Wireless M-Bus Documentation

Stack Reference Manual

STACKFORCE

embedded.connectivity.solutions

March 24, 2015

Contents

List of Tables

List of Figures

Acronym

AMI	Advanced Metering Infrastructure
API	Application Programming Interface
APL	Application Layer
DLL	Data Link Layer
ELL	Extended Data Link Layer
EN	European Norm
FAC	Frequent Access Cycle
FSK	Frequency Shift Keying
GFSK	Gaussian Frequency Shift Keying
HAL	Hardware Abstraction Layer
M-Bus	Meter-Bus
MCU	Microcontroller Unit
NRZ	Non-Return-to-Zero
PHY	Physical Layer
RF	Radio frequency, also commonly used as a synonym for the transceiver
RX	Receive
TPL	Transport Layer
TX	Transmit
WM-Bus	Wireless M-Bus

Chapter 1

Summary

In many regions of the world Meter-Bus (M-Bus) is recognized as a basis for new Advanced Metering Infrastructure (AMI) installations. The corresponding wireless implementation comes with a competitive advantage compared to the existing wired solution. WM-Bus products are easy to install and to maintain. The WM-Bus standard defines the wireless communication between several types of meters and actuators including water, gas, heat and electricity, and data concentrators. The STACKFORCE stack implements the protocols for WM-Bus. It has demonstrated its interoperability with modules of well-known manufacturers on the different protocol layers. It is an optimised solution between small footprint and excellent modularity and scalability. This document provides an introduction to WM-Bus and defines the Application Programming Interface (API) of the WM-Bus protocol stack of the STACKFORCE GmbH.

Chapter 2

Wireless M-Bus Basics

2.1 Standards

M-Bus is a field bus specialized for the transmission of metering data from gas, electricity, heat, water or other meters to a data collector. The European Norm (EN) 13757 defines the standard, which includes the specification of wired and wireless M-Bus. The specification contains five parts:

- EN 13757-1 [1]:
  Part 1: Data exchange
  The first part describes the basic communication between the meters and a central data collector. It provides an overview of the communication system.
- EN 13757-2 [2]:
  Part 2: Physical and link layer
  The second part includes the specification of the physical data transmission using wired connections. It also includes the description of the protocol to transmit the data.
- EN 13757-3 [3]:
  Part 3: Dedicated application layer
  The third part describes a standardized application protocol to enable multi-vendor capability. So devices of different manufacturers may be combined in one system.
- EN 13757-4 [4]:
  Part 4: Wireless meter readout (radio meter reading for operation in the 868 MHz to 870 MHz SRD band)
  This part specifies the wireless communication of M-Bus and is the main source document for the implementation of the WM-Bus stack from STACKFORCE. It includes the Physical Layer (PHY) and the Data Link Layer (DLL) for wireless devices. It corresponds to specification EN 13757-2 for wired communication.
- EN 13757-5 [5]:
  Part 5: Relaying
  This part includes different proposals for relaying data frames to overcome the range problem between remote meters and data collectors.

All parts of EN 13757 are compliant to the EN 870-5 [6].

2.2 The STACKFORCE Protocol Stack

The protocol stack consists of the following major parts:

Wireless M-Bus Stack The stack implements the PHY layer, DLL, Extended Data Link Layer (ELL), Transport Layer (TPL) and the Application Layer (APL).

RF driver The Radio frequency, also commonly used as a synonym for the transceiver (RF) driver configures and controls the connected transceiver. The RF driver somehow is part of the Hardware Abstraction Layer (HAL), as it is abstracting the access to transceiver for the stack, but in parallel the RF driver needs abstracted access to some hardware provided by the Microcontroller Unit (MCU), e.g. the SPI interface and GPIOs.

HAL The HAL is responsible to abstract all the hardware resources required by the stack and the RF driver. E.g. this includes SPI interface, UART interface, access to non-volatile memory, GPIO, …

Figure 2.1 provides an overview over the WM-Bus protocol stack implementation of STACKFORCE.

