SPECIFICATION

EnOcean Serial Protocol 3 (ESP3)

V1.30 / March 22, 2016

© EnOcean GmbH
Kolpingring 18a
82041 Oberhaching
Germany

Phone +49.89.67 34 689-0
Fax +49.89.67 34 689-50
info@enocean.com
www.enocean.com

Content

1 EnOcean Serial Protocol 3 (ESP3)

REVISION HISTORY

The following major modifications and improvements have been made to the first version of this document:

No.	Major Changes	Date	Who
1.0	First Version		
1.1	Corrections, added uses cases		
1.2	Added small correction in CMD_SA_LEARNDEVICE command.		
1.2	Reworked improved protocol		
1.3	Removed SMACK comments – rework needed		
1.4	Document Reviewed, performance measurements moved to EO3000I_API		
1.5	Added PacketType = 3		
1.6	Added types and defined commands		
1.7	New terms and definitions; extended description	2010-08-31	ap, op, jh
1.8	Modifications	2010-09-07	ap, op
1.9	Extracted from system spec document, removed internal info	2010-09-15	ASt
1.10	1st review	2010-09-28	op
1.11	2nd review (notes ap, mf)	2010-10-11	op
1.12	Minor modifications	2010-10-27	nm, op
1.13	Minor modifications	2010-11-03	nm, op
1.14	New event: CO_READY	2010-12-10	op
1.15	New optional data in CO_RD_IDBASE	2011-02-28	wh,ap
1.16	Corrected wrong CRC8D in CO_WR_RESET example Changed the REMOTE_MAN_COMMAND	2011-06-14	jh
1.17	Examples added for CO_WR_FILTER_ADD	2011-08-02	mho, wh
1.18	Fixed CO_RD_FILTER description	2012-03-16	jh
1.19	Common commands for secure devices added; minor modifications	2012-12-15	ap
1.20	Added CO_WR_MODE, type RADIO_ADVANCED to support advanced protocol	2013-01-06	jh
1.21	Minor editorial changes	2013-02-12	ap
1.22	Added type RADIO_MESSAGE, CO_RD_SECURE_DEVICE, CO_RD_NUM_SECURE_DEVICES. Updated types table	2013-03-04	ap
1.23	Fixed typos . Updated security commands, added commands: CO_WR_SECUREDEVICE_ADD_PSK, CO_WR_SECUREDEVICE_SENDTEACHIN, CO_WR_TEMPORARY_RLC_WINDOW	2013-08-19	jh
1.24	Added CO_RD_SECUREDEVICE_PSK	2013-10-14	jh
1.25	Modifications in filter and repeater common commands to enable filtered repeating functionality	2014-02-24	ap
1.26	Changed description of SA_RD_LEARNEDCLIENTS response, enhanced secure ESP 3 messages	2014-05-06	mhs
1.27	CO_WR_SECURE_DEVICE_ADD: Added optional byte defining if it is a PTM. Added CO_RD_DUTYCYCLE_LIMIT and CO_DUTYCYCLE_LIMIT.Remote management examples corrected. Many typos corrected. Renamed "Advanced Protocol" to ERP2.	2014-07-30	ap
1.28	Return code RET_LOCK_SET added.		ap
1.29	Reference to 2.4 Commands added.	2015-10-16	tm

	CO_SET_BAUDRATE and new supported higher baudrates added		
1.30	CO_GET_FREQUENCY_INFO, CO_GET_STEPCODE and 2.4 documentation added	2016-22-03	tm

Published by EnOcean GmbH, Kolpingring 18a, 82041 Oberhaching, Germany
www.enocean.com, info@enocean.com, phone ++49 (89) 6734 6890

Important!

This information describes the type of component and shall not be considered as assured characteristics. No responsibility is assumed for possible omissions or inaccuracies. Circuitry and specifications are subject to change without notice. For the latest product specifications, refer to the EnOcean website: http://www.enocean.com.

As far as patents or other rights of third parties are concerned, liability is only assumed for modules, not for the described applications, processes and circuits.
EnOcean does not assume responsibility for use of modules described and limits its liability to the replacement of modules determined to be defective due to workmanship. Devices or systems containing RF components must meet the essential requirements of the local legal authorities.

The modules must not be used in any relation with equipment that supports, directly or indirectly, human health or life or with applications that can result in danger for people, animals or real value.

Components of the modules are considered and should be disposed of as hazardous waste. Local government regulations are to be observed.

Packing: Please use the recycling operators known to you. By agreement we will take packing material back if it is sorted. You must bear the costs of transport. For packing material that is returned to us unsorted or that we are not obliged to accept, we shall have to invoice you for any costs incurred.

1.1 Terms & Abbreviations

Term / Abbr.	Description
µC	Microcontroller (external)
API	Application Programming Interface
APP	Application
BIST	Built-in self-test
CRC8	Cyclic redundancy check (CRC) or polynomial code checksum; CRC-8 = 9 bits polynomial lengths
CRC8D	CRC8 for Group ‘Data’ (incl. Optional Data)
CRC8H	CRC8 for Group ‘Header’
Data	Payload of ESP3 packet
EEP	EnOcean Equipment Profile
ERP1, ERP2	EnOcean Radio Protocol. There are versions 1 and 2.
ESP3	EnOcean Serial Protocol V3
Field	Identifier of Data subset / element
Group	Part of ESP3 packet (header, data, optional data)
Host	ESP3 communication device
LSB	Least significant bit
Mailbox	Message filing of the Postmaster for each Smart Ack Sensor/Client
MSB	Most significant bit
Offset	Byte position pointer of packet
Packet	ESP3 data unit
Packet Type	Type of ESP3 Packet (Command, Event, Radio, ...)
PM	Postmaster
Postmaster	Includes multiple mailboxes for each Smart Ack Sensor/Client
R-ORG	Unique identification of radio telegram types
R-ORG_EN	Addressed version of ’R-ORG’ (EN = encapsulation)
RS-232	Telecommunication standard for serial binary single-ended data and control signals; ESP3 use only the minimal "3-wire" RS-232 connection consisting only of transmit data, receive data, and ground. The full facilities of RS-232 are not required.
RSSI	Received signal strength indication (dBm)
Smart Ack	EnOcean standard for energy-optimized bidirectional transmission
Subtelegram	Smallest unit of data in radio transmission, using orthogonal structure
Sync Byte	Identifier for ESP3 packet start
UART	Universal Asynchronous Receiver Transmitter

1.2 Introduction

This document specifies the EnOcean Serial Protocol 3.0 (ESP3).
The ESP3 defines the serial communication between a host and EnOcean modules (based on Dolphin Platform). Hosts are external microcontrollers or PC's incl. software tools.
The physical interface between a host and a EnOcean RF module (UART) is a 3-wire connection (Rx, Tx, GND / software handshake / full-duplex), modelled on RS-232 serial interface.

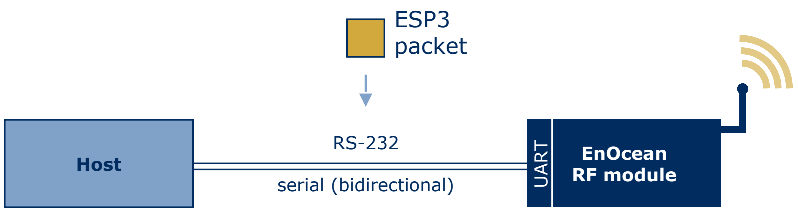


Figure 1

ESP3 enhances ESP2, adding future-proof structures and extending the data content. The new functional properties are:

- Transmission of the received radio signal strength and number of the received subtelegrams
- Future requirements can be realized flexibly with the packet group "Optional Data", without violating the compatibility
- Improved data security and consistency by CRC8 Data verification
- Higher reliable ESP3 packet detection at serial byte stream
- Approximately at least seven-time higher baud rate
- The standard baud rate is 57600. If the device supports an higher baudrate, the command SET_BAUDRATE can be used to modify it (1.10.38).

The ESP2/3 differences in summary:

	ESP 2.0	ESP 3.0
Subtelegram count	--	●
Receive signal strength (RSSI)	--	●
Upward compatible with 'Optional Data'	--	●
Data verification	Checksum	CRC8
UART Synchronization (packet detection)	2 bytes	6 bytes
Max. number of ESP packet types	8	256
Types of data	Radio, Command	Any type of data
Max. size of transferred data	28 bytes	65535 bytes
Communication speed	9600 baud	**57600 baud** 115200 baud 230400 baud 460800 baud

Table 1

1.3 Packet structure

ESP3 is a Point-to-Point protocol with a packet data structure.
This principle encapsulates actual user data (payload), Command, Event or Response messages.

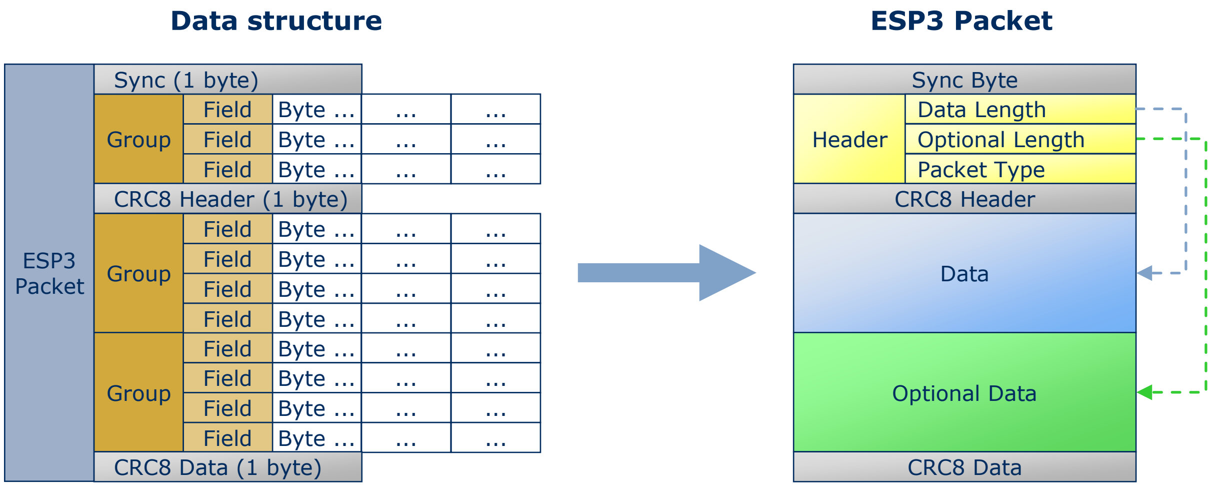


Figure 2

Every ESP3 packet consists of Header, Data and Optional Data.

The packet (frame) is divided into: Sync.-Byte (start), CRC8 for Header and CRC8 for Data (incl. Optional Data).

Every group consists of Fields, each with 1 or x bytes.

The ESP3 Header consists of the Fields:
- Data Length (number of bytes of the group Data)
- Optional Length (number of bytes of the group Optional Data)
- Packet Type (RADIO, RESPONSE, EVENT, COMMAND ...)

1.4 Upward compatibility

The ESP3 protocol is defined as a specific structure of Sync.-Byte, Header & CRC8, which should not be changed in future versions.

For each type of packet the content and the length of DATA is different.

Today's applications have to be compliant with later versions of the ESP3 protocol ensuring an upwards compatibility.

New software applications or devices might require the definition of new types of packet.

Existing packet types may be modified only via the field OPTIONAL_DATA. The field DATA is not to be changed.

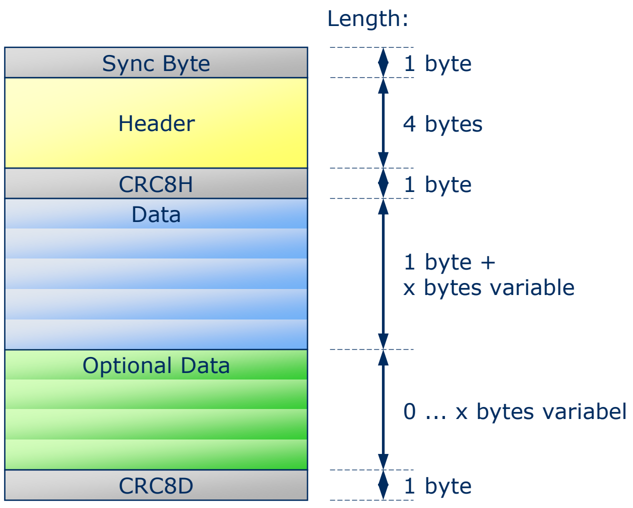


Figure 3

Existing devices will react as follows:

- Unknown packet types are confirmed with the RESPONSE message 'not supported' and will not be processed further.
- New fields in the Optional Data section of an existing packet type will be ignored; a RESPONSE message will not be sent.
- It is allowed to skip bytes (not transfer them) from optional fields when they are located at the end of the optional field.

Thus, backwards compatibility is secured.

1.5 UART framing

The UART of the EnOcean module has the framing: 8 data bits, no parity bit, one start bit (logical 0), one stop bit (logical 1). The line idle (≙ neutral) is logical 1 (standard).

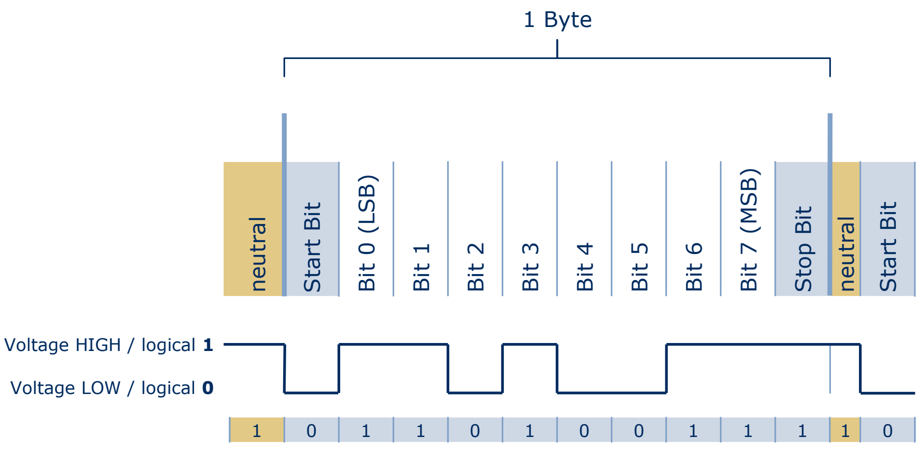


Figure 4

1.6 UART synchronization (packet detection)

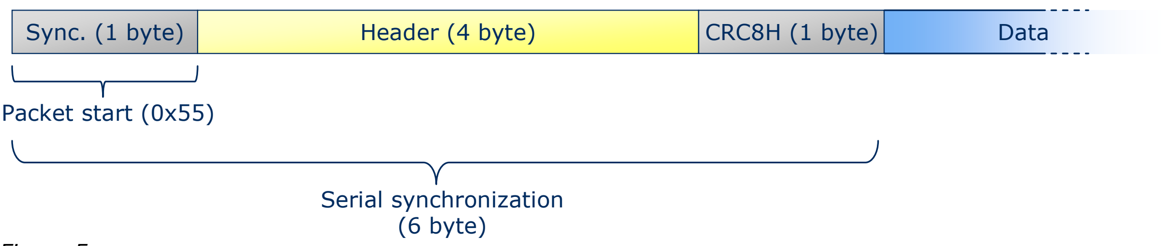


Figure 5

With ESP3 the reliability of the synchronization has been improved significantly:

As soon as a Sync.-Byte (value 0x55) is identified, the subsequent 4 byte-Header is compared with the corresponding CRC8H value.
If the result is a match the Sync.-Byte is correct. Consequently, the ESP3 packet is detected properly and the subsequent data will be passed.
If the Header does not match the CRC8H, the value 0x55 does not correspond to a Sync.-Byte. The next 0x55 within the data stream is picked and the verification is repeated.

The chapter 2.4 shows an example for a feasible implementation.

1.6.1 Packet description

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	Serial synchronization byte; always set to 0x55
Header	1	2	Data Length	0xnnnn	Specifies how many bytes in DATA must be interpreted
	3	1	Optional Length	0xnn	Specifies how many bytes in OPTIONAL_DATA must be interpreted
	4	1	Packet Type	0xnn	Specifies the packet type of DATA, respectively OPTIONAL_DATA
-	5	1	CRC8H	0xnn	CRC8 Header byte; calculated checksum for bytes: DATA_LENGTH, OPTIONAL_LENGTH and TYPE
Data	6	x	… … … … … …	… … … … … …	Contains the actual data payload with topics: - RawData (e.g. 1:1 radio telegram) - Function codes + optional parameters - Return codes + optional parameters - Event codes x = variable length of DATA / byte number
Optional Data	6+x	y	… … …	… … …	Contains additional data that extends the field DATA; y = variable length of OPTIONAL_DATA
-	6+x+y	1	CRC8D	0xnn	CRC8 Data byte; calculated checksum for whole byte groups: DATA and OPTIONAL_DATA

Table 2

1.6.2 Packet types

Depending on the field [Packet Type] a different kind of packet is transmitted.

