using the HCI commands (chapters 15.4.4, 15.4.5, 15.4.6 and 15.4.7) .

More detailed information about the measurement and the attenuation values can be found in the document #1 (ISO15118-3). Information can be found there in chapters A11.4.

6 Safety / Security Notes

1. Due to the bridging operation (ETH←→ PLC), all messages that are not directed to the WHITE beet module itself are sent out via the other interface. Therefore, actions may have to be taken on the host side to prevent access from the PLC network (e.g. firewall).
2. After system startup, the PLC network is joined from the PIB file (EV) or created (EVSE). For security reasons, it is advised to use a random network so that no access to the network is possible. This is also necessary after disconnecting the connection (CP).
3. To be protected against unwanted updates, it is recommended to use your own certificate and to adjust the configuration of the device. For more info see chapter 5.
4. WHITE Beet module was not designed for safety relevant applications. The user needs to take appropriate measures to avoid critical operation conditions due to WHITE Beet’s unexpected behaviour.

7 Firmware Update

With the help of the secure firmware update there are two possibilities:

1. Updating Firmware (STM32, QCA7005)
2. Module configuration (PIB-File, MAC-Address, …)

Please refer to the following chapters for further information or have a look to the examples in chapter 20.2 and 20.3!

7.0.1 Updating Firmware

The firmware of the QCA7005 and the STM32 can only be updated using a signed firmware update file provided by SEVENSTAX.

Details about the firmware upload procedure can be found in chapter 7.2.

7.0.2 Updating Configuration

The used configuration of the WHITE beet – module can be changed as described in chapter 5. This configuration can be used to generate a specific firmware update file. The firmware can be transferred to the module with the help of the secure firmware update tool.

Details on creating firmware update files can be found in chapter 7.1.

Details to the firmware upload can be found in chapter 7.2.

7.1 Generate Firmware Update (StxFwGen)

The following steps are necessary to generate a firmware update which can be written to the module using the firmware update tool:

1. Create configuration file for FW-Update. Please make sure to specify the exact paths as destination, which are given in the table (see chapter 2.5 - table 4).
2. Make sure that all source files are available in the specified location.
3. Note that the correct certificate (signing certificate for configured CA certificate) must be specified under Certification, which has to be used by the tool for signing the firmware. Please also note the information in Chapter 5.4.
4. Run the tool with the appropriate parameters. More information about the parameters you can find in Chapter 7.1.3.

Note: The parameters ‘maximum_containersize = 2000’ and ‘container_format_version = 1’ must not be changed.

7.1.1 Create Firmware Update Configuration file

In order to create a firmware update image, it must first be determined which files should be included and with which certificate the firmware should be signed. Please note that the signing certificate must be issued by the configured certificate in the device. Upon delivery, the device contains a standard certificate, which should be replaced. Until it has been replaced by an own certificate, the signing certificate included in the delivery must be used. Further information regarding the certificates can be found in the chapter 5.4.

The following figure shows a template for a firmware update configuration:

```
base_file = None
maximum_containersize = 2000
container_format_version = 1

modules = [
            [
                "INFO",
                [
                    [0x10, "CMP", "SEVENSTAX GmbH"],
                ]
            ],
            [
                "CERTIFICATION",
                {
                    "certificate": "SigningCert.crt.der",
                    "signature_scheme": "rsa-pkcs1"
                }
            ],
            [
                "FILE",
                {
                    "source" : "FileA.bin",
                    "destination" : "DestFilePathA.bin"
                }
            ],
            [
                "FILE",
                {
                    "source" : "FileB.bin",
                    "destination" : "DestFilePathB.bin"
                }
            ]
        ]
```

Figure 5: Firmware Update configuration template

Structure of the template:

Element	*Description*	*Info*
base_file	here an existing FWU file can be included in the own FWU file.	For creating own updates with WHITE beet firmware.
maximum_containersize	Firmware generator tool parameter which is used to define the maximum container size in FWU file.	Must not be modified.
container_format_version	Firmware generator tool parameter for internal container format.	Must not be modified.
modules	Section to specify the containers to be included in the FWU file.	-
INFO	Section to configure constraints as well as set version numbers	-
CERTIFICATION	Set parameters to certificate which should be used and included for the generation of the firmware update file.	-
FILE	Add sections to include files to the firmware update file.	To see which files can be exchanged with the help of the firmware

		update, please refer to chapter 2.5.

7.1.2 Firmware Update Configuration example for setting the MAC address

This section contains a sample configuration for generating a firmware file that configures the MAC addresses. In this example, also the default certificate for signing is given.

```
base_file = None
maximum_containersize = 2000
container_format_version = 1

modules = [
            [
                "INFO",
                [
                    [0x10, "CMP", "SEVENSTAX GmbH"],
                ]

            ],
            [
                "CERTIFICATION",
                {
                    "certificate": "DemoSignWhiteBeetPKI.crt.der",
                    "signature_scheme": "rsa-pkcs1"
                }
            ],
            [
                "FILE",
                {
                    "source" : "MacAddress.bin",
                    "destination" : "fs/dev/mac.bin"
                }
            ]
        ]
```

Figure 6: Example configuration for firmware udpate generation tool

7.1.3 Use StxFwGen for Firmware Update generation

The Firmware Generation Tool is used to create firmware update files and exists in two version. On the one hand as EXE file for Windows and on the other hand there is a Python variant, which ca be used on all systems where Python is available and the requirements of chapter 20.1 are met. This section provides information how to use it.

Note: Please note that the key file is deleted by the tool after the key is encrypted and placed in the execution directory. For the encryption, a separate password must be set, which is then always queried by the tool.

This tool supports the following parameters:

StxFwGen.exe [-h] -c CONFIGFILE [-o OUTFILE] [-k KEYFILE] [-u] [-t] [-l PKCS11LIB]

Table 13: Supported firmware generation tool parameters

Parameter	Description
-h	Help
-c	Firmware generation configuration file
-k	Key file from signing certificate (matching the certificate that is specified in the configuration file)

Parameter	Description
-u	Skip private key initialization and checks, no signed FWU possible (default: False)
-o	Output file (default: firmware.fwu).
-t	Sign the firmware using a PKCS#11 compatible crypto token (default: False)
-l	PKCS#11 interface implementation for crypto token in use (default: opensc-pkcs11.dll)

Example for FWU file generation:

```
StxFwGen.exe -k DemoSignWhiteBeetPKI.key -c fwuconfig_macaddress.py -o  mac_address.fwu
```

7.2 Uploading firmware via StxFwUpdater

The Firmware Upload Tool ‘StxFwUpdater’ is used to upload a firmware file onto the White beet module via Ethernet Frames.

The tool exists in two version. On the one hand as EXE file for Windows and on the other hand there is a Python variant, which ca be used on all systems where Python is available and the requirements of chapter 20.1 are met.

The StxFwUpdater supports the following parameters:

Table 14: Supported firmware update tool parameters

Parameter	Description
-h	Help
-f	FWU file for upload
-t	Target MAC address
-i	Interface

Example for uploading the firmware with ‘MAC address’-configuration:

```
STxFwUpdater.exe -f mac_address.fwu -t 00:01:01:63:77:33 -i eth0
```

8 Reference Application

In order to make commissioning as easy as possible, a Reference Application is currently being developed in Python. This Reference Application will be able to configure the WHITE beet module and then go through all the necessary steps (IEC, SLAC, SDP and V2G) in order to run through a simulated charge process.

Please note that the application is not finished yet, but will be available on Github soon. Please contact your local distributor for more information.

9 WHITE beet – Module Interfaces

Various communication interfaces are available on the WHITE beet module which can be used to retrieve information from the module or to control the module. Depending on the interface, some special features must be taken into account in order to send the commands to the module or receive them from the module in the Framing Protocol. The available commands are summarized in chapter 10 and divided into several subchapters according to the different services (e.g. Vehicle to Grid Service).

Information about the supported interfaces and their special properties can be found in the following subchapters (9.1 and 9.2).

The configuration for selecting the used Host Controller Interface is described in chapter 9.0.1.

9.0.1 Selection of the host controller interface (HCI)

In order to define the host controller interface (HCI) to be used for the communication, the both IF_SELECT_x pins must be configured according to table 15. In the current version of WHITE beet firmware SPI and Ethernet are available as Host Controller Interface.

The following table contains information about the interface configuration:

Table 15: Host Controller Interface selection

Pin	***Direction***	***WHITE beet pin***	***Description***
IF_SELECT_0	IN	PAD 85 (PC2)	HCI interface selection pin: 00: Ethernet 01: SPI 10: Reserved 11: Reserved Note: Must be configured before start-up!
IF_SELECT_1	IN	PAD 84 (PA4)	

Note: Only one interface is available at a time and never both. It can also not be changed at runtime. Changes are only applied after the restart!

9.1 Ethernet interface

For communication over the Ethernet interface, ordinary MAC frames (IEEE 802.3) are used, which are extended by means of the 'Control Frame Header' to transport the Framing Protocol over the Ethernet.

The structure of the headers is shown below.

0	1	2	3	4	5	6	7
Ethernet Protocol Header							
Ethernet Protocol Header						Control Frame Header	
Control Frame Header		Framing Protocol					
Framing Protocol							

Figure 7: Ethernet Frame Format

For using the HCI-API over Ethernet the frame must consist of the following three parts:

1. Ethernet Protocol Header
2. Control Frame Header
3. Framing Protocol (including the HCI-API commands)

9.1.1 Ethernet Header

0	1	2	3	4	5	6	7
Destination Address							
Source Address				Ethernet Type			

Figure 8: MAC Frame Format

The Ethernet header according to IEEE 802.3 consists of three parts, which are described in more detail in the following table.

Table 16: MAC Frame Header

Parameter	***Size (in Bytes)***	***Value***	***Description***
Destination Address	6	xx:xx:xx:xx:xx:xx	MAC address of the desired WHITE beet module.
Source Address	6	yy:yy:yy:yy:yy:yy	Own MAC address
Ethernet Type	2	0x6003	Fix Value (Control Frame Header)

The Ethernet header is followed by the *Control Frame Header*. The format of this header is described in chapter 9.1.2.

9.1.2 Control Header

For the transmission of the framing protocol another header is needed, which has the following structure.

0	1	2	3
Version	Message Type	Size	

Figure 9: Control Frame Format

Table 17: Control Header

Parameter	***Size (in Bytes)***	***Value***	***Description***
Version	1	0x00	Version number (fix value).
Message Type	1	0x04	Message Type for Framing Protocol.
Size	2	xx yy	Size of Payload.
Framing payload	8...n	xx yy zz	Framing Protocol data.

The format of the Framing Protocol (payload) is described in chapter 10.

9.2 SPI Interface

The WHITE beet module offers a SPI Slave interface which can also be used to control the WHITE beet module functionality. Therefore the Framing Protocol is transferred over the SPI Communication Protocol as described below.

9.2.1 Requirements

The SPI Slave interface can be used by a SPI master which fulfils the following requirements:

- SPI interface (MISO / MOSI / CLK / CS)
- two additional pins to signal the module status (RX_READY, TX_PENDING)
- SPI Mode 0 (CPOL = 0, CPHA = 0)
- Maximum SPI clock frequency of 8 MHz
- Maximum SPI Transfer Size 1504 bytes (=> Payloadsize 1500 bytes)

9.2.2 Pins

The host controller requires the following pins for communication with the module:

Pin	*Direction*	*Connect to WHITE beet*	*Description*
SPI_MISO	IN	PAD 35 (PB14)	Master In, Slave Out
SPI_MOSI	OUT	PAD 36 (PB15)	Master Out, Slave In
SPI_CLK	OUT	PAD 24 (PD3)	Clock
SPI_NSS	OUT	PAD 77 (PB9)	Neagtive Slave Select
SPI_RX_READY	IN	PAD 37 (PD4)	Slave is ready pin
SPI_TX_PENDING	IN	PAD 38 (PD11)	Slave Transfer is pending

9.2.2.1 SPI_RX_READY Pin

The 'SPI-RX-READY' pin is used by the WHITE beet module to signal the SPI master that the module is ready for an SPI transfer. The SPI master must take this pin into account and must not start an SPI transfer when this pin is low.

9.2.2.2 SPI_TX_PENDING Pin

If the SPI_TX_PENDING pin goes high, then the master should start an SPI transfer as soon as possible to fetch the data to be sent from the SPI slave.

A description of how data is fetched from the SPI slave is described in more detail in chapter 9.2.3.

9.2.3 SPI Communication Protocol

For the transmission of the Framing Protocol, by means of SPI, the communication must take place according to the following procedure.

1. In the first SPI Transfer, a **Size-Exchange-Frame** is sent to inform the other side of the number of data to be sent..
2. In the second SPI Transfer, the Framing Protocol data is transferred as payload from the **Data-Exchange-Frame**.
3. After transferring the Framing Protocol data, it starts again from Step 1.

Note: If the WHITE Beet module SPI communication gets out of sync the WHITE Beet module needs to be reset!

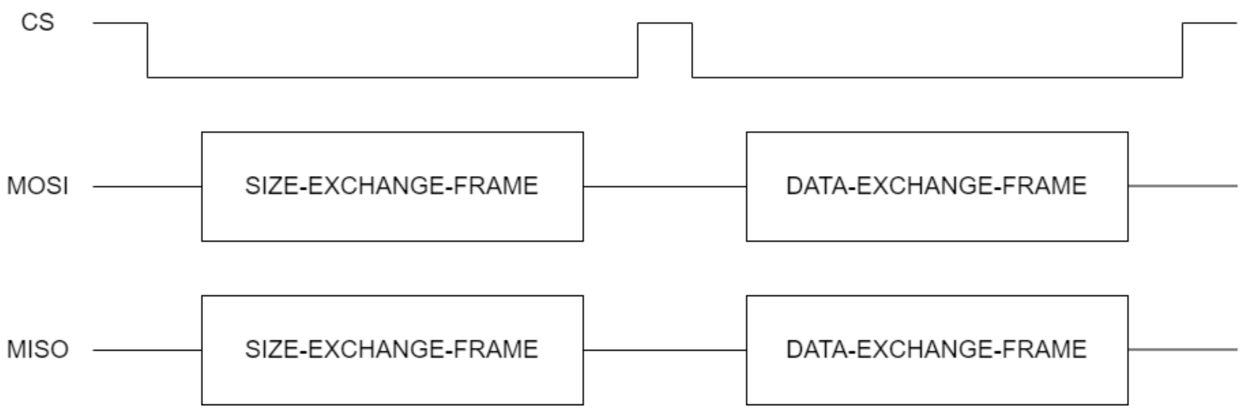


Figure 10: SPI Communication Protocol

9.2.4 Format Size-Exchange-Frame

0	1	2	3	4	5	6	7
Identifier (0xAA|0xAA)		**Size**					

Figure 11: Format Size-Exchange-Frame

The frame begins with an identifier of two bytes (0xAA, 0xAA) and is followed by the size of the payload (two bytes in network byte order) to be transmitted in the next Data Exchange frame.

After the SPI transfer the SPI-Master must determine how long the next SPI transfer will be. For this the master reads the size from the Size-Exchange-Frame of the SPI-Slave and stores the larger value as size for the next SPI transfer (Data-Exchange-Frame). This ensures that the SPI-Slave can also send its data.

Note: The size is transferred in network byte order!

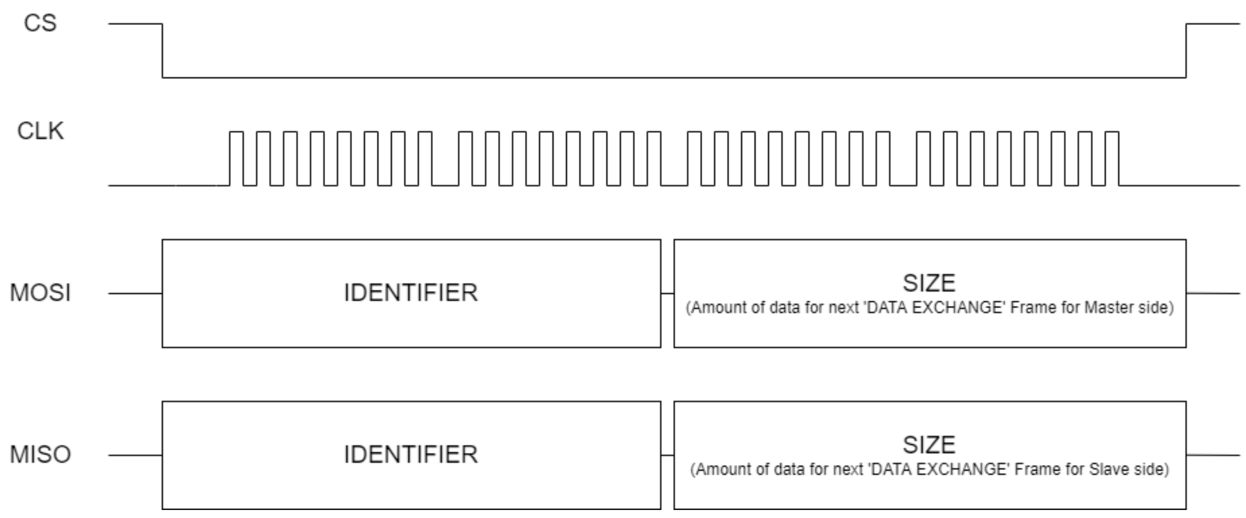

CS
CLK
MOSI
IDENTIFIER
SIZE
(Amount of data for next 'DATA EXCHANGE' Frame for Master side)
MISO
IDENTIFIER
SIZE
(Amount of data for next 'DATA EXCHANGE' Frame for Slave side)

Figure 12: Size-Exchange-Frame Transfer

9.2.5 Format Data-Exchange-Frame

0	1	2	3	4	5	6	7
Identifier (0x55|0x55|0x000x00)				**Framing Protocol Data**			
Framing Protocol Data							

Figure 13: Format Data-Exchange-Frame

This frame starts with an identifier of four bytes (0x55, 0x55, 0x00, 0x00) and is followed by the number of previously negotiated bytes of Framing Protocol Data.

Note: The SPI-Master must use the length from Size-Exchange-Frame as SPI transfer size, otherwise the SPI communication will get out of sync!

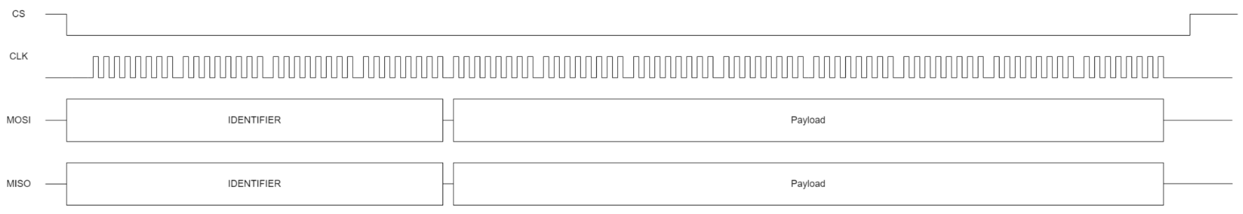


Figure 14: Data-Exchange-Frame Transfer

9.2.6 Timings

For a successful communication with the SPI Slave, the SPI Master must observe the following timings:

Parameter	***Description***	***Min.***	***Max.***
SPI Baudrate	Maximum SPI Baudrate	-	12 MHz
t_{Bit}	Bit Time	83,33 ns	-
$t_{DelayChipSelect}$	Delay after Chip select. Before beginning of data transmission)	20 µs	-

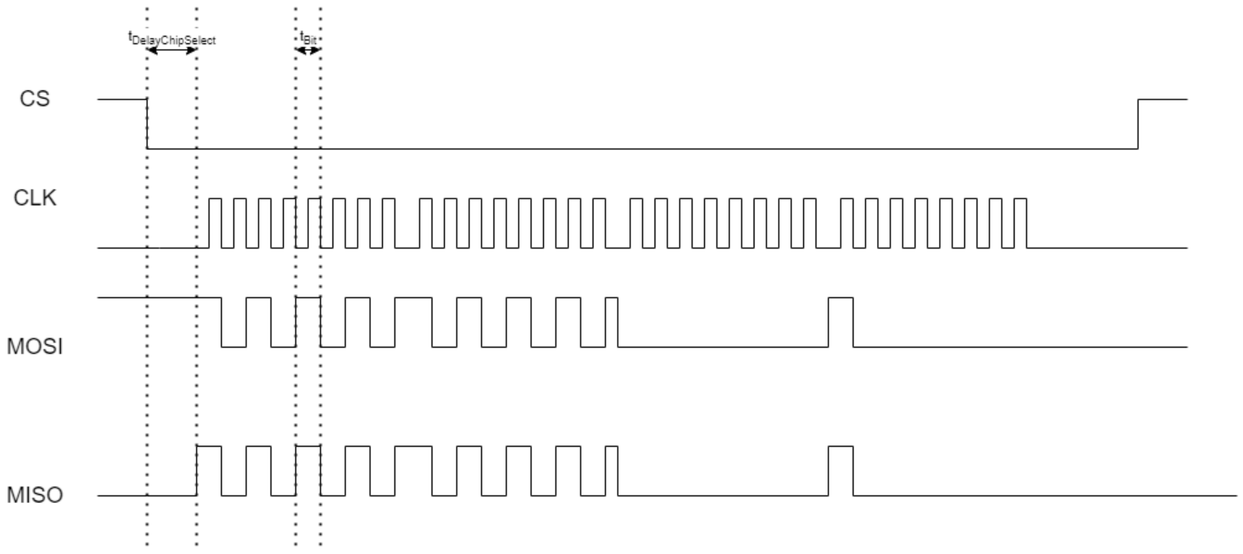
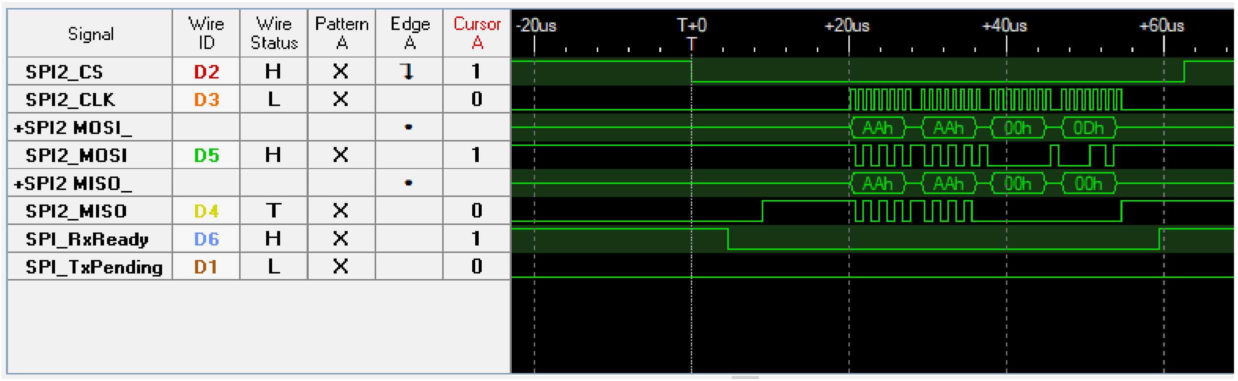


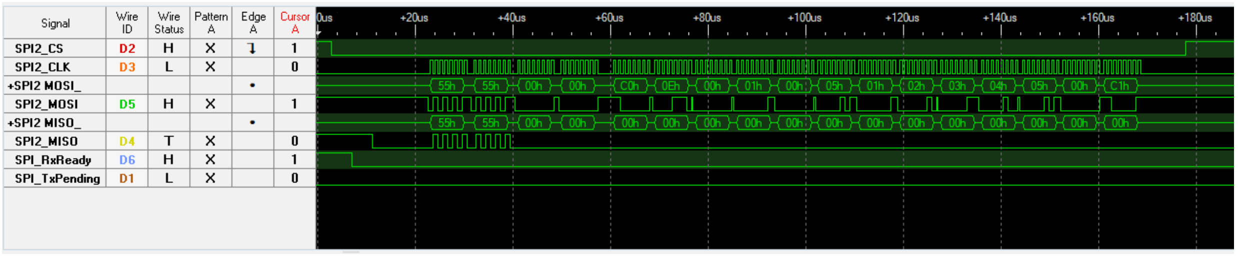
Figure 15: SPI Transfer Timings

9.2.7 Example Frames



Signal	Wire ID	Wire Status	Pattern A	Edge A	Cursor A
SPI2_CS	D2	H	X	↧	1
SPI2_CLK	D3	L	X		0
+SPI2 MOSI_				•	
SPI2_MOSI	D5	H	X		1
+SPI2 MISO_				•	
SPI2_MISO	D4	T	X		0
SPI_RxReady	D6	H	X		1
SPI_TxPending	D1	L	X		0

Figure 16: Example Size-Exchange-Frame



Signal	Wire ID	Wire Status	Pattern A	Edge A	Cursor A
SPI2_CS	D2	H	X	↧	1
SPI2_CLK	D3	L	X		0
+SPI2 MOSI_				•	
SPI2_MOSI	D5	H	X		1
+SPI2 MISO_				•	
SPI2_MISO	D4	T	X		0
SPI_RxReady	D6	H	X		1
SPI_TxPending	D1	L	X		0

0us
+20us
+40us
+60us
+80us
+100us
+120us
+140us
+160us
+180us
55h 55h 00h 00h C0h 0Eh 00h 01h 00h 05h 01h 02h 03h 04h 05h 00h C1h
55h 55h 00h 00h 00h 00h 00h 00h 00h 00h 00h 00h 00h 00h 00h 00h 00h

Figure 17: Example Data-Exchange-Frame

10 Framing Protocol

The framing protocol allows multiple data streams to be multiplexed on a single serial connection.

When using the framing protocol, all data is transferred in frames of limited size.

All values will be transferred in big-endian byte order (most significant byte first).

Note: The structure of this protocol is described in the following chapter in more detail. For sending HCI API commands, this overview is sufficient and the more detailed information are not absolutely necessary. The structure of these frames can be found in Chapter 19.