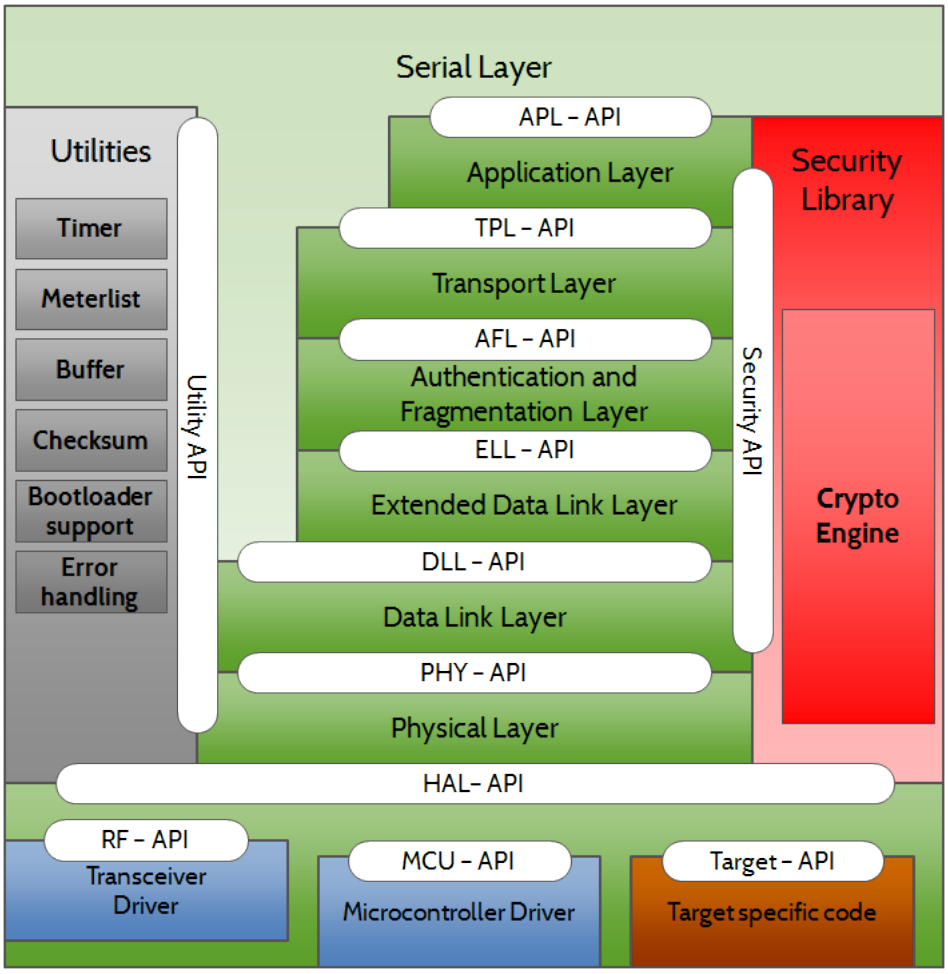

Figure 2.1: Architecture of the Wireless M-Bus Protocol Stack.

Please note, this picture illustrates the complete stack as available and developed by STACKFORCE. The received package may not include all parts that are illustrated. For details on which parts are shipped with this package please contact support at distributor of your package or STACKFORCE.

Please contact the STACKFORCE to obtain information about configurations, extensions and adaptations of the Wireless M-Bus Stack. Chapter 5 contains contact information.

2.3 Topology

In general, WM-Bus supports two types of devices [4]:

- "Meter" device:
  A "Meter" device is a device which collects and forwards information to an "Other" device. E.g., a meter could be an electric meter or a gas meter.
- "Other" device:
  An "Other" device is a system component which receives information from a "Meter" device. This could be "*mobile readout devices, stationary receivers, data collectors, multi-utility concentrators or system network components*" [4]. In the following, this document denotes an "Other" device as data collector.

WM-Bus favours asymmetric network topologies with low-cost or low-power metering devices on the one side and data collectors or gateways with higher performance on the other side. Currently, only point-to-point or star network topologies are supported.

2.4 WM-Bus Modes

Wireless M-Bus supports various comminication modes. The Wireless M-Bus modes define the communication flow and the configuration of the radio channel. Table 2.1 lists supported communication modes from the STACKFORCE Wireless M-Bus Stack.

Mode	Description
$S1, S2$	In the **S**tationary mode, the metering devices send their data several times a day. In this mode, the data collector may save power as the metering devices send a wakeup signal before transmitting their data.
$T1, T2$	In the Frequent **T**ransmit mode, the metering devices periodically send their data to collectors in range. The interval is configurable in terms of several seconds or minutes.
$C1, C2$	**C**ompact mode. This mode is similar to mode T but it allows for transmission of more data within the same energy budget and with the same duty cycle. It is suitable for walk-by and/or drive-by readout. The common reception of mode T and mode C frames with a single receiver is possible.
$N1(a-f), N2(a-f)$	**N**arrowband communication mode for long range transmissions.

Table 2.1: Wireless M-Bus operating modes

In general, Wireless M-Bus modes can have different data rates, data encodings (e.g. Non-Return-to-Zero (NRZ) or Manchester), frequency modulations (e.g. Frequency Shift Keying (FSK) or Gaussian Frequency Shift Keying (GFSK)) and carrier frequencies. Table 2.2 lists the related parameters with respect to the Wireless M-Bus mode, the device type and whether a device is in Receive (RX) or Transmit (TX) mode.

Mode	Meter		Collector		Data Rate	Endocing	Modulation	Frequency [MHz]
	RX	TX	RX	TX				
$N1a, N2a$	✓	✓	✓	✓	4.8 kbit/s	NRZ	GFSK	169.406250
$N1b, N2b$	✓	✓	✓	✓	4.8 kbit/s	NRZ	GFSK	169.418750
$N1c, N2c$	✓	✓	✓	✓	2.4 kbit/s	NRZ	GFSK	169.431250
$N1d, N2d$	✓	✓	✓	✓	2.4 kbit/s	NRZ	GFSK	169.443750
$N1e, N2e$	✓	✓	✓	✓	4.8 kbit/s	NRZ	GFSK	169.456250
$N1f, N2f$	✓	✓	✓	✓	4.8 kbit/s	NRZ	GFSK	169.468750
$T2$	✓			✓	32.768 kcps	Manchester	FSK	868.30
$T1, T2$		✓	✓		100 kcps	3-out-of-6	FSK	868.95
$S1, S2$	✓	✓	✓	✓	32.768 kcps	Manchester	FSK	868.30
$C2$	✓			✓	50 kcps	NRZ	GFSK	869.525
$C2$				✓	32.768 kcps	Manchester	FSK	868.30
$C2$			✓		100 kcps	3-out-of-6	FSK	869.95
$C1, C2$		✓	✓		100 kcps	NRZ	FSK	868.95

Table 2.2: Wireless M-Bus mode parameter settings.

Figure 2.2 provides a graphical overview over the TX frequencies in Wireless M-Bus modes S, T, C and N.

Figure 2.3 provides a graphical overview over the TX frequencies in Wireless M-Bus modes $N1(a-f)$ and $N2(a-f)$.