Type No.	Value hex	Name	Description
0	0x00	---	Reserved
1	0x01	RADIO_ERP1	Radio telegram
2	0x02	RESPONSE	Response to any packet
3	0x03	RADIO_SUB_TEL	Radio subtelegram
4	0x04	EVENT	Event message
5	0x05	COMMON_COMMAND	Common command
6	0x06	SMART_ACK_COMMAND	Smart Ack command
7	0x07	REMOTE_MAN_COMMAND	Remote management command
8	0x08	---	Reserved for EnOcean
9	0x09	RADIO_MESSAGE	Radio message
10	0x0A	RADIO_ERP2	ERP2 protocol radio telegram
11 … 15	0x0A … 0F	---	Reserved for EnOcean
16	0x10	RADIO_802_15_4	802_15_4_RAW Packet
17	0x11	COMMAND_2_4	2.4 GHz Command
18 … 127	0x12 … 7F	---	Reserved for EnOcean
128…255	0x80 … FF	available	MSC and messages

Table 3

1.6.3 Direction of packet types

The function and the properties of a packet type determine the direction of the ESP3 data traffic, and whether a RESPONSE message is required or not.

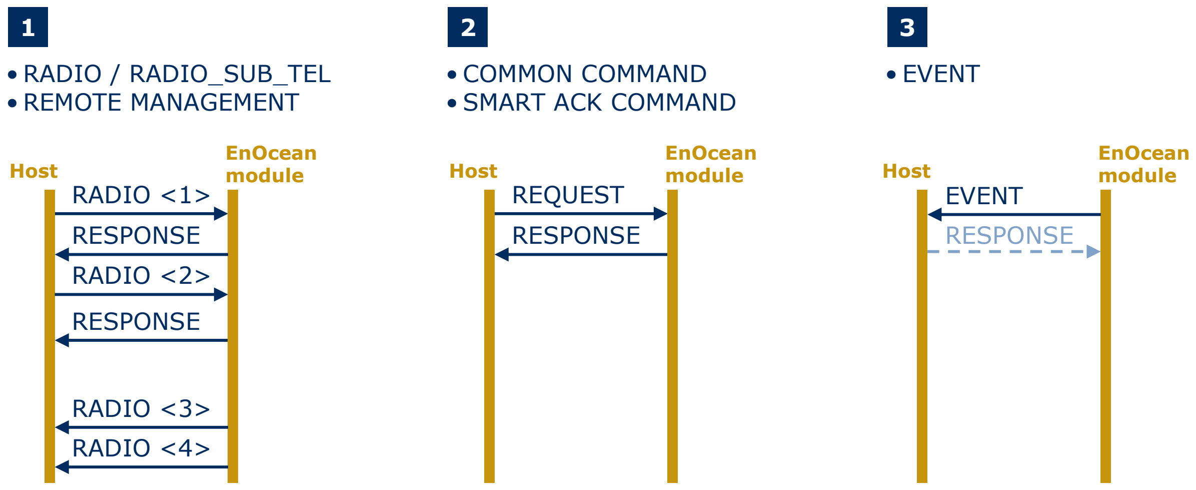


Figure 6

Case 1: ESP3 packets of the type RADIO_ERP1, RADIO_SUB_TEL or REMOTE_MAN pass bidirectionally across the serial interface. After sending a packet (host -> module) it is mandatory to wait for the RESPONSE message, only then the telegram is passed correctly via the radio interface.
After receiving (module -> host) a packet no RESPONSE is required (see RADIO_ERP1 no. <3> and <4>).

Case 2: Only a host sends a ESP3 COMMAND (COMMON, SMART ACK) to an EnOcean module. Each REQUEST is answered with a RESPONSE message (OK, error, etc.). The reverse direction module-to-host is not possible.

Case 3: Only an EnOcean module sends an EVENT to a host. The type of the EVENT defines whether a RESPONSE message is required or not.

1.6.4 ESP3 Timeout

A timeout in an ESP3 packet is defined as soon as the time between two characters exceeds 100ms.

If the answer time between REQUEST/EVENT and RESPONSE exceeds 500ms a timeout is identified as well.

1.7 Packet Type 1: RADIO_ERP1

1.7.1 Packet structure

The ERP1 radio telegram (raw data) is embedded into the ESP3 packet.
The actual user data (variable length) is a subset of the radio telegram.

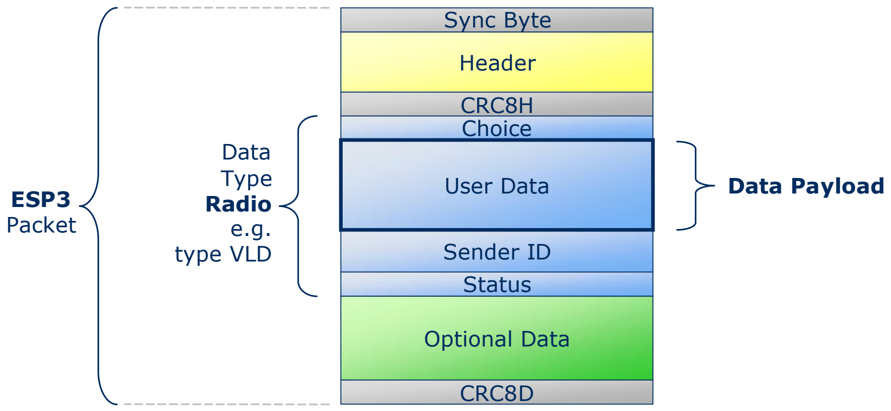


Figure 7

The following structure is applicable to all types of radio telegrams:

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0xnnnn	Variable length of radio telegram
	3	1	Optional Length	0x07	7 fields fixed
	4	1	Packet Type	0x01	RADIO_ERP1 = 1
-	5	1	CRC8H	0xnn	
Data	6	x	… …	… …	Radio telegram without checksum/CRC x = variable length / size
Optional Data	6+x	1	SubTelNum	0xnn	Number of subtelegram; Send: 3 / receive: 1 … y
	7+x	4	Destination ID	0xnnnnnnnn	Broadcast radio: FF FF FF FF ADT radio: Destination ID (= address)
	11+x	1	dBm	0xnn	Send case: FF Receive case: best RSSI value of all received subtelegrams (value decimal without minus)
	12+x	1	SecurityLevel	0x0n	0 = telegram unencrypted n = type of encryption (not supported any more)
-	13+x	1	CRC8D	0xnn	CRC8 Data byte; calculated checksum for whole byte groups: DATA and OPTIONAL_DATA

Table 4

When receiving a telegram, no RESPONSE has to be sent. When sending a telegram, a RESPOND has to be expected. In this case, the following **RESPONSE** message gives the return codes:
00 RET_OK
02 RET_NOT_SUPPORTED
03 RET_WRONG_PARAM
05 RET_LOCK_SET

Since no additional data are included, that have to be described, the standard RESPONSE structure is described in chapter 1.7.5

1.7.2 Radio variants (examples)

Out of the numerous variants of the RADIO_ERP1 packet, described in other documents, only a few examples are described here. These examples describe the structure of DATA on the ESP3 interface. On the radio link specifically the ADT telegram has a different structure (e.g. R-ORG_EN).

RADIO (VLD)

Group	Offset	Size	Field	Value hex	Description
Data	6	1	R-ORG	0xD2	Radio type VLD = D2
	7	x	User Data	0xnn...0xnn	**1 … 14** byte data payload
	7+x	4	Sender ID	0xnnnnnnnn	Unique device sender ID
	11+x	1	Status	0xnn	Telegram control bits – used in case of repeating, switch telegram encapsulation, checksum type identification

Table 5

RADIO (ADT) Addressing Destination Telegram

Group	Offset	Size	Field	Value hex	Description
Data	6	1	R-ORG	0xA6	Radio type, e.g. ADT = A6, 4BS = 0xA5
	7	x	User Data	0xnn...0xnn	**1 … 9** byte data payload
	7+x	4	Sender ID	0xnnnnnnnn	Unique device sender ID
	11+x	1	Status	0xnn	Telegram control bits – used in case of repeating, switch telegram encapsulation, checksum type identification
Optional Data	6+x	1	SubTelNum	0xnn	Number of sub telegram; Send: 3 / receive: 1 … y
	7+x	4	Destination ID	0xnnnnnnnn	ADT radio: Destination ID (= address)
	11+x	1	dBm	0xnn	Send case: FF Receive case: best RSSI value of all received subtelegrams (value decimal without minus)
	12+x	1	SecurityLevel	0x0n	0 = telegram unencrypted n = type of encryption (not supported any more)

Table 6

RADIO (4BS) / EEP profile 07-02-14

Group	Offset	Size	Field	Value hex	Description
Data	6	1	R-ORG	0xA5	Radio type 4BS
	7	1	Data Byte 3	0x00	Unused in this EEP profile
	8	1	Data Byte 2	0x00	Unused in this EEP profile
	9	1	Data Byte 1	0xnn	Temperature value 255 … 0
	10	1	Data Byte 0	0b0000**n**000	DB_0.BIT 3 = Learn Bit Normal mode = 1 / Teach In = 0
	11	4	Sender ID	0xnnnnnnnn	Unique device sender ID
	15	1	Status	0xnn	Telegram control bits – used in case of repeating, switch telegram encapsulation, checksum type identification

Table 7

Packet Type 2: RESPONSE

1.7.3 Packet structure

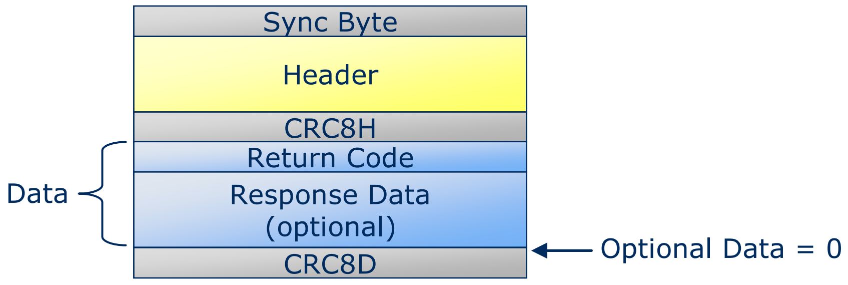


Figure 8

The properties of the preceding command and the re-delivered return-code determine whether optional response data are included, or only the return code itself.

1.7.4 List of Return Codes

Code	Name	Description
00	RET_OK	OK ... command is understood and triggered
01	RET_ERROR	There is an error occurred
02	RET_NOT_SUPPORTED	The functionality is not supported by that implementation
03	RET_WRONG_PARAM	There was a wrong parameter in the command
04	RET_OPERATION_DENIED	Example: memory access denied (code-protected)
05	RET_LOCK_SET	Duty cycle lock
06	RET_BUFFER_TO_SMALL	The internal ESP3 buffer of the device is too small, to handle this telegram
07	RET_NO_FREE_BUFFER	Currently all internal buffers are used.
> 128	---	Return codes greater than 0x80 are used for commands with special return information, not commonly useable.

Table 8

1.7.5 Example: RET_OK (without response data)

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0001	Data = 1 byte
	3	1	Optional Length	0x00	Optional Data = 0 byte
	4	1	Packet Type	0x02	RESPONSE = 2
-	5	1	CRC8H	0xnn	
Data	6	1	Return Code	0x00	RET_OK
-	7	1	CRC8D	0xnn	

Table 9

Specific variants of the response messages are described in the chapter of the command.

1.8 Packet Type 3: RADIO_SUB_TEL

This ESP3 packet type is functionality internal to EnOcean; it is applied for e.g. diagnosis or statistics. The packet design corresponds to the type RADIO_ERP1. The content of the OPTIONAL_DATA is altered slightly.

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0xnnnn	Variable length of radio subtelegram
	3	1	Optional Length	0xnn	9 + x + 3*s bytes x = variable length radio subtelegram s = number of subtelegram
	4	1	Packet Type	0x03	RADIO_SUB_TEL = 3
-	5	1	CRC8H	0xnn	
Data	6	x	… …	… …	Radio telegram without checksum/CRC x = variable length / size
Optional Data	6+x	1	SubTelNum	0xnn	actual sequence number of subtelegrams (1 … y); Repeated telegrams will be added
	7+x	4	Destination ID	0xnnnnnnnn	Broadcast radio: FF FF FF FF ADT radio: Destination ID (= address)
	11+x	1	dBm	0xnn	Send case: FF Receive case: best RSSI value of all received subtelegrams (value decimal without minus)
	12+x	1	SecurityLevel	0x0n	0 = telegram unencrypted n = type of encryption (not supported any more)
	13+x	2	TimeStamp	0xnnnn	Timestamp of 1st subtelegram is the system timer tick [ms] (2 byte lower address)
	15+x	1	Tick SubTel		Relative time [ms] of each subtelegram in relation to the TimeStamp
	15+x +1*s	1	dBm SubTel		RSSI value of each subtelegram
	15+x +2*s	1	Status SubTel	0xnn	Telegram control bits of each subtelegram – used in case of repeating, switch telegram encapsulation, checksum type identification
-	15+x +3*s	1	CRC8D	0xnn	

$\underbrace{\text{TimeStamp … Status SubTel}}_{s}$

Table 10

Every received subtelegram has the group **s** with fields in the order: Tick SubTel, dBm SubTel, Status SubTel (s = also number of subtelegram / multiplier to calculate the offset).

When receiving a telegram, no RESPONSE has to be sent. When sending a telegram, a RESPOND has to be expected. In this case, the following **RESPONSE** message gives the return codes:
00 RET_OK
02 RET_NOT_SUPPORTED
03 RET_WRONG_PARAM

05 RET_LOCK_SET

Since no additional data are included, that have to be described, the standard RESPONSE structure is described in chapter 1.7.5

1.9 Packet Type 4: EVENT

1.9.1 Structure

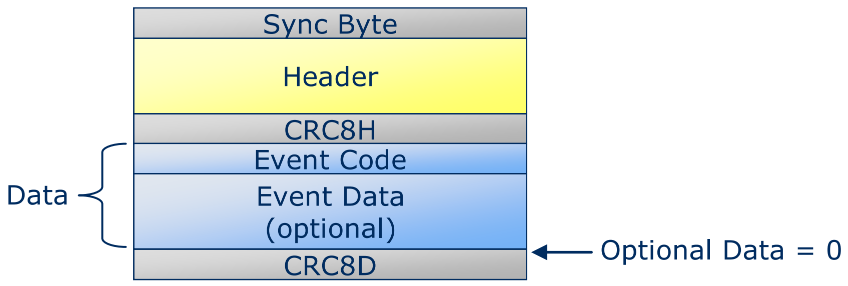


Figure 9

An EVENT is primarily a confirmation for processes and procedures, incl. specific data content.

1.9.2 List of EVENT Codes

Code	Name	Description
01	SA_RECLAIM_NOT_SUCCESSFUL	Informs the backbone of a Smart Ack Client to not successful reclaim.
02	SA_CONFIRM_LEARN	Used for SMACK to confirm/discard learn in/out
03	SA_LEARN_ACK	Inform backbone about result of learn request
04	CO_READY	Inform backbone about the readiness for operation
05	CO_EVENT_SECUREDEVICES	Informs about a secure device
06	CO_DUTYCYCLE_LIMIT	Informs about duty cycle limit
07	CO_TRANSMIT_FAILED	Informs that the device was not able to send a telegram.

Table 11

1.9.3 Code 01: SA_RECLAIM_NOT_SUCCESSFUL

Function: Informs the backbone of a Smart Ack Client to not successful reclaim.

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0001	1 byte
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x04	EVENT = 4
-	5	1	CRC8H	0xnn	
Data	6	1	Event Code	0x01	SA_RECLAIM_NOT_SUCCESSFUL = 1
-	7	1	CRC8D	0xnn	

Table 12

Following described **RESPONSE** applies to return codes:
00: RET_OK
01: RET_ERROR
02: RET_NOT_SUPPORTED
03: RET_WRONG_PARAM

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0001	1 byte
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x02	RESPONSE = 2
-	5	1	CRC8H	0xnn	
Data	6	1	Return Code	0xnn	00, 01, 02, 03
-	7	1	CRC8D	0xnn	

Table 13

1.9.4 Code 02: SA_CONFIRM_LEARN

Function: Request to backbone controller how to handle the received learn request.