10.1 Frame format

The frame consists of a marker for the start and a simple header, followed by zero or more bytes of payload, a checksum and an end marker. The frames have the following format (negative offset means number of bytes from end of frame):

0	**1**	**2**	**3**	**4**	**5**
Start (0xC0)	ModID	SubID	ReqID	Payload Size	
		... Payload Data ...			
				-2	**-1**
	... Payload Data ...			Checksum	End (0xC1)

Figure 18: Framing Protocol Format

The fields marked in yellow (Start, Checksum and End) are specific to the frame format.

The fields have the following meaning:

Table 18: Framing Header

Byte #	***Short name***	***Description***	***Section***
0	Start	Marker for the beginning of a frame (0xC0)	-
1	ModID	Module ID of the module on the WHITE beet - Module to communicate with	10.1.1
2	SubID	Module specific Sub-ID; used to distinguish between data transfers, configuration commands / return values and asynchronous status messages	10.1.2
3	ReqID	Request ID; Can be chosen freely by the host controller when sending requests to the WHITE beet - Module; will be send back in synchronous and asynchronous responses. When sending messages not directly related to a request by the host controller, the WHITE beet - Module will use the request ID 0xff.	10.1.3
4, 5	Payload Size	Size of the payload; may be zero.	-
-2	Checksum	The one´s complement of the one's complement sum of all frame	10.1.4

Byte #	Short name	Description	Section
		bytes; set to 0x00 to disable checksum verification.	
-1	End	Marker for end of frame (0xC1).	-

10.1.1 Module ID

The Module ID is used to distinguish the different software modules. When sending any request to the WHITE beet - Module, the host controller shall address a specific software module to handle the request.

The interpretation of the Sub-ID and the payload data contained in the frame depends on the module that is addressed in the request.

The following module IDs have currently been assigned:

Table 19: Module IDs

Module	Module ID	Section
Framing	0x0E	10.3
System	0x10	11
Network Configuration	0x05	12
Firmware Update	0x11	13
Control Pilot Service	0x29	14
SLAC Service	0x28	15
Vehicle to Grid Service	0x27	17
GPIO Module	0x0F	18
Framing protocol	0xFF	-

10.1.2 Sub-ID

The Sub-ID is used to distinguish different kinds of data transfer messages, configuration commands and asynchronous status messages.

Sub-IDs are module specific in general, but they are divided into the following groups:

Table 20: Sub-IDs

Range	Description
0x00 - 0x3F	Data Transfer
0x40 - 0x7F	Configuration commands and immediate responses (see also section 10.1.5)
0x80 - 0xFF	Asynchronous status messages and responses

In case of any error that cannot be handled by a module itself, a status message will be sent with the Module ID set to 0xFF and the Sub-ID denoting the reason of failure (section 10.3.1).

Any such status message may indicate a synchronization error and one of the procedures from section 10.2 should be followed if such a message is received.

10.1.3 Request ID

The request ID can be chosen freely by the host controller when sending a request. It will be returned in any immediate and asynchronous responses and will not be interpreted in any way. The purpose of the request ID is to allow the host controllers to recognize responses of sent requests.

When sending messages not related to any particular request by the host controller, the WHITE beet – module will set the Request ID to 0xFF.

10.1.4 Checksum

The integrity of frames can be verified using the checksum field. The checksum is optional and may be set to zero to indicate that no checksum has been computed.

The checksum is computed as the one´s complement of the one´s complement sum of all frame bytes, including the Start, all header fields, the payload data, the checksum field and the End. For the computation, the checksum field has to be set to zero. If the computed checksum is zero, it has to be inverted and send as 0xFF instead.

The checksum can be programmed as follows:

```c
uint8_t checksum(uint8_t aucData[], uint16_t usSize)
{
    uint32_t ulChecksum = 0U;
    uint16_t i;

    for (i = 0U; i < usSize; ++i)
    {
        ulChecksum += aucData[i];
    }

    ulChecksum = (ulChecksum & 0xFFFF) + (ulChecksum >> 16);
    ulChecksum = (ulChecksum & 0xFF)   + (ulChecksum >> 8);
    ulChecksum = (ulChecksum & 0xFF)   + (ulChecksum >> 8);

    if (ulChecksum != 0xFF)
    {
        ulChecksum = ~ulChecksum;
    }

    return (uint8_t)ulChecksum;
}
```

When a frame is received and the checksum is not zero, the same calculation can be performed to verify the integrity of the frame. For a frame with a valid checksum, the computation will always return the value 0xFF.

10.1.5 Responses to Configuration Commands

Configuration commands will always produce an immediate response. The responses are sent back to the Host Controller using the Sub-ID of the configuration command and the Request ID chosen by the Host Controller. The first byte of the payload data will denote whether the command was accepted and, in case of an error, the reason of failure.

The following result codes are used across all modules:

Table 21: Response Codes

Code	Description
0x00	Command accepted; data may follow if specified
0x01	Module is busy; try again later
0x02	Sub-ID is unknown
0x03	Command is unknown
0x04	Malformed data
0x05	Unexpected message; module is not in a state to execute the command
0xFF	Internal error

10.2 Recovering from Synchronization Errors

A framing error might signal that the synchronization has been lost and subsequent frames might not be correctly detected until resynchronization.

To help the system recover as fast as possible, the Host Controller should perform one of the following steps:

1. Stop sending any data for at least 2*00* ms (see below for possible caveats) or
2. Send Ping messages at regular intervals (e.g. *every 50* ms) until a response is received.

The second option can ensure that communication is synchronized, as it will receive a feedback when recovery from a synchronization error has taken place. Also, data transfers may be restarted slightly faster.

10.3 Framing Module

The framing module is able to answer Ping messages, so that the reachability of the Module can be tested and synchronization can be ensured.

The framing module has the Module-ID 0x0E

10.3.1 Sub-IDs used by the Framing Module

The framing module uses the following Sub-IDs:

Table 22: Framing Module Sub-ID´s

Generic Sub-IDs		
Sub-ID	**Description**	**Section**
0x00	Ping messages send from the host controller to the WHITE beet - Module.	10.3.2
0x01	Replies to ping messages send back to the host controller.	10.3.2
Status Messages		
Sub-ID	**Description**	**Section**
0xF0	RX timeout; a pause greater than 100 ms occurred while receiving a frame	10.2
0xF1	Missing start of frame	10.2
0xF2	Checksum error	10.2
0xF3	Missing end of frame	10.2
0xF4	Module Unknown	10.2
0xF5	Frame too big; Total frame size exceeds 4096 bytes	10.2
0xF6 - 0xFD	Reserved for future use; treat as synchronization error	10.2

10.3.2 Ping Messages and Replies

Ping messages may be sent by the host controller at any time. Whenever the framing module receives a Ping message, it will send back a reply.

Ping messages may contain an arbitrary amount of data. Data contained in a Ping message is not interpreted in any way, but is send back to the host controller in the reply.

Ping messages may be used to determine if the WHITE beet – Module is ready to receive further data or commands. They may also be used to ensure resynchronization after a framing error has occurred (see chapter 10.2).

11 System Module

The System module can be used to configure or read system variables, restart the device or reset the device to factory settings.

The System module has the Module-ID 0x10.

11.1 Sub-IDs used by the System Module

The System module uses the following Sub-IDs:

Generic Sub-IDs		
Sub-ID	**Description**	**Section**
-	-	-
Configuration Commands		
Sub-ID	**Description**	**Section**
0x40	Get Version	11.3.1
0x41	Get Firmware Version	11.3.2
0x45	Get MAC Address	11.3.3
0x46	Set Serial Number	11.3.4
0x47	Get Serial Number	11.3.5
0x48	Restart System	11.3.6
0x49	Factory Reset	11.3.7
0x4A	Set Save Mode	11.3.8
0x4B	Get Save Mode	11.3.9
0x4C	Save Configuration	11.3.10
0x57	Get API Info	11.3.11
0x58	Get Uptime	11.3.12
0x59	Get Kernel Version	11.3.13
0x5A	Get File Hash	11.3.14

11.2 Error Handling

Errors will be returned using the same sub ID as the message they are related to. When an error does not relate to any specific message, it will be send as a status message instead.

In addition to the generic error codes described in section 10.1.5, the following error codes will be used by the module:

0x10 – File not found

11.3 Configuration Commands

11.3.1 Get Version

Get Version		
Sub-ID	0x40	
Description	Get the system software version.	
Parameters		
-		
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5
Version	String[0...12]	Version Number String

Example:

Get Version:

```
C0 10 40 01 00 00                 (Get Version; ReqID: 1; Payload: none)
00 C1
```

Response:

```
C0 10 40 01 00 09                 (Response; ReqID: 1; Payload: 9 bytes)
00                                (Acknowledgement)
07 31 2e 31 2e 30 2e 30                  (Version 1.1.0.0)
00 C1
```

11.3.2 Get Firmware Version

Get Firmware Version		
Sub-ID	0x41	
Description	Get the firmware version.	
Parameters		
-		
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5
Version	String[0...32]	Version number

Example:

Get Firmware Version:

```
C0 10 41 02 00 00          (Get Firmware Version; ReqID: 2; Payload: none)
00 C1
```

Response:

```
C0 10 41 02 00 09          (Response; ReqID: 1; Payload: 9 bytes)
00                         (Acknowledgement)
07 31 2e 32 2e 30 2e 30          (Version 1.2.0.0)
00 C1
```

11.3.3 Get MAC-Address

Get MAC-Address		
Sub-ID	0x45	
Description	Get the MAC Address of the system.	
Parameters		
-		
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5
MAC	uint8[6]	MAC Address

Example:

Get MAC Address:

```
C0 10 45 06 00 00          (Get MAC Address; ReqID: 6; Payload: none)
00 C1
```

Response:

```
C0 10 45 06 00 08          (Response; ReqID: 6; Payload: 2 bytes)
00                         (Acknowledgement)
06 00 01 02 03 04 05               (MAC: 00:01:02:03:04:05)
00 C1
```

11.3.4 Set Serial Number

Set Serial Number		
Sub-ID	0x46	
Description	Set the system serial number. Note: The serial number can only be set once!	
Parameters		
Name	**Type**	**Description**
Serial Number	String[0...15]	System Serial Number.
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5

Example:

Set serial number:

```
C0 10 46 07 00 09          (Set Serial Number; ReqID: 7; Payload: 9 byte)
08 53 65 72 30 30 30 30 31 (Ser00001)
00 C1
```

Response:

```
C0 10 46 07 00 01          (Response; ReqID: 7; Payload: 1 byte)
00                         (Acknowledgement)
00 C1
```

11.3.5 Get Serial Number

Get Serial Number		
Sub-ID	0x47	
Description	Get the device serial number.	
Parameters		
-		
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5
Version	String[0...15]	Device Serial Number.

Example:

Get Serial Number:

```
C0 10 47 08 00 00          (Get Serial Number; ReqID: 8; Payload: none)
00 C1
```

Response:

```
C0 10 47 08 00 0A          (Response; ReqID: 8; Payload: 10 bytes)
00                         (Acknowledgement)
08 53 65 72 30 30 30 30 31 (Ser00001)
00 C1
```

11.3.6 Restart System

Restart System		
Sub-ID	0x48	
Description	Restart the device. Note: Unsaved data will be lost.	
Parameters		
Name	**Type**	**Description**
-	-	No parameters
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5

Example:

Restart the Device:

```
C0 10 48 09 00 00          (Restart Device; ReqID: 9; Payload: 0 byte)
00 C1
```

Response:

```
C0 10 48 09 00 01          (Response; ReqID: 9; Payload: 1 byte)
00                         (Acknowledgement)
00 C1
```

11.3.7 Factory Reset

Factory Reset		
Sub-ID	0x49	
Description	Initiate the reset to factory settings. Note: The serial number is not affected!	
Parameters		
Name	**Type**	**Description**
-	-	none
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5

Example:

Factory Reset:

```
C0 10 49 0A 00 00          (Factory Reset; ReqID: 10; Payload: 0 byte)
00 C1
```

Response:

```
C0 10 49 0A 00 01          (Response; ReqID: 10; Payload: 1 byte)
00                         (Acknowledgement)
00 C1
```

11.3.8 Set Save Mode

Set Save Mode		
Sub-ID	0x4A	
Description	Configure Module to preferred save mode. Setting the mode to '0' will enable manual save mode. Setting the mode to '1' will enable the automatic save mode. Manual means the user has to manually trigger the saving of the current configuration. Automatic means, the configuration will be saved automatically after every change.	
Parameters		
Name	**Type**	**Description**
Mode	uint8	Save Mode configuration: 0: Manual Save Mode. 1: Automatic Save Mode.
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5

Example:

Set Save Mode:

```
C0 10 4A 0B 00 01          (Set Save Mode; ReqID: 11; Payload: 1 byte)
01                         (Automatic Save Mode)
00 C1
```

Response:

```
C0 10 4A 0B 00 01          (Response; ReqID: 11; Payload: 1 byte)
00                         (Acknowledgement)
00 C1
```

11.3.9 Get Save Mode

Get Save Mode		
Sub-ID	0x4B	
Description	Get the configured save mode.	
Parameters		
-		
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5
Mode	uint8	Configured Mode: 0: Manual Save Mode. 1: Automatic Save Mode.

Example:

Get Save Mode:

```
C0 10 4B 0C 00 00          (Get Save Mode; ReqID: 12; Payload: none)
00 C1
```

Response:

```
C0 10 4B 0C 00 02          (Response; ReqID: 12; Payload: 2 bytes)
00                         (Acknowledgement)
01                         (Automatic Save Mode)
00 C1
```

11.3.10 Save Configuration

Save Configuration		
Sub-ID	0x4C	
Description	This command triggers the saving of the configuration.	
Parameters		
Name	**Type**	**Description**
-	-	-
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5

Example:

Save the Configuration:

```
C0 10 4C 0D 00 01          (Save Configuration; ReqID: 13; Payload: none)
00 C1
```

Response:

```
C0 10 4C 0D 00 01          (Response; ReqID: 13; Payload: 1 byte)
00                         (Acknowledgement)
00 C1
```

11.3.11 Get API Info

Get API Info		
Sub-ID	0x57	
Description	Get the Application protocol type, to identify the application.	
Parameters		
-		
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5
Type	string[0...32]	API Type.
Version	string[0...32]	API Version.

Example:

Get API Info:

```
C0 10 57 18 00 00          (Get API Info; ReqID: 24; Payload: none)
00 C1
```

Response:

```
C0 10 57 18 00 0D          (Response; ReqID: 24; Payload: 13 bytes)
00                         (Acknowledgement)
05 69 6f 74 2e 31          (iot.1)
05 31 2e 30 2e 30          (1.0.0)
00 C1
```

11.3.12 Get Uptime

Get Uptime		
Sub-ID	0x58	
Description	Get the system uptime in seconds.	
Parameters		
-		
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5
Uptime	uint32	Device uptime in seconds (from start of device).

Example:

Get connection state:

```
C0 10 58 19 00 00          (Get Status; ReqID: 25; Payload: none)
00 C1
```

Response:

```
C0 10 58 19 00 05          (Response; ReqID: 25; Payload: 5 bytes)
00                         (Acknowledgement)
00 00 00 0A                (10 seconds)
00 C1
```

11.3.13 Get Kernel Version

Get Kernel Version		
Sub-ID	0x59	
Description	Get the kernel version of the system	
Parameters		
-		
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5
Version	string[0...12]	Kernel version string

Example:

Get kernel version:

```
C0 10 59 1A 00 00          (Get kernel version; ReqID: 26; Payload: none)
00 C1
```

Response:

```
C0 10 59 1A 00 09          (Response; ReqID: 26; Payload: 9 bytes)
00                         (Acknowledgement)
07 31 2E 30 2E 30 2E 30            (Version 1.0.0.0)
9D C1
```

11.3.14 Get File Hash

Get File Hash		
Sub-ID	0x5A	
Description	Get the SHA 256 value for the requested file. The result will contain the file size and sha256 hash value. With this the file content can be verified, that is stored correctly in the flash file system.	
Parameters		
Name	**Type**	**Description**
Filename	string[0...255]	Path and Filename
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5 Module specific result code; see section 11.2.
File size	uint32	File size
File hash	uint8[32]	32 byte array with sha256 value of file.

Example:

Get file hash for fs/cert/fwu/prot/9b5af1c195d5118b5da7b193e8a8348c0886289b.der:

```
C0 10 5A FF 00 3E           (Get File Hash; ReqID: FF; Payload: 62 bytes)

3D 66 73 2F 63 65 72 74 2F 66 77 75 2F 70 72 6F 74 2F 39 62 35 61 66 31 63 31 39 35 64
35 31 31 38 62 35 64 61 37 62 31 39 33 65 38 61 38 33 34 38 63 30 38 38 36 32 38 39 62
2E 64 65 72
       (filename: fs/cert/fwu/prot/9b5af1c195d5118b5da7b193e8a8348c0886289b.der)
```

Response:

```
C0 10 5A FF 00 25           (Response; ReqID: FF; Payload: 37 bytes)

00                          (Acknowledgement)

00 00 03 EF                 (1007 bytes file size)

32 DB 4C 70 EC 44 E3 2B FA BC 04 8F D4 6C 6E 37 8C BA C1 D2 5E 55 1B F2 F5 DD 97 48 09
D1 01 D2                    (32 byte hash value)

BD C1
```

12 Network Configuration Module

The Network Configuration module allow to set up basic network parameters for the network interfaces of the USER-Module.

The Network Configuration module has the Module-ID 0x05

12.1 Sub-IDs used by the Network Configuration Module

The framing module uses the following Sub-IDs:

Configuration Commands		
Sub-ID	**Description**	**Section**
0x42	Set IPv6 Configuration	12.3.1
0x43	Get IPv6 Configuration	12.3.2
0x4A	Get Link Status	12.3.3
0x4B	Send Ping	12.3.4
0x4C	Set IPv6 Router	12.3.5
0x4D	Get IPv6 Router	12.3.6
0x50	Get Interface Information	12.3.7
0x51	Set Interface State	12.3.8
0x52	Get Interface State	12.3.9

12.2 Status Messages

Status messages will be send asynchronously, without being explicitly requested.

They will be send using the sub ID 0xFF. The data will consist of a single byte containing the status code, possibly followed by additional data.

The following status messages may be send by the module:

Status Messages		
Sub-ID	**Description**	**Section**
0x80	Interface Down	12.4.1
0x81	Interface Up	12.4.2
0x82	Echo Received	12.4.3
0x83	Ping Failed	12.4.4

12.3 Configuration Commands

12.3.1 Set IPv6 Configuration

Set IPv6 Configuration		
Sub-ID	0x42	
Description	Set the IPv6 configuration of an interface	
Parameters		
Name	**Type**	**Description**
Interface	uint8	Interface: 0x00 - Access Point 0x01 - Station
Command	uint8	0 – Delete given IPv6 address 1 – Add given IPv6 address
IP-Address	uint8[16]	IP-Address
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5

Example:

Set the configuration of the station interface:

```
C0 05 42 00 00 12                  (Set IPv6 Config; ReqID: 0; Payload: 18 bytes)
01                                 (Configure the station interface)
01                                 (Set the mode to Manual Configuration)
52 60 2A BD B2 21 BC C9 DB 83 47 F2 69 93 42 4D
                                   (IPv6: 5260:2abd:b221:bcc9:db83:47f2:6993:424d )
00 C1
```

Response:

```
C0 05 42 00 00 01                  (Response; ReqID: 0; Payload: 1 byte)
00                                 (Acknowledgement)
35 C1
```

12.3.2 Get IPv6 Configuration

Get IPv6 Configuration		
Sub-ID	0x43	
Description	Get the IPv6 configuration of an interface	
Parameters		
Name	**Type**	**Description**
Interface	uint8	Interface: 0x00 - Access Point 0x01 - Station
Index	uint16	IP table index to be queried.
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5
IPv6-Address	uint8[16]	IPv6 address on queried index

Example:

Get the configuration of the Access Point interface:

```
C0 05 43 00 00 01                  (Get IPv6 Config; ReqID: 0; Payload: 1 byte)
01                                 (Query configuration of station interface)
00 01                              (Query index 1)
00 C1
```

Response:

```
C0 05 43 00 00 11                  (Response; ReqID: 0; Payload: 17 bytes)
00                                 (Acknowledgement)
52 60 2A BD B2 21 BC C9 DB 83 47 F2 69 93 42 4D
                                   (IPv6: 5260:2abd:b221:bcc9:db83:47f2:6993:424d)
09 C1
```

12.3.3 Get Link Status

Get Link Status		
Sub-ID	0x4A	
Description	Get link status of an interface	
Parameters		
Name	**Type**	**Description**
Interface	uint8	Interface: 0x00 - Access Point 0x01 - Station
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5
Link Status	boolean	Link status

Example:

Get the link status of the Access Point interface:

```
C0 05 4A 0B 00 01                 (Set DHCP Config; ReqID: 11; Payload: 1 byte)
00                                (Access Point)
00 C1
```

Response:

```
C0 05 4A 0B 00 02                 (Response; ReqID: 11; Payload: 2 bytes)
00                                (Acknowledgement)
01                                (Link: True)
00 C1
```

12.3.4 Ping

Ping		
Sub-ID	0x4B	
Description	Send an ICMP ping	
Parameters		
Name	**Type**	**Description**
Interface	uint8	Interface: 0x00 - Access Point 0x01 - Station
IP-Address	uint8[4]	IP-Address
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5
Related Status Messages		
Name	**Code**	**Description**
Echo Received	0x82	An echo has been received (section 12.4.3)
Ping Failed	0x83	Host unreachable or failed to reply (section 12.4.4)

Example:

Send an ICMP ping:

```
C0 05 4B 0C 00 05                 (Ping; ReqID: 12; Payload: 5 byte)
01                                (Station)
C0 A8 64 17                       (IP-Address: 192.168.100.23)
00 C1
```

Response:

```
C0 05 4B 0C 00 01                 (Response; ReqID: 12; Payload: 1 byte)
00                                (Acknowledgement)
00 C1
```

Asynchronous Status message:

```
C0 05 82 0C 00 01                 (Echo Received; ReqID: 12; Payload: 2 byte)
00 2A                             (RTT: 42ms)
00 C1
```

12.3.5 Set IPv6 Router Configuration

Set IPv6 Router Configuration		
Sub-ID	0x4C	
Description	Set IPv6 router configuration of an interface	
Parameters		
Name	**Type**	**Description**
Interface	uint8	Interface: 0x00 - Access Point 0x01 - Station
Command	uint8	0 – Delete given router IP 1 – Add given router IP
IPv6-Address	uint8[16]	IPv6-Address
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5

Example:

Set the router configuration of the station interface:

```
C0 05 4C 00 00 12                  (Set IPv6 Config; ReqID: 0; Payload: 18 bytes)
01                                 (Configure the station interface)
01                                 (Add the given address)
52 60 2A BD B2 21 BC C9 DB 83 47 F2 69 93 42 4D
                                   (IPv6: 5260:2abd:b221:bcc9:db83:47f2:6993:424d)
00 C1
```

Response:

```
C0 05 4C 00 00 01                  (Response; ReqID: 0; Payload: 1 byte)
00                                 (Acknowledgement)
2B C1
```

12.3.6 Get IPv6 Router Configuration

Get IPv6 Router Configuration		
Sub-ID	0x4D	
Description	Get the IPv6 router configuration of an interface	
Parameters		
Name	**Type**	**Description**
Interface	uint8	Interface: 0x00 - Access Point 0x01 - Station
Index	uint16	IP table index to be queried.
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5
IPv6-Address	uint8[16]	IPv6 address on queried index

Example:

Get the configuration of the Access Point interface:

```
C0 05 4D 00 00 02                 (Get IPv6 Config; ReqID: 0; Payload: 2 bytes)
01                                (Query configuration of station interface)
01                                (Query index 1)
00 C1
```

Response:

```
C0 05 4D 00 00 11                 (Response; ReqID: 0; Payload: 17 bytes)
00                                (Acknowledgement)
52 60 2A BD B2 21 BC C9 DB 83 47 F2 69 93 42 4D
                                  (IPv6: 5260:2abd:b221:bcc9:db83:47f2:6993:424d)
FE C1
```

12.3.7 Get Interface Info

Get Interface Info		
Sub-ID	0x50	
Description	Get information about the interfaces.	
Parameters		
Name	**Type**	**Description**
-	-	-
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5
Following elements are repeated for the number of interfaces		
Index	uint8	Index of the interface
Type	uint8	Type of the interface (0xFF is uninitialized) 0 – UART 1 – LAN 2 – WLAN 3 – RNDIS
Name	string[0-16]	Name of the interface
Mode	uint8	Mode of the interface 0 – unspecified 1 – Station Interface 2 – Access Point Interface

Example:

Get the configuration of the Access Point interface:

```
C0 05 50 00 00 00                 (Get interface info; ReqID: 0; Payload: 0 bytes)
00 C1
```

Response:

```
C0 05 50 00 00 1A                 (Response; ReqID: 0; Payload: 26 bytes)
00                                (Acknowledgement)
00 01 04 65 74 68 30 00           (Index: 0, Type: 1, Name: eth0, Mode: 0)
01 01 04 65 74 68 31 00           (Index: 1, Type: 1, Name: eth1, Mode: 0)
02 01 05 77 6C 61 6E 30 00        (Index: 2, Type: 1, Name: wlan0, Mode: 0)
01                                (Captive mode 1 active)
31 C1
```

12.3.8 Set Interface State

Set Interface State		
Sub-ID	0x51	
Description	Sets the state of the interface	
Parameters		
Name	**Type**	**Description**
Index	uint8	Index of the interface where the mode should be set.
State	uint8	State to set for the interface on given index 0 – off 1 – on
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5

Example:

Set the router configuration of the station interface:

```
C0 05 51 00 00 02                 (Set interface state; ReqID: 0; Payload: 2 bytes)
01                                (Set interface on index 1)
01                                (Set state on)
00 C1
```

Response:

```
C0 05 51 00 00 01                 (Response; ReqID: 0; Payload: 1 byte)
00                                (Acknowledgement)
26 C1
```

12.3.9 Get Interface State

Get Interface State		
Sub-ID	0x52	
Description	Get the current state of an interface.	
Parameters		
Name	**Type**	**Description**
Index	uint8	The interface to get the state of.
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5
State	uint8	Currently set state 0 – off 1 – on

Example:

Get the configuration of the Access Point interface:

```
C0 05 52 00 00 00                  (Get captive mode; ReqID: 0; Payload: 0 bytes)
00 C1
```

Response:

```
C0 05 52 00 00 02                  (Response; ReqID: 0; Payload: 2 bytes)
00                                 (Acknowledgement)
01                                 (Interface state is on)
23 C1
```

12.4 Status Messages

12.4.1 Interface Down

Interface Down		
Sub-ID	0x80	
Description	An interface went down	
Parameters		
Name	**Type**	**Description**
Interface	uint8	Interface: 0x00 - Access Point 0x01 - Station

12.4.2 Interface Up

Interface Down		
Sub-ID	0x81	
Description	An interface went up	
Parameters		
Name	**Type**	**Description**
Interface	uint8	Interface: 0x00 - Access Point 0x01 - Station

12.4.3 Echo Received

Echo Received		
Sub-ID	0x82	
Description	An echo has been received	
Response To	Ping (section 12.3.4)	
Parameters		
Name	**Type**	**Description**
RTT	uint16	RTT in milliseconds

12.4.4 Ping Failed

Ping failed	
Sub-ID	0x83
Description	No response from host
Response To	Ping (section 12.3.4)

13 Firmware Update Module

The Firmware Update module enables the host controller to update the files from device filesystem or/and firmwares from MCU via the framing interface.