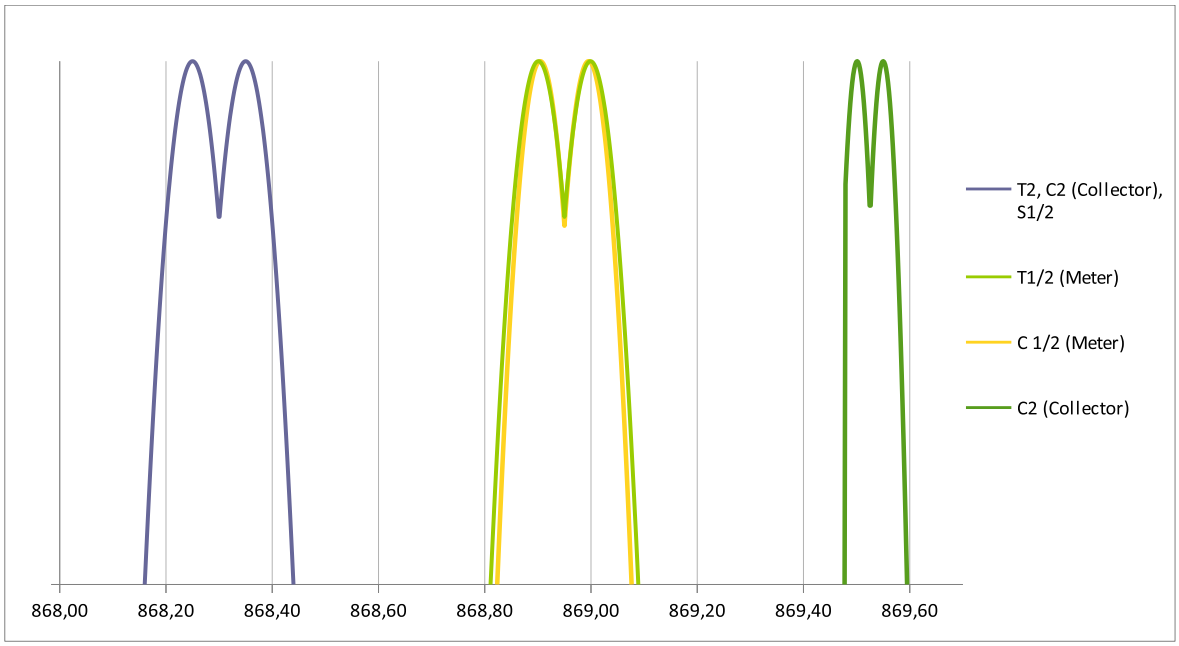

Figure 2.2: Frequencies of Wireless M-Bus modes S, T and C.

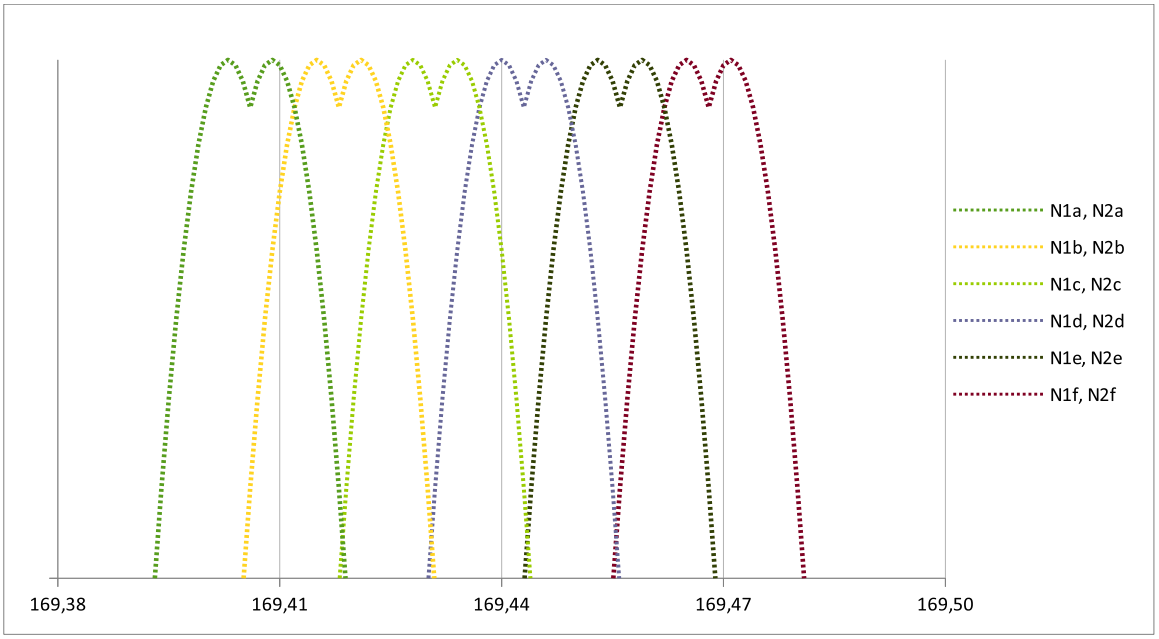

Figure 2.3: Frequencies of Wireless M-Bus mode N.

Table 2.3 specifies the compatibility of different Wireless M-Bus modes.

	Meter S	**Meter** T	**Meter** C	**Meter** $N(a-f)$
Collector S	✓			
Collector T		✓		
Collector C		✓	✓	
Collector $N(a-f)$				✓

Table 2.3: Compatibility matrix for Wireless M-Bus modes.

In summary, the Wireless M-Bus modes S and N (in mode N, every sub-mode $a-f$ is only compatible with the same sub-mode) are only compatible with the same Wireless M-Bus mode. But T is partially compatible to C - a data collector in mode C is able to communicate with a meter device in mode T.

2.5 Unidirectional and Bidirectional Communication

Based on the WM-Bus mode, two communicatino models are available:

1. Unidirectional
2. Bidirectinoal

Unidirectional WM-Bus modes support only data transmission from a meter device to a data collector. The advantages of those modes is the low overhead implementation of the single devices - a meter device must only transmit data, and a data collector must only receive data.