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0011	17 bytes
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x04	EVENT = 4
-	5	1	CRC8H	0xnn	
Data	6	1	Event Code	0x02	SA_CONFIRM_LEARN = 2
	7	1	Priority of the postmaster candidate	0xnn	Already post master 0b xxxx 1xxx Place for mailbox 0b xxxx x1xx Good RSSI 0b xxxx xx1x Local 0b xxxx xxx1
	8	1	2^2 ... 2^0: Manufacturer ID 2^7 ... 2^3: Res.	0b00000nnn	nnn = Most significant 3 bits of the Manufacturer ID 00000 = reserved
	9	1	Manufacturer ID	0xnn	Least significant bits of the Manufact. ID
	10	3	EEP	0xnnnnnn	Code of used EEP profile
	13	1	RSSI	0xnn	Signal strength; Send case: FF Receive case: actual RSSI
	14	4	Postmaster Candidate ID	0xnnnnnnnn	Device ID of the Post master candidate
	18	4	Smart Ack ClientID	0xnnnnnnnn	This sensor would be Learn IN
	22	1	Hop Count	0xnn	Numbers of repeater hop
-	23	1	CRC8D	0xnn	

Table 14

Following described **RESPONSE** applies to return code:
00: RET_OK

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0004	4 byte
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x02	RESPONSE = 2
-	5	1	CRC8H	0xnn	
Data	6	1	Return Code	0x00	RET_OK = 0
	7	2	Response time	0xnnnn	Response time for Smart Ack Client in ms in which the controller can prepare the data and send it to the postmaster. Only actual if learn return code is Learn IN
	9	1	Confirm code	0xnn	0x00 Learn IN 0x11 Discard Learn IN, EEP not accepted 0x12 Discard Learn IN, PM has no place for further mailbox 0x13 Discard Learn IN, Controller has no place for new sensor 0x14 Discard Learn IN, RSSI was not good enough 0x20 Learn OUT 0xFF Function not supported
-	10	1	CRC8D	0xnn	

Table 15

For **RESPONSE** with return codes: 01 RET_ERROR, 02 RET_NOT_SUPPORTED, 03 RET_WRONG_PARAM is the structure described by the chapter: 1.7.5

1.9.5 Code 03: SA_LEARN_ACK

Function: Informs Smart Ack client about the result of a previous sent learn request.

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0004	4 bytes
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x04	EVENT = 4
-	5	1	CRC8H	0xnn	
Data	6	1	Event Code	0x03	SA_LEARN_ACK = 3
	7	2	Response time	0xnnnn	Response time for Smart Ack Client in ms in which the controller can prepare the data and send it to the postmaster. Only actual if learn return code is Learn IN
	9	1	Confirm code	0xnn	0x00 Learn IN 0x11 Discard Learn IN, EEP not accepted 0x12 Discard Learn IN, PM has no place for further MB 0x13 Discard Learn IN, Controller has no place for new sensor 0x14 Discard Learn IN, RSSI was not good enough 0x20 Learn OUT
-	10	1	CRC8D	0xnn	

Table 16

In this case, the following **RESPONSE** message gives the return codes:
00 RET_OK
02 RET_NOT_SUPPORTED
03 RET_WRONG_PARAM
Since no additional data are included, that have to be described, the standard RESPONSE structure is described in chapter 1.7.5

1.9.6 Code 04: CO_READY

Function: Informs backbone about the readiness for operation.

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0002	2 bytes
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x04	EVENT = 4
-	5	1	CRC8H	0xnn	
Data	6	1	Event Code	0x04	CO_READY = 4
	7	1	Reset Cause	0xnn	**00** = Voltage supply drop or indicates that VDD > VON **01** = Reset caused by usage of the reset pin (is set also after downloading the program with the programmer) **02** = Watchdog timer counter reached the timer period **03** = Flywheel timer counter reached the timer period **04** = Parity error **05** = HW Parity error in the Internal or External Memory **06** = A memory request from the CPU core does not correspond to any valid memory location. This error may be caused by a S/W malfunction. **07** = Wake-up pin 0 activated **08** = Wake-up pin 1 activated **09** = Unknown reset source – reset reason couldn't be detected
-	8	1	CRC8D	0xnn	

Table 17

This EVENT does not require any RESPONSE message.

1.9.7 Code 05: CO_EVENT_SECUREDEVICES

Function: Informs backbone about events regarding a secure device

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0006	6 bytes
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x04	EVENT = 4
-	5	1	CRC8H	0xnn	
Data	6	1	Event Code	0x05	CO_EVENT_SECUREDEVICES = 5
	7	1	Event Cause	0xnn	**00** = Teach in failed, because no more space available **01** = reserved **02** = Resynchronization attempt with wrong private key **03** = Configured count of telegrams with wrong CMAC received **04** = Teach-In failed. Telegram corrupted. **05** = PSK Teach-In failed. No PSK is set for the device **06** = Teach-In failed. Trying to teach-in without Pre-Shared Key even if the PSK is set for the device **07...255** = reserved
	8	4	Device ID	0xnnnnnnnn	Device ID
-	12	1	CRC8D	0xnn	

Table 18

This EVENT does not require any RESPONSE message.

1.9.8 Code 06: CO_DUTYCYCLE_LIMIT

Function: Informs backbone about duty cycle limit

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0002	2 bytes
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x04	EVENT = 4
-	5	1	CRC8H	0xnn	
Data	6	1	Event Code	0x06	CO_DUTYCYCLE_LIMIT = 6
	7	1	Event Cause	0xnn	**00** = Duty cycle limit released, it's possible to send telegrams **01** = Duty cycle limit reached, no more telegram will be sent **02...255** = reserved
-	8	1	CRC8D	0xnn	

Table 19

This EVENT does not require any RESPONSE message.

1.9.9 Code 07: CO_TRANSMIT_FAILED

Function: Informs backbone about duty cycle limit

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0002	2 bytes
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x04	EVENT = 4
-	5	1	CRC8H	0xnn	
Data	6	1	Event Code	0x07	CO_TRANSMIT_FAILED = 7
	7	1	Event Cause	0xnn	**00** = CSMA failed, channel was never free **01** = No Acknowledge received, telegram was transmitted, but no ack received. **02...255** = reserved
-	8	1	CRC8D	0xnn	

Table 20

This EVENT does not require any RESPONSE message.

1.10 Packet Type 5: COMMON_COMMAND

1.10.1 Structure

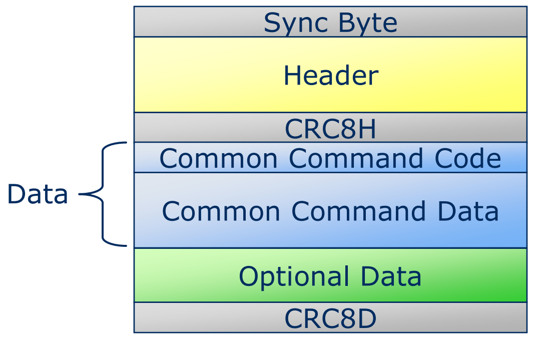


Figure 10

1.10.2 List of COMMON_COMMAND Codes

Code	Function Name	Description
01	CO_WR_SLEEP	Order to enter in energy saving mode
02	CO_WR_RESET	Order to reset the device
03	CO_RD_VERSION	Read the device (SW) version / (HW) version, chip ID etc.
04	CO_RD_SYS_LOG	Read system log from device databank
05	CO_WR_SYS_LOG	Reset System log from device databank
06	CO_WR_BIST	Perform Flash BIST operation
07	CO_WR_IDBASE	Write ID range base number
08	CO_RD_IDBASE	Read ID range base number
09	CO_WR_REPEATER	Write Repeater Level off,1,2
10	CO_RD_REPEATER	Read Repeater Level off,1,2
11	CO_WR_FILTER_ADD	Add filter to filter list
12	CO_WR_FILTER_DEL	Delete filter from filter list
13	CO_WR_FILTER_DEL_ALL	Delete all filter
14	CO_WR_FILTER_ENABLE	Enable/Disable supplied filters
15	CO_RD_FILTER	Read supplied filters
16	CO_WR_WAIT_MATURITY	Waiting till end of maturity time before received radio telegrams will transmitted
17	CO_WR_SUBTEL	Enable/Disable transmitting additional subtelegram info
18	CO_WR_MEM	Write x bytes of the Flash, XRAM, RAM0 ….
19	CO_RD_MEM	Read x bytes of the Flash, XRAM, RAM0 ….
20	CO_RD_MEM_ADDRESS	Feedback about the used address and length of the config area and the Smart Ack Table
21	CO_RD_SECURITY	Read own security information (level, key)
22	CO_WR_SECURITY	Write own security information (level, key)
23	CO_WR_LEARNMODE	Enable/disable learn mode
24	CO_RD_LEARNMODE	Read learn mode
25	CO_WR_SECUREDEVICE_ADD	Add a secure device
26	CO_WR_SECUREDEVICE_DEL	Delete a secure device
27	CO_RD_SECUREDEVICE_ BY_INDEX	Read secure device by index
28	CO_WR_MODE	Sets the gateway transceiver mode
29	CO_RD_NUMSECUREDEVICES	Read number of taught in secure devices
30	CO_RD_SECUREDEVICE_	Read secure device by ID

	BY_ID	
31	CO_WR_SECUREDEVICE_ ADD_PSK	Add Pre-shared key for inbound secure device.
32	CO_WR_SECUREDEVICE_ SENDTEACHIN	Send secure Teach-In message.
33	CO_WR_TEMPORARY_RLC_ WINDOW	Set the temporary rolling-code window for every taught-in device
34	CO_RD_SECUREDEVICE_PSK	Read PSK
35	CO_RD_DUTYCYCLE_LIMIT	Read parameters of actual duty cycle limit

Table 21

1.10.3 Code 01: CO_WR_SLEEP

Function: Order to enter the energy saving mode.

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0005	5 bytes
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x05	COMMON_COMMAND = 5
-	5	1	CRC8H	0xnn	
Data	6	1	COMMAND Code	0x01	CO_WR_SLEEP = 1
	7	4	Deep sleep period	0x00nnnnnn	Period in 10 ms units 00000000 = default max. value = max. data range 00 FF FF FF (~ 46h); After waking up, the module generate an internal hardware reset
-	11	1	CRC8D	0xnn	

Table 22

In this case, the following **RESPONSE** message gives only the return codes:
00 RET_OK
02 RET_NOT_SUPPORTED
Since no additional data are included which require description the standard RESPONSE structure is detailed in chapter 1.7.5

1.10.4 Code 02: CO_WR_RESET

Function: Order to reset the device.

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0001	1 byte
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x05	COMMON_COMMAND = 5
-	5	1	CRC8H	0xnn	
Data	6	1	COMMAND Code	0x02	CO_WR_RESET = 2
-	7	1	CRC8D	0xnn	

Table 23

In this case, the following **RESPONSE** message gives only the return codes:
00 RET_OK
01 RET_ERROR
02 RET_NOT_SUPPORTED
Since no additional data are included which require description the standard RESPONSE structure is detailed in chapter 1.7.5

1.10.5 Code 03: CO_RD_VERSION

Function: Read the device SW version / HW version, chip-ID, etc.

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0001	1 byte
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x05	COMMON_COMMAND = 5
-	5	1	CRC8H	0xnn	
Data	6	1	COMMAND Code	0x03	CO_RD_VERSION = 3
-	7	1	CRC8D	0xnn	

Table 24

Following described **RESPONSE** applies to return code:
00: RET_OK

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0021	33 bytes
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x02	RESPONSE = 2
-	5	1	CRC8H	0xnn	
Data	6	1	Return Code	0x00	RET_OK = 0
	7	4	APP version	0xnnnnnnnn	Application Byte 1: Main version Byte 2: Beta version Byte 3: Alpha version Byte 4: Build
	11	4	API version	0xnnnnnnnn	Application Programming Interface Byte 1: Main version Byte 2: Beta version Byte 3: Alpha version Byte 4: Build
	15	4	Chip ID	0xnnnnnnnn	Unique ID
	19	4	Chip Version	0xnnnnnnnn	Reserved for internal use
	23	16	App. description	char. ASCII	8 bit ASCII / 16 characters; Null-terminated string
-	39	1	CRC8D	0xnn	

Table 25

For **RESPONSE** with return code:
02 RET_NOT_SUPPORTED is the standard structure described by the chapter 1.7.5

1.10.6 Code 04: CO_RD_SYS_LOG

Function: Read System Log from device databank.

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0001	1 byte
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x05	COMMON_COMMAND = 5
-	5	1	CRC8H	0xnn	
Data	6	1	COMMAND Code	0x04	CO_RD_SYS_LOG = 4
-	7	1	CRC8D	0xnn	

Table 26

Following described **RESPONSE** applies to return code:
00: RET_OK

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0xnnnn	1+x bytes
	3	1	Optional Length	0xnn	y bytes
	4	1	Packet Type	0x02	RESPONSE = 2
-	5	1	CRC8H	0xnn	
Data	6	1	Return Code	0x00	RET_OK = 0
	7	x	API Log entry 000 API Log entry 001 API Log entry 002 … … …	0xnn 0xnn 0xnn … … …	Log entry 000 - xxx in DATA: Log counter of API
Optional Data	7+x	y	APP Log entry 000 APP Log entry 001 APP Log entry 002 … … …	0xnn 0xnn 0xnn … … …	Log entry 000 - xxx in OPTIONAL_DATA: Log counter of APP
-	7+x+y	1	CRC8D	0xnn	

Table 27

After a reset, the counters starts with FF and decrement with each new EVENT down to 00 and will stopped. With a reset command the counter starts again with FF.

For **RESPONSE** with return code:
02 RET_NOT_SUPPORTED is the standard structure described by the chapter 1.7.5

1.10.7 Code 05: CO_WR_SYS_LOG

Function: Reset System Log from device databank.

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0001	1 byte
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x05	COMMON_COMMAND = 5
-	5	1	CRC8H	0xnn	
Data	6	1	COMMAND Code	0x05	CO_WR_SYS_LOG = 5
-	7	1	CRC8D	0xnn	

Table 28

In this case, the following **RESPONSE** message gives only the return codes:
00 RET_OK
02 RET_NOT_SUPPORTED
Since no additional data are included which require description the standard RESPONSE structure is detailed in chapter 1.7.5

1.10.8 Code 06: CO_WR_BIST

Function: Perform Flash BIST operation (Built-in-self-test).

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0001	1 byte
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x05	COMMON_COMMAND = 5
-	5	1	CRC8H	0xnn	
Data	6	1	COMMAND Code	0x06	CO_WR_BIST = 6
-	7	1	CRC8D	0xnn	

Table 29

Following described **RESPONSE** applies to return code:
00: RET_OK

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0002	2 bytes
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x02	RESPONSE = 2
-	5	1	CRC8H	0xnn	
Data	6	1	Return Code	0x00	RET_OK = 0
	7	1	BIST result	0xnn	BIST OK = 0, BIST failed = other value
-	8	1	CRC8D	0xnn	

Table 30

For **RESPONSE** with return code:
02 RET_NOT_SUPPORTED is the standard structure described by the chapter 1.7.5

1.10.9 Code 07: CO_WR_IDBASE

Function: Write ID range base number.

IMPORTANT: This function can only be called 10 times to change the base ID. There is no possibility to reset this constraint. Also power off/on will not allow more than 10 changes!

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0005	5 bytes
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x05	COMMON_COMMAND = 5
-	5	1	CRC8H	0xnn	
Data	6	1	COMMAND Code	0x07	CO_WR_IDBASE = 7
	7	4	Base ID	0xFFnnnnnn	Range between 0xFF800000 and 0xFFFFFF80
-	11	1	CRC8D	0xnn	

Table 31

RESPONSE:

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0001	1 byte
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x02	RESPONSE = 2
-	5	1	CRC8H	0xnn	
Data	6	1	Return Code	0xnn	RET_OK = 0x00 RET_NOT_SUPPORTED = 0x02 FLASH_HW_ERROR = 0x82 The write/erase/verify process failed, the flash page seems to be corrupted BASEID_OUT_OF_RANGE = 0x90 BASEID_MAX_REACHED = 0x91 (BaseID was changed 10 times, no more changes are allowed)
-	7	1	CRC8D	0xnn	

Table 32

1.10.10 Code 08: CO_RD_IDBASE

Function: Read ID range base number.