The Firmware Update module has the Module-ID 0x11.

13.1 Sub-IDs used by the Firmware Update Module

The Firmware Update module uses the following Sub-IDs:

Generic Sub-IDs		
Sub-ID	**Description**	**Section**
0x04	FWU Data Frame	13.3.1
Configuration Commands		
Sub-ID	**Description**	**Section**
0x40	Get Status	13.4.1
0x41	Set Mode	13.4.2

13.2 Status Messages

Status messages will be send asynchronously, without being explicitly requested.

They will be send using the sub ID 0xFF. The data will consist of a single byte containing the status code, possibly followed by additional data.

The following status messages may be send by the module:

Status Messages		
Sub-ID	**Description**	**Section**
0x80	Firmware Update ready (success)	13.5.1
0x81	Timeout (Firmware Update failed)	13.5.2
0x82	Error (Firmware Update failed)	13.5.3
0x83	Request missing packets	13.5.4

13.3 Sending Data

Once the firmware update process was started the host controller can send data from the FWU-File.

The host controller can use the Sub-ID 0x04 (FWU Data Frame) to send arbitrary amounts of payload data.

If all data was received the module will send a notification. A notification is also send if an error or timeout is detected.

13.3.1 FWU Data Frame

FWU Data Frame		
Sub-ID	0x04	
Description	Sends firmware update data (FWU file) to the device. If the device detects a frame with to high packet number, it will send a status message 13.5.4.	
Parameters		
Name	**Type**	**Description**
PktNum	uint32	Packet number. Note: Used to detect missing frames.
Data	uint8[0..2000]	Payload data
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.2

Example:

Send data for Firmware Update:

```
C0 11 04 00 00 0C              (Send FWU Data; ReqID: 0; Payload: 12 bytes)
00 00 00 00                    (Packet Number 0)
30 31 32 33 34 35 36 37        (Data: 01234567)
00 C1
```

Response:

```
C0 11 04 00 00 01              (Response; ReqID: 0; Payload: 1 byte)
00                             (Acknowledgement)
00 C1
```

13.4 Configuration Commands

13.4.1 Get Status

Get Status		
Sub-ID	0x40	
Description	Get the status of the firmware update process.	
Parameters		
-		
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5
Status	uint8	Status of the firmware update process: 0x00 – Ready for update 0x01 – Update Active 0x02 – Checking for new firmware 0x03 – Update Done 0x04 – Update failed
Progress	uint8	Progress in percent.

Example:

Get connection state:

```
C0 11 40 01 00 00                  (Get Status; ReqID: 1; Payload: none)
00 C1
```

Response:

```
C0 11 40 01 00 03                  (Response; ReqID: 1; Payload: 3 bytes)
00                                 (Acknowledgement)
01                                 (Update active)
62                                 (Progress: 98%)
00 C1
```

13.4.2 Set Mode

Set Mode		
Sub-ID	0x41	
Description	Trigger functionalities by selected mode. Setting the mode to '0' will trigger a request to get the current firmware version on the server. Setting the mode to '1' will trigger an execution of the automatic firmware update. Setting the mode to '2' will set the firmware update module to the state that a firmware can be received from the host controller.	
Parameters		
Name	**Type**	**Description**
Mode	uint8	The mode will trigger one of the following functionalities: 0: Check via HTTP-Client if update is available (not supported) 1: Start Update using the HTTP-Client. (not supported) 2: Start Update via Framing-API.
Version	string	Version (Optional. Only used for Firmware update via HTTP-Client)
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5
RxBuffer	uint16	Receive Buffer size (Parameter only in answers for Mode 2 available).

Example:

Enable connection:

```
C0 11 41 02 00 01             (Set State; ReqID: 2; Payload: 1 byte)
02                            (Start firmware update via Framing)
00 C1
```

Response:

```
C0 11 41 02 00 01             (Response; ReqID: 2; Payload: 1 byte)
00                            (Acknowledgement)
09 CF                         (max. 2500 Bytes available space in RX Buffer)
00 C1
```

13.5 Status Messages

13.5.1 Firmware Update Ready

Firmware Update Ready		
Sub-ID	0x80	
Description	Firmware Update was successful	
Response To	Set Mode (13.4.2), Send Data (13.3.1)	
Parameters		
Name	**Type**	**Description**
-	-	none

Example:

Status message that Firmware Update was successful:

```
C0 11 80 FF 00 00          (SUCCESS; ReqID: 255; Payload: 0 byte)
00 C1
```

13.5.2 Timeout (Update failed)

Timeout		
Sub-ID	0x81	
Description	Timeout was detected.	
Response To	Set Mode (13.4.2), Send Data (13.3.1)	
Parameters		
Name	**Type**	**Description**
-	-	none

Example:

Status message that the firmware update failed because a timeout was detected.:

```
C0 11 81 FF 00 00          (Timeout; ReqID: 255; Payload: 0 byte)
00 C1
```

13.5.3 Error (Update failed)

Error	
Sub-ID	0x82
Description	Error was detected.
Response To	Set Mode (13.4.2), Send Data (13.3.1)

Example:

Status message that the firmware update failed because an error was detected.:

```
C0 11 82 FF 00 00          (Error; ReqID: 255; Payload: 0 byte)
00 C1
```

13.5.4 Request packets

Request missing packets		
Sub-ID	0x83	
Description	Missing packet was detected. Inform host for retransmission of frames.	
Response To	FWU Data Frame – Version 2 (13.3.1)	
Parameters		
Name	**Type**	**Description**
ReqType	uint8	Request type: 0: Repeat the packet starting from the specified number.
PktNum	uint32	Packet number in network byte order

Example:

Status message that the firmware update failed because an error was detected.:

```
C0 11 83 FF 00 00          (SUCCESS; ReqID: 255; Payload: 0 byte)
00                         (Repeat since given packet number)
00 00 00 10                (Repeat each frame starting from packet number 16)
00 C1
```

14 Control Pilot Service

The control pilot (CP) service is used to control and read the duty cycle of the generated PWM signal and to control the voltage level on the CP. Depending on the configuration of the service only a part of the functionality is available. In EV mode the PWM duty cycle can only be read and in EVSE mode the resistor level which is used to control the voltage on the CP cannot be set.

When starting the service the resistor level and the duty cycle of the PWM start with their default values. PWM duty cycle is starting at 100% which corresponds to state X1 and the resistor level will start at level 0 where only the 2,74k resistor is present.

The CP service has the Module-ID 0x29.

14.1 Sub-IDs used by the CP Service

The CP service uses the following Sub-IDs:

Configuration Commands		
Sub-ID	**Description**	**Section**
0x40	Set Mode	14.4.1
0x41	Get Mode	14.4.2
0x42	Start	14.4.3
0x43	Stop	14.4.4
0x44	Set PWM duty cycle (only EVSE)	14.4.5
0x45	Get PWM duty cycle	14.4.6
0x46	Set resistor level (only EV)	14.4.7
0x47	Get resistor level (only EV)	14.4.8
0x48	Get state	14.4.9
0x49	Set duty cycle notification threshold (only EV)	14.4.10
0x50	Get duty cycle notification threshold (only EV)	14.4.11
0x4B	Get CP AD value	14.4.12

14.2 Error Handling

Errors will be returned using the same sub ID as the message they are related to. If an error does not relate to any specific message, it will be send as a status message instead.

14.3 Status Messages

Status messages will be send asynchronously, without being explicitly requested.

They will be sent using the sub ID 0xFF. The data will consist of a single byte containing the status code, possibly followed by additional data.

The following status messages may be send by the module:

Status Messages		
Sub-ID	**Description**	**Section**
0x80	Duty cycle changed	14.5.1
0x81	State changed	14.5.2

14.4 Configuration Commands

This chapter describes the commands that can be sent to the module to change the configuration.

14.4.1 Set Mode

Set Mode		
Sub-ID	0x40	
Description	If the CP module supports both modes (EV and EVSE) this command can be used to change the mode. The mode can only be changed before the service is started. If the service was already started the service will respond with the result code 0x01 “Module is busy”. In this case the service needs to be stopped before using this command. If only one of the modes is supported the service will respond to this command with the result code 0x05 “Unexpected message”.	
Parameters		
Name	**Type**	**Description**
Mode	uint8 (0-1)	0: EV, 1: EVSE, 255: Mode not yet set
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code (0x00, 0x01, 0x05); see section 10.1.5

Example:

Set the mode of the CP-Service:

```
C0 29 40 00 00 01          (Set Mode; ReqID: 0; Payload: 1 byte)
01                         (Set Mode to EVSE)
00 C1
```

Response:

```
C0 29 40 00 00 01          (Response; ReqID: 0; Payload: 1 byte)
00                         (Acknowledgement)
00 C1
```

14.4.2 Get Mode

Get Mode	
Sub-ID	0x41
Description	This command can be used to determine the mode of the service is in.

Parameters		
Name	**Type**	**Description**
-	-	-
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code (0x00); see section 10.1.5
Mode	uint8 (0-1)	0: EV, 1: EVSE

Example:

Request the current mode from the CP-Service:

```
C0 29 41 01 00 00          (Get Mode; ReqID: 1; Payload: 0 byte)
00 C1
```

Response:

```
C0 29 41 01 00 02          (Response; ReqID: 1; Payload: 2 byte)
00                         (Acknowledgement)
01                         (Mode is EVSE)
00 C1
```

14.4.3 Start

Start		
Sub-ID	0x42	
Description	This command starts the CP service. As long as the service is not started setting the duty cycle of the PWM signal or setting the resistor level do not have any effects, but the service will store these values and use them as soon as it is started. If the values are not set beforehand the default values will be used. As EV the resistor level will be set initially to level 0 (only resistor 2,74k is connected). As EVSE the duty cycle of the PWM will be set to 100% (state X1). If the service was already started the result code 0x05 “Unexpected message” will be returned.	
Parameters		
Name	**Type**	**Description**
-	-	-
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code (0x00, 0x05); see section 10.1.5

Example:

Start CP-Service process:

```
C0 29 42 02 00 00          (Start CP-Service; ReqID: 2; Payload: 0 byte)
00 C1
```

Response:

```
C0 29 42 02 00 01        (Response; ReqID: 2; Payload: 1 byte)
00                       (Acknowledgement)
00 C1
```

14.4.4 Stop

Stop		
Sub-ID	0x43	
Description	This command stops the CP service. The service will return to the default state. As EV the resistor level will be set to level 0 (only resistor 2,74k is connected). As EVSE the duty cycle of the PWM will be set to 100% (state X1). If the service was not started beforehand the result code 0x05 “Unexpected message” will be returned.	
Parameters		
Name	**Type**	**Description**
-	-	-
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code (0x00, 0x05); see section 10.1.5

Example:

Stop CP-Service process:

```
C0 29 43 03 00 00        (Stop CP-Service; ReqID: 3; Payload: 0 byte)
00 C1
```

Response:

```
C0 29 43 03 00 01        (Response; ReqID: 3; Payload: 1 byte)
00                       (Acknowledgement)
00 C1
```

14.4.5 Set PWM duty cycle (only EVSE)

Set PWM duty cycle		
Sub-ID	0x44	
Description	Sets the duty cycle percentage of the PWM signal. This command is only available if the mode is set to EVSE. Setting the duty cycle to 100% (12V continuously) corresponds to state X1, for state X2 set a duty cycle between 0-99%. Setting it to 0% will lead to the fault state F (-12V continuously). If the command is used in EV mode the result code 0x05 “Unexpected message” will be returned.	
Parameters		
Name	**Type**	**Description**
Duty cycle	uint16 (0-1000)	Duty cycle of the PWM signal in permill.
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code (0x00, 0x05); see section 10.1.5

Example:

Set the PWM Duty Cylce to 5%:

```
C0 29 44 04 00 02          (Set Duty-Cycle; ReqID: 4; Payload: 2 bytes)
00 32                      (Set Duty Cycle to 5%)
00 C1
```

Response:

```
C0 29 44 04 00 01          (Response; ReqID: 4; Payload: 1 byte)
00                         (Acknowledgement)
00 C1
```

14.4.6 Get PWM duty cycle

Get PWM duty cycle		
Sub-ID	0x45	
Description	This command can be used to determine the duty cycle of the PWM signal.	
Parameters		
Name	**Type**	**Description**
-	-	-
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code (0x00); see section 10.1.5
Duty cycle	uint16 (0-1000)	Duty cycle of the PWM signal in permill.

Example:

Request the current Duty-Cylce from the CP-Service:

```
C0 29 45 05 00 00          (Get Duty-Cycle; ReqID: 5; Payload: 0 byte)
```

```
00 C1
```

Response:

```
C0 29 45 05 00 03          (Response; ReqID: 5; Payload: 3 bytes)
00                         (Acknowledgement)
00 32                      (Duty Cycle is 5%)
00 C1
```

14.4.7 Set resistor level (only EV)

Set resistor level		
Sub-ID	0x46	
Description	Sets the resistors to achieve a specific state on the CP. 0: state B, 1: state C, 2: state D. This command is only available when the mode is set to EV. If the command is used in EVSE mode the result code 0x05 "Unexpected message" will be returned.	
Parameters		
Name	**Type**	**Description**
Resistor level	uint8 (0-2)	The resistor level that should be set. 0: State B 1: State C 2: State D
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code (0x00, 0x05); see section 10.1.5

Example:

Set the resistor level to state C:

```
C0 29 46 06 00 01          (Set CP state to B; ReqID: 6; Payload: 1 byte)
01                         (Set resistors for state C)
00 C1
```

Response:

```
C0 29 46 06 00 01          (Response; ReqID: 6; Payload: 1 byte)
00                         (Acknowledgement)
00 C1
```

14.4.8 Get resistor level (only EV)

Get resistor level	
Sub-ID	0x47
Description	This command can be used to determine the current resistor level. If the command is used in EVSE mode the result code 0x05 "Unexpected message" will be returned.

Parameters		
Name	**Type**	**Description**
-	-	-
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code (0x00, 0x05); see section 10.1.5
Resistor level	uint8 (0-2)	The resistor level that is currently set.

Example:

Request the current resistor level from the CP-Service:

```
C0 29 47 07 00 00          (Get resistor level; ReqID: 7; Payload: 0 byte)
00 C1
```

Response:

```
C0 29 47 07 00 02          (Response; ReqID: 7; Payload: 2 byte)
00                         (Acknowledgement)
01                         (Resistor level is set for state C)
00 C1
```

14.4.9 Get state

Get state		
Sub-ID	0x48	
Description	This command can be used to determine the current state on the CP. The state depends on the voltage level that is measured on the CP line. 12V: state A, 9V: state B, 6V: state C, 3V: state D, 0V: state E, -12V: state F. The EV will only detect modes B-E. States A and F can only be seen in EVSE mode due to the way the CP circuit is designed. If state unknown is returned the voltage level couldn't be assigned to a state with enough confidence.	
Parameters		
Name	**Type**	**Description**
-	-	-
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code (0x00); see section 10.1.5
State	uint8 (0-5)	0: state A, 1: state B, 2: state C, 3: state D, 4: state E, 5: state F, 6: state unknown

Example:

Request the current detected state of the CP-Service:

```
C0 29 48 08 00 00          (Get current state; ReqID: 8; Payload: 0 byte)
00 C1
```

Response:

```
C0 29 48 08 00 02          (Response; ReqID: 8; Payload: 2 byte)
00                         (Acknowledgement)
02                         (State C)
00 C1
```

14.4.10 Set PWM duty cycle notification threshold (only EV)

Set PWM duty cycle notification threshold		
Sub-ID	0x49	
Description	Sets the threshold for receiving notifications about changes in PWM duty cycle. I.e. if the current duty cycle is 50% and the threshold is set to 5% a notification is sent if the PWM duty cycle changes to a value equal or greater than 55% or equal or less than 45%. If the command is used in EVSE mode the result code 0x05 “Unexpected message” will be returned.	
Parameters		
Name	**Type**	**Description**
Threshold level	uint16 (0-1000)	The threshold level for detecting changes in duty cycle in permill.
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code (0x00, 0x05); see section 10.1.5

Example:

Set threshold for PWM duty cycle notification to 1%:

```
C0 29 49 09 00 02          (Set threshold to 1%; ReqID: 9; Payload: 2 bytes)
00 0A                      (1%)
00 C1
```

Response:

```
C0 29 49 09 00 01          (Response; ReqID: 9; Payload: 1 byte)
00                         (Acknowledgement)
00 C1
```

14.4.11 Get PWM duty cycle notification threshold (only EV)

Get PWM duty cycle notification threshold		
Sub-ID	0x4A	
Description	This command can be used to determine the current threshold for notification about changes in PWM duty cycle. If the command is used in EVSE mode the result code 0x05 “Unexpected message” will be returned.	
Parameters		
Name	**Type**	**Description**
-	-	-
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code (0x00, 0x05); see section 10.1.5
Threshold level	uint16 (0-1000)	The threshold level for detecting changes in duty cycle in permill.

Example:

Request the current threshold level for detecting changes in duty cycle:

```
C0 29 4A 0A 00 00          (Get threshold; ReqID: 10; Payload: 0 byte)
00 C1
```

Response:

```
C0 29 4A 0A 00 03          (Response; ReqID: 10; Payload: 3 bytes)
00                         (Acknowledgement)
00 14                      (2 %)
00 C1
```

14.4.12 Get CP AD value

Get CP AD value		
Sub-ID	0x4B	
Description	This command can be used to AD value from CP pin.	
Parameters		
Name	**Type**	**Description**
-	-	-
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code (0x00); see section 10.1.5
AD value	uint32	AD value for CP pin.

Example:

Request the current AD value of the CP-Service:

```
C0 29 4B 0B 00 00          (Get AD value; ReqID: 11; Payload: 0 byte)
00 C1
```

Response:

```
C0 29 4B 0B 00 05          (Response; ReqID: 11; Payload: 5 byte)
00                         (Acknowledgement)
00 00 0E 98                (3736)
00 C1
```

14.5 Status messages

This chapter describes the commands that can be sent to the module to change the configuration.

14.5.1 Duty cycle changed

Duty cycle changed		
Sub-ID	0x80	
Description	This status message is sent if the duty cycle of the PWM changed and exceeded the configured threshold.	
Parameters		
Name	**Type**	**Description**
Duty cycle	uint16 (0-1000)	The current value of the duty cycle in permill.

Example:

Status message that the duty cycle has changed to 5%:

```
C0 29 80 FF 00 02          (Changed duty cycle; ReqID: 255; Payload: 2 bytes)
00 32                      (5 %)
00 C1
```

14.5.2 State changed

State changed		
Sub-ID	0x81	
Description	This status message is sent if the state on CP changed.	
Parameters		
Name	**Type**	**Description**
State	uint8 (0-5)	The current state on the CP.

Example:

Status message that the state has changed to C:

```
C0 29 81 FF 00 01          (Changed state; ReqID: 255; Payload: 1 byte)
```

```
02                                (State C)
00 C1
```

15 SLAC-Service

The SLAC-Service is used to control the SLAC module.

SLAC-Sevice's Module-ID is 0x28.

15.1 Sub-IDs used by the SLAC module

The SLAC Sevice module uses the following Sub-IDs:

Table 23: SLAC Service Sub-IDs

Configuration Commands		
Sub-ID	**Description**	**Section**
0x42	Start SLAC	15.4.1
0x43	Stop SLAC	15.4.2
0x44	Start SLAC matching process	15.4.3
0x46	Set AttrnRx values for EVSE	15.4.4
0x47	Get AttrnRx values	15.4.5
0x48	Set AttrnTx Reference values for EV	15.4.6
0x49	Get AttrnTx values	15.4.7
0x4A	Set BCB Toggle Result	15.4.8
0x4B	Set Validation Configuration	15.4.9
0x4C	Get Validation Configuration	15.4.10

15.2 Error Handling

Errors will be returned using the same sub ID like the messages they are related to. If an error does not relate to any specific message, it will be sent as a status message instead.

In addition to the generic error codes described in section 10.1.5.

15.3 Status Messages

Status messages will be sent asynchronously, without being explicitly requested.

They will be sent with the request ID 0xFF if the message can not be assigned to a previous message. If the message can be assigned, then the request ID from the corresponding message is used.

Table 24: SLAC-Service Status Message Sub-IDs

Status Message Sub-IDs	
Sub-ID	**Description**
0x80	SLAC process was successful
0x81	SLAC process failed
0x82	BCB Toggling started
0x83	BCB Toggling finished
0x84	Status for joining HPGP network

15.3.1 SLAC process was successful

SLAC process was successful		
Sub-ID	0x80	
Description	SLAC Matching Process was finished successful.	
Parameters		
Name	**Type**	**Description**
-	-	-

Example:

Status message that SLAC matching was processed successful:

```
C0 28 80 FF 00 00         (SUCCESS; ReqID: 255; Payload: 0 byte)
00 C1
```

15.3.2 SLAC process failed

SLAC process failed		
Sub-ID	0x81	
Description	SLAC Matching Process was finished with an error.	
Parameters		
Name	**Type**	**Description**
-	-	-

Example:

Status message that SLAC matching was processed failed:

```
C0 28 81 FF 00 00         (FAIL; ReqID: 255; Payload: 0 byte)
00 C1
```

15.3.3 BCB Toggling started

BCB Toggling started	
Sub-ID	0x82

Description	BCB toggling was started on EV side. Now try to detect state changes on CP for given window and send immediately message to set detected number of BCB toggles.	
Parameters		
Name	**Type**	**Description**
BCB Time Window	uint16	Time window for toggling

Example:

Status message that BCB toggling started for validation:

```
C0 28 82 FF 00 02          (BCB toggling started; ReqID: 255; Payload: 2 byte)
07 D0                      (2000 ms)
00 C1
```

15.3.4 BCB Toggling finished

BCB Toggling finished		
Sub-ID	0x83	
Description	BCB toggling was finished (Timeout). **Note:** The response for validation must be sent before this message was received! The application has to use the timeout from the start message.	
Parameters		
Name	**Type**	**Description**
-	-	-

Example:

Status message that BCB toggling finished for validation:

```
C0 28 83 FF 00 00          (End of BCB toggling; ReqID: 255; Payload: 0 byte)
00 C1
```

15.3.5 Status for joining HPGP network.

Status for joining HPGP network.		
Sub-ID	0x84	
Description	Joining HPGP network finished.	
Parameters		
Name	**Type**	**Description**
Status	uint8	Status for joining network. 0: Failed 1: Success

Example:

Status message that BCB toggling started for validation:

```
C0 28 84 FF 00 01          (Status for joining; ReqID: 255; Payload: 1 byte)
01                         (Success)
00 C1
```

15.4 Configuration Commands

15.4.1 Start SLAC

Table 25: Start SLAC Command

Start SLAC		
Sub-ID	0x42	
Description	Start the SLAC module in selected mode. **Note**: For starting SLAC matching process them command 'Start Matching Process' must be send (see section 15.4.3).	
Parameters		
Name	**Type**	**Description**
Mode	uint8	0x00 – EV 0x01 – EVSE
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5.

Example:

Start SLAC process:

```
C0 28 42 02 00 01          (Start SLAC; ReqID: 2; Payload: 1 byte)
01                         (Start SLAC as EVSE)
00 C1
```

Response:

```
C0 28 42 02 00 01          (Response; ReqID: 2; Payload: 1 byte)
00                         (Acknowledgement)
00 C1
```

15.4.2 Stop SLAC

Table 26: Stop SLAC Command

Stop SLAC	
Sub-ID	0x43

Description	Stop the SLAC module. If the result code 0x10 is returned the module is already stopped or was not started yet.	
Parameters		
Name	**Type**	**Description**
-	-	-
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5.

Example:

Stop SLAC process:

```
C0 28 43 03 00 00         (Stop SLAC; ReqID: 3; Payload: 0 byte)
00 C1
```

Response:

```
C0 28 43 03 00 01         (Response; ReqID: 3; Payload: 1 byte)
00                        (Acknowledgement)
00 C1
```

15.4.3 Start Matching Process

Table 27: Start SLAC matching process Command

Start SLAC matching process		
Sub-ID	0x44	
Description	Start the SLAC matching process. **EVSE**: Sets EVSE to SLAC ready state for 50 seconds. If no matching has taken place until timeout, an error message is sent – see section 19.3. **EV**: The EV will start sending SLAC messages (e.g. CM_SLAC_PARAM.REQ). **Note**: SLAC module must be started before SLAC matching process can be started (see section 15.4.1).	
Parameters		
Name	**Type**	**Description**
-	-	-
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5.

Example:

Start the SLAC matching process:

```
C0 28 44 04 00 00   (Start Matching Process; ReqID: 4; Payload: 0 byte)
00 C1
```

Response:

```
C0 28 44 04 00 01          (Response; ReqID: 4; Payload: 1 byte)
00                         (Acknowledgement)
00 C1
```

15.4.4 Set AttnRx Values (EVSE)

Table 28: Set AttnRx Value Command

Set AttnRx Values		
Sub-ID	0x46	
Description	Sets the AttnRx Values. (Only for EVSE)	
Parameters		
Name	**Type**	**Description**
AttnRx	uint8[58]	AttnRx values for all groups (58).
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5.