In contrast, bidirectional WM-Bus modes addiontally support a communication from a data collector to a meter device. Data collectors support bidirectional modes only and are able to request information from a bidirectional meter devices. Figure 2.4 shows a typical communication flow. The data collector sends a telegram "*Request User Data 2*" to the collector. The meter device receives the telegram and answers with a "*Response User Data 2*" telegram containing related information to the collector. The answer of the meter will be repeated until the collector closes the communication or a timeout occurs (according to the Frequent Access Cycle (FAC) defined in [4]).

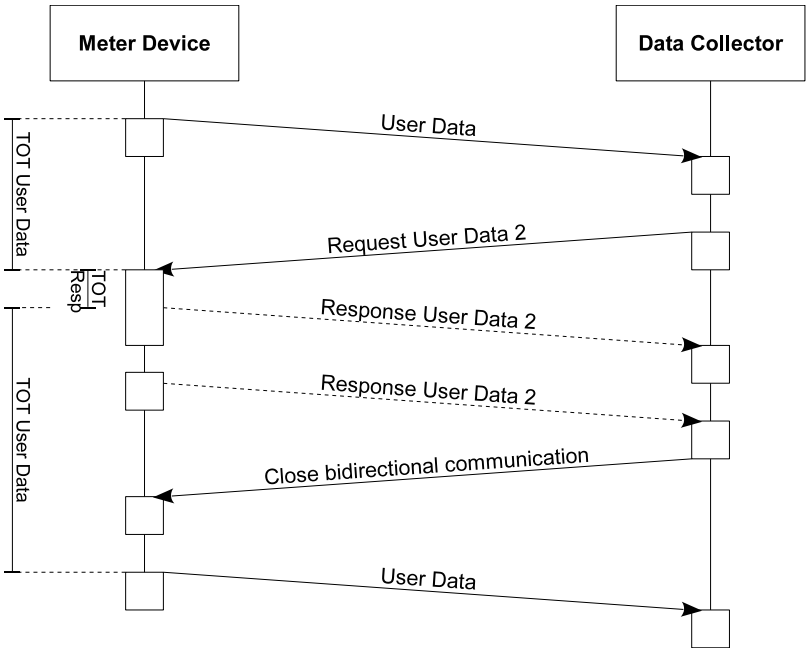

Figure 2.4: Bidirectional communication example for WM-Bus modes $S2$, $T2$, $C2$ and $N2(a-f)$.

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

File Documentation

4.1 inc/pub/tpl/wmbus_tpl_api.h File Reference

Transport Layer Application Programming Interface of Wireless M-Bus.

```
#include "wmbus_global.h"
```

Data Structures

- struct s_tpl_header_t
- struct s_tpl_headerLong_t
- struct s_tpl_headerShort_t
- struct s_tpl_headerExtI_t
- struct s_tpl_headerExtII_t
- struct s_tpl_meterEntry_t
- struct s_tpl_meterList_t
- struct s_tpl_startAttr_t
- struct s_tpl_addMeterRet_t
- struct s_tpl_tlgAttr_t

Macros

- #define **__DECL_TRANSPORTLAYER_H__** extern
- #define **AES_SIZE_OF_KEY** 0x10U
- #define **AES_DEFAULT_NETWORK_KEY**
- #define DES_DYNAMIC_IV_DECRYPT_ENABLED FALSE
- #define DES_DYNAMIC_IV_ENCRYPT_ENABLED FALSE
- #define DES_STATIC_IV_DECRYPT_ENABLED FALSE

- #define DES_STATIC_IV_ENCRYPT_ENABLED FALSE
- #define TPL_FIELD_CI_RESET 0x50U
- #define TPL_FIELD_CI_DATA_SEND 0x51U
- #define TPL_FIELD_CI_SLAVE_SELECT 0x52U
- #define TPL_FIELD_CI_CMD_TO_DEVICE_SHORT 0x5AU
- #define TPL_FIELD_CI_CMD_TO_DEVICE_LONG 0x5BU
- #define TPL_FIELD_CI_TIME_SYNC_1 0x6CU
- #define TPL_FIELD_CI_TIME_SYNC_2 0x6DU
- #define TPL_FIELD_CI_APL_ERROR_SHORT 0x6EU
- #define TPL_FIELD_CI_APL_ERROR_LONG 0x6FU
- #define TPL_FIELD_CI_APL_ERROR 0x70U
- #define TPL_FIELD_CI_ALARM 0x71U
- #define TPL_FIELD_CI_HEADER_LONG 0x72U
- #define TPL_FIELD_CI_APL_ALARM_SHORT 0x74U
- #define TPL_FIELD_CI_APL_ALARM_LONG 0x75U
- #define TPL_FIELD_CI_HEADER_NO 0x78U
- #define TPL_FIELD_CI_HEADER_SHORT 0x7AU
- #define TPL_FIELD_CI_LINK_TO_DEVICE_LONG 0x80U
- #define TPL_FIELD_CI_LINK_FROM_DEVICE_SHORT 0x8AU
- #define TPL_FIELD_CI_LINK_FROM_DEVICE_LONG 0x8BU
- #define TPL_FIELD_CI_EXT_DLL_I 0x8CU
- #define TPL_FIELD_CI_EXT_DLL_II 0x8DU
- #define TPL_FIELD_CI_MANU_NO 0xA1U
- #define TPL_FIELD_CI_MANU_SHORT 0xA2U
- #define TPL_FIELD_CI_MANU_LONG 0xA3U
- #define TPL_FIELD_CI_MANU_SHORT_RF 0xA4U
- #define TPL_FIELD_CI_NULL 0xFFU
- #define TPL_FIELD_CI_LEN 0x01U
- #define TPL_ENCRYPTION_MODE_0 0x00U
- #define TPL_ENCRYPTION_MODE_4 0x04U
- #define TPL_ENCRYPTION_MODE_5 0x05U
- #define TPL_ENCRYPTION_MODE_7 0x07U
- #define TPL_ENCRYPTION_MODE_15 0x0FU