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0001	1 byte
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x05	COMMON_COMMAND = 5
-	5	1	CRC8H	0xnn	
Data	6	1	COMMAND Code	0x08	CO_RD_IDBASE = 8
-	7	1	CRC8D	0xnn	

Table 33

Following described **RESPONSE** applies to return code:
00: RET_OK

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0005	5 bytes
	3	1	Optional Length	0x01	1 byte
	4	1	Packet Type	0x02	RESPONSE = 2
-	5	1	CRC8H	0xnn	
Data	6	1	Return Code	0x00	RET_OK = 0
	7	4	Base ID	0xFFnnnnnn	Range between 0xFF800000 and 0xFFFFFF80
Optional Data	8	1	Remaining write cycles for Base ID	0xnn	Remaining write cycles for Base ID
-	9	1	CRC8D	0xnn	

Table 34

For **RESPONSE** with return code:
02 RET_NOT_SUPPORTED is the standard structure described by the chapter 1.7.5

1.10.11 Code 09: CO_WR_REPEATER

Function: Write Repeater Level OFF, 1, 2.

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0003	3 bytes
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x05	COMMON_COMMAND = 5
-	5	1	CRC8H	0xnn	
Data	6	1	COMMAND Code	0x09	CO_WR_REPEATER = 09
	7	1	REP_ENABLE	0x00...0x02	Repeater OFF = 0, ON all = 1, ON filtered = 2
	8	1	REP_LEVEL	0x00...0x02	When Repeater OFF must be 0, when ON then 1 for Level-1 , 2 for Level-2
-	9	1	CRC8D	0xnn	

Table 35

In this case, the following **RESPONSE** message gives only the return codes:
00 RET_OK
02 RET_NOT_SUPPORTED
03 RET_WRONG_PARAM

Since no additional data are included which require description the standard RESPONSE structure is detailed in chapter 1.7.5

1.10.12 Code 10: CO_RD_REPEATER

Function: Read Repeater Level OFF, 1, 2.

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0001	1 byte
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x05	COMMON_COMMAND = 5
-	5	1	CRC8H	0xnn	
Data	6	1	COMMAND Code	0x0A	CO_RD_REPEATER = 10
-	7	1	CRC8D	0xnn	

Table 36

Following described **RESPONSE** applies to return code:
00: RET_OK

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0003	3 bytes
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x02	RESPONSE = 2
-	5	1	CRC8H	0xnn	
Data	6	1	Return Code	0x00	RET_OK = 0
	7	1	REP_ENABLE	0x00…0x02	Repeater OFF = 0, ON all = 1, ON filtered = 2
	8	1	REP_LEVEL	0x00…0x02	Repeater OFF = 0, 1 for Level-1, 2 for Level-2
-	9	1	CRC8D	0xnn	

Table 37

For **RESPONSE** with return code:
02 RET_NOT_SUPPORTED is the standard structure described by the chapter: 1.7.5

1.10.13 Code 11: CO_WR_FILTER_ADD

Function: Add filter to filter list.

The FILTER module is used to filter received telegrams according ID, R-ORG or received dBm value. Adding a filter, activates the filter module.

When the filter is active telegrams which matches to the data specified with the filter are dropped and the application does not receive them.

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0007	7 bytes
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x05	COMMON_COMMAND = 5
-	5	1	CRC8H	0xnn	
Data	6	1	COMMAND Code	0x0B	CO_WR_FILTER_ADD = 11
	7	1	Filter type	0x00…0x03	Device source ID = 0, R-ORG = 1, dBm = 2, destination ID = 3
	8	4	Filter value	0xnnnnnnnn	Value of filter function 'compare': - device source or destination ID - R-ORG - dBm value RSSI of radio telegram (unsigned, but interpreted as negative dBm value)
	12	1	Filter kind	0x00 0x80 0x40 0xC0	blocks radio interface = 0x00 apply radio interface = 0x80 blocks filtered repeating = 0x40 apply filtered repeating = 0xC0
-	13	1	CRC8D	0xnn	

Table 38

In this case, the following **RESPONSE** message gives the return codes:
00 RET_OK
01 RET_ERROR (memory space full)
02 RET_NOT_SUPPORTED
03 RET_WRONG_PARAM
Since no additional data are included, that have to be described, the standard RESPONSE structure is described in chapter: 1.7.5

Some examples for filters:

```
//BLOCKS specified ID
Filter_type = 0x0 (ID)
Filter_value = 0x12345678 (device source ID)
Filter_kind = 0x00 (block)

//BLOCKS all other IDs besides specified ID
Filter_type = 0x00 (ID)
Filter_value = 0x12345678 (device source ID)
Filter_kind = 0x80 (apply)

//BLOCKS telegrams with specified R-ORG
Filter_type = 0x01 (R-ORG)
Filter_value = 0xA5 (4BS)
Filter_kind = 0x00 (block)

//BLOCKS all other telegrams besides telegrams with specified R-ORG
Filter_type = 0x01 (R-ORG)
Filter_value = 0xA5 (4BS)
Filter_kind = 0x80 (apply)

//BLOCKS signals weaker than -70dBm
Filter_type = 0x02 (dBm)
Filter_value = 0x00000046 (dec 70)
Filter_kind = 0x00 (block)

//BLOCKS signals stronger than -70dBm
Filter_type = 0x02 (dBm)
Filter_value = 0x00000046 (dec 70)
Filter_kind = 0x80 (apply)


//Repeats only specified ID (when filtered repeating is ON)
Filter_type = 0x00 (ID)
Filter_value = 0x12345678 (device source ID)
Filter_kind = 0xC0 (apply for filtered repeating)

//Does not repeat telegrams with specified R-ORG (when filtered repeating is ON)
Filter_type = 0x01 (R-ORG)
Filter_value = 0xA5 (4BS)
Filter_kind = 0x40 (block for filtered repeating)

// Does not repeat signals stronger than -70dBm (when filtered repeating is ON)
Filter_type = 0x02 (dBm)
Filter_value = 0x00000046 (dec 70)
Filter_kind = 0xC0 (apply for filtered repeating)
```

1.10.14 Code 12: CO_WR_FILTER_DEL

Function: Delete filter from filter list.

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0006	6 bytes
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x05	COMMON_COMMAND = 5
-	5	1	CRC8H	0xnn	
Data	6	1	COMMAND Code	0x0C	CO_WR_FILTER_DEL = 12
	7	1	Filter type	0x00…0x03	Device source ID = 0, R-ORG = 1, dBm = 2, destination ID = 3
	8	4	Filter value	0xnnnnnnnn	Value of filter function 'compare': - device source or destination ID - R-ORG - dBm value RSSI of radio telegram (unsigned, but interpreted as negative dBm value)
-	12	1	CRC8D	0xnn	

Table 39

In this case, the following **RESPONSE** message gives the return codes:
00 RET_OK
01 RET_ERROR
02 RET_NOT_SUPPORTED
03 RET_WRONG_PARAM
Since no additional data are included which require description the standard RESPONSE structure is detailed in chapter: 1.7.5

1.10.15 Code 13: CO_WR_FILTER_DEL_ALL

Function: Delete all filters from filter list.

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0001	1 byte
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x05	COMMON_COMMAND = 5
-	5	1	CRC8H	0xnn	
Data	6	1	COMMAND Code	0x0D	CO_WR_FILTER_DEL = 13
-	7	1	CRC8D	0xnn	

Table 40

In this case, the following **RESPONSE** message gives only the return codes:
00 RET_OK
02 RET_NOT_SUPPORTED
Since no additional data are included which require description the standard RESPONSE structure is detailed in chapter: 1.7.5

1.10.16 Code 14: CO_WR_FILTER_ENABLE

Function: Enable/Disable all supplied filters.

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0003	3 bytes
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x05	COMMON_COMMAND = 5
-	5	1	CRC8H	0xnn	
Data	6	1	COMMAND Code	0x0E	CO_WR_FILTER_ENABLE = 14
	7	1	Filter ON/OFF	0x00 0x01	All radio interface filter disable = 0 (OFF) All radio interface filter enable = 1 (ON)
	8	1	Filter Operator	0x00 0x01 0x08 0x09	OR composition of all filters = 0 AND composition of all filters = 1 OR for radio interface filters; AND for filtered repeating filters = 8 AND for radio interface filters; OR for filtered repeating filters = 9
-	9	1	CRC8D	0xnn	

Table 41

In this case, the following **RESPONSE** message gives the return codes:
00 RET_OK
02 RET_NOT_SUPPORTED
03 RET_WRONG_PARAM
Since no additional data are included which require description the standard RESPONSE structure is detailed in chapter: 1.7.5

1.10.17 Code 15: CO_RD_FILTER

Function: Read supplied filters.

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0001	1 byte
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x05	COMMON_COMMAND = 5
-	5	1	CRC8H	0xnn	
Data	6	1	COMMAND Code	0x0F	CO_RD_FILTER = 15
-	7	1	CRC8D	0xnn	

Table 42

Following described **RESPONSE** applies to return code:
00: RET_OK

Group	Offset	Size	Field	Value hex	Description	
-	0	1	Sync. Byte	0x55		
Header	1	2	Data Length	0xnnnn	1 + 5*f bytes (f = number of filters)	
	3	1	Optional Length	0x00	0 byte	
	4	1	Packet Type	0x02	RESPONSE = 2	
-	5	1	CRC8H	0xnn		
Data	6	1	Return Code	0x00	RET_OK = 0	
	7+5*f	1	Filter type	0xnn	Device ID = 0, R-ORG = 1, dBm = 2	**f**
	8+5*f	4	Filter value	0xnnnnnnnn	Value of filter function 'compare': - device ID - R-ORG - RSSI of radio telegram in dBm	**f**
-	12+5*f	1	CRC8D	0xnn		

Table 43

Every supplied filter has the group **f** with fields in the order: filter type, filter value.

For **RESPONSE** with return code:
02 RET_NOT_SUPPORTED is the standard structure described by the chapter 1.7.5

1.10.18 Code 16: CO_WR_WAIT_MATURITY

Function: Waiting till end of maturity time before received radio telegrams will transmit.

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0002	2 bytes
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x05	COMMON_COMMAND = 5
-	5	1	CRC8H	0xnn	
Data	6	1	COMMAND Code	0x10	CO_WR_WAIT_MATURITY = 16
	7	1	Wait End Maturity	0xnn	0: Radio telegrams are send immediately 1: Radio telegrams are send after the maturity time is elapsed
-	8	1	CRC8D	0xnn	

Table 44

In this case, the following **RESPONSE** gives the return codes:
00 RET_OK
02 RET_NOT_SUPPORTED
03 RET_WRONG_PARAM

Since no additional data are included which require description the standard RESPONSE structure is detailed in chapter 1.7.5

1.10.19 Code 17: CO_WR_SUBTEL

Function: Enable/Disable transmitting additional subtelegram info.

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0002	2 bytes
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x05	COMMON_COMMAND = 5
-	5	1	CRC8H	0xnn	
Data	6	1	COMMAND Code	0x11	CO_WR_SUBTEL = 17
	7	1	Enable	0xnn	Enable = 1 Disable = 0
-	8	1	CRC8D	0xnn	

Table 45

In this case, the following **RESPONSE** gives the return codes:
00 RET_OK
02 RET_NOT_SUPPORTED
03 RET_WRONG_PARAM
Since no additional data are included which require description the standard RESPONSE structure is detailed in chapter 1.7.5

1.10.20 Code 18: CO_WR_MEM

Function: Write x bytes of the Flash, RAM0, DATA, IDATA, XDATA.

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0xnnnn	6 + x bytes
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x05	COMMON_COMMAND = 5
-	5	1	CRC8H	0xnn	
Data	6	1	COMMAND Code	0x12	CO_WR_MEM = 18
	7	1	Memory type	0xnn	Flash 0x00 RAM 0 0x01 data RAM 0x02 idata RAM 0x03 xdata RAM 0x04
	8	4	Memory address	0xnnnnnnnn	Start address to write
	12	x	Memory data	0xnn ... 0xnn	Data content to write ... ...
-	12+x	1	CRC8D	0xnn	

Table 46

In this case, the following **RESPONSE** gives the return codes:
00 RET_OK
02 RET_NOT_SUPPORTED
03 RET_WRONG_PARAM (address outside range of values)
04 RET_OPERATION_DENIED (memory access denied / code-protected)
Since no additional data are included which require description the standard RESPONSE structure is detailed in chapter: 1.7.5

1.10.21 Code 19: CO_RD_MEM

Function: Read x bytes of the Flash, RAM0, DATA, IDATA, XDATA.

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0xnn08	8 bytes
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x05	COMMON_COMMAND = 5
-	5	1	CRC8H	0xnn	
Data	6	1	COMMAND Code	0x13	CO_RD_MEM = 19
	7	1	Memory type	0xnn	Flash 0x00 RAM 0 0x01 data RAM 0x02 idata RAM 0x03 xdata RAM 0x04
	8	4	Memory address	0xnnnnnnnn	Start address to read
	12	2	Data lenght	0xnnnn	Length to be read
-	14	1	CRC8D	0xnn	

Table 47

Following described **RESPONSE** applies to return code:
00: RET_OK

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0xnnnn	1 + x bytes
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x02	RESPONSE = 2
-	5	1	CRC8H	0xnn	
Data	6	1	Return Code	0x00	RET_OK = 0
	7	x	Memory data	0xnn … 0xnn	Of read memory contents … …
-	7+x	1	CRC8D	0xnn	

Table 48

For **RESPONSE** with return codes:
02 RET_NOT_SUPPORTED
03 RET_WRONG_PARAM (address outside range of values)
04 RET_OPERATION_DENIED (memory access denied / code-protected)
is the standard structure described by the chapter 1.7.5

1.10.22 Code 20: CO_RD_MEM_ADDRESS

Function: Feedback about the used address and length of the config area and the Smart Ack table.

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0002	2 bytes
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x05	COMMON_COMMAND = 5
-	5	1	CRC8H	0xnn	
Data	6	1	COMMAND Code	0x14	CO_RD_MEM_ADDRESS = 20
	7	1	Memory area	0xnn	Config area = 0 Smart Ack Table = 1 System error log = 2
-	8	1	CRC8D	0xnn	

Table 49

Following described **RESPONSE** applies to return code:
00: RET_OK

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x000A	10 bytes
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x02	RESPONSE = 2
-	5	1	CRC8H	0xnn	
Data	6	1	Return Code	0x00	RET_OK = 0
	7	1	Memory type	0xnn	Flash 0x00 RAM 0 0x01 data RAM 0x02 idata RAM 0x03 xdata RAM 0x04
	8	4	Memory address	0xnnnnnnnn	Start address of config area / Smart Ack table / system error log
	12	4	Memory length	0xnnnnnnnn	Data length of config area / Smart Ack table / system error log
-	16	1	CRC8D	0xnn	

Table 50

For **RESPONSE** with return codes:
02 RET_NOT_SUPPORTED
03 RET_WRONG_PARAM
04 RET_OPERATION_DENIED (memory access denied / code-protected)
is the standard structure described by the chapter 1.7.5

1.10.23 Code 21: CO_RD_SECURITY

Function: Read security information (level, keys). This function does not support the actual security concept and should not be used any more.

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0001	1 byte
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x05	COMMON_COMMAND = 5
-	5	1	CRC8H	0xnn	
Data	6	1	COMMAND Code	0x15	CO_RD_SECURITY = 21
-	7	1	CRC8D	0xnn	

Table 51

Following described **RESPONSE** applies to return code:
00: RET_OK

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x000A	10 bytes
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x02	RESPONSE = 2
-	5	1	CRC8H	0xnn	
Data	6	1	Return Code	0x00	RET_OK = 0
	7	1	SEC LEVEL	0x0n	Type no. of encryption
	8	4	KEY	0xnnnnnnnn	Security key
	12	4	Rolling Code	0x00000000	Reserved
-	16	1	CRC8D	0xnn	

Table 52

For **RESPONSE** with return code:
02 RET_NOT_SUPPORTED is the standard structure described by the chapter 1.7.5

1.10.24 Code 22: CO_WR_SECURITY

Function: Write security information (level, keys). This function does not support the actual security concept and should not be used any more.

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x000A	10 bytes
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x05	COMMON_COMMAND = 5
-	5	1	CRC8H	0xnn	
Data	6	1	COMMAND Code	0x16	CO_WR_SECURITY = 22
	7	1	SEC LEVEL	0x0n	Type no. of encryption
	8	4	KEY	0xnnnnnnnn	Security key
	12	4	Rolling Code	0x00000000	Reserved
-	16	1	CRC8D	0xnn	

Table 53

In this case, the following **RESPONSE** gives the return codes:
00 RET_OK
01 RET_ERROR
02 RET_NOT_SUPPORTED
03 RET_WRONG_PARAM
Since no additional data are included which require description the standard RESPONSE structure is detailed in chapter 1.7.5

1.10.25 Code 23: CO_WR_LEARNMODE

Function: Enables or disables learn mode of Controller.