Example:

Set AttnRx values for EVSE:

```
C0 28 46 06 00 00          (Sets AttnRx values; ReqID: 6; Payload: 58 byte)
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F      (AttRx values for all 58 groups)
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F
30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F
40 41 42 43 44 45 46 47 48 49
00 C1
```

Response:

```
C0 28 46 06 00 01          (Response; ReqID: 6; Payload: 1 byte)
00                         (Acknowledgement)
00 C1
```

15.4.5 Get AttnRx Values (EVSE)

Table 29: Get AttnRx Value Command

Get AttnRx Values		
Sub-ID	0x47	
Description	Get the actual AttnRx values. (Only for EVSE)	
Parameters		
Name	**Type**	**Description**
-	-	-
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5.
AttnRx	uint8[58]	AttnRx values for all groups (58).

Example:

Get AttnRx values:

```
C0 28 47 07 00 00           (Gets AttnRx values; ReqID: 7; Payload: 0 byte)

00 C1
```

Response:

```
C0 28 47 07 00 01           (Response; ReqID: 7; Payload: 59 byte)

00                          (Acknowledgement)

10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F      (AttRx values for all 58 groups)
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F
30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F
40 41 42 43 44 45 46 47 48 49

00 C1
```

15.4.6 Set AttnTxRef Values (EV)

Table 30: Set AttnTxRef Value Command

Set AttnTxRef Values		
Sub-ID	0x48	
Description	Sets the AttnTx reference Values for EV. (Only for EV)	
Parameters		
Name	**Type**	**Description**
AttnTxRef	uint8[58]	AttnTxRef values for all groups (58).
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5.

Example:

Set AttnTxRef values for EV:

```
C0 28 48 08 00 00          (Sets AttnTxRef; ReqID: 8; Payload: 58 byte)

10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F (AttnTxRef values for all 58 groups)
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F
30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F
40 41 42 43 44 45 46 47 48 49

00 C1
```

Response:

```
C0 28 48 08 00 01          (Response; ReqID: 8; Payload: 1 byte)

00                         (Acknowledgement)

00 C1
```

15.4.7 Get AttnTxRef Values (EV)

Table 31: Get AttnTxRef Value Command

Get AttnTxRef Values		
Sub-ID	0x49	
Description	Get the actual AttnTx reference Values. (Only for EV)	
Parameters		
Name	**Type**	**Description**
-	-	-
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5.
AttnTxRef	uint8[58]	AttnTxRef values for all groups (58).

Example:

Get AttnTxRef values for EV:

```
C0 28 49 09 00 00          (Sets AttnTxRef; ReqID: 9; Payload: 0 byte)

00 C1
```

Response:

```
C0 28 49 09 00 01          (Response; ReqID: 9; Payload: 59 byte)

00                         (Acknowledgement)

10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F  (AttTxRef values for all 58 groups)
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F
30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F
40 41 42 43 44 45 46 47 48 49

00 C1
```

15.4.8 Set BCB Toggle Result

Set BCB Toggle Result		
Sub-ID	0x4A	
Description	Sets the result from BCB toggling. Message has to be send as soon as possible after SLAC BCB toggle time window.	
Parameters		
Name	**Type**	**Description**
Mode	uint8	Number of detected toggles in time window. 0x00 – No toggles detected 0x01...0x03 – Number of detected toggles 0x04...0xFF – Invalid values
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5.

Example:

Set number of detected BCB toggles in SLAC Validation:

```
C0 28 4A 0B 00 01          (Set BCB toggle result; ReqID: 11; Payload: 1
                           byte)
02                         (Two detected BCB state toggles)
00 C1
```

Response:

```
C0 28 4A 0B 00 01          (Response; ReqID: 11; Payload: 1 byte)
00                         (Acknowledgement)
00 C1
```

15.4.9 Set Validation Configuration

Set Validation Configuration		
Sub-ID	0x4B	
Description	Configure the SLAC service if Validation process is enabled.	
Parameters		
Name	**Type**	**Description**
Mode	uint8	0x00 – Disable 0x01 – Enable
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5.

Example:

Enable SLAC Validation:

```
C0 28 4B 0B 00 01          (Configure Validation; ReqID: 11; Payload: 1 byte)
01                         (Enable Validation)
00 C1
```

Response:

```
C0 28 4B 0B 00 01          (Response; ReqID: 11; Payload: 1 byte)
00                         (Acknowledgement)
00 C1
```

15.4.10 Get Validation Configuration

Get Validation Configuration		
Sub-ID	0x4C	
Description	Get the SLAC validation configuration.	
Parameters		
Name	**Type**	**Description**
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5
State	uint8	0x00 – Disabled 0x01 – Enabled

Example:

Get SLAC validation configuration:

```
C0 28 4C 0C 00 00          (Get Validation Config; ReqID: 12; Payload: none)
00 C1
```

Response:

```
C0 28 4C 0C 00 02          (Response; ReqID: 12; Payload: 2 byte)
00                         (Acknowledgement)
01                         (SLAC Validation is enabled)
00 C1
```

16 PLC-Service

The PLC-Service is used to control the PLC driver module.

PLC-Sevice’s Module-ID is 0x2A.

16.1 Sub-IDs used by the PLC module

The PLC Sevice module uses the following Sub-IDs:

Table 32: PLC Service Sub-IDs

Configuration Commands		
Sub-ID	**Description**	**Section**
0x40	Join HPGP network	16.4.1
0x41	Change PIB file	16.4.2
0x42	Read PIB file	16.4.3

16.2 Error Handling

Errors will be returned using the same sub ID like the messages they are related to. If an error does not relate to any specific message, it will be sent as a status message instead.

16.3 Status Messages

Status messages will be sent asynchronously, without being explicitly requested.

They will be sent with the request ID 0xFF if the message can not be assigned to a previous message. If the message can be assigned, then the request ID from the corresponding message is used.

Table 33: PLC-Service Status Message Sub-IDs

Status Message Sub-IDs	
Sub-ID	**Description**
0x80	Status for joining HPGP network

16.3.1 Status for joining HPGP network.

Status for joining HPGP network.		
Sub-ID	0x80	
Description	Joining HPGP network finished.	
Parameters		
Name	**Type**	**Description**
Status	uint8	Status for joining network. 0: Failed 1: Success

Example:

Status message that BCB toggling started for validation:

```
C0 2A 80 FF 00 01          (Status for joining; ReqID: 255; Payload: 1 byte)
01                         (Success)
00 C1
```

16.4 Configuration Commands

16.4.1 Join HPGP network

Table 34: Join HPGP network Command

Join HPGP network		
Sub-ID	0x40	
Description	Join a HPGP network by using given NID and NMK. **Note:** If command returns 'Command Accepted' a status message is sent if joining was successful or failed (16.3.1)!	
Parameters		
Name	**Type**	**Description**
NID	uint8[7]	Network ID. **Note:** Please note that the bits 7 and 8 from the last byte must be zero! If bits are not set to zero they will be ignored!
NMK	uint8[16]	Network Managment Key.
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5.

Example:

Set AttnRx values for EVSE:

```
C0 2A 40 00 00 17            (Join HPGP network; ReqID: 0; Payload: 23 byte)
01 02 03 04 05 06 07         (NID)
00 01 02 03 04 05 06 07 08 09
0A 0B 0C 0D 0E 0F            (NMK)

00 C1
```

Response:

```
C0 2A 40 00 00 01            (Response; ReqID: 0; Payload: 1 byte)
00                           (Acknowledgement)
00 C1
```

16.4.2 Change PIB file

Table 35: Change PIB file Command

Change PIB file		
Sub-ID	0x41	
Description	Modify actual PIB file.	
Parameters		
Name	**Type**	**Description**
Offset	uint16	Offset of parameter in the PIB Data area (+ 0x3C0 from the beginning of PIB file)
Length	uint16	Data length
Data	uint8[1...600]	Data byte sequence
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5.

Example:

Modify PIB file (Change MAC address):

```
C0 2A 41 01 00 0A          (Modify PIB; ReqID: 1; Payload: 10 byte)
00 0C                      (Offset 12 Bytes - MAC Address)
00 06                      (Length 6 Bytes - MAC Address length)
C4 93 00 00 00 01          (MAC Address C4:93:00:00:00:01)
00 C1
```

Response:

```
C0 2A 41 01 00 01          (Response; ReqID: 1; Payload: 1 byte)
00                         (Acknowledgement)
00 C1
```

16.4.3 Read from PIB file

Table 36: Read PIB file Command

Read PIB file command		
Sub-ID	0x42	
Description	Read data from actual PIB file.	
Parameters		
Name	**Type**	**Description**
Offset	uint16	Offset for reading parameters in the PIB Data area (+ 0x3C0 from the beginning of PIB file)
Length	uint16	Number of bytes to read from PIB file
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5.
Data	uint8[0..600]	Read data from PIB file.

Example:

Read data from PIB file:

```
C0 2A 42 02 00 04          (Sets AttnTxRef; ReqID: 9; Payload: 4 byte)

00 00                      (Offset 0 Bytes)

00 20                      (Length 32 Bytes)

00 C1
```

Response:

```
C0 2A 42 02 00 21          (Response; ReqID: 9; Payload: 33 byte)

00                         (Acknowledgement)

00 00 00 00 00 00 00 00 00 00 00 00 C4 93 00 00  (Data from PIB file)
00 01 FF FF FF FF FF FF FF FF FF FF FF FF FF FF

00 C1
```

17 Vehicle to Grid Service

The V2G service has the Module-ID 0x27.

The module can be configured using the configuration commands. These can only be used as long as the module wasn’t started. If the module was already started it needs to be stopped before the configuration can be changed.

17.1 Specific types

Type	*Size*	*Description*
exponential	3 bytes	This type consists of a sint16 value and a sint8 exponent. The final value is: value * 10 ^ exponent. The exponent must be in the range -3 to 3.

17.2 Common Sub-IDs

The V2G service uses the following Sub-IDs:

Configuration Commands		
Sub-ID	**Description**	**Section**
0x40	Set Mode	17.8.1
0x41	Get Mode	17.8.2
0x42	Start	17.8.3
0x43	Stop	17.8.4

17.3 EV Sub-IDs

EV mode currently not supported.

17.4 EVSE Sub-IDs

The V2G service uses the following Sub-IDs:

Configuration Commands		
Sub-ID	**Description**	**Section**
0x60	Set supported protocols	17.10.1
0x61	Get supported protocols	17.10.2
0x62	Set SDP config	17.10.3
0x63	Get SDP config	17.10.4
0x64	Set payment options	17.10.5
0x65	Get payment options	17.10.6
0x66	Set energy transfer modes	17.10.7

0x67	Get energy transfer modes	17.10.8
0x68	Set EVSE ID	17.10.9
0x69	Set Authorization Status	17.10.10
0x6A	Set Discovery Charge Parameters	17.10.11
0x6B	Set Schedules	17.10.12
0x6C	Set Cable Check Status	17.10.13
0x6D	Set Cable Check Parameters	17.10.14
0x6E	Set Pre Charge Parameters	17.10.15
0x6F	Set Start Charging Status	17.10.16
0x70	Set Charge Loop Parameters	17.10.17
0x71	Set Stop Charging Status	17.10.18
0x72	Set Post Charge Parameters	17.10.19

17.5 Error Handling

Errors will be returned using the same sub ID as the message they are related to. If an error does not relate to any specific message, it will be send as a status message instead.

17.6 EV Status Messages

EV mode currently not supported.

17.7 EVSE Status Messages

Status messages will be send asynchronously, without being explicitly requested.

The following status messages may be send by the module:

Status Messages		
Sub-ID	**Description**	**Section**
0x80	Session started	17.12.1
0x81	Session stopped	17.12.2
0x82	Request EVSE ID	17.12.3
0x83	Request Authorization Status	17.12.4
0x84	Request Discovery Charge Parameters	17.12.5
0x85	Request Schedules	17.12.6
0x86	Request Cable Check Status	17.12.7
0x87	Request Cable Check Parameters	17.12.8
0x88	Request Pre Charge Parameters	17.12.9
0x89	Request Start Charging	17.12.10
0x8A	Request Charge Loop Parameters	17.12.11

0x8B	Request Stop Charging	17.12.12
0x8C	Request Post Charge Parameters	17.12.13

17.8 Common Configuration Commands

This chapter describes the commands that can be sent to the module to change the configuration.

17.8.1 Set Mode

Set Mode		
Sub-ID	0x40	
Description	If the V2G module supports both modes (EV and EVSE) this command can be used to change the mode. The mode can only be changed before the service is started. If the service was already started the service will respond with the result code 0x01 “Module is busy”. In this case the service needs to be stopped before using this command. If only one of the modes is supported the service will respond to this command with the result code 0x05 “Unexpected message”.	
Parameters		
Name	**Type**	**Description**
Mode	uint8 (0-1)	0: EV, 1: EVSE
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5

Example:

Set mode to EV:

```
C0 30 40 00 00 01          (Set mode to EV; ReqID: 0; Payload: 1 byte)
00                         (Set mode to EV)
00 C1
```

Response:

```
C0 30 40 00 00 01          (Response; ReqID: 0; Payload: 1 byte)
00                         (Acknowledgement)
00 C1
```

17.8.2 Get Mode

Get Mode		
Sub-ID	0x41	
Description	This command can be used to determine the mode the service is in.	
Parameters		
Name	**Type**	**Description**
-	-	-
Returned Result		
Name	**Type**	**Description**
Mode	uint8 (0-1)	0: EV, 1: EVSE
Code	uint8	Generic result code; see section 10.1.5

Example:

Request the current mode:

```
C0 30 41 01 00 00          (Get mode; ReqID: 1; Payload: 0 byte)
00 C1
```

Response:

```
C0 30 41 01 00 02          (Response; ReqID: 1; Payload: 2 byte)
00                         (Acknowledgement)
00                         (Mode is EV)
00 C1
```

17.8.3 Start

Start		
Sub-ID	0x42	
Description	This command starts the V2G service. If the service was already started the result code 0x05 “Unexpected message” will be returned.	
Parameters		
Name	**Type**	**Description**
-	-	-
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5

Example:

Start V2G-Service:

```
C0 30 42 02 00 00          (Start V2G-Service; ReqID: 2; Payload: 0 byte)
00 C1
```

Response:

```
C0 30 42 02 00 01          (Response; ReqID: 2; Payload: 1 byte)
00                         (Acknowledgement)
00 C1
```

17.8.4 Stop

Stop		
Sub-ID	0x43	
Description	This command stops the V2G service. If the service was not started beforehand the result code 0x05 “Unexpected message” will be returned.	
Parameters		
Name	**Type**	**Description**
-	-	-
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5

Example:

Start V2G-Service:

```
C0 30 43 03 00 00          (Stop V2G-Service; ReqID: 3; Payload: 0 byte)
00 C1
```

Response:

```
C0 30 43 03 00 01          (Response; ReqID: 3; Payload: 1 byte)
00                         (Acknowledgement)
00 C1
```

17.9 EV Configuration Commands

EV mode currently not supported.

17.10 EVSE Configuration Commands

This chapter describes the commands that can be sent to the module to change the configuration.

17.10.1 Set supported protocols

Set supported protocols	
Sub-ID	0x60
Description	The supported protocols can only be changed before the service is started. If the service was already started the service will respond with the result code 0x01

	"Module is busy". In this case the service needs to be stopped before using this command.	
Parameters		
Name	**Type**	**Description**
Count	uint8 (1-4)	Number of supported protocols. The following parameter has to be repeated this many times.
Protocol	uint8 (0-3)	0: DIN70121-2:2012 1: ISO15118-2:2010 (currently not supported) 2: ISO15118-2:2014 3: ISO15118-20:2020 (currently not supported)
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5

Example:

Set supported protocols:

```
C0 30 60 00 00 03          (Set protocols; ReqID: 0; Payload: 3 byte)
02                         (2 protocols)
00                         (DIN70121-2:2012)
02                         (ISO15118-2:2014)
00 C1
```

Response:

```
C0 30 60 00 00 01          (Response; ReqID: 0; Payload: 1 byte)
00                         (Acknowledgement)
00 C1
```

17.10.2 Get supported protocols

Get supported protocols		
Sub-ID	0x61	
Description	This command can be used to retrieve a list of supported protocols.	
Parameters		
Name	**Type**	**Description**
-	-	-
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5
Count	uint8 (1-4)	Number of supported protocols. The following parameter is repeated this many times.
Supported protocol	uint8 (0-3)	0: DIN70121-2:2012 1: ISO15118-2:2010 (currently not supported) 2: ISO15118-2:2014 3: ISO15118-20:2020 (currently not supported)

Example:

Get supported protocols:

```
C0 30 61 01 00 03         (Set protocols; ReqID: 1; Payload: 0 byte)
00 C1
```

Response:

```
C0 30 61 01 00 01         (Response; ReqID: 1; Payload: 4 byte)
00                        (Acknowledgement)
02                        (2 protocols)
00                        (DIN70121-2:2012)
02                        (ISO15118-2:2014)
00 C1
```

17.10.3 Set SDP config

Set SDP config		
Sub-ID	0x62	
Description	This configures the SDP server parameters. The ports for unsecure and secure connections cannot be the same. The default ports for the SDP server are generated randomly in the range of dynamic ports 49152-65535 and can be overwritten by this command.	
Parameters		
Name	**Type**	**Description**
Allow unsecure	boolean	True if unsecure connections are allowed. The next parameter is only present if this is set to true.
Unsecure port	uint16 (49152-65535)	Listen port for unsecure connections. This parameter is only present if “Allow unsecure” parameter is set to true.
Allow secure	boolean	True if secure connections are allowed. The next parameter is only present if this is set to true.
Secure port	uint16 (49152-65535)	Listen port for secure connections. This parameter is only present if “Allow secure” parameter is set to true.
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5

Example:

Set SDP Configuration:

```
C0 30 62 02 00 04         (Set config; ReqID: 2; Payload: 4 byte)
01                        (Unsecure connections allowed)
C0 00                     (Port 49152)
00                        (Secure connections are not allowed)
00 C1
```

Response:

```
C0 30 62 02 00 01         (Response; ReqID: 2; Payload: 1 byte)
```

```
00                          (Acknowledgement)
00 C1
```

17.10.4 Get SDP config

Get SDP config		
Sub-ID	0x63	
Description	Retrieves the current configuration of the SDP server.	
Parameters		
Name	**Type**	**Description**
-	-	-
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5
Allow unsecure	boolean	True if unsecure connections are allowed. The next parameter “Unsecure port” is only present if this is set to true.
Unsecure port	uint16 (49152-65535)	Listen port for unsecure connections. This parameter is only present if “Allow unsecure” parameter is set to true.
Allow secure	boolean	True if secure connections are allowed. The next parameter “Secure port” is only present if this is set to true.
Secure port	uint16 (49152-65535)	Listen port for secure connections. This parameter is only present if “Allow secure” parameter is set to true.

Example:

Get SDP Configuration:

```
C0 30 63 03 00 00           (Get config; ReqID: 3; Payload: 0 byte)
00 C1
```

Response:

```
C0 30 63 03 00 05           (Response; ReqID: 3; Payload: 5 byte)
00                          (Acknowledgement)
01                          (Unsecure connections allowed)
C0 00                       (Port 49152)
00                          (Secure connections are not allowed)
00 C1
```

17.10.5 Set payment options

Set payment options	
Sub-ID	0x64
Description	Sets the available payment options.
Parameters	

Name	Type	Description
Authorization required	boolean	True if authorization is required. If this parameter is set to false the following parameter list needs to be omitted.
Count	uint8 (1-2)	Number of supported payment methods. The following parameter has to be repeated this many times.
Payment option	uint8 (0-1)	0: External payment 1: Contract payment (currently not supported)
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5

Example:

Set payment options:

```
C0 30 64 04 00 03          (Set options; ReqID: 4; Payload: 3 byte)
01                         (Authorization required)
01                         (1 method)
00                         (External Payment)
00 C1
```

Response:

```
C0 30 64 04 00 01          (Response; ReqID: 4; Payload: 1 byte)
00                         (Acknowledgement)
00 C1
```

17.10.6 Get payment options

Get payment options		
Sub-ID	0x65	
Description	Retrieves the currently set payment options.	
Parameters		
Name	**Type**	**Description**
-	-	-
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5
Authorization required	boolean	True if authorization is required. If this parameter is set to false the following parameter list is omitted.
Count	uint8 (1-2)	Number of supported payment methods. The following parameter is repeated this many times.
Payment option	uint8 (0-1)	0: External payment 1: Contract payment (currently not supported)

Example:

Get payment options:

```
C0 30 65 05 00 00          (Get options; ReqID: 5; Payload: 0 byte)
00 C1
```

Response:

```
C0 30 65 05 00 04          (Response; ReqID: 5; Payload: 4 byte)
00                         (Acknowledgement)
01                         (Authorization required)
01                         (1 method)
00                         (External Payment)
00 C1
```

17.10.7 Set energy transfer modes

Set energy transfer modes		
Sub-ID	0x66	
Description	Sets the available energy transfer modes.	
Parameters		
Name	**Type**	**Description**
Count	uint8 (1-4)	Number of energy transfer modes. The following parameter has to be repeated this many times.
Energy transfer mode	uint8 (0-5)	0: DC core, 1: DC extended 2: DC combo core 3: DC unique 4: AC single phase 5: AC three phase
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5

Example:

Set energy transfer mode:

```
C0 30 66 06 00 02          (Set modes; ReqID: 6; Payload: 2 byte)
01                         (1 energy transfer mode)
00                         (DC Core)
00 C1
```

Response:

```
C0 30 66 06 00 01          (Response; ReqID: 6; Payload: 1 byte)
00                         (Acknowledgement)
00 C1
```

17.10.8 Get energy transfer modes

Get energy transfer modes		
Sub-ID	0x67	
Description	Retrieves the currently set payment options.	
Parameters		
Name	**Type**	**Description**
-	-	-
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5
Count	uint8 (1-4)	Number of energy transfer modes. The following parameter is repeated this many times.
Energy transfer mode	uint8 (0-5)	0: DC core, 1: DC extended 2: DC combo core 3: DC unique 4: AC single phase 5: AC three phase

Example:

Get energy transfer mode:

```
C0 30 67 07 00 00          (Get modes; ReqID: 7; Payload: 0 byte)
00 C1
```

Response:

```
C0 30 67 07 00 03          (Response; ReqID: 7; Payload: 3 byte)
00                         (Acknowledgement)
01                         (1 energy transfer mode)
00                         (DC Core)
00 C1
```

17.10.9 Set EVSE ID

Set EVSE ID		
Sub-ID	0x68	
Description	This command has to be used to set the EVSI ID after it was requested by the status message Request EVSE ID.	
Parameters		
Name	**Type**	**Description**
Code	uint8	Result code: 0: Success 1: No EVSE ID available
EVSE ID	string[0-38]	EVSE ID in the requested format. Only present if Code is set to 0: Success.
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5

Example:

Set EVSE ID:

```
C0 30 68 08 00 0B           (Set EVSE-ID; ReqID: 8; Payload: 11 byte)
00                          (Success)
31 32 33 34 35 36 37 38 39 30     (ID = 1234567890)
00 C1
```

Response:

```
C0 30 68 08 00 01           (Response; ReqID: 8; Payload: 1 byte)
00                          (Acknowledgement)
00 C1
```

17.10.10 Set Authorization Status

Set Authorization Status		
Sub-ID	0x69	
Description	This command has to be used to set the authorization status after it was requested by the status message Request Authorization Status.	
Parameters		
Name	**Type**	**Description**
Code	uint8	Result code: 0: Authorization succeeded 1: Authorization failed
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5

Example:

Set Authorization Status:

```
C0 30 69 09 00 01        (Set Auth-Status; ReqID: 9; Payload: 1 byte)
00                       (Authorization succeeded)
00 C1
```

Response:

```
C0 30 69 09 00 01        (Response; ReqID: 9; Payload: 1 byte)
00                       (Acknowledgement)
00 C1
```

17.10.11 Set Discovery Charge Parameters

Set Discovery Charge Parameters		
Sub-ID	0x6A	
Description	This command has to be used to set the discovery charge parameters after they were requested by the status message Request Discovery Charge Parameters.	
Parameters		
Name	**Type**	**Description**
Code	uint8	Result code: 0: EV parameters accepted 1: EV parameters invalid 2: Unable to deliver EVSE parameters
Type	uint8	The type of the requested discovery charge parameters. 0: DC 1: AC
DC		
Isolation level	uint8 (0-4)	The present isolation level: 0: Invalid, 1: Valid, 2: Warning, 3: Fault, 4: No IMD
Max. current	exponential	Maximum current supported by the EVSE.
Min. current	exponential	Minimum current supported by the EVSE.
Max. voltage	exponential	Maximum voltage supported by the EVSE.
Min. voltage	exponential	Minimum voltage supported by the EVSE.
Max. power	exponential	Maximum power supported by the EVSE.
Current regul. tolerance present	boolean	Whether or not the next parameter “Current regul. tolerance” is present.
Current regul. tolerance	exponential	Optional: Absolute magnitude of the regulation tolerance.
Peak current ripple	exponential	Peak-to-peak magnitude of the current ripple.
Energy to be delivered present	boolean	Whether or not the next parameter “Energy to be delivered” is present.
Energy to be	exponential	Optional: Amount of energy to be delivered.

delivered		
AC		
RCD status	boolean	Indicates the current status of the Residual Current Device (RCD). If RCD is equal to true, the RCD has detected an error. If RCD is equal to false, the RCD has not detected an error. This status flag is for informational purpose only.
Nominal voltage	exponential	Line voltage supported by the EVSE. This is the voltage measured between one phases and neutral. If the EVSE supports multiple phase charging the EV might easily calculate the voltage between phases. This parameter is also used as reference for calculating the corresponding maximum charging current out of the PMax values in the SASchedule entities.
Max. current	exponential	Maximum allowed line current restriction set by the EVSE per phase. If the PWM ratio is set to 5% ratio then this is the only line current restriction processed by the EVCC. Otherwise the EVCC applies the smaller current constraint from the EVSEMaxCurrent value and the PWM ratio information.
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5