- #define **TPL_MAX_HEADER_LENGTH** 12U
- #define TPL_STATUS_BUSY 0x01U
- #define TPL_STATUS_CONNECTED 0x08U
- #define TPL_TLG_FLAG_CLR 0x0000U
- #define TPL_TLG_FLAG_SIGNATURE 0x000CU
- #define TPL_TLG_FLAG_SIGNATURE_UNKNOWN 0x0004U
- #define TPL_TLG_FLAG_SIGNATURE_ERROR 0x0008U
- #define TPL_TLG_FLAG_SIGNATURE_NO 0x000CU
- #define TPL_TLG_FLAG_HOP_COUNTER 0x0010U
- #define TPL_TLG_FLAG_REPEATER_ACCESS 0x0020U
- #define **TPL_TLG_FLAG_CONTENT_OF_MESSAGE_0** 0x0040U
- #define **TPL_TLG_FLAG_CONTENT_OF_MESSAGE_1** 0x0080U
- #define **TPL_TLG_FLAG_ENCRYPTION_MODE_MASK** 0x1F00U
- #define **TPL_TLG_FLAG_ENCRYPTION_MODE_5** 0x0500U
- #define **TPL_TLG_FLAG_ENCRYPTION_MODE_7** 0x0700U
- #define **TPL_TLG_FLAG_ENCRYPTION_MODE_13** 0x0D00U
- #define TPL_TLG_FLAG_SYNC 0x2000U
- #define TPL_TLG_FLAG_ACCESSIBILITY 0x4000U
- #define TPL_TLG_FLAG_BIDIRECTIONAL 0x8000U
- #define TPL_FIELD_CONF_WORD_HOP_COUNTER 0x0001U
- #define TPL_FIELD_CONF_WORD_REPEATER_ACCESS 0x0002U
- #define TPL_FIELD_CONF_WORD_CONTENT_OF_MESSAGE_MASK 0x000CU
- #define TPL_FIELD_CONF_WORD_CONTENT_OF_MESSAGE_0 0x0004U
- #define TPL_FIELD_CONF_WORD_CONTENT_OF_MESSAGE_1 0x0008U
- #define TPL_FIELD_CONF_WORD_MODE_7_CONTENT_OF_MESSAGE_0 0x4000U
- #define TPL_FIELD_CONF_WORD_MODE_7_CONTENT_OF_MESSAGE_1 0x8000U
- #define TPL_FIELD_CONF_WORD_ENCRYPTION_BLOCK_MASK 0x00F0U
- #define TPL_FIELD_CONF_WORD_ENCRYPTION_MODE_MASK 0x1F00U
- #define TPL_FIELD_CONF_WORD_NO_ENCRYPTION 0x0000U
- #define TPL_FIELD_CONF_WORD_ENCRYPTION_MODE_2 0x0200U
- #define TPL_FIELD_CONF_WORD_ENCRYPTION_MODE_3 0x0300U
- #define TPL_FIELD_CONF_WORD_ENCRYPTION_MODE_4 0x0400U
- #define TPL_FIELD_CONF_WORD_ENCRYPTION_MODE_5 0x0500U

- #define TPL_FIELD_CONF_WORD_ENCRYPTION_MODE_7 0x0700U
- #define TPL_FIELD_CONF_WORD_ENCRYPTION_MODE_13 0x0D00U
- #define TPL_FIELD_CONF_WORD_ENCRYPTION_MODE_15 0x0F00U
- #define TPL_FIELD_CONF_WORD_BI_ACCESSIBILITY_MASK 0xC000U
- #define TPL_FIELD_CONF_WORD_BIDIRECTIONAL 0x8000U
- #define TPL_FIELD_CONF_WORD_ACCESSIBILITY 0x4000U
- #define TPL_FIELD_CONF_WORD_SYNC 0x2000U
- #define **TPL_METERLIST_ENTRY_LEN**

Enumerations

- enum E_TPL_CRYPT_RET_t { E_TPL_CRYPT_RET_OK, E_TPL_CRYPT_RET_ERR, E_TPL_CRYPT_RET_UNKNOWN, E_TPL_CRYPT_RET_TOO_FEW_BYTES, E_TPL_CRYPT_RET_INVALID }
- enum E_TPL_RET_t { E_TPL_RET_OK = 0U, E_TPL_RET_ERR, E_TPL_RET_NOT_READY, E_TPL_RET_DUPLICATED }
- enum E_TPL_HEADER_TYPE_t { E_TPL_HEADER_TYPE_NO, E_TPL_HEADER_TYPE_SHORT, E_TPL_HEADER_TYPE_LONG, E_TPL_HEADER_TYPE_EXT_I, E_TPL_HEADER_TYPE_EXT_II, E_TPL_HEADER_TYPE_INVALID }
- enum E_TPL_STATUS_t { E_TPL_STATUS_ERROR, E_TPL_STATUS_INVALID_PARAM_ERROR, E_TPL_STATUS_NOT_INIT_ERROR, E_TPL_STATUS_SUCCESS }
- enum E_TPL_AUTH_RESULT_t { E_TPL_AUTH_RESULT_OK, E_TPL_AUTH_RESULT_NONE, E_TPL_AUTH_RESULT_ERROR }

Functions

- E_TPL_HEADER_TYPE_t wmbus_tpl_evt_getCiHeader (uint8_t c_ci)

  Transmits a CI field and requires the type of headers.

- void wmbus_tpl_evt_mtr_sendUserData (void)

  Meter device only. This function is called if periodical user data has to be sent. For this a new telegram has to be created and filled with user data. Be careful: In mode T2 a telegram with C-field DLL_FIELD_C_PRM_AD should be sent, in other modes DLL_FIELD_C_PRM_UD_NOREPL. This telegram has to be sent from higher layers because of the different M-Bus specifications as OMS or DSMR.