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0006	6 bytes
	3	1	Optional Length	0x01	1 byte
	4	1	Packet Type	0x05	COMMON_COMMAND = 5
-	5	1	CRC8H	0xnn	
Data	6	1	COMMAND Code	0x17	CO_WR_LEARNMODE = 23
	7	1	Enable	0x0n	Start Learn mode = 1 End Learn mode = 0
	8	4	Timeout	0xnnnnnnnn	Time-Out for the learn mode in ms. When time is 0 then default period of 60’000 ms is used
Optional Data	12	1	Channel	0xnn	0..0xFD = Channel No. absolute 0xFE = Previous channel relative 0xFF = Next channel relative
-	-	1	CRC8D	0xnn	

Table 54

In this case, the following **RESPONSE** message gives the return codes:
00 RET_OK
02 RET_NOT_SUPPORTED
03 RET_WRONG_PARAM
Since no additional data are included which require description the standard RESPONSE structure is detailed in chapter 1.7.5

1.10.26 Code 24: CO_RD_LEARNMODE

Function: Reads the learn-mode state of Controller.

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0001	1 byte
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x05	COMMON_COMMAND = 5
-	5	1	CRC8H	0xnn	
Data	6	1	COMMAND Code	0x18	CO_RD_LEARNMODE = 24
-	7	1	CRC8D	0xnn	

Table 55

Following described **RESPONSE** applies to return code:
00: RET_OK

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0002	2 bytes
	3	1	Optional Length	0x01	1 byte
	4	1	Packet Type	0x02	RESPONSE = 2
-	5	1	CRC8H	0xnn	
Data	6	1	Return Code	0x00	RET_OK = 0
	7	1	Enable	0x0n	Learn mode not active = 0 Learn mode active = 1
Optional Data	8	1	Channel	0xnn	0..0xFD = Channel No. absolute 0xFE = not used 0xFF = not used
-	-	1	CRC8D	0xnn	

Table 56

For **RESPONSE** with return code:
02 RET_NOT_SUPPORTED
is the standard structure described by the chapter 1.7.5

1.10.27 Code 25: CO_WR_SECUREDEVICE_ADD

Function: Add secure device to controller. It is possible to add only one or more rocker with this function.

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0019	25 bytes
	3	1	Optional Length	0x02	2 bytes
	4	1	Packet Type	0x05	COMMON_COMMAND = 5
-	5	1	CRC8H	0xnn	
Data	6	1	COMMAND Code	0x19	CO_WR_SECUREDEVICE_ADD = 25
	7	1	SLF	0xnn	Security Level Format
	8	4	ID	0xnnnnnnnn	Device ID
	12	16	Private key	0xnnnnnnnn 0xnnnnnnnn 0xnnnnnnnn 0xnnnnnnnn	16 bytes private key of the device
	28	3	Rolling code	0xnnnnnn	If a 16 bit rolling code is defined in SLF, the MSB is undefined
Optional Data	31	1	Direction	0xnn	Add device security information to: 0x00 = Inbound table (default) 0x01 = Outbound table ID = Device ID 0x02 = Outbound table broadcast ID = Gateway SourceID which can be ChipID or one of BaseIDs 0x02..0xFF = not used
	32	1	PTM Sender	0xnn	0x00 = not PTM sender 0x01 = PTM sender 0x02..0xFF = Not Used.
-	-	1	CRC8D	0xnn	

Table 57

In this case, the following **RESPONSE** message gives only the return codes:
00 RET_OK
01 RET_ERROR (memory space full)
02 RET_NOT_SUPPORTED
03 RET_WRONG_PARAM (added device known, but private key wrong)
Since no additional data are included which require description the standard RESPONSE structure is detailed in chapter 1.7.5

1.10.28 Code 26: CO_WR_SECUREDEVICE_DEL

Function: Delete secure device from controller. It is only possible to delete ALL rockers of a secure device. If there was a Pre-Shared Key entry specified for that device then it will be removed as well.

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0005	5 bytes
	3	1	Optional Length	0x01	1 byte
	4	1	Packet Type	0x05	COMMON_COMMAND = 5
-	5	1	CRC8H	0xnn	
Data	6	1	COMMAND Code	0x1A	CO_WR_SECUREDEVICE_DEL = 26
	7	4	ID	0xnnnnnnnn	Device ID. If it is the broadcast ID (0xFFFFFFFF), then delete all secure devices from controller
Optional Data	8	1	Direction	0xnn	Remove secure device from: 0x00 = Inbound table (default) 0x01 = Outbound table 0x02..0xFF = not used
-	-	1	CRC8D	0xnn	

Table 58

In this case, the following **RESPONSE** message gives the return codes:
00 RET_OK
01 RET_ERROR (device not in list)
02 RET_NOT_SUPPORTED
Since no additional data are included which require description the standard RESPONSE structure is detailed in chapter: 1.7.5

1.10.29 Code 27: CO_RD_SECUREDEVICE_BY_INDEX

Function: Read secure device by index

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0002	2 bytes
	3	1	Optional Length	0x01	1 byte
	4	1	Packet Type	0x05	COMMON_COMMAND = 5
-	5	1	CRC8H	0xnn	
Data	6	1	COMMAND Code	0x1B	CO_RD_SECUREDEVICE = 27
	7	1	Index	0x00…0xFE	Index of secure device to read, starting with 0…254
Optional Data	8	1	Direction	0xnn	Read device security information from: 0x00 = Inbound table (default) 0x01 = Outbound table 0x02..0xFF = not used
-	9	1	CRC8D	0xnn	

Table 59

Following described **RESPONSE** applies to return code:
00: RET_OK

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0006	6 bytes
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x02	RESPONSE = 2
-	5	1	CRC8H	0xnn	
Data	6	1	Return Code	0x00	RET_OK = 0
	7	1	SLF	0xnn	Security Level Format
	8	4	ID	0xnnnnnnnn	Device ID
Optional Data	12	16	Private Key	0xnnnnnnnn 0xnnnnnnnn 0xnnnnnnnn 0xnnnnnnnn	16 bytes private key of the device
	28	3	Rolling Code	0xnnnnnn	If a 16 bit rolling code is defined in SLF, the MSB is undefined
	31	16	PSK	0xnnnnnnnn 0xnnnnnnnn 0xnnnnnnnn 0xnnnnnnnn	16 bytes pre-shared key of the device
-	47	1	CRC8D	0xnn	

Table 60

For **RESPONSE** with return code:
01 RET_ERROR (device not in list)
02 RET_NOT_SUPPORTED is the standard structure described by the chapter 1.7.5

If PSK was not set, it will be not included in the packet. If in the future response will be extended, all bytes of non existing PSK will be set to 0x00.

1.10.30 Code 28: CO_WR_MODE

Function: Sets the gateway transceiver mode.

There are two modes available:

- Compatible mode - ERP1 - gateway uses Packet Type 1 to transmit and receive radio telegrams – for ASK products with ERP2 radio protocol
- Advanced mode – ERP2 - gateway uses Packet Type 10 to transmit and receive radio telegrams – for FSK products with ERP2 radio protocol

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0002	2 bytes
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x05	COMMON_COMMAND = 5
-	5	1	CRC8H	0xnn	
Data	6	1	COMMAND Code	0x1C	CO_WR_MODE = 28
	6	1	Mode	0xnn	0x00 – Compatible mode (default) - ERP1 0x01 – Advanced mode - ERP2
-	7	1	CRC8D	0xnn	

Table 61

In this case, the following **RESPONSE** message gives the return codes:
00 RET_OK
01 RET_ERROR (device not in list)
02 RET_NOT_SUPPORTED
Since no additional data are included which require description the standard RESPONSE structure is detailed in chapter: 1.7.5

1.10.31 Code 29: CO_RD_NUMSECUREDEVICES

Function: Read number of taught in secure devices

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0001	1 byte
	3	1	Optional Length	0x00...0x01	1 byte
	4	1	Packet Type	0x05	COMMON_COMMAND = 5
-	5	1	CRC8H	0xnn	
Data	6	1	COMMAND Code	0x1D	CO_RD_NUMSECUREDEVICES = 29
Optional Data	7	1	Direction	0xnn	Get the device count for: 0x00 = Inbound table (default) 0x01 = Outbound table 0x02..0xFF = not used
-	8	1	CRC8D	0xnn	

Table 62

Following described **RESPONSE** applies to return code:
00: RET_OK

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0002	2 bytes
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x02	RESPONSE = 2
-	5	1	CRC8H	0xnn	
Data	6	1	Return Code	0x00	RET_OK = 0
	7	1	Number	0xnn	Number of secure devices teached in
-	8	1	CRC8D	0xnn	

Table 63

For **RESPONSE** with return code:
02 RET_NOT_SUPPORTED is the standard structure described by the chapter 1.7.5

1.10.32 Code 30: CO_RD_SECUREDEVICE_BY_ID

Function: Read secure device by ID

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0005	5 bytes
	3	1	Optional Length	0x00...0x01	1 byte
	4	1	Packet Type	0x05	COMMON_COMMAND = 5
-	5	1	CRC8H	0xnn	
Data	6	1	COMMAND Code	0x1E	CO_RD_SECUREDEVICE_BY_ID= 30
	7	4	ID	0xnnnnnnnn	Device ID
Optional Data	11	1	Direction	0xnn	Read device security information from: 0x00 = Inbound table (default) 0x01 = Outbound table 0x02..0xFF = not used
-	12	1	CRC8D	0xnn	

Table 64

Following described **RESPONSE** applies to return code:
00: RET_OK

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0002	2 bytes
	3	1	Optional Length	0x00...0x01	0 byte
	4	1	Packet Type	0x02	RESPONSE = 2
-	5	1	CRC8H	0xnn	
Data	6	1	Return Code	0x00	RET_OK = 0
	7	1	SLF	0xnn	Security Level Format
Optional Data	8	1	Index	0x00…0xFE	Index of secure device in the devices table, starting with 0…254
-	9	1	CRC8D	0xnn	

Table 65

For **RESPONSE** with return code:
01 RET_ERROR (device not in the list)
02 RET_NOT_SUPPORTED is the standard structure described by the chapter 1.7.5

1.10.33 Code 31: CO_WR_SECUREDEVICE_ADD_PSK

Function: Add Pre-shared key for inbound secure device.

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0015	21 bytes
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x05	COMMON_COMMAND = 5
-	5	1	CRC8H	0xnn	
Data	6	1	COMMAND Code	0x1F	CO_WR_SECUREDEVICE_ADD_PSK = 31
	8	4	ID	0xnnnnnnnn	Device ID
	12	16	Pre-Shared Key	0xnnnnnnnn 0xnnnnnnnn 0xnnnnnnnn 0xnnnnnnnn	16 bytes pre-shared key of the device
-	-	1	CRC8D	0xnn	

Table 66

In this case, the following **RESPONSE** message gives only the return codes:
00 RET_OK
01 RET_ERROR (memory space full)
02 RET_NOT_SUPPORTED
03 RET_WRONG_PARAM (added device known, but private key wrong)
Since no additional data are included which require description the standard RESPONSE structure is detailed in chapter 1.7.5

1.10.34 Code 32: CO_WR_SECUREDEVICE_SENDTEACHIN

Function: Send secure Teach-In message. The device has to exist in the outbound table. Use CO_WR_SECUREDEVICE_ADD to add outbound device.

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0005	5 bytes
	3	1	Optional Length	0x00...0x01	1 byte
	4	1	Packet Type	0x05	COMMON_COMMAND = 5
-	5	1	CRC8H	0xnn	
Data	6	1	COMMAND Code	0x20	CO_WR_SECUREDEVICE_SENDTEACHIN = 32
	8	4	ID	0xnnnnnnnn	Device ID
Optional Data	8	1	TeachInInfo	0xnn	Teach-In Info
-	-	1	CRC8D	0xnn	

Table 67

In this case, the following **RESPONSE** message gives only the return codes:
00 RET_OK
01 RET_ERROR (memory space full)
02 RET_NOT_SUPPORTED
03 RET_WRONG_PARAM (added device known, but private key wrong)
Since no additional data are included which require description the standard RESPONSE structure is detailed in chapter 1.7.5

1.10.35 Code 33: CO_WR_TEMPORARY_RLC_WINDOW

Function: Set the temporary rolling-code window for every taught-in device but only one time for every devices next incoming telegram.

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0006	6 bytes
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x05	COMMON_COMMAND = 5
-	5	1	CRC8H	0xnn	
Data	6	1	COMMAND Code	0x21	CO_WR_TEMPORARY_RLC_WINDOW=33
	7	1	Enable	0xnn	0x00 - Disables the temporary RLC window 0x01 - Enables the temporary RLC window
	8	4	RLC Window	0xnnnnnnnn	Temporary rolling code window size. Only applied when Enabled = 0x01
-	-	1	CRC8D	0xnn	

Table 68

In this case, the following **RESPONSE** message gives the return codes:
00 RET_OK
01 RET_ERROR (device not in list)
02 RET_NOT_SUPPORTED
Since no additional data are included which require description the standard RESPONSE structure is detailed in chapter: 1.7.5

1.10.36 Code 34: CO_RD_SECUREDEVICE_PSK

Function: Read Pre-shared key for inbound secure device or for the module itself.

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0005	5 bytes
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x05	COMMON_COMMAND = 5
-	5	1	CRC8H	0xnn	
Data	6	1	COMMAND Code	0x22	CO_RD_SECUREDEVICE_PSK = 34
	8	4	ID	0xnnnnnnnn	Device ID 0x00000000: will return the module PSK other ID: will return inbound device PSK
-	-	1	CRC8D	0xnn	

Table 69

In this case, the following **RESPONSE** message gives only the return codes:
00 RET_OK

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0011	17 bytes
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x02	RESPONSE = 2
-	5	1	CRC8H	0xnn	
Data	6	1	Return Code	0x00	RET_OK = 0
	7	16	PSK	0xnnnnnnnn 0xnnnnnnnn 0xnnnnnnnn 0xnnnnnnnn	16-bytes Pre-Shared Key
-	12	1	CRC8D	0xnn	

01 RET_ERROR (no PSK assigned to the ID)
02 RET_NOT_SUPPORTED
03 RET_WRONG_PARAM (added device known, but private key wrong)
Since no additional data are included which require description the standard RESPONSE structure is detailed in chapter 1.7.5

1.10.37 Code 35: CO_RD_DUTYCYCLE_LIMIT

Function: Read actual duty cycle limit values.

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0001	1 bytes
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x05	COMMON_COMMAND = 5
-	5	1	CRC8H	0xnn	
Data	6	1	COMMAND Code	0x23	CO_RD_DUTYCYCLE_LIMIT = 35
-	-	1	CRC8D	0xnn	

Table 70

In this case, the following **RESPONSE** message gives only the return codes:
00 RET_OK

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0008	8 bytes
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x02	RESPONSE = 2
-	5	1	CRC8H	0xnn	
Data	6	1	Return Code	0x00	RET_OK = 0
	7	1	Available duty cycle	0..0x64	Total load of available 1% duty cycle from 0..100%
	8	1	Slots	0xnn	Total number of duty cycle slots
	9	2	Slot period	0xnnnn	Period of one slot in seconds
	11	2	Actual slot left	0xnnnn	Time left in actual slot in seconds
	13	1	Load after actual	0..0x64	Load available when period ends from 0..100%
-	14	1	CRC8D	0xnn	

02 RET_NOT_SUPPORTED
Since no additional data are included which require description the standard RESPONSE structure is detailed in chapter 1.7.5

1.10.38 Code 36: CO_SET_BAUDRATE

Function: Modifies the baud rate of the EnOcean device.
Note: The standard start up baud rate is 57600 and not all devices support the faster baudrate.

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0002	2 bytes
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x05	COMMON_COMMAND = 5
-	5	1	CRC8H	0xnn	
Data	6	1	COMMAND Code	0x24	CO_SET_BAUDRATE = 36
	7	1	BAUDRATE	0xnn	0 = 57600 BAUD 1 = 115200 BAUD 2 = 230400 BAUD 3 = 460800 BAUD

-	-	1	CRC8D	0xnn	

Table 71

In this case, the following **RESPONSE** message gives only the return codes:
00 RET_OK
02 RET_NOT_SUPPORTED
Since no additional data are included which require description the standard RESPONSE structure is detailed in chapter 1.7.5

Note: the response is send with the current Baud rate settings. If the baud rate can be modified the response is then send afterwards again with the new baudrate.

1.10.39 Code 37: CO_GET_FREQUENCY_INFO

Function: Reads Frequency and protocol of the Device.