Example:

Set Discovery Charge Parameters:

```
C0 30 6A 0A 00 09          (Set Disc-Param.; ReqID: 10; Payload: 9 byte)
00                         (EV parameters accepted)
01                         (AC)
00                         (RCD - no error detected)
01 90 00                   (400 V)
00 0C 00                   (12 A)
00 C1
```

Response:

```
C0 30 6A 0A 00 01          (Response; ReqID: 10; Payload: 1 byte)
00                         (Acknowledgement)
00 C1
```

17.10.12 Set Schedules

Set Schedules		
Sub-ID	0x6B	
Description	This command has to be used to set the schedules after they were requested by the status message Request Schedules.	
Parameters		
Name	**Type**	**Description**
Code	uint8	Result code: 0: OK 1: Unable to deliver schedules
Time anchor	uint64	UTC time of the start of the first schedule.
Count	uint16	Number of schedule entries in the following list. The following parameters have to be repeated this many times.
ID	uint16	ID of the schedule.
Interval	uint16	Interval in seconds where the schedule is valid.
Power	exponential	The power available in this time interval.
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5

Example:

Set Schedules:

```
C0 30 6B 0B 00 11          (Set Schedules; ReqID: 11; Payload: 17 byte)
00                         (ok)
00 01 02 03 04 05 06 07    (UTC Time)
01                         (1 entry)
00 01                      (Schedule ID: 1)
0E 10                      (3600 Seconds)
00 0C 03                   (12 KW)
00 C1
```

Response:

```
C0 30 6B 0B 00 01          (Response; ReqID: 11; Payload: 1 byte)
00                         (Acknowledgement)
00 C1
```

17.10.13 Set Cable Check Status

Set Cable Check Status	
Sub-ID	0x6C
Description	This command has to be used to set the cable check status after it was requested by the status message Request Cable Check Status.
Parameters	

Name	Type	Description
Code	uint8	Result code: 0: Cable check succeeded 1: Cable check failed
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5

Example:

Set Cable Check Status:

```
C0 30 6C 0C 00 01          (Set Cable-Check-Status; ReqID: 12; Payload: 1 byte)
00                         (Cable check succeeded)
00 C1
```

Response:

```
C0 30 6C 0C 00 01          (Response; ReqID: 12; Payload: 1 byte)
00                         (Acknowledgement)
00 C1
```

17.10.14 Set Cable Check Parameters

Set Cable Check Parameters		
Sub-ID	0x6D	
Description	This command has to be used to set the cable check parameters after they were requested by the status message Request Cable Check Parameters.	
Parameters		
Name	**Type**	**Description**
Code	uint8	Result code: 0: EV parameters accepted 1: EV parameters invalid 2: Unable to deliver EVSE parameters
Type	uint8	The type of the requested cable check parameters. 0: DC
DC		
Isolation level	uint8 (0-4)	The present isolation level: 0: Invalid, 1: Valid, 2: Warning, 3: Fault, 4: No IMD
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5

Example:

Set Cable Check Parameters:

```
C0 30 6D 0D 00 03          (Set Cable-Check-Param; ReqID: 13; Payload: 3 byte)
00                         (EV parameters accepted)
00                         (DC)
02                         (Warning)
00 C1
```

Response:

```
C0 30 6D 0D 00 01          (Response; ReqID: 13; Payload: 1 byte)
00                         (Acknowledgement)
00 C1
```

17.10.15 Set Pre Charge Parameters

Set Pre Charge Parameters		
Sub-ID	0x6E	
Description	This command has to be used to set the pre charge parameters after they were requested by the status message Request Pre Charge Parameters.	
	Parameters	
Name	**Type**	**Description**
Code	uint8	Result code: 0: EV parameters accepted 1: EV parameters invalid 2: Unable to deliver EVSE parameters
Type	uint8	The type of the requested pre charge parameters. 0: DC
	DC	
Isolation level	uint8 (0-4)	The present isolation level: 0: Invalid, 1: Valid, 2: Warning, 3: Fault, 4: No IMD
Present voltage	exponential	Output voltage of the EVSE.
	Returned Result	
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5

Example:

Set Pre Charge Parameters:

```
C0 30 6E 0E 00 03          (Set Pre Charge Param; ReqID: 14; Payload: 6 byte)
00                         (EV parameters accepted)
00                         (DC)
02                         (Warning)
00 04 02                   (400V)
```

```
00 C1
```

Response:

```
C0 30 6E 0E 00 01           (Response; ReqID: 14; Payload: 1 byte)
00                          (Acknowledgement)
00 C1
```

17.10.16 Set Start Charging Status

Set Start Charging Status		
Sub-ID	0x6F	
Description	This command has to be used to set the start charging status after it was requested by the status message Request Start Charging.	
Parameters		
Name	**Type**	**Description**
Code	uint8	Result code: 0: Charging started 1: Charging could not be started
DC		
Isolation level	uint8 (0-4)	The present isolation level: 0: Invalid, 1: Valid, 2: Warning, 3: Fault, 4: No IMD
AC		
RCD status	boolean	Indicates the current status of the Residual Current Device (RCD). If RCD is equal to true, the RCD has detected an error. If RCD is equal to false, the RCD has not detected an error. This status flag is for informational purpose only.
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5

Example:

Set Start Charging Status:

```
C0 30 6F 0F 00 02           (Set Start Status; ReqID: 15; Payload: 2 byte)
00                          (Charging started)
01                          (Valid)
00 C1
```

Response:

```
C0 30 6F 0F 00 01           (Response; ReqID: 15; Payload: 1 byte)
00                          (Acknowledgement)
00 C1
```

17.10.17 Set Charge Loop Parameters

Set Charge Loop Parameters		
Sub-ID	0x70	
Description	This command has to be used to set the charge loop parameters after they were requested by the status message Request Charge Loop Parameters.	
Parameters		
Name	**Type**	**Description**
Code	uint8	Result code: 0: EV parameters accepted 1: EV parameters invalid 2: Unable to deliver EVSE parameters
Type	uint8	The type of the requested charge loop parameters. 0: DC 1: AC
DC		
Isolation level	uint8 (0-4)	The present isolation level: 0: Invalid, 1: Valid, 2: Warning, 3: Fault, 4: No IMD
Present voltage	exponential	Output voltage of the EVSE.
Present current	exponential	Output current of the EVSE.
Max. current present	boolean	Whether or not the next parameter “Max. current” is present.
Max. current	exponential	(Optional) Maximum current supported by the EVSE.
Max. voltage present	boolean	Whether or not the next parameter “Max. voltage” is present.
Max. voltage	exponential	(Optional) Maximum voltage supported by the EVSE.
Max. power present	boolean	Whether or not the next parameter “Max. power” is present.
Max. power	exponential	(Optional) Maximum power supported by the EVSE.
Max. current limit reached	boolean	If set to true, the current limit is reached.
Max. voltage limit reached	boolean	If set to true, the voltage limit is reached.
Max. power limit reached	boolean	If set to true, the power limit is reached.
AC		
Max. current present	boolean	Whether or not the next parameter “Max. current” is present.
Max. current	exponential	Optional: This element is used by the SECC to indicate the maximum line current per phase the EV can draw.
RCD status	boolean	Indicates the current status of the Residual Current Device (RCD). If RCD is equal to true, the RCD has detected an error. If RCD is equal to false, the RCD has not detected an error. This status flag is for informational purpose only.
Returned Result		
Name	**Type**	**Description**

Code	uint8	Generic result code; see section 10.1.5

Example:

Set Start Charge Loop Parameters:

```
C0 30 70 10 00 07          (Set Start Loop Param; ReqID: 16; Payload: 7 byte)
00                         (EV parameters accepted)
01                         (AC)
01                         (Max Current present)
00 0C 00                   (12 A)
00                         (RCD has not detected an error)
00 C1
```

Response:

```
C0 30 70 10 00 01          (Response; ReqID: 16; Payload: 1 byte)
00                         (Acknowledgement)
00 C1
```

17.10.18 Set Stop Charging Status

Set Stop Charging Status		
Sub-ID	0x71	
Description	This command has to be used to set the stop charging status after it was requested by the status message Request Stop Charging.	
	Parameters	
Name	**Type**	**Description**
Code	uint8	Result code: 0: Charging stopped 1: Charging could not be stopped
	DC	
Isolation level	uint8 (0-4)	The present isolation level: 0: Invalid, 1: Valid, 2: Warning, 3: Fault, 4: No IMD
	AC	
RCD status	boolean	Indicates the current status of the Residual Current Device (RCD). If RCD is equal to true, the RCD has detected an error. If RCD is equal to false, the RCD has not detected an error. This status flag is for informational purpose only.
	Returned Result	
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5

Example:

Set Stop Charging Status:

```
C0 30 71 11 00 03          (Set Stop Status; ReqID: 17; Payload: 3 byte)
```

```
00                          (Charging stopped)
01                          (DC)
01                          (Valid)
00 C1
```

Response:

```
C0 30 71 11 00 01           (Response; ReqID: 17; Payload: 1 byte)
00                          (Acknowledgement)
00 C1
```

17.10.19 Set Post Charge Parameters

Set Post Charge Parameters		
Sub-ID	0x72	
Description	This command has to be used to set the post charge parameters after they were requested by the status message Request Post Charge Parameters.	
Parameters		
Name	**Type**	**Description**
Code	uint8	Result code: 0: EV parameters accepted 1: EV parameters invalid 2: Unable to deliver EVSE parameters
Type	uint8	The type of the requested post charge parameters. 0: DC
DC		
Isolation level	uint8 (0-4)	The present isolation level: 0: Invalid, 1: Valid, 2: Warning, 3: Fault, 4: No IMD
Present voltage	exponential	Output voltage of the EVSE.
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5

Example:

Set Post Charge Parameters:

```
C0 30 72 12 00 06           (Set Stop Status; ReqID: 18; Payload: 6 byte)
00                          (EV parameters accepted)
01                          (DC)
01                          (Valid)
01 90 00                    (400 V)
00 C1
```

Response:

```
C0 30 72 12 00 01           (Response; ReqID: 18; Payload: 1 byte)
00                          (Acknowledgement)
```

```
00 C1
```

17.11 EV Status messages

EV mode currently not supported.

17.12 EVSE Status messages

This chapter describes the status messages sent by the service.

17.12.1 Session started

Session started		
Sub-ID	0x80	
Description	This status message is sent if a new session is started.	
Parameters		
Name	**Type**	**Description**
Protocol	uint8 (0-3)	0: DIN70121-2:2012 1: ISO15118-2:2010 2: ISO15118-2:2014 3: ISO15118-20:2020
Session ID	uint8[8]	The session ID that is used for the current session
EVCC ID	uint8[6-20]	The ID of the EVCC.

Example:

Status message if a new session was started:

```
C0 28 80 FF 00 00          (started; ReqID: 255; Payload: 0 byte)
00                         (DIN)
01 02 03 04 05 06 07       (Session ID)
30 31 32 33 34 35          (EVCC ID)
00 C1
```

17.12.2 Session stopped

Session stopped		
Sub-ID	0x81	
Description	This status message is sent if a session is stopped.	
Parameters		
Name	**Type**	**Description**
Result	boolean	True if successful, false otherwise.

Example:

Status message if a new session was staopped:

```
C0 28 81 FF 00 00           (stopped; ReqID: 255; Payload: 0 byte)
01                          (TRUE)
00 C1
```

17.12.3 Request EVSE ID

Request EVSE ID		
Sub-ID	0x82	
Description	This status message is sent if the EVSE ID needs to be sent to the EV.	
Parameters		
Name	**Type**	**Description**
Timeout	uint32	Timeout in milliseconds until the EVSE ID needs to be delivered. If this timeout is exceeded the communication session will be closed.
Format	uint8 (0-1)	This is the requested format (<Country Code> <S> <EVSE Operator ID> <S> <ID Type> <Power Outlet ID>) of the EVSE ID: 0: i.e. “+49*123*456*789” (max length: 32) 1: i.e. “DE*A23*E45B*78C” (max. length: 38)

Example:

Status message for requesting EVSE ID:

```
C0 28 82 FF 00 00           (started; ReqID: 255; Payload: 0 byte)
00 00 01 F4                 (Timeout 500 ms)
01                          (Format 1)
00 C1
```

17.12.4 Request Authorization Status

Request Authorization Status		
Sub-ID	0x83	
Description	This status message is sent if external payment was selected and the EV requested the authorization.	
Parameters		
Name	**Type**	**Description**
Timeout	uint32	Timeout in milliseconds until an authorization has to be performed. If this timeout is exceeded the communication session will be closed.

Example:

Status message for requesting Authorization Status:

```
C0 28 83 FF 00 00           (Req. Auth. Status; ReqID: 255; Payload: 0 byte)
00 00 01 F4                 (Timeout 500 ms)
00 C1
```

17.12.5 Request Discovery Charge Parameters

Request Discovery Charge Parameters		
Sub-ID	0x84	
Description	This status message is sent when the EV transmitted its discovery charge parameters and the EVSE discovery charge parameters need to be sent.	
Parameters		
Name	**Type**	**Description**
Timeout	uint32	Timeout in milliseconds. If this timeout is exceeded the communication session will be closed.
Type	uint8	The type of the discovery charge parameters. See the following tables for the specific parameters of the selected type. 0: DC 1: AC
DC		
Name	**Type**	**Description**
Max. current	exponential	Maximum current supported by the EV.
Min. current present	boolean	Whether or not the next parameter “Min. current” is present.
Min. current	exponential	(Optional) Minimum current supported by the EV.
Max. power present	boolean	Whether or not the next parameter “Max. power” is present.
Max. power	exponential	(Optional) Maximum power supported by the EV.
Min. power present	boolean	Whether or not the next parameter “Min. power” is present.
Min. power	exponential	(Optional) Minimum power supported by the EV.
Max. voltage	exponential	Maximum voltage supported by the EV.
Min. voltage present	boolean	Whether or not the next parameter “Min. voltage” is present.
Min. voltage	exponential	(Optional) Minimum voltage supported by the EV.
Full SOC present	boolean	Whether or not the next parameter “Full SOC” is present.
Full SOC	uint8 (0-100)	SOC at which the EV considers the battery to be fully charged.
Bulk SOC present	boolean	Whether or not the next parameter “Bulk SOC” is present.
Bulk SOC	uint8 (0-100)	SOC at which the EV considers a fast charge process to end.
SOC	uint8 (0-100)	State of charge of the EV’s battery.
AC		
Energy amount	exponential	Amount of energy reflecting the EV's estimate how much energy is needed to fulfill the user configured charging goal for the current charging session. This might include energy for other purposes than solely charging the HV battery of an EV.
Max. voltage	exponential	The RMS of the maximal nominal voltage the vehicle can accept, measured between one phase and neutral.

Max. current	exponential	Maximum current supported by the EV per phase.
Min. current	exponential	EVMinCurrent is used to indicate to the SECC that charging below this minimum is not energy/cost efficient for the EV. It is recommended that the SECC considers this value during the target setting process (e.g. sale tariff table should account for this value). However, if there is physical limitations or limitations indicated by the PWM signal these limitations overwrite the EVMinCurrent the EV indicated. It is implementation specific whether a vehicle chooses not charge if the EVMinCurrent is higher than the physical limitations for efficiency reasons.

Example:

Status message for requesting dicovery charge parameters:

```
C0 28 84 FF 00 00          (Req. Disc. Chrg Para; ReqID: 255; Payload: 0 byte)
00 00 01 F4                (Timeout 500 ms)
01                         (AC)
2E E0 00                   (Energy Amount)
01 90 00                   (Max voltage 400 V)
00 0C 00                   (Max current 12 A)
00 01 00                   (Min current 1 A)
00 C1
```

17.12.6 Request Schedules

Request Schedules		
Sub-ID	0x85	
Description	This status message is sent when the EVSE needs to send its schedules.	
Parameters		
Name	**Type**	**Description**
Timeout	uint32	Timeout in milliseconds. If this timeout is exceeded the communication session will be closed.
Max entries	uint16	Maximum number of entries in the schedule list. The host has to limit the list of schedules to this many entries.
Timestamp	uint64	Current UTC timestamp in ms.

Example:

Status message for requesting Shedules:

```
C0 28 85 FF 00 00          (started; ReqID: 255; Payload: 0 byte)
00 00 01 F4                (Timeout 500 ms)
00 05                      (Max. 5 entries)
00 00 01 78 34 D1 8A 78    (1615793851000 ms)
00 C1
```

17.12.7 Request Cable Check Status

Request Cable Check Status		
Sub-ID	0x86	
Description	This status message is sent when the EV requested the cable check to be started.	
Parameters		
Name	**Type**	**Description**
Timeout	uint32	Timeout in milliseconds until the cable check has to be finished. If this timeout is exceeded the communication session will be closed.

Example:

Status message for requesting Cable Check Status:

```
C0 28 86 FF 00 00          (started; ReqID: 255; Payload: 0 byte)
00 00 01 F4                (Timeout 500 ms)
00 C1
```

17.12.8 Request Cable Check Parameters

Request Cable Check Parameters		
Sub-ID	0x87	
Description	This status message is sent when the EV transmitted its cable check parameters and the EVSE cable check parameters need to be sent.	
Parameters		
Name	**Type**	**Description**
Timeout	uint32	Timeout in milliseconds. If this timeout is exceeded the communication session will be closed.
Type	uint8	The type of the cable check parameters. See the following tables for the specific parameters of the selected type. 0: DC
DC		
Name	**Type**	**Description**
SOC	uint8 (0-100)	State of charge of the EV’s battery.

Example:

Status message for requesting Cable Check Parameters:

```
C0 28 87 FF 00 00          (Req. Cable Check Param; ReqID: 255; Payload: 0 byte)
00 00 01 F4                (Timeout 500 ms)
00                         (Type DC)
00                         (State 0)
00 C1
```

17.12.9 Request Pre Charge Parameters

Request Pre Charge Parameters		
Sub-ID	0x88	
Description	This status message is sent when the EV transmitted its pre charge parameters and the EVSE pre charge parameters need to be sent.	
Parameters		
Name	**Type**	**Description**
Timeout	uint32	Timeout in milliseconds. If this timeout is exceeded the communication session will be closed.
Type	uint8	The type of the pre charge parameters. See the following tables for the specific parameters of the selected type. 0: DC
DC		
Name	**Type**	**Description**
Target voltage	exponential	Target voltage requested by the EV.
Target current	exponential	Target current requested by the EV.
SOC	uint8 (0-100)	State of charge of the EV’s battery.

Example:

Status message for requesting Cable Check Parameters:

```
C0 28 88 FF 00 00          (Req. Pre. Chrg Param; ReqID: 255; Payload: 0 byte)
00 00 01 F4                (Timeout 500 ms)
00                         (Type DC)
01 90 00                   (Target voltage 400 V)
00 0C 00                   (Target current 12 A)
00                         (State 0)
00 C1
```

17.12.10 Request Start Charging

Request Start Charging		
Sub-ID	0x89	
Description	This request message is sent when the EV requested to start charging.	
Parameters		
Name	**Type**	**Description**
Timeout	uint32	Timeout in milliseconds until the charging has to be started. If this timeout is exceeded the communication session will be closed.
Schedule ID	uint8	The schedule ID selected by the EV.
Time anchor	uint64	UTC time of the start of the first EV power profile.
Count	uint16	Number of EV power profile entries in the following list. The following parameters Interval and Power have to be repeated this many times.
Interval	uint16	Interval in seconds of this EV power profile entry.
Power	exponential	Power used in this EV power profile entry.
Type	uint8	The type of the requested charge loop parameters. 0: DC 1: AC
DC		
Name	**Type**	**Description**
SOC present	boolean	Whether or not the next parameter “SOC” is present.
SOC	uint8 (0-100)	(Optional) State of charge of the EV’s battery.
Charging complete present	boolean	Whether or not the next parameter “Charging complete” is present.
Charging complete	boolean	(Optional) If true the EV indicates that charging is completed.
Bulk charging complete present	boolean	Whether or not the next parameter “Bulk charging complete” is present.
Bulk charging complete	boolean	(Optional) If true the EV indicates that bulk charging is completed.
AC		
-	-	empty

Example:

Status message for requesting start charging:

```
C0 28 89 FF 00 00          (Req. Start Charging; ReqID: 255; Payload: 0 byte)
00 00 01 F4                (Timeout 500 ms)
00                         (Shedule ID 0)
00                         (State 0)
00 00 01 78 34 D1 8A 78    (1615793851000 ms)
00 02                      (0 EV power profiles)
```

```
2E E0 00                              (Power)
01                            (AC)
00 C1
```

17.12.11 Request Charge Loop Parameters

Request Charge Loop Parameters		
Sub-ID	0x8A	
Description	This status message is sent when the EV transmitted its charge loop parameters and the EVSE charge loop parameters need to be sent.	
Parameters		
Name	**Type**	**Description**
Timeout	uint32	Timeout in milliseconds. If this timeout is exceeded the communication session will be closed.
Type	uint8	The type of the charge loop parameters. See the following tables for the specific parameters of the selected type. 0: DC 1: AC
DC		
Name	**Type**	**Description**
Max. current present	boolean	Whether or not the next parameter “Max. current” is present.
Max. current	exponential	(Optional) Maximum current supported by the EV.
Max. voltage present	boolean	Whether or not the next parameter “Max. voltage” is present.
Max. voltage	exponential	(Optional) Maximum voltage supported by the EV.
Max. power present	boolean	Whether or not the next parameter “Max. power” is present.
Max. power	exponential	(Optional) Maximum power supported by the EV.
Target voltage	exponential	Target voltage requested by the EV.
Target current	exponential	Target current requested by the EV.
SOC	uint8 (0-100)	State of charge of the EV’s battery.
Charging complete	boolean	If true the EV indicates that charging is completed.
Bulk charging complete present	boolean	Whether or not the next parameter “Bulk charging complete” is present.
Bulk charging complete	boolean	(Optional) If true the EV indicates that bulk charging is completed.
Remaining time to full SOC present	boolean	Whether or not the next parameter “Remaining time to full SOC” is present.
Remaining time to full SOC	exponential	(Optional) Estimated or calculated time until charging is completed.
Remaining time to bulk SOC	boolean	Whether or not the next parameter “Remaining time to bulk SOC” is present.

present		
Remaining time to bulk SOC	exponential	(Optional) Estimated or calculated time until bulk charging is completed.
AC		
-	-	empty

Example:

Status message for requesting charge loop parameters:

```
C0 28 8A FF 00 00          (Req. Chrg Loop Param; ReqID: 255; Payload: 0 byte)
00 00 01 F4                (Timeout 500 ms)
01                         (AC)
00 C1
```

17.12.12 Request Stop Charging

Request Stop Charging		
Sub-ID	0x8B	
Description	This request message is sent when the EV requested to stop charging.	
Parameters		
Name	**Type**	**Description**
Timeout	uint32	Timeout in milliseconds until the charging has to be started. If this timeout is exceeded the communication session will be closed.
Type	uint8	The type of the requested charge loop parameters. 0: DC 1: AC
DC		
Name	**Type**	**Description**
SOC present	boolean	Whether or not the next parameter “SOC” is present.
SOC	uint8 (0-100)	(Optional) State of charge of the EV’s battery.
Charging complete present	boolean	Whether or not the next parameter “Charging complete” is present.
Charging complete	boolean	(Optional) If true the EV indicates that charging is completed.
Bulk charging complete present	boolean	Whether or not the next parameter “Bulk charging complete” is present.
Bulk charging complete	boolean	(Optional) If true the EV indicates that bulk charging is completed.
AC		
-	-	empty

Example:

Status message for requesting stop charging:

```
C0 28 8B FF 00 00          (Req. Chrg Loop Param; ReqID: 255; Payload: 0 byte)
00 00 01 F4                (Timeout 500 ms)
01                         (AC)
00 C1
```

17.12.13 Request Post Charge Parameters

Request Post Charge Parameters		
Sub-ID	0x8C	
Description	This status message is sent when the EV transmitted its post charge parameters and the EVSE post charge parameters need to be sent.	
Parameters		
Name	**Type**	**Description**
Timeout	uint32	Timeout in milliseconds. If this timeout is exceeded the communication session will be closed.
Type	uint8	The type of the post charge parameters. See the following tables for the specific parameters of the selected type. 0: DC
DC		
Name	**Type**	**Description**
SOC	uint8 (0-100)	State of charge of the EV’s battery.

Example:

Status message for requesting post charge parameters:

```
C0 28 8C FF 00 00          (Req. Post Chrg Param; ReqID: 255; Payload: 0 byte)
00 00 01 F4                (Timeout 500 ms)
00                         (DC)
01                         (State 1)
00 C1
```

18 GPIO Module

The GPIO module can be used to configure pins of the module. Furthermore the state can be set and read.

The GPIO module has the Module-ID 0x0F.