- void wmbus_tpl_evt_col_tlgAvailable (E_WMBUS_RX_t e_status, uint8_t c_tlgReqId, uint8_t c_tlgId)

This function is called if a telegram is available. Collector device.

- void wmbus_tpl_evt_mtr_tlgAvailable (E_WMBUS_RX_t e_status, uint8_t c_tlgReqId, uint8_t c_tlgId)

  This function is called if a telegram is available. Meter device.

- void wmbus_tpl_evt_col_opened (uint8_t c_tlgId)

  This function is called if a bidirectional communication with a meter device is started. It is the acknowledgement to wmbus_dll_open().

- void wmbus_tpl_evt_mtr_tx (uint8_t c_tlgId)

  This function is called if a telegram is sent successfully. Meter deivce.

- void wmbus_tpl_evt_col_tx (uint8_t c_tlgId)

  This function is called if a telegram is sent successfully. Collector device.

- void wmbus_tpl_init (void)

  Initializes the transport layer.

- E_TPL_STATUS_t wmbus_tpl_start (s_tpl_startAttr_t *s_startAttr)

  Starts the transport layer.

- void wmbus_tpl_run (void)

  Default running function of the transport layer. This function has to be called as often as possible.

- uint8_t wmbus_tpl_getStatus (void)

  Returns the current status of the transport layer.

- E_TPL_HEADER_TYPE_t wmbus_tpl_getHeaderType (uint8_t c_ci)

  Reads the header type of the CI field.

- bool_t wmbus_tpl_setKey (uint8_t *pc_key, uint8_t c_len)

  Writes the local encryption key.

- bool_t wmbus_tpl_getKey (uint8_t *pc_key, uint8_t c_len)

  Reads the local encryption key.

- bool_t wmbus_tpl_encryptPrepare (uint8_t c_tlgId)

  Prepares data encryption.

- E_TPL_CRYPT_RET_t wmbus_tpl_encrypt (uint8_t c_tlgId)

  Encrypts a complete telegrams.

- E_TPL_CRYPT_RET_t wmbus_tpl_decrypt (uint8_t c_tlgId)

  Decrypts a complete telegrams.

- uint8_t wmbus_tpl_createTlg (uint8_t c_type, s_wmbus_addr_t *ps_meterAddr, uint8_t c_ci)

  Creates a new telegram.

- bool_t wmbus_tpl_writeTlg (uint8_t c_tlgId, uint8_t *pc_data, uint16_t i_len, uint16_t i_offset, bool_t b_reverse)

  Writes data to a telegram.

- uint16_t wmbus_tpl_cntDataBytes (uint8_t c_tlgId)

  Counts the number of telegram data bytes.

- void wmbus_tpl_setTxWait (uint32_t l_timeout)

  Sets a timeout to wait before sending the next telegram.

- bool_t wmbus_tpl_sendTlg (uint8_t c_tlgId)

  Sends a telegram. If the telegram is sent successfully, it is destroyed by the data link layer automatically.

- bool_t wmbus_tpl_destroyTlg (uint8_t c_tlgId)

  Destroys a telegram.

- uint16_t wmbus_tpl_receiveTlg (uint8_t c_tlgId, uint8_t *pc_data, uint16_t i_len, uint16_t i_offset, bool_t b_reverse)

  Receives data from a telegram.

- bool_t wmbus_tpl_getTlgAttr (uint8_t c_tlgId, s_tpl_tlgAttr_t *ps_tlgAttrRecv)

  Reads the attributes of a received telegram.

- bool_t wmbus_tpl_setHeader (uint8_t c_tlgId, s_tpl_header_t *ps_header)

  Sets the header values of a telegram.

- bool_t wmbus_tpl_getHeader (uint8_t c_tlgId, s_tpl_header_t *ps_header)

  Reads the header of a telegram.

- void wmbus_tpl_setAccessDemandFlag (void)

  Signalizes that user data class 1 is available. This flag is reset if there is an incoming user data class 1 request.

- bool_t wmbus_tpl_open (void)

  Start a bidirectional communication the next time as possible. Function used by collector device only.

- bool_t wmbus_tpl_sendQueued (uint8_t c_tlgId, uint16_t i_meterId, bool_t b_append)

  Adds a telegram to the collector queue. Function used by collector device only.

- uint8_t wmbus_tpl_close (uint16_t i_meterId, bool_t b_sendNke)

  Resets the flag to start a bidirectional communication. Function used by collector device only.

- s_tpl_addMeterRet_t wmbus_tpl_meterAdd (s_tpl_meterEntry_t *ps_meterEntry)

  Adds a meter device to the meter list. Function used by collector device only.

- E_TPL_RET_t wmbus_tpl_meterRemove (s_wmbus_addr_t *ps_meter)

  Removes a meter device from the meter list. Function used by collector device only.

- E_TPL_RET_t wmbus_tpl_meterSetRfAdapter (s_wmbus_addr_t *ps_meter, s_wmbus_addr_t *ps_rfadapter)

  Updates the RF adapter address of the meter device. Function used by collector device only.

- bool_t wmbus_tpl_meterGetRfAdapter (s_wmbus_addr_t *ps_meter, s_wmbus_addr_t *ps_rfadapter)

  Reads the rf adapter address of the meter. Function used by collector device only.

- E_TPL_RET_t wmbus_tpl_meterSetKey (s_wmbus_addr_t *ps_meter, uint8_t *pc_key, uint8_t c_len)

  Updates the meter specific key. Function used by collector device only.

- bool_t wmbus_tpl_meterGetKey (s_wmbus_addr_t *ps_meter, uint8_t *pc_key, uint8_t c_len)

  Reads the key of the meter. Function used by collector device only.

- uint16_t wmbus_tpl_meterGetNum (void)

  Gets the number of connected meter devices. Function used by collector device only.