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0001	1 bytes
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x05	COMMON_COMMAND = 5
-	5	1	CRC8H	0xnn	
Data	6	1	COMMAND Code	0x25	CO_GET_FREQUENCY_INFO = 37
-	7	1	CRC8D	0xnn	

Table 72

In this case, the following **RESPONSE** message gives only the return codes:
00 RET_OK

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0002	2 bytes
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x02	RESPONSE = 2
-	5	1	CRC8H	0xnn	
Data	6	1	Return Code	0x00	RET_OK = 0
	7	1	Frequency	0xnn	0x00 315Mhz 0x01 868.3Mhz 0x02 902.875Mhz 0x03 925 Mhz 0x04 928 Mhz 0x20 2.4 Ghz
	8	1	Protocol	0xnn	0x00 ERP1 0x01 ERP2 0x10 802.15.4 0x20 Bluetooth 0x30 Long Range
-	9	1	CRC8D	0xnn	

02 RET_NOT_SUPPORTED

Since no additional data are included which require description the standard RESPONSE structure is detailed in chapter 1.7.5

1.10.40 Code 38: Reserved

Reserved for future use.

1.10.41 Code 37: CO_GET_STEPCODE

Function: Reads Hardware Step code and Revision of the Device

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0001	1 bytes
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x05	COMMON_COMMAND = 5
-	5	1	CRC8H	0xnn	
Data	6	1	COMMAND Code	0x27	CO_GET_STEPCODE = 39
-	7	1	CRC8D	0xnn	

Table 73

In this case, the following **RESPONSE** message gives only the return codes:
00 RET_OK

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0003	3 bytes
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x02	RESPONSE = 2
-	5	1	CRC8H	0xnn	
Data	6	1	Return Code	0x00	RET_OK = 0
	7	1	HW- Revision	0xnn	e.g. 0xDA ,0xCA …
	8	1	Stepcode	0xnn	
-	9	1	CRC8D	0xnn	

02 RET_NOT_SUPPORTED
Since no additional data are included which require description the standard RESPONSE structure is detailed in chapter 1.7.5

1.11 Packet Type 6: SMART_ACK_COMMAND

1.11.1 Structure

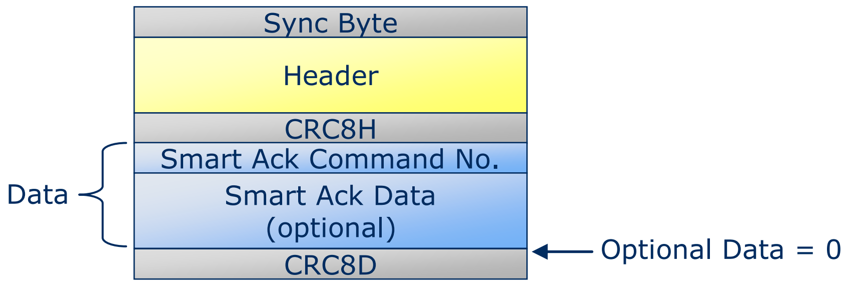


Figure 11

In the current version of ESP3 the packet type SMART_ACK_COMMAND carries no Optional Data.

1.11.2 List of SMART ACK Codes

Code	Function Name	Description
01	SA_WR_LEARNMODE	Set/Reset Smart Ack learn mode
02	SA_RD_LEARNMODE	Get Smart Ack learn mode state
03	SA_WR_LEARNCONFIRM	Used for Smart Ack to add or delete a mailbox of a client
04	SA_WR_CLIENTLEARNRQ	Send Smart Ack Learn request (Client)
05	SA_WR_RESET	Send reset command to a Smart Ack client
06	SA_RD_LEARNEDCLIENTS	Get Smart Ack learned sensors / mailboxes
07	SA_WR_RECLAIMS	Set number of reclaim attempts
08	SA_WR_POSTMASTER	Activate/Deactivate Post master functionality

Table 74

1.11.3 Code 01: SA_WR_LEARNMODE

Function: Enables or disables learn mode of Smart Ack Controller.

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0007	7 bytes
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x06	SMART_ACK_COMMAND = 6
-	5	1	CRC8H	0xnn	
Data	6	1	SMART_ACK Code	0x01	SA_WR_LEARNMODE = 1
	7	1	Enable	0x0n	Start Learnmode = 1 End Learnmode = 0
	8	1	Extended	0x0n	Simple Learnmode = 0 Advance Learnmode = 1 Advance Learnmode select Rep. = 2
	9	4	Timeout	0xnnnnnnnn	Time-Out for the learn mode in ms. When time is 0 then default period of 60’000 ms is used
-	13	1	CRC8D	0xnn	

Table 75

In this case, the following **RESPONSE** message gives the return codes:
00 RET_OK
02 RET_NOT_SUPPORTED
03 RET_WRONG_PARAM
Since no additional data are included which require description the standard RESPONSE structure is detailed in chapter 1.7.5

1.11.4 Code 02: SA_RD_LEARNMODE

Function: Reads the learnmode state of Smart Ack Controller.

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0001	1 byte
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x06	SMART_ACK_COMMAND = 6
-	5	1	CRC8H	0xnn	
Data	6	1	SMART_ACK Code	0x02	SA_RD_LEARNMODE = 2
-	7	1	CRC8D	0xnn	

Table 76

Following described **RESPONSE** applies to return code:
00: RET_OK

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0003	3 bytes
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x02	RESPONSE = 2
-	5	1	CRC8H	0xnn	
Data	6	1	Return Code	0x00	RET_OK = 0
	7	1	Enable	0x0n	Learnmode not active = 0 Learnmode active = 1
	8	1	Extended	0x0n	Simple Learnmode = 0 Advance Learnmode = 1 Advance Learnmode select Rep. = 2
-	9	1	CRC8D	0xnn	

Table 77

For **RESPONSE** with return code:
02 RET_NOT_SUPPORTED
is the standard structure described by the chapter 1.7.5

1.11.5 Code 03: SA_WR_LEARNCONFIRM

Function: Send smart ack learn answer to modify mailbox at postmaster.

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x000C	12 bytes
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x06	SMART_ACK_COMMAND = 6
-	5	1	CRC8H	0xnn	
Data	6	1	SMART_ACK Code	0x03	SA_WR_LEARNCONFIRM = 3
	7	2	Response time	0xnnnn	Response time for sensor in ms in which the controller can prepare the data and send it to the postmaster. Only actual, if learn return code is Learn IN.
	9	1	Confirm code	0xnn	Learn IN: 0x00 Learn OUT: 0x20
	10	4	Postmaster Candidate ID	0xnnnnnnnn	Device ID of the used Post master
	14	4	Smart Ack Client ID	0xnnnnnnnn	Device ID of the learned IN/OUT Smart Ack Client
-	18	1	CRC8D	0xnn	

Table 78

In this case, the following **RESPONSE** message gives only the return codes:
00 RET_OK
02 RET_NOT_SUPPORTED
03 RET_WRONG_PARAM
Since no additional data are included which require description the standard RESPONSE structure is detailed in chapter 1.7.5

1.11.6 Code 04: SA_WR_CLIENTLEARNRQ

Function: Sends Smart Ack Learn Request telegram to Smart Ack Controller. This function will only be used in a Smart Ack Client.

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0006	6 bytes
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x06	SMART_ACK_COMMAND = 6
-	5	1	CRC8H	0xnn	
Data	6	1	SMART_ACK Code	0x04	SA_WR_ CLIENTLEARNRQ = 4
	7	1	2^2 ... 2^0: Manufacturer ID 2^7 ... 2^3: Reserved	0b11111nnn	nnn = Most significant 3 bits of the Manufacturer ID 11111 = reserved / default values
	8	1	Manufacturer ID	0xnn	Least significant bits of the Manufacturer ID
	9	3	EEP	0xnnnnnn	EEP of the Smart Ack client, who wants to Teach IN.
-	12	1	CRC8D	0xnn	

Table 79

In this case, the following **RESPONSE** message gives the return codes:
00 RET_OK
02 RET_NOT_SUPPORTED
03 RET_WRONG_PARAM
Since no additional data are included which require description the standard RESPONSE structure is detailed in chapter 1.7.5

1.11.7 Code 05: SA_WR_RESET

Function: Send reset command to a Smart Ack Client.

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0005	5 bytes
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x06	SMART_ACK_COMMAND = 6
-	5	1	CRC8H	0xnn	
Data	6	1	SMART_ACK Code	0x05	SA_WR_ RESET = 5
	7	4	Smart Ack Client ID	0xnnnnnnnn	Device ID of the Smart Ack Client
-	11	1	CRC8D	0xnn	

Table 80

In this case, the following **RESPONSE** message gives the return codes:
00 RET_OK
02 RET_NOT_SUPPORTED
03 RET_WRONG_PARAM
Since no additional data are included which require description the standard RESPONSE structure is detailed in chapter 1.7.5

1.11.8 Code 06: SA_RD_LEARNEDCLIENTS

Read mailbox information at the Post Master device, about all learned Smart Ack clients.

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0001	1 byte
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x06	SMART_ACK_COMMAND = 6
-	5	1	CRC8H	0xnn	
Data	6	1	SMART_ACK Code	0x06	SA_RD_ LEARNEDCLIENTS = 6
-	7	1	CRC8D	0xnn	

Table 81

Following described **RESPONSE** applies to return code:
00: RET_OK

Group	Offset	Size	Field	Value hex	Description	
-	0	1	Sync. Byte	0x55		
Header	1	2	Data Length	0xnnnn	1 + 9*c bytes (c = number of clients)	
	3	1	Optional Length	0x00	0 byte	
	4	1	Packet Type	0x02	RESPONSE = 2	
-	5	1	CRC8H	0xnn		
Data	6	1	Return Code	0x00	RET_OK = 0	
	7	4	Smart Ack Client ID	0xnnnnnnnn	Device ID of the Smart Ack Client	c
	7 +4*c	4	Controller ID	0xnnnnnnnn	Controller ID dedicated Smart Ack Client	c
	7 +8*c	1	Mailbox index	0xnn	Internal counter of Post master (0x00 … 0x0E)	c
-	7 +9*c	1	CRC8D	0xnn		

Table 82

Every learned Smart Ack Client has the group **c** with fields in the order: Controller ID, Smart Ack Client ID, Mailbox index (c = also number of clients / multiplier to calculate the offset).

For **RESPONSE** with return code:
02 RET_NOT_SUPPORTED
is the standard structure described by the chapter 1.7.5

1.11.9 Code 07: SA_WR_RECLAIMS

Function: Set the amount of reclaim tries in Smart Ack Client.

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0002	2 bytes
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x06	SMART_ACK_COMMAND = 6
-	5	1	CRC8H	0xnn	
Data	6	1	SMART_ACK Code	0x07	SA_WR_RECLAIMS = 7
	7	1	Reclaim count	0xnn	Presetting for the number of required reclaim tries
-	8	1	CRC8D	0xnn	

Table 83

In this case, the following **RESPONSE** message gives the return codes:
00 RET_OK
02 RET_NOT_SUPPORTED
03 RET_WRONG_PARAM
Since no additional data are included which require description the standard RESPONSE structure is detailed in chapter 1.7.5

1.11.10 Code 08: SA_WR_POSTMASTER

Function: Enables/Disables postmaster function of device.

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0002	2 bytes
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x06	SMART_ACK_COMMAND = 6
-	5	1	CRC8H	0xnn	
Data	6	1	SMART_ACK Code	0x08	SA_WR_POSTMASTER = 8
	7	1	Mailbox count	0xnn	Amount of mailboxes available, 0 = disable post master functionality; Maximum 28 mailboxes can be created. This upper limit is for each firmware restricted and may be smaller.
-	8	1	CRC8D	0xnn	

Table 84

In this case, the following **RESPONSE** message gives the return codes:
00 RET_OK
02 RET_NOT_SUPPORTED
03 RET_WRONG_PARAM
Since no additional data are included which require description the standard RESPONSE structure is detailed in chapter 1.7.5

1.12 Packet Type 7: REMOTE_MAN_COMMAND

1.12.1 Structure

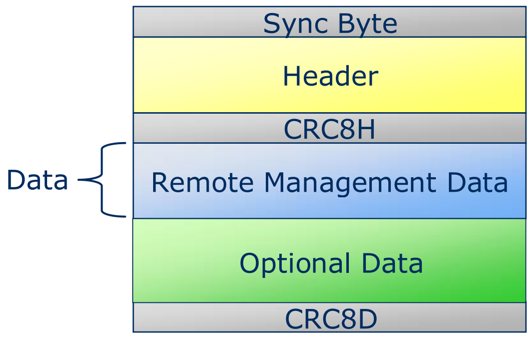


Figure 12

This section describes the remote management command structure. This structure is applied for the send as well as the receive case.

1.12.2 Description

Function: Remote Management send or receive message.

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0xnnnn	4 + x bytes
	3	1	Optional Length	0x0A	10 bytes
	4	1	Packet Type	0x07	REMOTE_MAN_COMMAND = 7
-	5	1	CRC8H	0xnn	
Data	6	2	Function No.	0x0nnn	Range: 0x0000 … 0x0FFF
	8	2	Manufacturer ID	0x0nnn	Range: 0x0000 … 0x07FF
	10	x	Message data	…	0 … 511 bytes
Optional Data	10+x	4	Destination ID	0xnnnnnnnn	Destination ID Broadcast ID: FF FF FF FF
	14+x	4	Source ID	0xnnnnnnnn	Receive case: Source ID of the sender Send case: 0x00000000
	18+x	1	dBm	0xnn	Send case: 0xFF Receive case: Best RSSI value of all received sub telegrams (value decimal without minus)
	19+x	1	Send With Delay	0x0n	1: if the first message has to be sent with random delay. When answering to broadcast message this has to be 1, otherwise 0. Default: 0
-	20+x	1	CRC8D	0xnn	CRC8 Data byte; calculated checksum for whole byte groups: DATA and OPTIONAL_DATA

Table 85

The receive case has no RESPONSE.

The send case has the following **RESPONSE** with the return codes:
00 RET_OK
02 RET_NOT_SUPPORTED
03 RET_WRONG_PARAM

Since no additional data are included, which require description the standard RESPONSE structure is detailed in chapter 1.7.5

1.13 Packet Type 9: RADIO_MESSAGE

1.13.1.1 Packet structure

The radio message (payload data without any radio telegram contents) is embedded into the ESP3 packet.

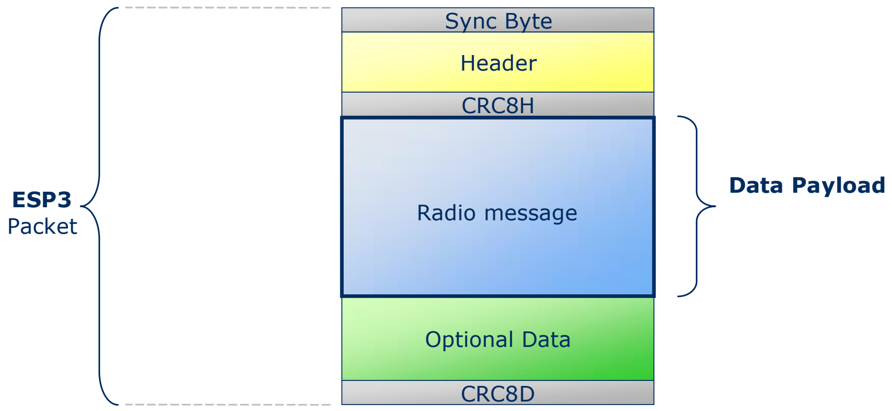


Figure 13

The following structure is applicable to all types of radio messages:

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0xnnnn	Variable length of message
	3	1	Optional Length	0x0A	Optional Data = 9 bytes
	4	1	Packet Type	0x09	RADIO_MESSAGE = 9
-	5	1	CRC8H	0xnn	
Data	6	1	Message RORG	0xnn	RORG
Data	7	x	Message Data	...	Message Data Content
Optional Data	7+x	4	Destination ID	0xnnnnnnnn	Destination ID Broadcast ID: FF FF FF FF
	11+x	4	Source ID	0xnnnnnnnn	Receive case: Source ID of the sender Send case: 0x00000000
	15+x	1	dBm	0xnn	Send case: 0xFF Receive case: Best RSSI value of all received sub telegrams (value decimal without minus)
-	13+x	1	CRC8D	0xnn	CRC8 Data byte; calculated checksum for whole byte groups: DATA and OPTIONAL_DATA

Table 86

When receiving a message, no RESPONSE has to be sent. When sending a message, a RESPOND has to be expected. In this case, the following **RESPONSE** message gives the return codes:
00 RET_OK
02 RET_NOT_SUPPORTED
05 RET_LOCK_SET

Since no additional data are included, that have to be described, the standard RESPONSE structure is described in chapter 1.7.5

1.14 Packet Type 10: RADIO_ERP2

1.14.1 Packet structure

The ERP2 radio protocol telegram (raw data without LEN) is embedded into the ESP3 packet.