18.1 Sub-IDs used by the GPIO Module

The GPIO module uses the following Sub-IDs:

Generic Sub-IDs		
Sub-ID	**Description**	**Section**
-	-	-
Configuration Commands		
Sub-ID	**Description**	**Section**
0x40	Set Mode	11.3.1
0x41	Get Mode	11.3.2
0x42	Set State	18.2.3
0x43	Get State	18.2.4

18.2 Configuration Commands

18.2.1 Set Mode

Set Mode		
Sub-ID	0x40	
Description	Set GPIO mode to input or output.	
Parameters		
Name	**Type**	**Description**
GPIO	uint8	GPIO number
Mode	uint8	Mode: 0: Output 1: Input
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5

Example:

Set GPIO Mode:

```
C0 0F 40 01 00 02                 (Set GPIO mode; ReqID: 1; Payload: 2 bytes)
14                                (GPIO number 20)
00                                (output)
00 C1
```

Response:

```
C0 0F 40 01 00 01                 (Response; ReqID: 1; Payload: 1 byte)
00                                (Acknowledgement)
00 C1
```

18.2.2 Get mode

Get Mode		
Sub-ID	0x41	
Description	Get GPIO mode to input or output.	
Parameters		
Name	**Type**	**Description**
GPIO	uint8	GPIO number
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5
Mode	uint8	Mode: 0: Output 1: Input

Example:

Get GPIO Mode:

```
C0 0F 41 02 00 00                 (Get GPIO mode; ReqID: 2; Payload: 1 byte)
15                                (GPIO pin 21)
00 C1
```

Response:

```
C0 0F 41 02 00 02                 (Response; ReqID: 1; Payload: 2 bytes)
00                                (Acknowledgement)
01                                (input)
00 C1
```

18.2.3 Set State

Set State		
Sub-ID	0x42	
Description	Set GPIO state	
	Parameters	
Name	**Type**	**Description**
GPIO	uint8	GPIO number
State	uint8	State: 0: Off 1: On
	Returned Result	
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5

Example:

Set GPIO state:

```
C0 0F 42 03 00 02                  (Set GPIO state; ReqID: 3; Payload: 2 byte)
14                                 (GPIO 20)
01                                 (on)
00 C1
```

Response:

```
C0 0F 42 03 00 01                  (Response; ReqID: 3; Payload: 1 byte)
00                                 (Acknowledgement)
00 C1
```

18.2.4 Get State

Get State		
Sub-ID	0x43	
Description	Get GPIO state	
Parameters		
Name	**Type**	**Description**
GPIO	uint8	GPIO number
Returned Result		
Name	**Type**	**Description**
Code	uint8	Generic result code; see section 10.1.5
State	uint8	State: 0: Off 1: On

Example:

Get GPIO state

```
C0 0F 43 04 00 00                 (Get State; ReqID: 4; Payload: 1 byte)
14                                (GPIO pin 20)
00 C1
```

Response:

```
C0 0F 43 04 00 0A                 (Response; ReqID: 4; Payload: 2 bytes)
00                                (Acknowledgement)
01                                (on)
00 C1
```

19 Application Examples

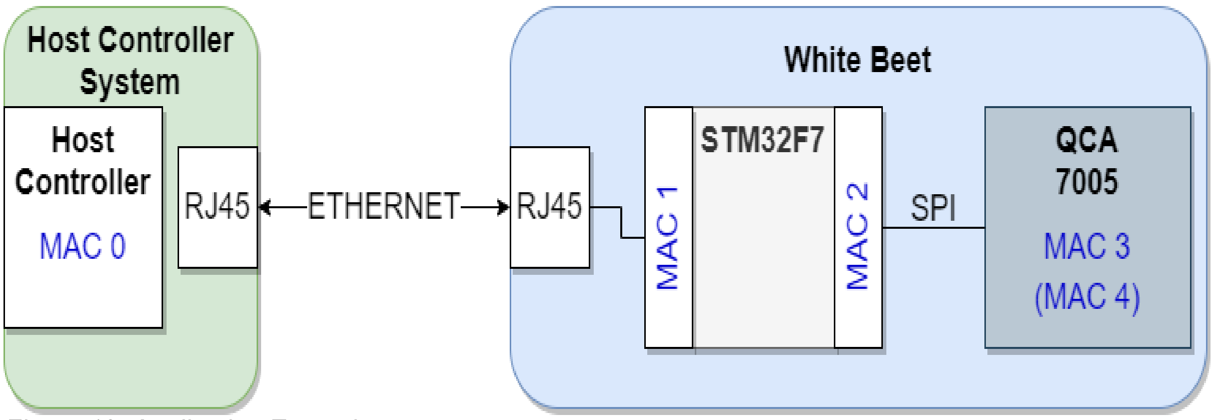


Figure 19: Application Example

19.1 Example configuration

Used parameters for all examples as follows:

19.1.1 EVSE

Table 37: Example Configuration EVSE

Parameter	***Description***	***Value***
MAC 0	MAC address of the host controller system (HLE). Note: Example MAC address depends from the host controller!	For the following examples the following MAC is used: 00:22:22:22:22:22
MAC 1	Ethernet interface MAC address of the WHITE beet module (EVSE). Note: Must be replaced by own MAC address.	For the following examples the following MAC is used: 00:01:01:63:77:33
MAC 2	SPI-Interface MAC address of the WHITE beet module (EVSE) for communication with QCA700x controller. Note: Must be replaced by own MAC address.	For the following examples the following MAC is used: 00:01:01:63:77:32
MAC 3	Individual MAC address of the QCA7005.	Individual MAC address depends from configured PIB file.

MAC 4	Default MAC address of the QCA7005.	The default MAC address of the QCA7005 is: 00:B0:52:00:00:01

19.1.2 EV

Table 38: Example Configuration EV

Parameter	***Description***	***Value***
MAC 0	MAC address of the host controller system (HLE). Note: Example MAC address depends from the host controller!	For the following examples the following MAC is used: 00:44:44:44:44:44
MAC 1	Ethernet interface MAC address of the WHITE beet module (EVSE). Note: Must be replaced by own MAC address.	For the following examples the following MAC is used: 00:01:01:63:77:31
MAC 2	SPI-Interface MAC address of the WHITE beet module (EVSE) for communication with QCA700x controller. Note: Must be replaced by own MAC address.	For the following examples the following MAC is used: 00:01:01:63:77:30
MAC 3	Individual MAC address of the QCA7005.	Individual MAC address depends from configured PIB file.
MAC 4	Default MAC address of the QCA7005.	The default MAC address of the QCA7005 is: 00:B0:52:00:00:01

19.2 Control Pilot (CP) Service

19.2.1 Interaction Diagram (EVSE)

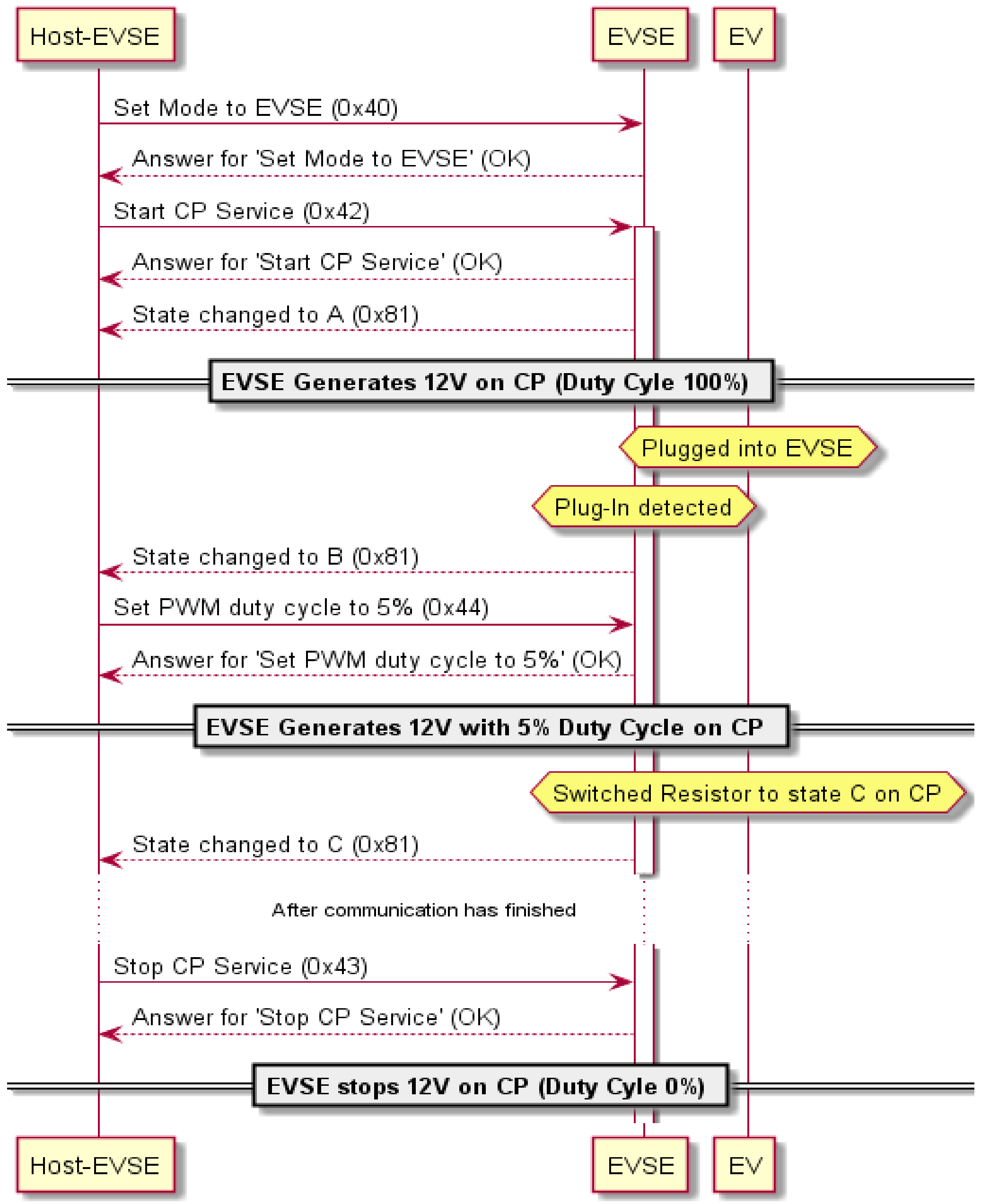

Host-EVSE
EVSE
EV
Set Mode to EVSE (0x40)
Answer for 'Set Mode to EVSE' (OK)
Start CP Service (0x42)
Answer for 'Start CP Service' (OK)
State changed to A (0x81)
EVSE Generates 12V on CP (Duty Cyle 100%)
Plugged into EVSE
Plug-In detected
State changed to B (0x81)
Set PWM duty cycle to 5% (0x44)
Answer for 'Set PWM duty cycle to 5%' (OK)
EVSE Generates 12V with 5% Duty Cycle on CP
Switched Resistor to state C on CP
State changed to C (0x81)
After communication has finished
Stop CP Service (0x43)
Answer for 'Stop CP Service' (OK)
EVSE stops 12V on CP (Duty Cyle 0%)
Host-EVSE
EVSE
EV

Figure 20: EVSE: Controlling the Control Pilot

19.2.2 Process description

After Module was initialized and started successful, the host controller can send the Set-Mode-command to the EVSE WHITE beet module to configure the device as EVSE. In EVSE mode the WHITE beet module will interact as a Voltage source on Control Pilot. To enable the voltage on CP the host controller has to send the Start-command to EVSE. This will enable the ADC for state detection and forces the EVSE to generate a 12V voltage on the CP line. If no EV is connected the ADC of the EVSE will detect state A and the Control Pilot Service will send a State-Changed info message to the host.

If a vehicle is now connected to the EVSE, the internal resistor of the EV will reduce the voltage on CP so that the ADC on EVSE side will detect state B. The Control Pilot Service will send again a ***State-Changed*** info message to the host. Now the host controller should send a ***Set PWM duty cycle*** command to the EVSE to set the duty cycle to 5% if High Level Communication is intended.

After the loading process has been completed or if an error has occurred, the module can be stopped using the ***Stop*** command.

The commands of the CP service are described in chapter 14.

19.3 EVSE: Successful SLAC matching process (EVSE was selected by EV)

19.3.1 Interaction Diagram

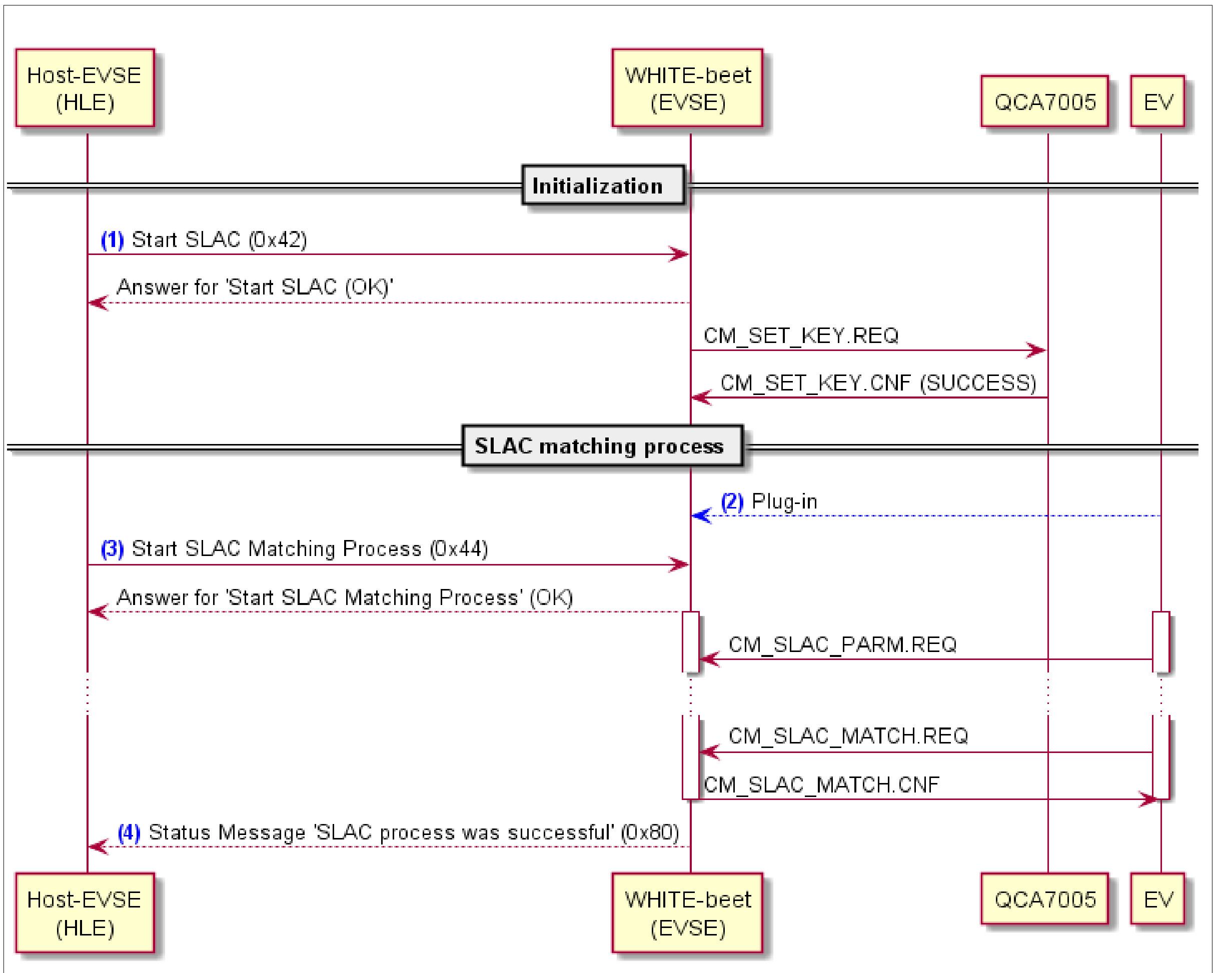


Figure 21: EVSE: Successful SLAC matching process

19.3.2 Process description

(1) The HLE sends a control frame to the module with the command ‘Start SLAC’ and the role (EVSE) for the SLAC process. The module will then initiate joining a random HPGP logical network which is communicated to the remote station in the event of a successful SLAC process. This network will be used for the later communication.

Ethernet Frame:

00 01 01 63 77 33 00 22 22 22 22 22 60 03 00 04 00 09 C0 28 42 00 00 01 01 00 C1

Table 39: EVSE: Start SLAC

Parameter	Value
Destination MAC Address	00:01:01:63:77:33
Source MAC Address	00:22:22:22:22:22
Ethernet Type	60 03
Version	00
Message Type	04
Size	00 09
Framing payload	C0 28 42 00 00 01 01 00 C1 => 15.4.1

(2) The charging station recognizes that a vehicle has connected to it’s logical network.

(3) The HLE sends a control frame to the module with the command ‘Start SLAC Matching Process’ to start the SLAC process. The EVSE starts timer for timeout detection and gets ready for receiving SLAC messages. The WHITE beet – module performs the SLAC process as described in the ISO 15118-3.

Ethernet Frame:

00 01 01 63 77 33 00 22 22 22 22 22 60 03 00 04 00 08 C0 28 44 00 00 00 00 C1

Table 40: EVSE: Start SLAC Matching Process

Parameter	Value
Destination MAC Address	00:01:01:63:77:33
Source MAC Address	00:22:22:22:22:22
Ethernet Type	60 03
Version	00
Message Type	04
Size	00 08
Framing payload	C0 28 44 00 00 00 00 C1 => 15.4.3

(4) After the SLAC process has finished successfully and the EV joined EVSE's logical network, the WHITE beet – module sends a control frame to inform the HLE. The HLE can now start with the higher layer communication (SDP, V2GTP). When the WHITE beet - module detects a timeout or any error it will send an error message to HLE.

Ethernet Frame (Success):

00 22 22 22 22 22 00 01 01 63 77 33 60 03 00 04 00 08 C0 28 80 00 00 00 00 C1

Table 41: EVSE: Response for successful SLAC matching process

Parameter	***Value***
Destination MAC Address	00:22:22:22:22:22
Source MAC Address	00:01:01:63:77:33
Ethernet Type	60 03
Version	00
Message Type	04
Size	00 08
Framing payload	C0 28 80 00 00 00 00 C1 => 15.3

Ethernet Frame (Failed):

00 22 22 22 22 22 00 01 01 63 77 33 60 03 00 04 00 08 C0 28 81 00 00 00 00 C1

Table 42: EVSE: Response for failed SLAC matching process

Parameter	***Value***
Destination MAC Address	00:22:22:22:22:22
Source MAC Address	00:01:01:63:77:33
Ethernet Type	60 03
Version	00
Message Type	04
Size	00 08
Framing payload	C0 28 8100 00 00 00 C1 => 15.3

19.4 EV: Successful SLAC matching process

19.4.1 Interaction Diagram

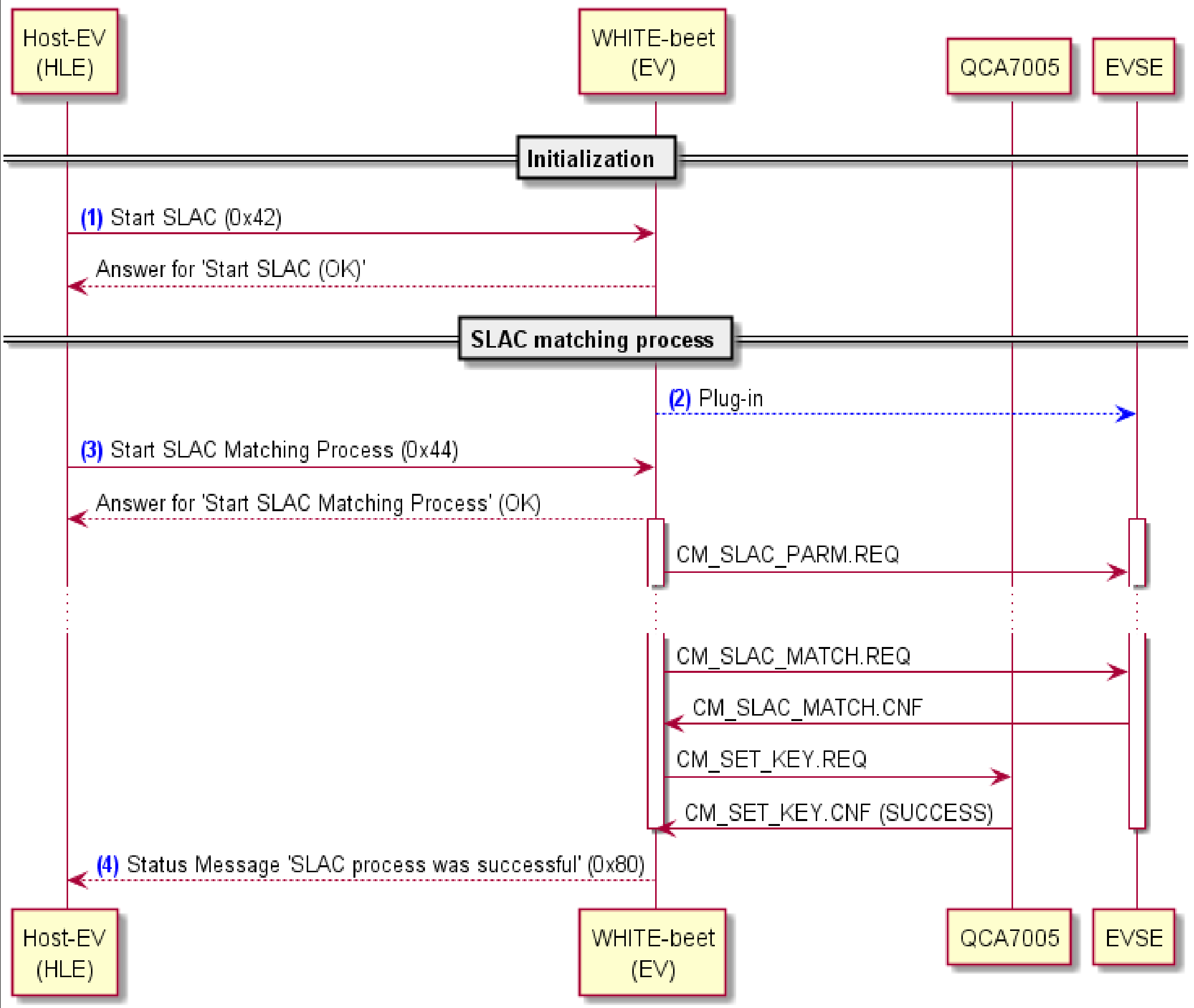


Figure 22: EV: Successful SLAC matching process

19.4.2 Process description

(1) The HLE sends a control frame to the module with the command 'Start SLAC' and the role (EV) for the SLAC process. The module will then be prepared for the SLAC matching process.

Ethernet Frame:

00 01 01 63 77 31   00 44 44 44 44 44   60 03   00   04   00 09   C0 28 42 00 00 01 00 00 C1

Table 43: EV: Start SLAC

Parameter	Value
Destination MAC Address	00:01:01:63:77:31
Source MAC Address	00:44:44:44:44:44
Ethernet Type	60 03
Version	00
Message Type	04
Size	00 09
Framing payload	C0 28 42 00 00 01 00 00 C1 => 15.4.1

(2) The EV wants to charge at the connected EVSE.

(3) The HLE sends a control frame to the module with the command 'Start SLAC Matching Process' for starting the SLAC matching process on WHITE beet Module. The EV will start sending SLAC messages. The WHITE beet – module performs the SLAC process as described in the ISO 15118-3.

Ethernet Frame:

00 01 01 63 77 31   00 44 44 44 44 44   60 03   00   04   00 08   C0 28 44 00 00 00 00 C1

Table 44: EV: Start SLAC matching process

Parameter	Value
Destination MAC Address	00:01:01:63:77:31
Source MAC Address	00:44:44:44:44:44
Ethernet Type	60 03
Version	00
Message Type	04
Size	00 08
Framing payload	C0 28 44 00 00 00 00 C1 => 15.4.3

(4) After the SLAC process has finished successfully and the EV joined EVSE's logical network, the WHITE beet - module sends a control frame to inform the HLE. The HLE can now start with the higher layer communication (SDP, V2G). If the WHITE beet - module detects a timeout or any error it will send an error message to HLE.

Ethernet Frame (Success):

00 44 44 44 44 44   00 01 01 63 77 31   60 03   00   04   00 08   C0 28 80 00 00 00 00 C1

Table 45: EV: Response for successful SLAC matching process

Parameter	Value
Destination MAC Address	00:44:44:44:44:44
Source MAC Address	00:01:01:63:77:31
Ethernet Type	60 03
Version	00
Message Type	04
Size	00 08
Framing payload	C0 28 80 00 00 00 00 C1 => 15.3

Ethernet Frame (Failed):

00 44 44 44 44 44   00 01 01 63 77 33   60 03   00   04   00 08   C0 28 81 00 00 00 00 C1

Table 46: EV: Response for failed SLAC matching process

Parameter	Value
Destination MAC Address	00:44:44:44:44:44
Source MAC Address	00:01:01:63:77:31
Ethernet Type	60 03
Version	00
Message Type	04
Size	00 08
Framing payload	C0 28 8100 00 00 00 C1 => 15.3

19.5 EVSE: Set AttnRx values

19.5.1 Interaction Diagram

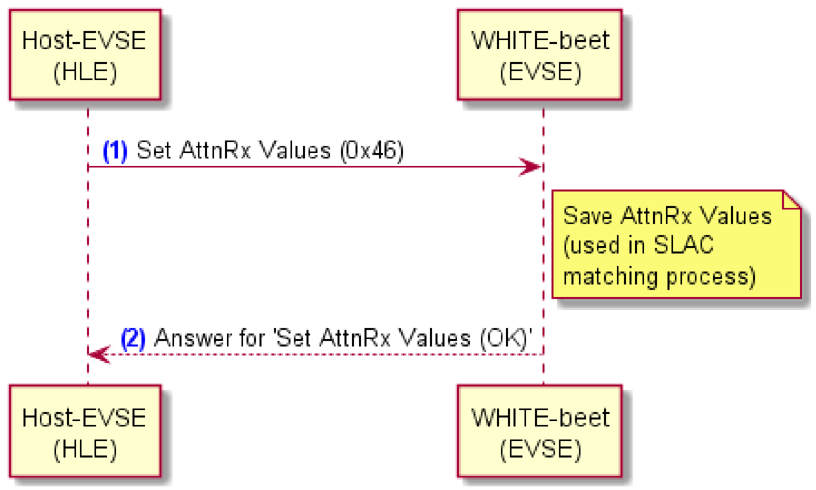

Host-EVSE (HLE)
WHITE-beet (EVSE)
(1) Set AttnRx Values (0x46)
Save AttnRx Values (used in SLAC matching process)
(2) Answer for 'Set AttnRx Values (OK)'
Host-EVSE (HLE)
WHITE-beet (EVSE)

Figure 23: EVSE: Set AttnRX values

19.5.2 Process description

(1) The HLE sends a the control frame 'Set AttnRx Values' with AttnRx values for all 58 groups to the module. The module will use these values for SLAC matching process and save them until reset.

Ethernet Frame:

00 01 01 63 77 33 00 22 22 22 22 22 60 03 00 04 00 42 C0 28 46 00 00 3A xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx 00 C1

Table 47: EVSE: Set AttnRx Values

Parameter	***Value***
Destination MAC Address	00:01:01:63:77:33
Source MAC Address	00:22:22:22:22:22
Ethernet Type	60 03
Version	00
Message Type	04
Size	00 42
Framing payload	C0 28 46 00 00 3A xx (58 times) 00 C1 => 15.4.4

(2) The module will send an answer for the command.