- bool_t wmbus_tpl_meterGetAddr (uint16_t i_index, s_wmbus_addr_t *ps_meterAddr)

  Reads the address of a meter device. Function used by collector device only.

- uint16_t wmbus_tpl_meterGetIndex (s_wmbus_addr_t *ps_meterAddr)

  Reads the index of a meter device. Function used by collector device only.

- void wmbus_tpl_getAddrOwn (s_wmbus_addr_t *ps_meterAddr)

  Returns the address of the device.

- void wmbus_tpl_setAddrOwn (s_wmbus_addr_t *ps_meterAddr)

  Sets the address of the meter device.

- uint32_t wmbus_tpl_getInterval (void)

  Returns the periodical interval of a meter device. If the device is a collector always 0 is returned.

- void wmbus_tpl_setInterval (uint32_t l_interval)

  Sets the periodical interval of a meter device.

- void wmbus_tpl_setConnected (bool_t b_connected)

  Sets the device connected or disconnected.

- void wmbus_tpl_setCollectorAddr (s_wmbus_addr_t *ps_collectorAddr)

  Sets the address of the data collector. (needed for OMS)

- void wmbus_tpl_getCollectorAddr (s_wmbus_addr_t *ps_collectorAddr)

  Reads the address of the data collector. (needed for OMS)

- E_WMBUS_SLEEP_RESULT_t wmbus_tpl_sleep (E_WMBUS_SLEEP_MODE_t e_sleepMode)

  Checks if the dll is able to go in low power-mode.

- void wmbus_tpl_wakeUp (E_WMBUS_SLEEP_MODE_t e_sleepMode)

Wake up the target.

- void wmbus_tpl_setTlgFormat (uint8_t c_tlgId, E_WMBUS_FRAME_t e_frameType)

  Sets the frame type of the telegram. This function should only be used in C-mode. Only the C-mode supports packets in format A and B.

- void wmbus_tpl_setTlgMode (uint8_t c_tlgId, E_WMBUS_MODE_t e_mode)

  Sets the mode of the telegram.

- void wmbus_tpl_setTxPower (uint8_t c_txPower)

  Sets the transmission power of the rf-module. Please have a look in the corresponding data sheet of the selected transceiver to choose a supported transmission power.

- uint8_t wmbus_tpl_getTxPower (void)

  Reads the transmission power.

- void wmbus_tpl_evt_col_ACDBitSet (uint8_t c_tlgId)

  This function is called if the ACD bit of a meter telegram is set. Function used by collector device only.

- bool_t wmbus_tpl_setRfChannel (E_RADIO_CHANNEL_INDEX_t e_channel)

  Sets the radio channel of the rf-module. This function should only be used in mode N to set the correct channel for the device.

- E_RADIO_CHANNEL_INDEX_t wmbus_tpl_getRfChannel (void)

  Reads the radio channel of the rf-module.

- uint8_t wmbus_tpl_accessNoIncrementBy (uint8_t c_numberOfIncrements)

  Increments the access number which is used from the tpl and the apl.

- uint8_t wmbus_tpl_getAccessNo (void)

  Returns the current access number which is used from the tpl and the apl.

- void wmbus_tpl_setAccessNo (uint8_t c_accessNo)

  Sets the current access number which is used from the tpl and the apl.

- uint8_t wmbus_tpl_getHeaderStatus (void)

  Returns the current status byte which is used from the tpl and the apl.

- void wmbus_tpl_setHeaderStatusFlag (uint8_t c_flag)

  Sets a flag in the current status byte which is used from the tpl and the apl.

- void wmbus_tpl_clrHeaderStatusFlag (uint8_t c_flag)

  Clears a flag in the current status byte which is used from the tpl and the apl.

- E_WMBUS_ACCESSIBILITY_t wmbus_tpl_getAccessibility (void)

  Returns the global accessibility setting of the meter device. Default = E_WMBUS_ACCESSIBILITY_BIDIRECTIONAL_ACCESS Function used by bidirectional meter device only.

- void wmbus_tpl_setAccessibility (E_WMBUS_ACCESSIBILITY_t e_access)

  This function is only for meter devices. Set the acessibiolity type of the meter to save energy. Function used by bidirectional meter device only.

- void wmbus_tpl_bufCleanUp (void)

  Cleans up all buffers.

- uint8_t wmbus_tpl_sendAck (uint8_t c_accNo, uint16_t i_meterId)

  Sends an Ack message.

- void wmbus_tpl_col_clearTlgQueue (void)

  This function resets the complete queue.

- E_WMBUS_COL_QUEUE_RET_t wmbus_tpl_col_clearMtrTlgQueue (uint16_t i_meterId)

  Removes all telegrams from the queue for a specific meter.

- bool_t wmbus_tpl_mtr_setResponseDelay (E_WMBUS_RESPONSE_DELAY_t e_respDelay)

  Sets the response delay to be used by the bidirectional meter device of WMBus mode C or WMBus mode N. The user given input may be overwritten by the collector the meter is connected to.

4.1.1 Detailed Description

Transport Layer Application Programming Interface of Wireless M-Bus.
Copyright

STACKFORCE GmbH, Heitersheim, Germany, http://www.stackforce.de
Author

STACKFORCE

4.1.2 Data Structure Documentation

4.1.2.1 struct s_tpl_header_t

Structure of an application header.

Data Fields

E_TPL_HEADER_TYPE_t	e_type	Header type.

uint8_t	pc_data[TPL_↩ MAX_HEADER↩ _LENGTH]	Default length

4.1.2.2 struct s_tpl_headerLong_t

Structure of an long application header.

Data Fields

uint8_t	c_accNo	Access number.
uint8_t	c_status	Status byte.
E_TPL_HEAD↩ ER_TYPE_t	e_type	Header type.
uint16_t	i_signature	Signature word.
s_wmbus_↩ addr_t	s_addr	Application address.