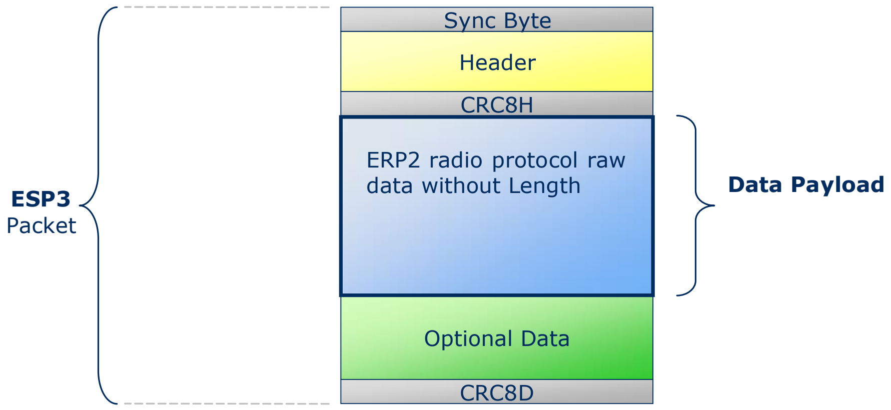


Figure 14

The following structure is applicable to all types of radio telegrams:

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0xnnnn	Variable length of radio telegram
	3	1	Optional Length	0x02	2 fields fixed
	4	1	Packet Type	0x0A	RADIO_ERP2 = 10
-	5	1	CRC8H	0xnn	
Data	6	x	Raw data	… …	ERP2 radio protocol telegram without the first Length byte. For sending the ERP2 protocol CRC8 byte can be set to any value. x = Data Length
Optional Data	6+x	1	SubTelNum	0xnn	Number of sub telegram; Send: 3 / receive: 1 … y
	7+x	1	dBm	0xnn	Send case: FF Receive case: best RSSI value of all received sub telegrams (value decimal without minus)
-	8+x	1	CRC8D	0xnn	CRC8 Data byte; calculated checksum for whole byte groups: DATA and OPTIONAL_DATA

Table 87

When receiving a telegram, no RESPONSE has to be sent. When sending a telegram, a RESPOND has to be expected. In this case, the following **RESPONSE** message gives the return codes:
00 RET_OK
02 RET_NOT_SUPPORTED
03 RET_WRONG_PARAM
Since no additional data are included, that have to be described, the standard RESPONSE structure is described in chapter 1.7.5

1.15 Packet Type 16: RADIO_802_15_4 : 802.15.4 Physical Raw Packet

This packet is send from a 802.15.4 sniffer to a client after receiving a valid 802.15.4 frame.
If a gateway receives such a telegram, the packet will be send via the air interface.

1.15.1 Packet structure

The whole 802.15.4 mac frame omitting the FCS is embedded into the ESP3 packet.

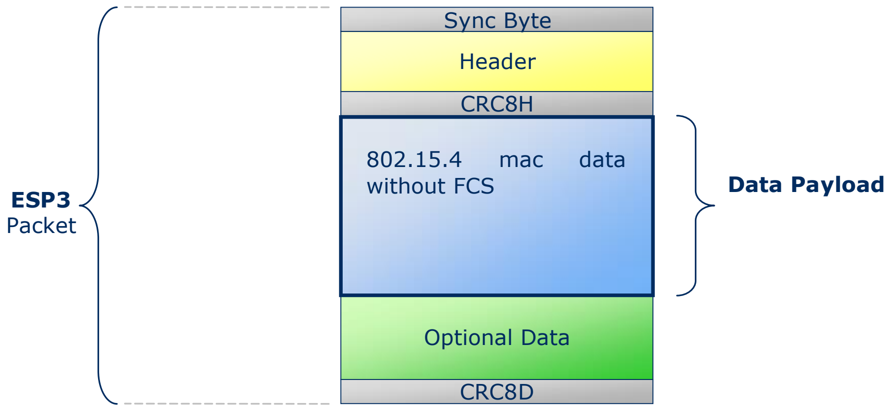
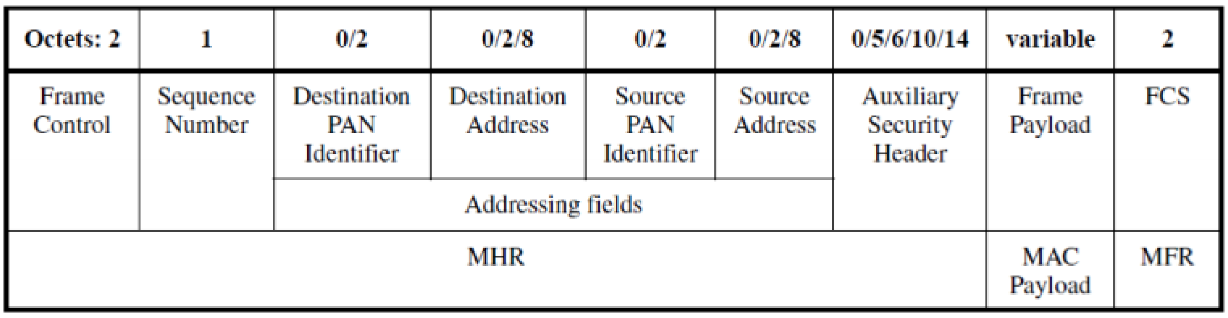


Figure 15 Packet Type 0x10



Octets: 2	1	0/2	0/2/8	0/2	0/2/8	0/5/6/10/14	variable	2
Frame Control	Sequence Number	Destination PAN Identifier	Destination Address	Source PAN Identifier	Source Address	Auxiliary Security Header	Frame Payload	FCS
		Addressing fields						
MHR							MAC Payload	MFR

Figure 16 Mac Data Format

The following structure is applicable to all types of radio telegrams:

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
	1	2	Data Length	0xnnnn	Variable length of radio telegram

Header	3	1	Optional Length	0x01	1 byte
	4	1	Packet Type	0x10	RADIO_802_15_4= 16
-	5	1	CRC8H	0xnn	
Data	6	x	Raw data	… …	802.15.4 MHR+ mac payload without the FCS
Optional Data	6+x	1	RSSI	0xnn	Send case: don’t care Receive case: - RSSI
-	8+x	1	CRC8D	0xnn	CRC8 Data byte; calculated checksum for whole byte groups: DATA and OPTIONAL_DATA

Table 88

When receiving a telegram, no RESPONSE has to be sent. When sending a telegram, a RESPOND has to be expected. In this case, the following RESPONSE message gives the return codes:

00	RET_OK
02	RET_NOT_SUPPORTED
03	RET_WRONG_PARAM
07	RET_NO_FREE_BUFFER

Since no additional data are included, that have to be described, the standard RESPONSE structure is described in chapter 1.7.5

1.16 Packet Type 17: COMMAND_2_4

1.16.1 List of EnOcean 2.4 commands

Code	Command Name	Description
01	SET_CHANNEL	Sets the current 802.15.4 channel
02	GET_CHANNEL	Gets the current 802.15.4 channel

Table 89

1.16.2 SET_CHANNEL

Function: Set 802.15.4 channel

The command structure is following:

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0002	2 bytes
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x11	COMMAND_2_4 = 0x11
-	5	1	CRC8H	0xnn	
Data	6	1	COMMAND Code	0x01	SET_CHANNEL = 0x01
	7	1	Channel	11-26	The 802.15.4 channel to use
-	8	1	CRC8D	0xnn	

Table 90

Following described **RESPONSE** applies to return code:
00: RET_OK

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0001	1 byte
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x02	RESPONSE = 2
-	5	1	CRC8H	0xnn	
Data	6	1	Return Code	0xnn	00
-	7	1	CRC8D	0xnn	

Table 91

For **RESPONSE** with return code:
02: RET_NOT_SUPPORTED
Since no additional data are included, that have to be described, the standard RESPONSE structure is described in chapter 1.7.5

1.16.3 GET_CHANNEL

Function: Get the device's current used channel

The command structure is following:

Group	Offset	Size	Field	Value hex	Description
-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0001	1 byte
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x11	COMMAND_2_4 = 0x11
-	5	1	CRC8H	0xnn	
Data	6	1	COMMAND Code	0x02	GET_CHANNEL = 0x02
-	7	1	CRC8D	0xnn	

Table 92

Following described **RESPONSE** applies to return code:
00: RET_OK

Group	Offset	Size	Field	Value hex	Description

-	0	1	Sync. Byte	0x55	
Header	1	2	Data Length	0x0002	2 bytes
	3	1	Optional Length	0x00	0 byte
	4	1	Packet Type	0x02	RESPONSE_PACKET = 0x02
-	5	1	CRC8H	0xnn	
Data	6	1	Return Code	0	OK
	7	1	Channel	11..26	Used Channel
-	8	1	CRC8D	0xnn	

Table 93

For **RESPONSE** with return code:
02: RET_NOT_SUPPORTED
Since no additional data are included, that have to be described, the standard RESPONSE structure is described in chapter 1.7.5

2 Appendix

2.1 ESP3 Data flow sequences

The following examples illustrate the ESP3 traffic. In particular the flow of the Smart Ack commands is more complex.

2.1.1 Client data request

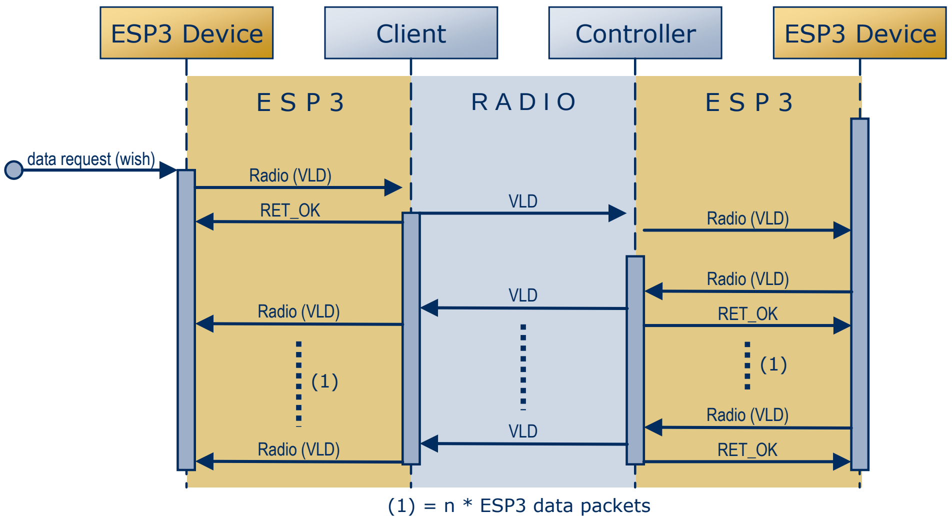


Figure 17

2.1.2 Teach IN via VLL

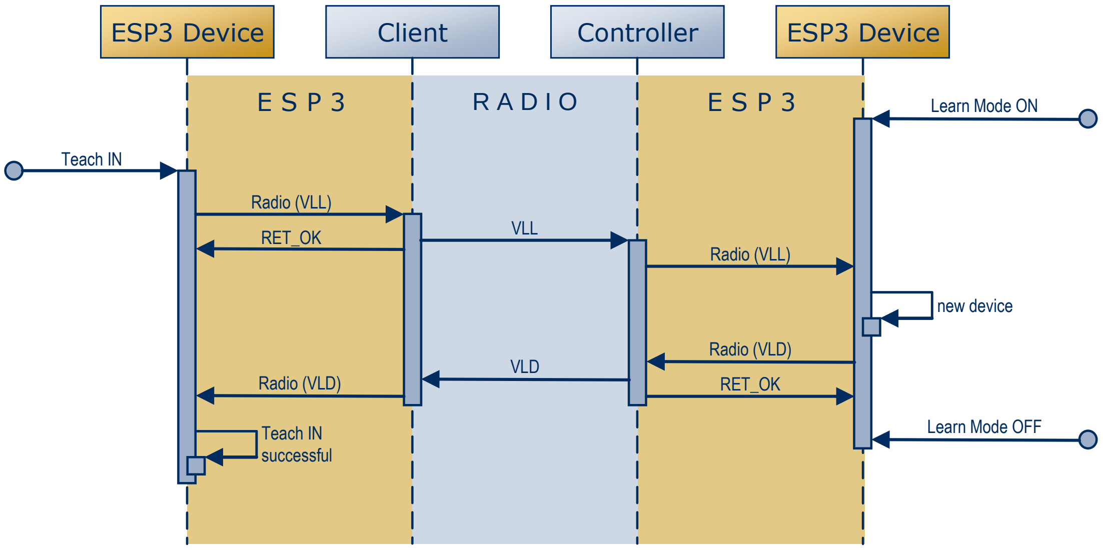
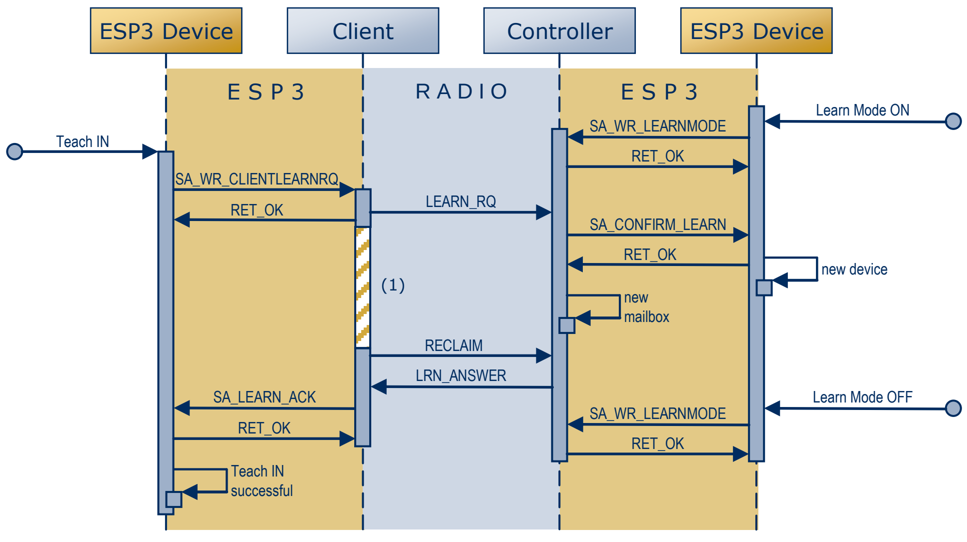


Figure 18

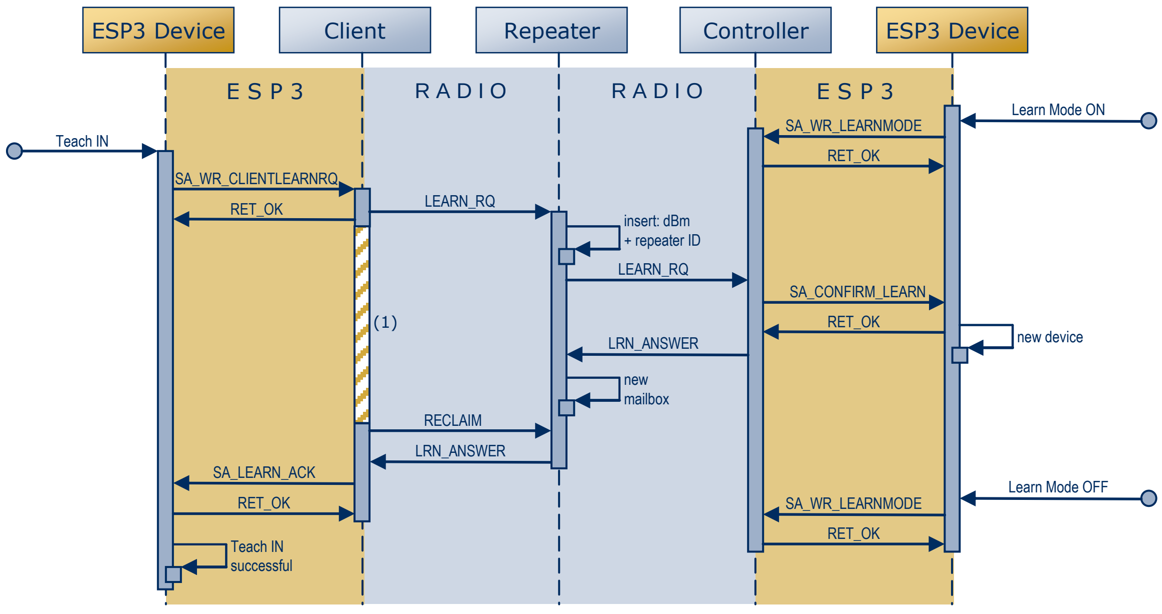
2.1.3 Teach IN via Smart Ack



(1) = Short Term Sleep

Figure 19

2.1.4 Teach IN via Smart Ack incl. repeater



(1) = Short Term Sleep

Figure 20

2.2 ESP3 telegram examples

2.2.1 Packet: Radio VLD

Sy	Header	CR C8	Data	Optional Data	CR C8
55	00 0F 07 01	2B	D2 DD DD DD DD DD DD DD DD DD 00 80 35 C4 00	03 FF FF FF FF 4D 00	36