Ethernet Frame (Success):

00 22 22 22 22 22 00 01 01 63 77 33 60 03 00 04 00 09 C0 28 46 00 00 01 00 00 C1

Table 48: EVSE: Response for Set AttnRx Values

Parameter	***Value***
Destination MAC Address	00:22:22:22:22:22
Source MAC Address	00:01:01:63:77:33
Ethernet Type	60 03
Version	00
Message Type	04
Size	00 09
Framing payload	C0 28 46 00 00 01 00 00 C1 => 15.4.4

19.6 EV: Set AttnTx Reference values

19.6.1 Interaction Diagram

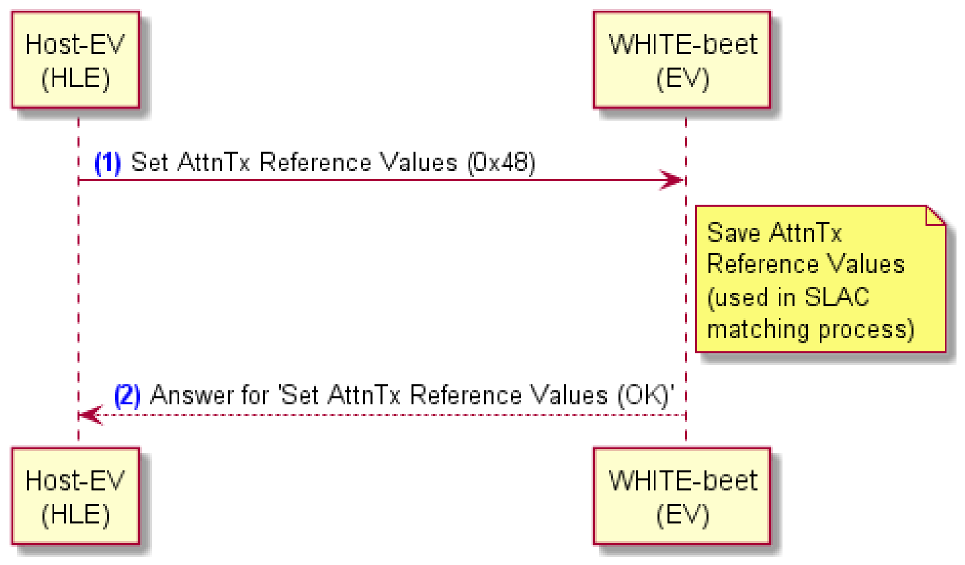

Host-EV
(HLE)
WHITE-beet
(EV)
(1) Set AttnTx Reference Values (0x48)
Save AttnTx Reference Values (used in SLAC matching process)
(2) Answer for 'Set AttnTx Reference Values (OK)'
Host-EV
(HLE)
WHITE-beet
(EV)

Figure 24: EV: Set AttnTx reference values

19.6.2 Process description

(1) The HLE sends a the control frame 'Set AttnTx Reference Values' with AttnTx Reference values for all 58 groups to the module. The module will use these values for SLAC matching process and save them until reset.

Ethernet Frame:

00 01 01 63 77 31 00 44 44 44 44 44 60 03 00 04 00 42 C0 28 48 00 00 3A xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx 00 C1

Table 49: EV: Set AttnTx Reference Values

Parameter	Value
Destination MAC Address	00:01:01:63:77:31
Source MAC Address	00:44:44:44:44:44
Ethernet Type	60 03
Version	00
Message Type	04
Size	00 42
Framing payload	C0 28 48 00 00 3A xx (58 times) 00 C1 => 15.4.6

(2) The module will send an answer for the command.

Ethernet Frame (Success):

00 44 44 44 44 44 00 01 01 63 77 31 60 03 00 04 00 09 C0 28 48 00 00 01 00 00 C1

Table 50: EV: Response for Set AttnTx Reference Values

Parameter	Value
Destination MAC Address	00:44:44:44:44:44
Source MAC Address	00:01:01:63:77:31
Ethernet Type	60 03
Version	00
Message Type	04
Size	00 09
Framing payload	C0 28 48 00 00 01 00 00 C1 => 15.4.6

19.7 SLAC Validation (EVSE)

19.7.1 Interaction Diagram

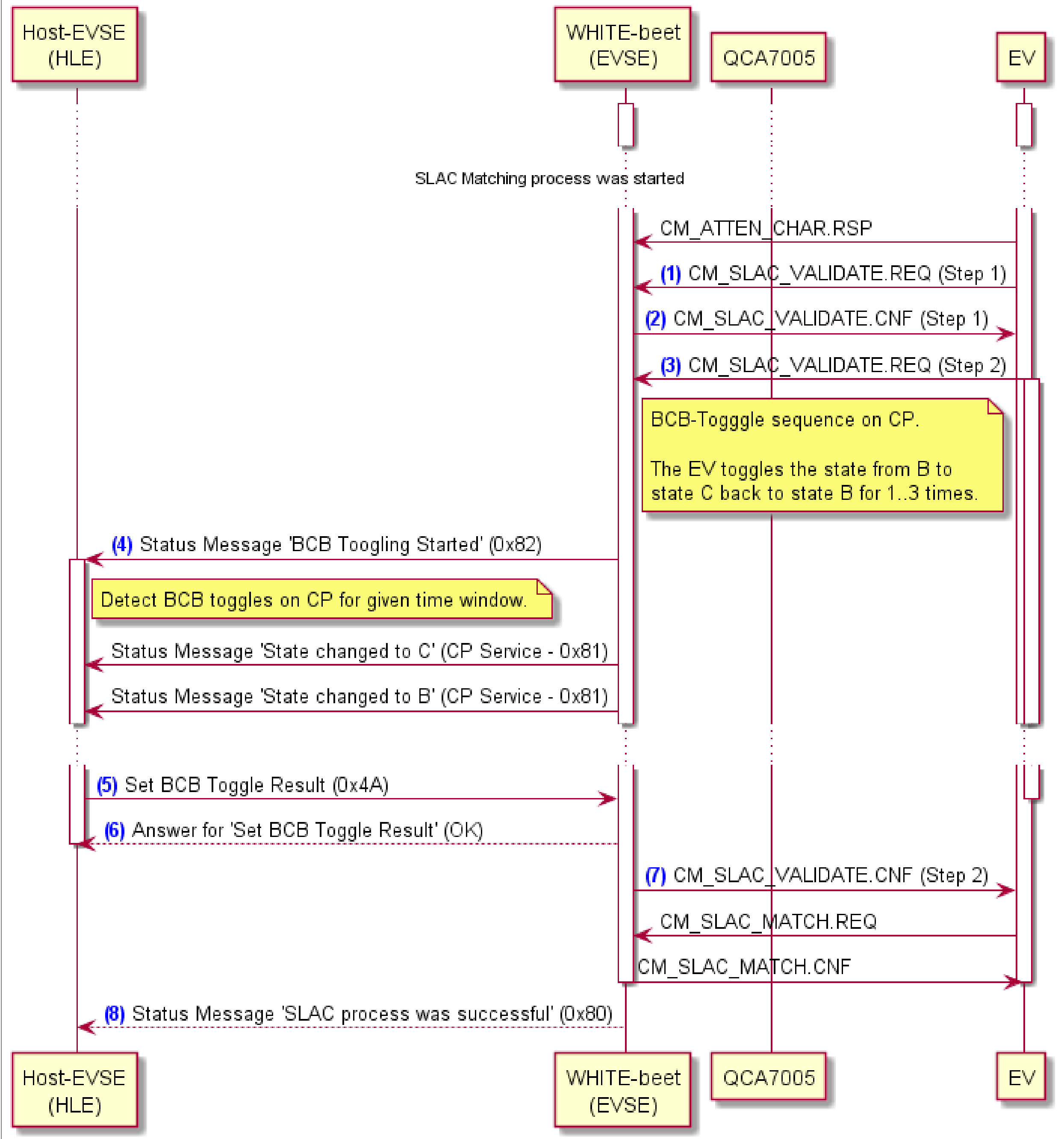

Host-EVSE (HLE)
WHITE-beet (EVSE)
QCA7005
EV
SLAC Matching process was started
CM_ATTEN_CHAR.RSP
(1) CM_SLAC_VALIDATE.REQ (Step 1)
(2) CM_SLAC_VALIDATE.CNF (Step 1)
(3) CM_SLAC_VALIDATE.REQ (Step 2)
BCB-Togggle sequence on CP.
The EV toggles the state from B to state C back to state B for 1..3 times.
(4) Status Message 'BCB Toogling Started' (0x82)
Detect BCB toggles on CP for given time window.
Status Message 'State changed to C' (CP Service - 0x81)
Status Message 'State changed to B' (CP Service - 0x81)
(5) Set BCB Toggle Result (0x4A)
(6) Answer for 'Set BCB Toggle Result' (OK)
(7) CM_SLAC_VALIDATE.CNF (Step 2)
CM_SLAC_MATCH.REQ
CM_SLAC_MATCH.CNF
(8) Status Message 'SLAC process was successful' (0x80)
Host-EVSE (HLE)
WHITE-beet (EVSE)
QCA7005
EV

Figure 25: EVSE: SLAC validation

19.7.2 Process description

After the SLAC matching was started on both sides (EV and EVSE) the EV will receive the attenuation from the EVSE. Now the EV can decide if it is necessary to execute the validation process to ensure that the EVSE is really connected to the EV. Therefore it can send CM_VALIDATE Requests.

(1) The EV sends the first CM_VALIDATE.REQ to the EVSE.

(2) The EVSE answers the CM_VALIDATE.REQ with a CM_VALIDATE.CNF. This contains a different status depending on configuration and state.

- If validation is deactivated at the charging station side, the vehicle receives the information that validation is not possible. The vehicle can now decide whether to continue with the matching process or test another charging station.

- If the validation is activated, the vehicle receives the information that the validation is possible and continues with step **(3)**.

(3) The EV sends the second CM_VALIDATE.REQ to the EVSE. This contains a time window in which the BCB phase is performed.

(4) The EVSE informs the Host that the BCB toggle phase has been started and in which period of time it must listen for state changes. For this the host is informed about status messages from the CP service.

(5) After the time is up, the host must report the result of the counted state change to the charging station as soon as possible.

(6) The EVSE then sends an ACK to the host.

(7) The vehicle continues the matching process as known if the result is correct and sends the message CM_SLAC_MATCH.REQ.

(8) If everything went well, the charging station sends the status message ‘SLAC matching successful’.

19.8 V2G Charging Session Example

19.8.1 Configuration

Before the V2G service can be used, it needs to be configured. Use the configuration commands described in 17.9 and 17.10 to set the supported protocols, the SDP configuration, the payment options and the available energy transfer modes. After the service was configured it can be started. This will initialize the V2G module which is then listening for incoming SDP requests and TCP connections on the configured port.

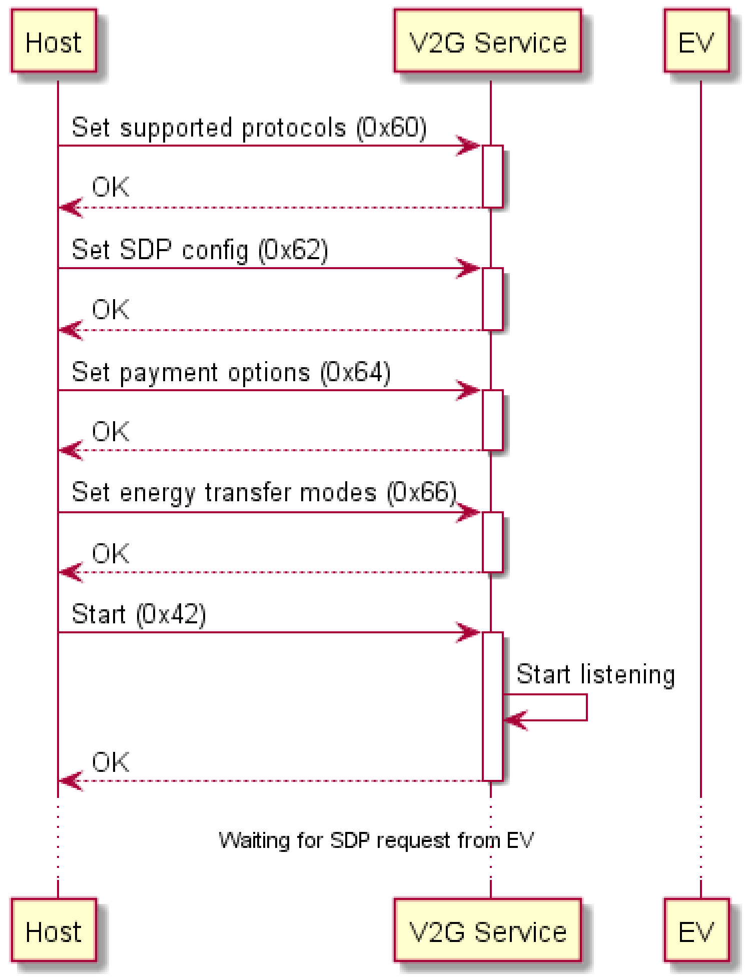

Host
V2G Service
EV
Set supported protocols (0x60)
OK
Set SDP config (0x62)
OK
Set payment options (0x64)
OK
Set energy transfer modes (0x66)
OK
Start (0x42)
Start listening
OK
Waiting for SDP request from EV
Host
V2G Service
EV

Figure 26: Configuring and starting the V2G service

19.8.2 Session Start and Stop

When an EV discovered the TCP port by using the SDP protocol it will request to start a session by sending a SessionSetupReq message. The host will be informed with the Session started status message 17.12.1 and the EVSE will be requested. When the session was completed by the SessionStopReq message or an error occurred during the communication session the host will be informed by the Session stopped status message 17.12.2.

After the session was started the EV requests the services, that were already set during configuration, with the ServiceDiscoveryReq. It also requests detailed information in the ServiceDetailReq. After the services were requested the EV selects a charging service and a payment method in the PaymentServiceSelectionReq message.

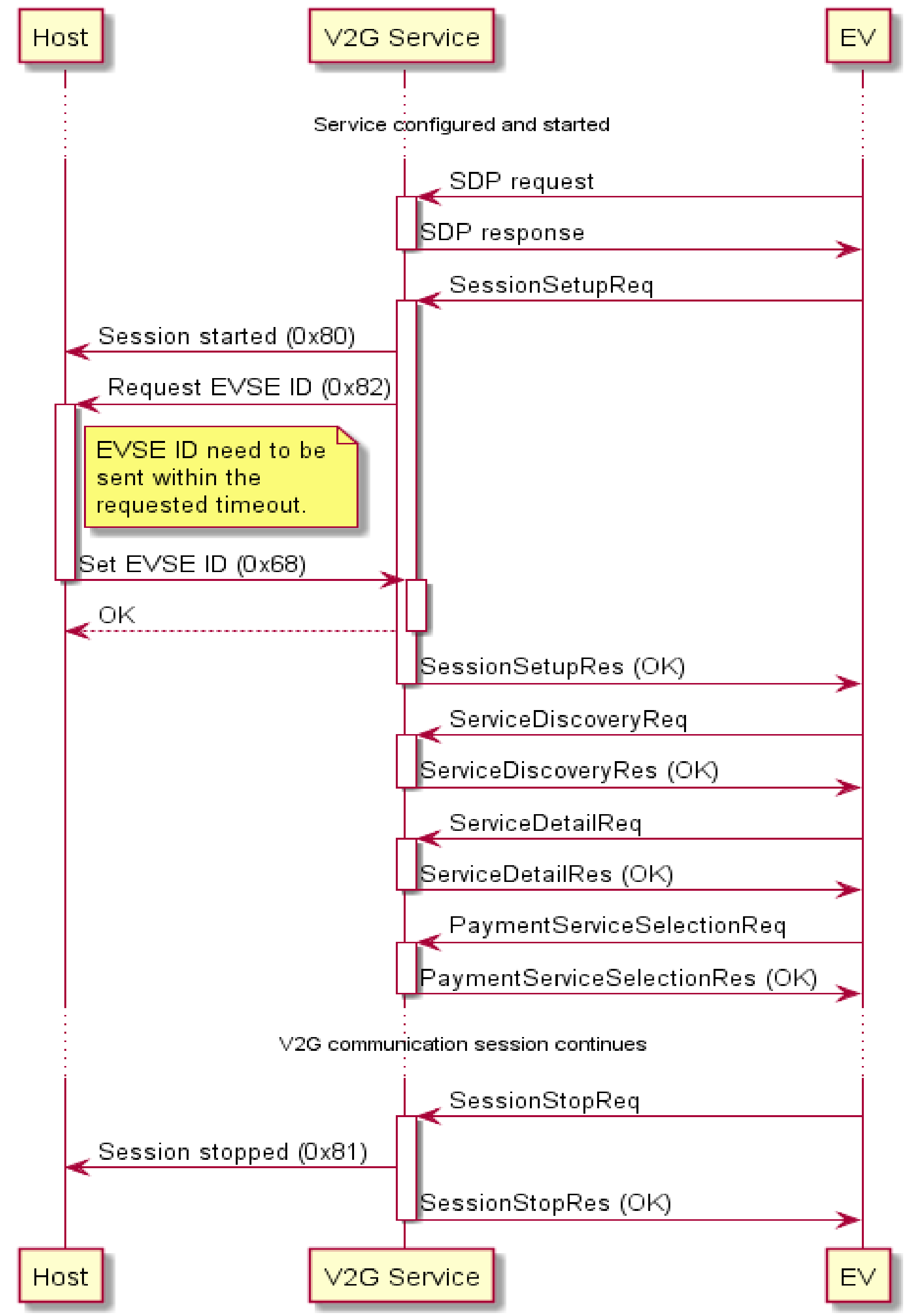


Figure 27: Start and end of a V2G communication session

19.8.3 Authorization

The EV continues with the authorization. The authorization is also requested from the host. The host has to respond to the request in the given timeout. Since the timeout on the V2G message level is lower than on the application level it can happen that a response has to be sent to the EV before the host returned the authorization result. In that case a parameter in the response will be set to ongoing and the EV will send an additional request until the authorization was completed.

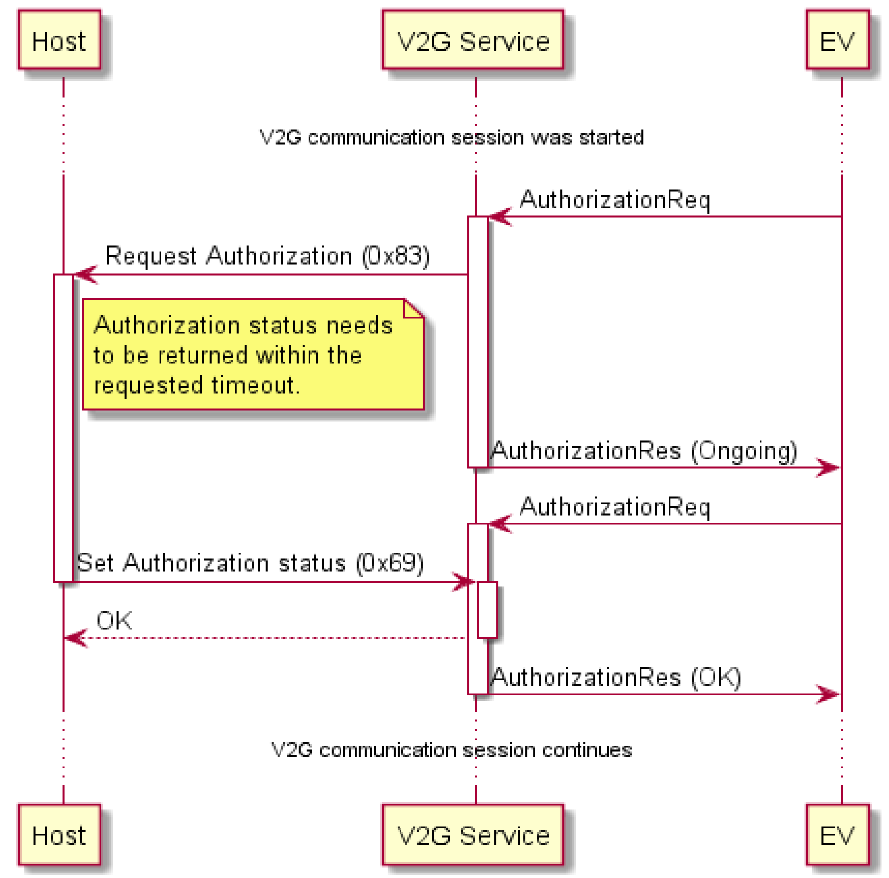

Host
V2G Service
EV
V2G communication session was started
AuthorizationReq
Request Authorization (0x83)
Authorization status needs to be returned within the requested timeout.
AuthorizationRes (Ongoing)
AuthorizationReq
Set Authorization status (0x69)
OK
AuthorizationRes (OK)
V2G communication session continues
Host
V2G Service
EV

Figure 28: Authorization of the EV

19.8.4 Charge Parameter Discovery

Following the authorization the EV sends the ChargeParameterDiscoveryReq message. EV parameters are reported to the host and EVSE parameters are requested. In this message the EVSE also reports the schedules which have to be returned by the host.

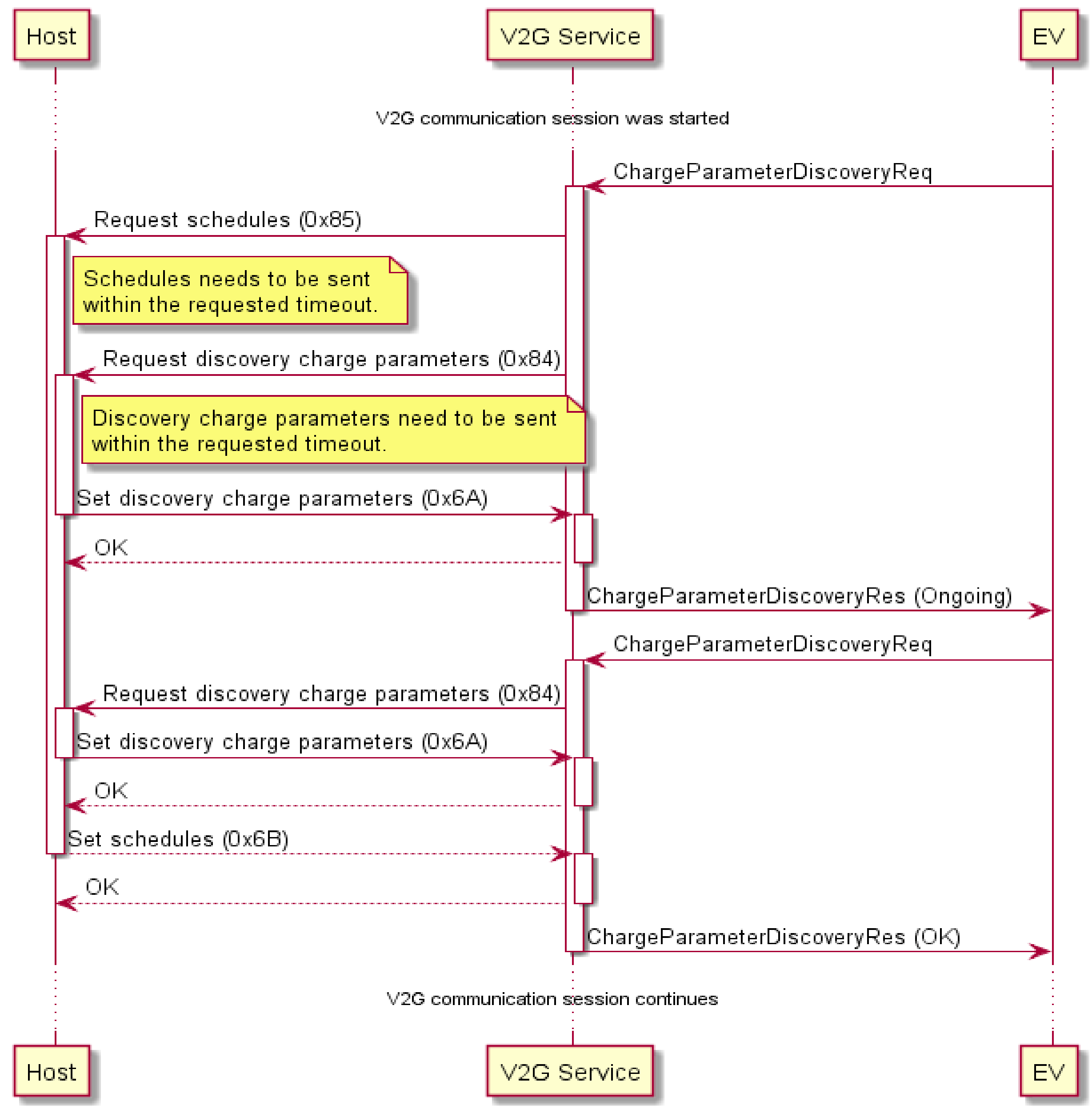

Host
V2G Service
EV
V2G communication session was started
ChargeParameterDiscoveryReq
Request schedules (0x85)
Schedules needs to be sent within the requested timeout.
Request discovery charge parameters (0x84)
Discovery charge parameters need to be sent within the requested timeout.
Set discovery charge parameters (0x6A)
OK
ChargeParameterDiscoveryRes (Ongoing)
ChargeParameterDiscoveryReq
Request discovery charge parameters (0x84)
Set discovery charge parameters (0x6A)
OK
Set schedules (0x6B)
OK
ChargeParameterDiscoveryRes (OK)
V2G communication session continues
Host
V2G Service
EV

Figure 29: Discovery of the charge parameters

19.8.5 Cable Check

When the charge parameter discovery was completed the cable check is requested by the EV. During the cable check the EV parameters are reported and EVSE parameters have to be returned by the host.