2.2.2 Packet: CO_WR_SLEEP

Sy	Header	CR C8	Data	CR C8
55	00 05 00 05	DB	01 00 00 00 0A	54

```
Periode = 10 (0x0A)
```

2.2.3 Packet: CO_WR_RESET

Sy	Header	CR C8	Data	CR C8
55	00 01 00 05	70	02	0E

2.2.4 Packet: CO_RD_IDBASE

Sy	Header	CR C8	Data	CR C8
55	00 01 00 05	70	08	38

```
Response RET_OK:
```

Sy	Header	CR C8	Data	CR C8
55	00 05 00 02	CE	00 FF 80 00 00	DA

2.2.5 Packet: REMOTE_MAN_COMMAND

```
Example dummy command:
Function      = 0x0876
Manufacture   = 0x07FF
Message data  = 0x000102030405060708090a0b0c0d0e0f
DestinationID = Broadcast = 0xFFFFFFFF
SendWithDelay = 0
```

Sy	Header	CR C8	Data	
				Message data
55	00 14 0A 07	9E	08 76 07 FF	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

Optional Data	CR C8
FF FF FF FF 00 00 00 00 FF 00	86

Example QueryID:

Sy	Header	CR C8	Data	CR C8
55	00 04 00 07	BE	00 04 07 FF	33

2.3 CRC8 calculation

The polynomial **$G(x) = x^8 + x^2 + x^1 + x^0$** is used to generate the CRC8 table, needed for the CRC8 calculation. Following C code illustrates how the CRC8 value is calculated:

Implementation:

```
uint8 u8CRC8Table[256] = {
    0x00, 0x07, 0x0e, 0x09, 0x1c, 0x1b, 0x12, 0x15,
    0x38, 0x3f, 0x36, 0x31, 0x24, 0x23, 0x2a, 0x2d,
    0x70, 0x77, 0x7e, 0x79, 0x6c, 0x6b, 0x62, 0x65,
    0x48, 0x4f, 0x46, 0x41, 0x54, 0x53, 0x5a, 0x5d,
    0xe0, 0xe7, 0xee, 0xe9, 0xfc, 0xfb, 0xf2, 0xf5,
    0xd8, 0xdf, 0xd6, 0xd1, 0xc4, 0xc3, 0xca, 0xcd,
    0x90, 0x97, 0x9e, 0x99, 0x8c, 0x8b, 0x82, 0x85,
    0xa8, 0xaf, 0xa6, 0xa1, 0xb4, 0xb3, 0xba, 0xbd,
    0xc7, 0xc0, 0xc9, 0xce, 0xdb, 0xdc, 0xd5, 0xd2,
    0xff, 0xf8, 0xf1, 0xf6, 0xe3, 0xe4, 0xed, 0xea,
    0xb7, 0xb0, 0xb9, 0xbe, 0xab, 0xac, 0xa5, 0xa2,
    0x8f, 0x88, 0x81, 0x86, 0x93, 0x94, 0x9d, 0x9a,
    0x27, 0x20, 0x29, 0x2e, 0x3b, 0x3c, 0x35, 0x32,
    0x1f, 0x18, 0x11, 0x16, 0x03, 0x04, 0x0d, 0x0a,
    0x57, 0x50, 0x59, 0x5e, 0x4b, 0x4c, 0x45, 0x42,
    0x6f, 0x68, 0x61, 0x66, 0x73, 0x74, 0x7d, 0x7a,
    0x89, 0x8e, 0x87, 0x80, 0x95, 0x92, 0x9b, 0x9c,
    0xb1, 0xb6, 0xbf, 0xb8, 0xad, 0xaa, 0xa3, 0xa4,
    0xf9, 0xfe, 0xf7, 0xf0, 0xe5, 0xe2, 0xeb, 0xec,
    0xc1, 0xc6, 0xcf, 0xc8, 0xdd, 0xda, 0xd3, 0xd4,
    0x69, 0x6e, 0x67, 0x60, 0x75, 0x72, 0x7b, 0x7c,
    0x51, 0x56, 0x5f, 0x58, 0x4d, 0x4a, 0x43, 0x44,
    0x19, 0x1e, 0x17, 0x10, 0x05, 0x02, 0x0b, 0x0c,
    0x21, 0x26, 0x2f, 0x28, 0x3d, 0x3a, 0x33, 0x34,
    0x4e, 0x49, 0x40, 0x47, 0x52, 0x55, 0x5c, 0x5b,
    0x76, 0x71, 0x78, 0x7f, 0x6A, 0x6d, 0x64, 0x63,
    0x3e, 0x39, 0x30, 0x37, 0x22, 0x25, 0x2c, 0x2b,
    0x06, 0x01, 0x08, 0x0f, 0x1a, 0x1d, 0x14, 0x13,
    0xae, 0xa9, 0xa0, 0xa7, 0xb2, 0xb5, 0xbc, 0xbb,
    0x96, 0x91, 0x98, 0x9f, 0x8a, 0x8D, 0x84, 0x83,
    0xde, 0xd9, 0xd0, 0xd7, 0xc2, 0xc5, 0xcc, 0xcb,
    0xe6, 0xe1, 0xe8, 0xef, 0xfa, 0xfd, 0xf4, 0xf3
    };

#define proccrc8(u8CRC, u8Data)  (u8CRC8Table[u8CRC ^ u8Data])
```

Example:

```
u8CRC = 0;
for (i = 0 ; i < u16DataSize ; i++)
    u8CRC = proccrc8(u8CRC, u8Data[i]);
printf("CRC8 = %02X\n", u8CRC);
```

2.4 UART Synchronization (example c-code)

Please notice, that the example c-code in this chapter is written for big endian systems only. If you have a little endian system you have to make changes for proper functionality.

2.4.1 ESP3 Packet Structure

```
  //! Packet structure (ESP3)
    typedef struct
    {
        // Amount of raw data bytes to be received. The most significant byte is sent/received first
            uint16   u16DataLength;
        // Amount of optional data bytes to be received
            uint8     u8OptionLength;
        // Packe type code
            uint8     u8Type;
        // Data buffer: raw data + optional bytes
            uint8     *u8DataBuffer;
    } PACKET_SERIAL_TYPE;
```

2.4.2 Get ESP3 Packet

```
//! \file uart_getPacket.c

#include "EO3000I_API.h"
#include "proc.h"
#include "uart.h"
#include "time.h"

/*
ESP3 packet structure through the serial port.

Protocol bytes are generated and sent by the application

Sync = 0x55
CRC8H
CRC8D


    1              2                1           1           1      u16DataLen + u8OptionLen       1
+------+------------------+---------------+-----------+-----------+-------------/------------+-----------+
| 0x55 |      u16DataLen    |   u8OptionLen |  u8Type    |  CRC8H     |             DATAS          |   CRC8D    |
+------+------------------+---------------+-----------+-----------+-------------/------------+-----------+


DATAS structure:

                   u16DataLen                    u8OptionLen
+--------------------------------------------+----------------------+
|                   Data                      |     Optional          |
+--------------------------------------------+----------------------+
*/


RETURN_TYPE uart_getPacket(PACKET_SERIAL_TYPE *pPacket, uint16 u16BufferLength)
{
    //! uart_getPacket state machine states.
     typedef enum
     {
         //! Waiting for the synchronisation byte 0x55
         GET_SYNC_STATE=0,
         //! Copying the 4 after sync byte: raw data length (2 bytes), optional data length (1), type (1).
         GET_HEADER_STATE,
         //! Checking the header CRC8 checksum. Resynchronisation test is also done here
         CHECK_CRC8H_STATE,
         //! Copying the data and optional data bytes to the paquet buffer
         GET_DATA_STATE,
         //! Checking the info CRC8 checksum.
```

```
        CHECK_CRC8D_STATE,

    } STATES_GET_PACKET;

    //! UART received byte code
    uint8  u8RxByte;
    //! Checksum calculation
    static uint8 u8CRC = 0;
    //! Nr. of bytes received
    static uint16 u16Count = 0;
    //! State machine counter
    static STATES_GET_PACKET u8State = GET_SYNC_STATE;
    //! Timeout measurement
    static uint8 u8TickCount = 0;
    // Byte buffer pointing at the paquet address
    uint8 *u8Raw = (uint8*)pPacket;
    // Temporal variable
    uint8     i;

    // Check for timeout between two bytes
    if (((uint8)ug32SystemTimer) - u8TickCount > SER_INTERBYTE_TIME_OUT)
    {
        // Reset state machine to init state
        u8State = GET_SYNC_STATE;
    }

   // State machine goes on when a new byte is received
    while (uart_getByte(&u8RxByte) == OK)
    {
        // Tick count of last received byte
        u8TickCount = (uint8)ug32SystemTimer;

        // State machine to load incoming packet bytes
        switch(u8State)
        {
            // Waiting for packet sync byte 0x55
            case GET_SYNC_STATE:

                if (u8RxByte == SER_SYNCH_CODE)
                {
                    u8State = GET_HEADER_STATE;
                    u16Count = 0;
                    u8CRC    = 0;
                }

                break;


            // Read the header bytes
            case GET_HEADER_STATE:

                // Copy received data to buffer
                u8Raw[u16Count++] = u8RxByte;
                u8CRC = proc_crc8(u8CRC, u8RxByte);

                // All header bytes received?
                if(u16Count == SER_HEADER_NR_BYTES)
                {
                   u8State = CHECK_CRC8H_STATE;
                }

                break;


            // Check header checksum & try to resynchonise if error happened
            case CHECK_CRC8H_STATE:

                // Header CRC correct?
                if (u8CRC != u8RxByte)
                {
                    // No. Check if there is a sync byte (0x55) in the header
                    int a = -1;
                    for (i = 0 ; i < SER_HEADER_NR_BYTES ; i++)
                        if (u8Raw[i] == SER_SYNCH_CODE)
                        {
                            // indicates the next position to the sync byte found
                            a=i+1;
                            break;
                        };

                    if ((a == -1) && (u8RxByte != SER_SYNCH_CODE))
                    {
                        // Header and CRC8H does not contain the sync code
                        u8State = GET_SYNC_STATE;
                        break;
                    }
                    else if((a == -1) && (u8RxByte == SER_SYNCH_CODE))
                    {
                        // Header does not have sync code but CRC8H does.
                        // The sync code could be the beginning of a packet
                        u8State = GET_HEADER_STATE;
                        u16Count = 0;
                        u8CRC    = 0;
                        break;
                    }

                    // Header has a sync byte. It could be a new telegram.
                    // Shift all bytes from the 0x55 code in the buffer.
                    // Recalculate CRC8 for those bytes
                    u8CRC = 0;
                    for (i = 0 ; i < (SER_HEADER_NR_BYTES - a) ; i++)
                    {
                        u8Raw[i] = u8Raw[a+i];
                        u8CRC = proc_crc8(u8CRC, u8Raw[i]);
                    }
                    u16Count = SER_HEADER_NR_BYTES - a;
                    // u16Count = i; // Seems also valid and more intuitive than u16Count -= a;

                    // Copy the just received byte to buffer
                    u8Raw[u16Count++] = u8RxByte;
                    u8CRC = proc_crc8(u8CRC, u8RxByte);

                    if(u16Count < SER_HEADER_NR_BYTES)
                    {
                        u8State = GET_HEADER_STATE;
                        break;
                    }

                    break;
                }

                // CRC8H correct. Length fields values valid?
                if((pPacket->u16DataLength + pPacket->u8OptionLength) == 0)
                {
                    //No. Sync byte received?
                    if((u8RxByte == SER_SYNCH_CODE))
                    {
                        //yes
                        u8State = GET_HEADER_STATE;
                        u16Count = 0;
                        u8CRC    = 0;
                        break;
                    }

                    // Packet with correct CRC8H but wrong length fields.
                    u8State = GET_SYNC_STATE;
                    return OUT_OF_RANGE;
                }

                // Correct header CRC8. Go to the reception of data.
                u8State = GET_DATA_STATE;
                u16Count = 0;
                u8CRC    = 0;

                break;


            // Copy the information bytes
            case GET_DATA_STATE:

                // Copy byte in the packet buffer only if the received bytes have enough room
                if(u16Count < u16BufferLength)
                {
                    pPacket->u8DataBuffer[u16Count] = u8RxByte;
                    u8CRC = proc_crc8(u8CRC, u8RxByte);
                }
```

```
                    // When all expected bytes received, go to calculate data checksum
                    if( ++u16Count == (pPacket->u16DataLength + pPacket->u8OptionLength) )
                    {
                         u8State = CHECK_CRC8D_STATE;
                    }

                    break;


               // Check the data CRC8
               case CHECK_CRC8D_STATE:

                    // In all cases the state returns to the first state: waiting for next sync byte
                    u8State = GET_SYNC_STATE;

                    // Received packet bigger than space to allocate bytes?
                    if (u16Count > u16BufferLength)  return OUT_OF_RANGE;

                    // Enough space to allocate packet. Equals last byte the calculated CRC8?
                    if (u8CRC == u8RxByte)       return OK;                // Correct packet received

                    // False CRC8.
                    // If the received byte equals sync code, then it could be sync byte for next paquet.
                    if((u8RxByte == SER_SYNCH_CODE))
                    {
                         u8State = GET_HEADER_STATE;
                         u16Count = 0;
                         u8CRC    = 0;
                    }

                    return NOT_VALID_CHKSUM;


               default:

                    // Yes. Go to the reception of info.
                   u8State = GET_SYNC_STATE;
                    break;
          }
     }

     return (u8State == GET_SYNC_STATE) ? NO_RX_TEL : NEW_RX_BYTE;
}
```

2.4.3 Send ESP3 Packet

```c
//! \file uart_sendPacket.c


#include "EO3000I_API.h"
#include "proc.h"
#include "uart.h"

/*
ESP3 packet structure through the serial port.

Protocol bytes are generated and sent by the application

Sync = 0x55
CRC8H
CRC8D


    1              2                1           1           1      u16DataLen + u8OptionLen       1
+------+------------------+---------------+-----------+-----------+-------------/------------+-----------+
| 0x55 |     u16DataLen   |   u8OptionLen | u8Type    |    CRC8H  |            DATAS         |    CRC8D  |
+------+------------------+---------------+-----------+-----------+-------------/------------+-----------+


DATAS structure:

                   u16DataLen                    u8OptionLen
+--------------------------------------------+----------------------+
|                   Data                     |     Optional         |
+--------------------------------------------+----------------------+

*/

RETURN_TYPE uart_sendPacket(PACKET_SERIAL_TYPE *pPacket)
{
    uint16 i;
    uint8  u8CRC;

    // When both length fields are 0, then this telegram is not allowed.
    if((pPacket->u16DataLength || pPacket->u8OptionLength) == 0)
    {
        return OUT_OF_RANGE;
    }
    // Sync
    while(uart_sendByte(0x55) != OK);

    // Header
    while(uart_sendBuffer((uint8*)pPacket, 4) != OK);

    // Header CRC
    u8CRC = 0;
    u8CRC = proc_crc8(u8CRC, ((uint8*)pPacket)[0]);
    u8CRC = proc_crc8(u8CRC, ((uint8*)pPacket)[1]);
    u8CRC = proc_crc8(u8CRC, ((uint8*)pPacket)[2]);
    u8CRC = proc_crc8(u8CRC, ((uint8*)pPacket)[3]);
    while(uart_sendByte(u8CRC) != OK);

    // Data
    u8CRC = 0;
    for (i = 0 ; i < (pPacket->u16DataLength + pPacket->u8OptionLength) ; i++)
    {
        u8CRC = proc_crc8(u8CRC, pPacket->u8DataBuffer[i]);
        while(uart_sendByte(pPacket->u8DataBuffer[i]) != OK);
    }

    // Data CRC
    while(uart_sendByte(u8CRC)!=OK);

    return OK;
}
```