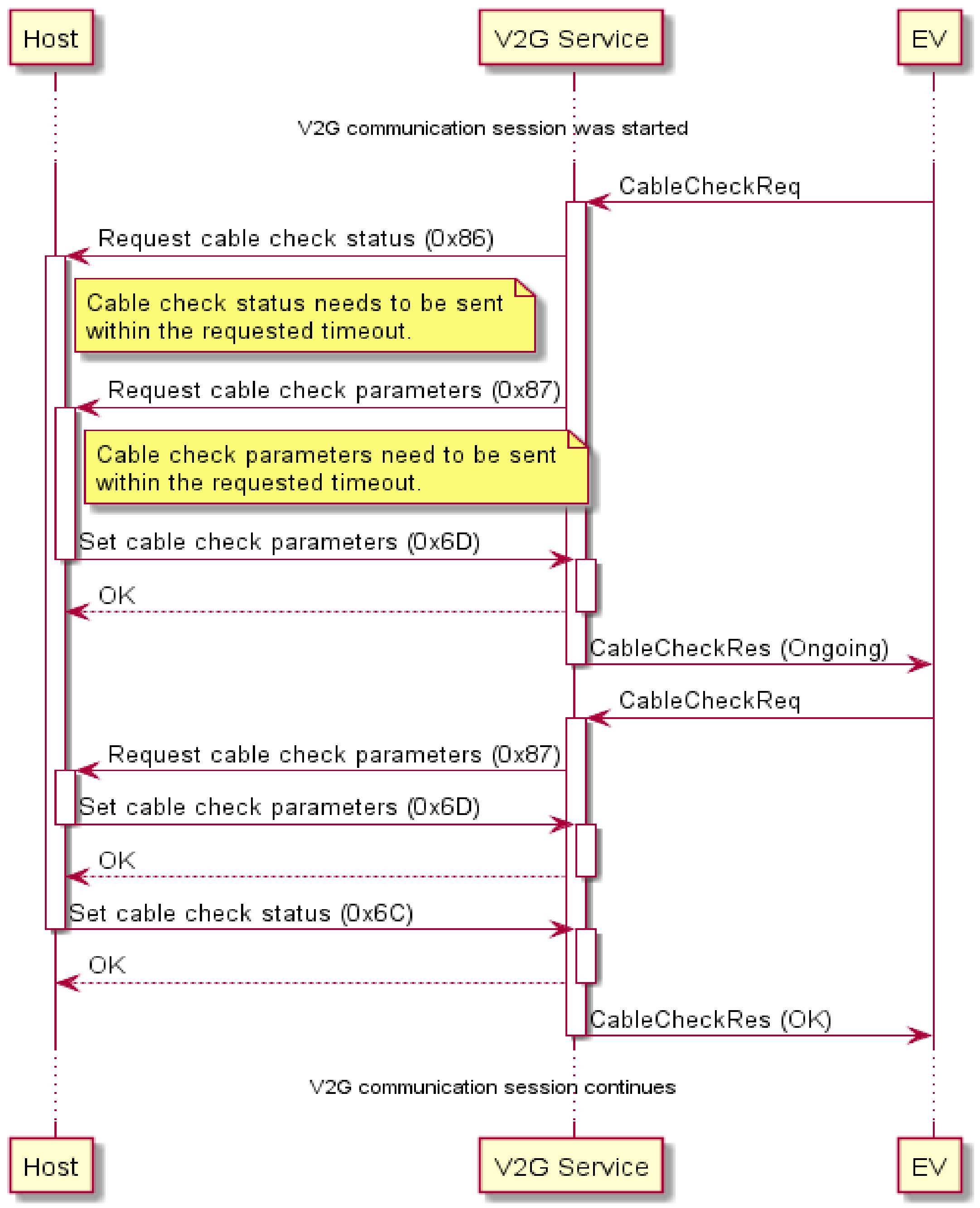

Host
V2G Service
EV
V2G communication session was started
CableCheckReq
Request cable check status (0x86)
Cable check status needs to be sent within the requested timeout.
Request cable check parameters (0x87)
Cable check parameters need to be sent within the requested timeout.
Set cable check parameters (0x6D)
OK
CableCheckRes (Ongoing)
CableCheckReq
Request cable check parameters (0x87)
Set cable check parameters (0x6D)
OK
Set cable check status (0x6C)
OK
CableCheckRes (OK)
V2G communication session continues
Host
V2G Service
EV

Figure 30: Processing of the cable check

19.8.6 Pre-Charge

After the cable check was completed by the host returning the cable check status and the result is the cable check being successful the EV continues by sending PreChargeReq message where again EV and

EVSE pre charge parameters are exchanged. The PreChargeReq message is repeated until the EV decides that the parameters sent by the EVSE are sufficient.

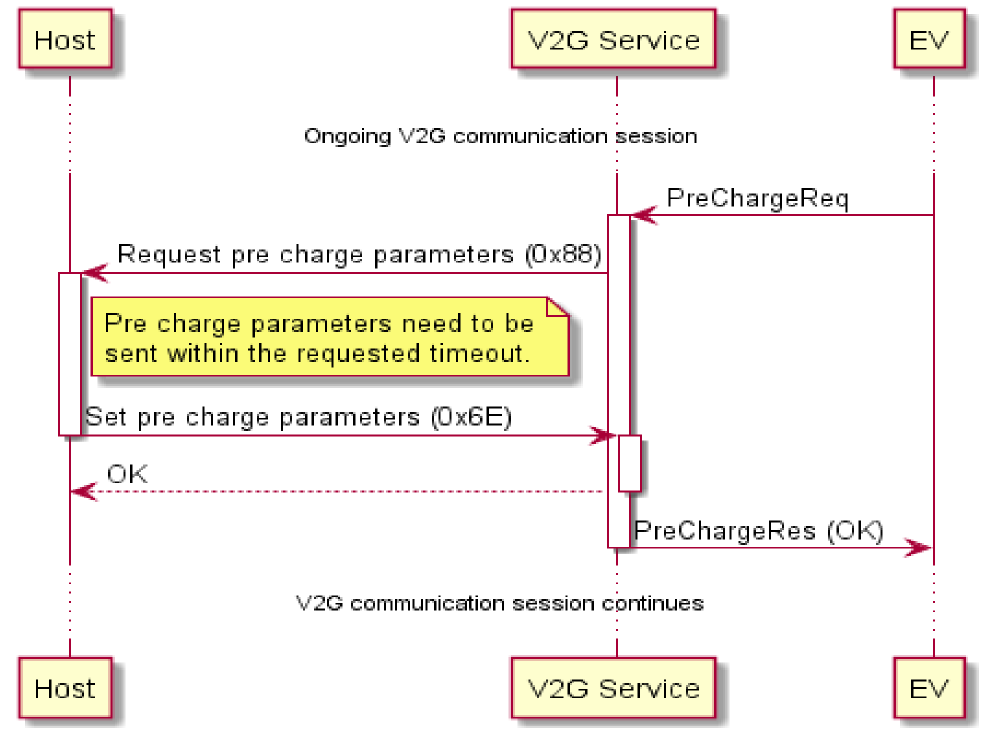

Host
V2G Service
EV
Ongoing V2G communication session
PreChargeReq
Request pre charge parameters (0x88)
Pre charge parameters need to be sent within the requested timeout.
Set pre charge parameters (0x6E)
OK
PreChargeRes (OK)
V2G communication session continues
Host
V2G Service
EV

Figure 31: Exchange sequence of pre-charge parameters

19.8.7 Start Charging

When the EV has decided that the pre charge parameters are sufficient it will send a PowerDeliveryReq message to request the charging to be started. This is also requested from the host.

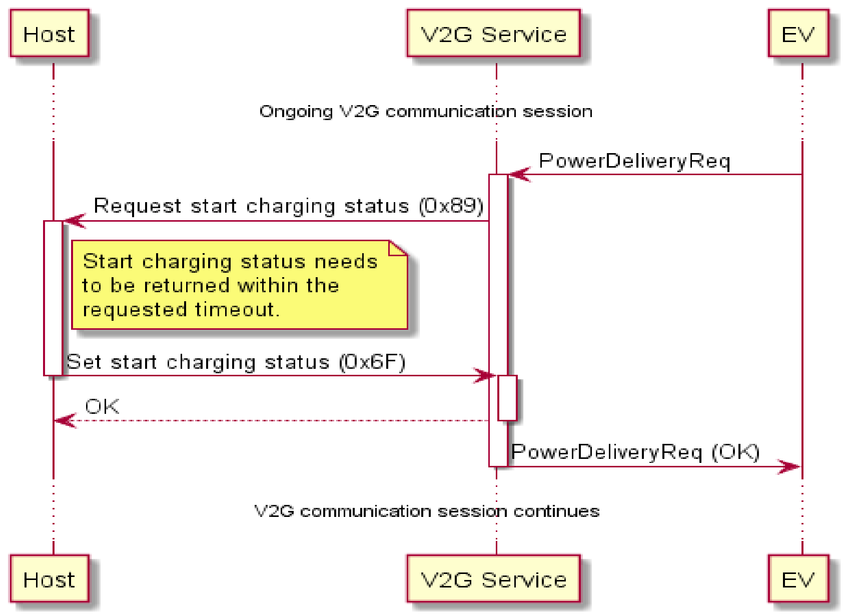

Host
V2G Service
EV
Ongoing V2G communication session
PowerDeliveryReq
Request start charging status (0x89)
Start charging status needs to be returned within the requested timeout.
Set start charging status (0x6F)
OK
PowerDeliveryReq (OK)
V2G communication session continues
Host
V2G Service
EV

Figure 32: Start sequence of the charging process

19.8.8 Charge Loop

When charging was started successfully the EV sends the CurrentDemandReq as long as charging is active. EV parameters are reported to the host and EVSE parameters are requested.

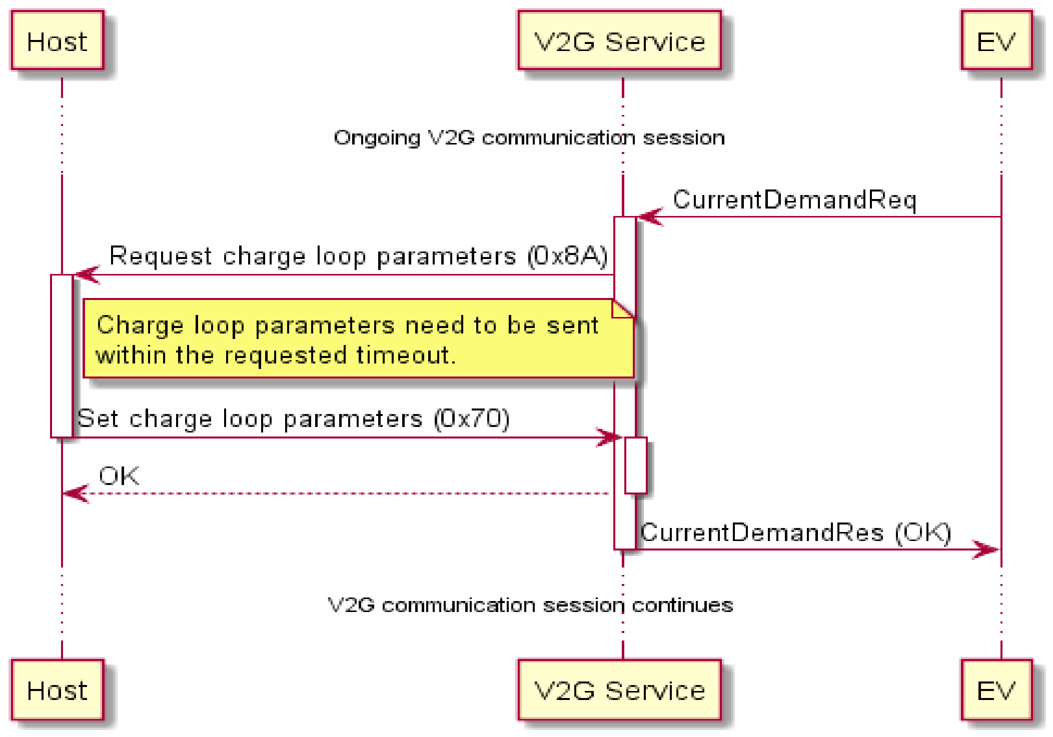

Host
V2G Service
EV
Ongoing V2G communication session
CurrentDemandReq
Request charge loop parameters (0x8A)
Charge loop parameters need to be sent within the requested timeout.
Set charge loop parameters (0x70)
OK
CurrentDemandRes (OK)
V2G communication session continues
Host
V2G Service
EV

Figure 33: Messages during the charge loop

19.8.9 Stop Charging

When the vehicle decides to stop the charging process it will send a PowerDeliveryReq message. The request is forwarded to the host by requesting to stop the charging. The host needs to send the response message within the requested timeout.

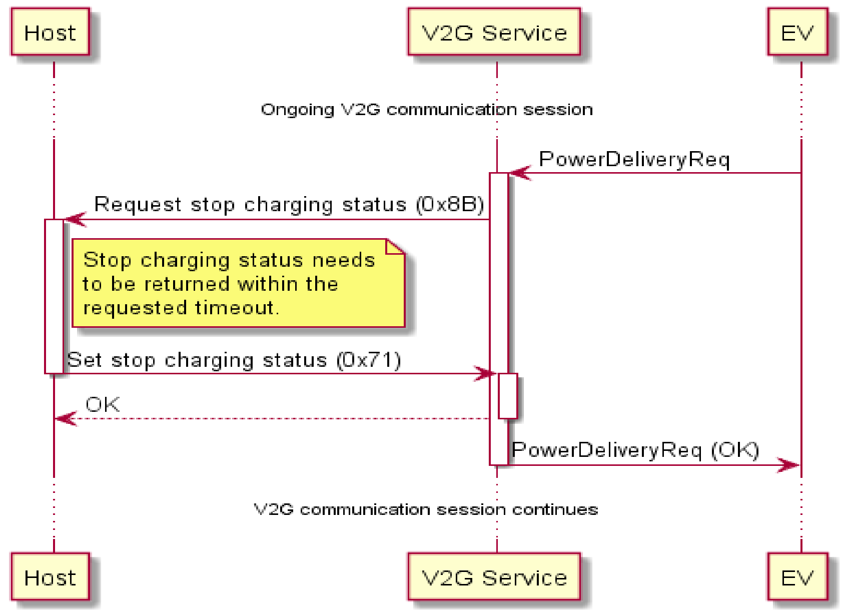

Host
V2G Service
EV
Ongoing V2G communication session
PowerDeliveryReq
Request stop charging status (0x8B)
Stop charging status needs to be returned within the requested timeout.
Set stop charging status (0x71)
OK
PowerDeliveryReq (OK)
V2G communication session continues
Host
V2G Service
EV

Figure 34: Stop sequence of the charging process

19.8.10 Post Charge

After the charging was stopped, the EV will send the WeldingDetectionReq message where again EV parameters are reported and EVSE parameters are requested as long as the EV is connected.

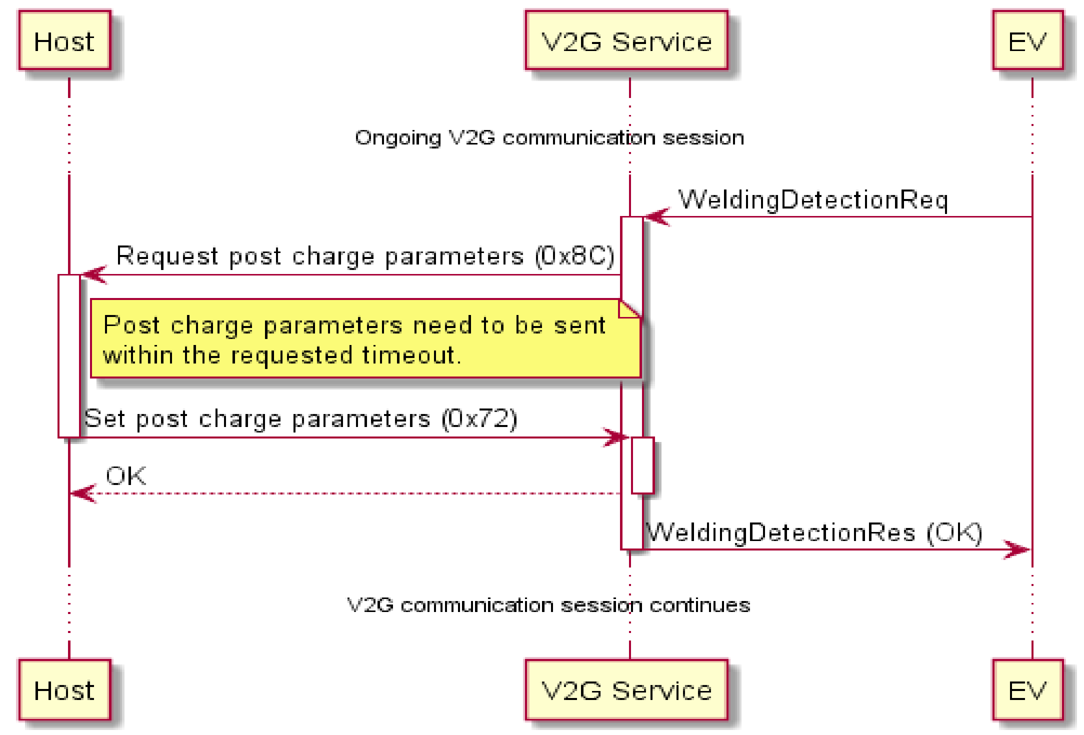

Host
V2G Service
EV
Ongoing V2G communication session
WeldingDetectionReq
Request post charge parameters (0x8C)
Post charge parameters need to be sent within the requested timeout.
Set post charge parameters (0x72)
OK
WeldingDetectionRes (OK)
V2G communication session continues
Host
V2G Service
EV

Figure 35: Exchange of post charge parameters

20 Appendix

20.1 Python tool dependencies

All White Beet tools also exit as Python variant and each of these tools has its own requirements which are described in the following chapters. They can be executed on all systems where Python is available (Windows, Linux, ...). The prerequisite is that the tool specific Python packages must be available for the system and they must be installed before running the script.

20.1.1 StxCfgGen

Required Python Version:

To run the StxCfgGen tool it is recommended to use Python 3.9. The tool was tested with Python 3.9.1 on Windows and 3.9.7 under Linux.

Required python packages for using the Python script:

No dependencies on packages outside the python distribution

20.1.2 StxFwGen

Required Python Version:

To run the StxFwGen tool it is recommended to use Python 3.9. The tool was tested with Python 3.9.1 on Windows and 3.9.7 under Linux.

Required python packages for using the Python script:

```
cffi==1.14.6
cryptography==35.0.0
pycparser==2.20
PyKCS11==1.5.10
pyOpenSSL==21.0.0
six==1.16.0
```

Figure 36: Required python packages for StxFwGen.

20.1.3 StxFwUpdater

Required Python Version:

To run the StxFwUpdater tool it is recommended to use Python 3.9. The tool was tested with Python 3.9.1 on Windows and 3.9.7 under Linux.

Required python packages for using the Python script:

```
cffi==1.14.6
cryptography==35.0.0
hkdf==0.0.3
psutil==5.8.0
pycparser==2.20
pyserial==3.5
pythoncrc==1.21
scapy==2.4.5
```

Figure 37: Required python packages for StxFwUpdater.

20.1.4 Dir2StxFs

Required Python Version:

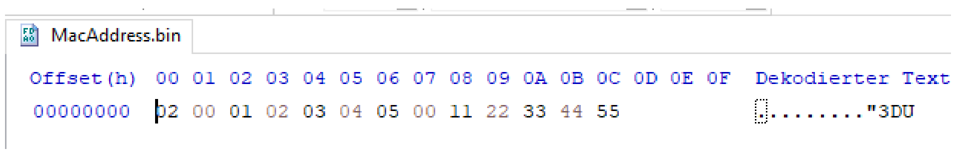
To run the Dir2StxFs tool it is recommended to use Python 3.9. The tool was tested with Python 3.9.1 on Windows and 3.9.7 under Linux.

Required python packages for using the Python script:

No dependencies on packages outside the python distribution

20.2 Example for changing MAC addresses

1. The first thing you have to do for changing the MAC addresses of the White Beet module is that you have to create a binary file, which can then look as follows:



```
MacAddress.bin
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Dekodierter Text
00000000   02 00 01 02 03 04 05 00 11 22 33 44 55           ........."3DU
```

   The file starts with a byte containing the number of MAC addresses in the file (here two) and follows with the two MAC addresses (each 6 bytes). For more info, see Chapter 5.2.

2. Create firmware update file with the generated binary file as input.
   1. For creating a firmware update (FWU file), a configuration file for the FW Update is needed.

      The configuration (fwu_cfg.py) for changing MAC address could look like this:

```
base_file = None
maximum_containersize = 2000
container_format_version = 1

modules = [
            [
                "INFO",
                [
                    [0x10, "CMP", "SEVENSTAX GmbH"]
                ]
            ],
            [
                "CERTIFICATION",
                {
                    "certificate": "DemoSignWhiteBeetPKI.crt.der",
                    "signature_scheme": "rsa-pkcs1"
                }
            ],
            [
                "FILE",
                {
                    "source" : "MacAddress.bin",
                    "destination" : "fs/dev/mac.bin"
                }
            ]
        ]
```
Figure 38: Example configuration for MAC address firmware udpate.

   2. Now run the StxFwGen-Tool to generate the FWU file.

      For this purpose the following command must be executed:

      StxFwGen.exe -k DemoSignWhiteBeetPKI.key -c fwu_cfg.py -o MacAddress.fwu

Please Note that the filenames must be adapted depending from used files.

3. Upload the FWU file to the White Beet module by using the StxFwUpdater-Tool:
   StxFwUpdate.exe -f MacAddress.fwu -t 00:01:01:63:77:33 -i “Ethernet”

   More information about the firmware update can be found in chapter 7.

20.3 Example for uploading PIB file for QCA7005

1. If not available, first create or modify a PIB file.
2. Create firmware update file (FWU file) with the PIB file as input.

   The StxFwGen configuration (fwu_cfg.py) for uploading a PIB file could look like this:

```
base_file = None
maximum_containersize = 2000
container_format_version = 1

modules = [
            [
                "INFO",
                [
                    [0x10, "CMP", "SEVENSTAX GmbH"],
                    [0x11, "CMP", "ISO15118 (EVSE)"],
                ]
            ],
            [
                "CERTIFICATION",
                {
                    "certificate": "DemoSignWhiteBeetPKI.crt.der",
                    "signature_scheme": "rsa-pkcs1"
                }
            ],
            [
                "FILE",
                {
                    "source" : "QCA7005.pib",
                    "destination" : "fs/config/qca/fw2/evse.pib"
                }
            ]
        ]
```

Figure 39: Example configuration for PIB upload firmware udpate.

3. Now run the StxFwGen-Tool to generate the FWU file.

   For this purpose the following command must be executed:

   StxFwGen.exe -k DemoSignWhiteBeetPKI.key -c fwu_cfg.py -o PibFile.fwu

   Please Note that the filenames must be adapted depending from used files.
4. Upload the FWU file to the White Beet module by using the StxFwUpdater-Tool:
   StxFwUpdate.exe -f PibFile.fwu -t 00:01:01:63:77:33 -i “Ethernet”

More information about the firmware update can be found in chapter 7.

20.4 Example for uploading White Beet Firmware Update

20.4.1 Environmental conditions used in example

Ethernet MAC Address White Beet:	00:01:01:63:77:33
PC ethernet interface name:	Ethernet
FW-Update file name:	WhiteBeet_FwUpdate.fwu

20.4.2 Procedure for the update

1. The firmware updates for the White Beet module do not have to be generated by yourself, but are provided by the technical support for White Beet.
2. Upload the FWU file to the White Beet module by using the StxFwUpdater-Tool:

   ```
   StxFwUpdate.exe -f WhiteBeet_FwUpdate.fwu -t 00:01:01:63:77:33 -i “Ethernet”
   ```

   More information about the firmware update can be found in chapter 7.

21 Change History

Vers.	Date	by	Change description
0.1	2020-09-14	bbr	First version
0.6	2020-10-01	bbr	Added new commands and changes for WHITE beet module.
0.7	2020-10-27	bbr	Added firmware update chapter.
1.0	2020-10-30	bbr	Added additional commands and improved the existing description.
1.1	2020-11-20	jpo	Add chapter for V2G service framing interface and add application example.
1.2	2020-12-01	bbr	Added chapter for control pilot service (framing interface and application example). Revised Chapter 2 - Product Description. Revised figures in chapter 19.3 and chapter 19.4.
1.3	2020-12-10	bbr	Added required messages for SLAC validation in chapter 15 and added example (19.7).
1.4	2020-12-17	cba	Spelling and layout fixes.
1.5	2021-03-15	bbr	Added message examples for SLAC, CP and V2G service.
1.6	2021-03-24	bbr	Updated with new informations to new WB-CARRIER-BOARD. Merging of both varieties (SLAC Bridging and ISO15118) started. Added HCI-Commands for GPIO control and network configuration.
1.7	2021-03-26	jpo	Duty cycle handling was changed from percent to permill.
1.8	2021-04-01	bbr	Revised the manual and added new HCI-Commands for Firmware Update.
1.9	2021-04-23	bbr	Fix for broken references and image added to chapter ‘Product Description’.
2.0	2021-04-26	jpo	Spelling fixes.
2.1	2021-05-03	bbr	Added chapter for SPI interface.
2.2	2021-06-25	bbr	Added description for HCI select pin. Added command for getting AD value for CP.
2.3	2021-06-28	bbr	Documentation update with revised pin assignment at SPI, GPIOs and HCI select pins.
2.4	2021-07-23	bbr	Fixed parameters for Firmware Update command ‘FWU Data Frame’ Fixed wrong module ID for V2G Fix missing parameter in example messages of V2G. Add range for exponent of exponential type of V2G. Add missing booleans for optional types in ‘Request Start Charging’. Add PLC Service commands. Added chapter ‘Safety instructions’.
2.5	2021-08-24	jpo	Fixed spelling in GPIO and FWU modules. Descriptions in Fimrware Update chapter adapted to WHITE beet module. Updated and renamed chapter 6.
2.6	2021-09-23	bbr	Fixed description of MAC addresses in chapter 5.2.
2.7	2021-10-08	bbr	Improvements for White Beet tools usage. Added appendix with examples and information for python tool

Vers.	*Date*	*by*	*Change description*
			dependencies.