4.1.2.3 struct s_tpl_headerShort_t

Structure of an short application header.

Data Fields

uint8_t	c_accNo	Access number.
uint8_t	c_status	Status byte.
E_TPL_HEAD↩ ER_TYPE_t	e_type	Header type.
uint16_t	i_signature	Signature word.

4.1.2.4 struct s_tpl_headerExtI_t

Structure of an extended link layer header version I.

Data Fields

uint8_t	c_accNo	Access number.
uint8_t	c_commControl	Communication control field
E_TPL_HEAD↩ ER_TYPE_t	e_type	Header type.

4.1.2.5 struct s_tpl_headerExtII_t

Structure of an extended link layer header version II.

Data Fields

uint8_t	c_accNo	Access number.
uint8_t	c_commControl	Communication control field
E_TPL_HEAD↩ ER_TYPE_t	e_type	Header type.
uint16_t	i_payloadCrc	Payload CRC
uint32_t	l_sessionNr	Session number.

4.1.2.6 struct s_tpl_meterEntry_t

Structure of one meter entry in the list.

Data Fields

E_WMBUS_M↩ ODE_t	e_wmbusMode	Wmbus mode
uint8_t	pc_meter↩ Key[0x10U]	Key of the meter device
s_wmbus_↩ addr_t	s_meterAddr	Address of the meter device
s_wmbus_↩ addr_t	s_rfAdapter	RF adapter

4.1.2.7 struct s_tpl_meterList_t

Structure of the meter list. Please note, that the meter entry array must be a packed list!.

Data Fields

uint16_t	i_numberOf↩ Meters	Number of meter devices in the list.
s_tpl_meter↩ Entry_t *	ps_meterEntry	Pointer to the first meter entry.

4.1.2.8 struct s_tpl_startAttr_t

Start structure to start a meter device

Data Fields

s_wmbus_↩ addr_t *	s_deviceAddr	Address of the device
sint16_t	si_freqOffset	Frequency offset of the carrier. If the parameter is not known set it to 0.

4.1.2.9 struct s_tpl_addMeterRet_t

Structure of the return value of the function wmbus_mem_mgmt_mlMeterAdd.

Data Fields

E_TPL_RET_t	e_ret	
uint16_t	i_meterId	

4.1.2.10 struct s_tpl_tlgAttr_t

Data Fields

bool_t	b_access↩ Demand	TRUE if user data class 1 is available.
bool_t	b_flowControl	Data Collector only: TRUE if further requests may be sent. FALSE if further requests could cause an overflow.

uint8_t	c_accNo	Access number of the received telegram.
uint8_t	c_controlField	Type of the telegram (C-field).
uint8_t	c_controlInfo	Type of the data (CI-field). If no CI field is received, this value is APL_FIELD_CI_NULL.
uint8_t	c_quality	Quality of a received telegram. FF no link quality available FE -254 dBm ... 00 0 dBm
uint8_t	c_status	Status bytes, read from status byte of the telegram.
uint8_t	c_tlgId	Id of the telegram.
E_TPL_AUTH_RESULT_t	e_aflAuthenticationResult	
E_WMBUS_FRAME_t	e_frameType	Defines the type of frame format.
E_WMBUS_MODE_t	e_mode	Mode in which the telegram was received. [S, T, C] In C mode also T packets would be received
uint16_t	i_flag	Telegram communication flag register. Bit 0,1 Reserved Bit 2,3 Signature 00 No signature error 01 Signature unknown 10 Encryption error 11 No encryption Bit 4 Hop counter Bit 5 Repeater access Bit 6,7 Content of message Bit 8-12 Encryption mode Bit 13 Synchronous comminication Bit 14 Accessibility Bit 15 Bidirectional communication
uint16_t	i_len	Number of received data bytes. 0 if no data is available.
s_wmbus_addr_t	s_longHeaderAddr	Address information used for the long telegram header
s_wmbus_addr_t	s_meterAddr	Address of the meter device the telegram was received from. If the target is a meter device, this field includes invalid data.

4.1.3 Macro Definition Documentation

4.1.3.1 #define AES_DEFAULT_NETWORK_KEY

Value:

```
{0xFF, 0xFF, 0xFF, 0xFF,\
                                0xFF, 0xFF, 0xFF, 0xFF,\
                                0xFF, 0xFF, 0xFF, 0xFF,\
                                0xFF, 0xFF, 0xFF, 0xFF}
```

4.1.3.2 #define DES_DYNAMIC_IV_DECRYPT_ENABLED FALSE

DES + CBC decryption with dynamic IV disabled.

4.1.3.3 #define DES_DYNAMIC_IV_ENCRYPT_ENABLED FALSE

DES + CBC encryption with dynamic IV disabled.

4.1.3.4 #define DES_STATIC_IV_DECRYPT_ENABLED FALSE

DES + CBC decryption with static IV disabled.

4.1.3.5 #define DES_STATIC_IV_ENCRYPT_ENABLED FALSE

DES + CBC encryption with static IV disabled.

4.1.3.6 #define TPL_ENCRYPTION_MODE_0 0x00U

Use no encryption.

4.1.3.7 #define TPL_ENCRYPTION_MODE_15 0x0FU

AES + CBC with dynamic IV for DSMR

4.1.3.8 #define TPL_ENCRYPTION_MODE_4 0x04U

AES + CBC

4.1.3.9 #define TPL_ENCRYPTION_MODE_5 0x05U

AES + CBC with dynamic IV

4.1.3.10 #define TPL_ENCRYPTION_MODE_7 0x07U

AES + CBC with dynamic Key

4.1.3.11 #define TPL_FIELD_CI_ALARM 0x71U

report of alarms (EN 13757-3)

4.1.3.12 #define TPL_FIELD_CI_APL_ALARM_LONG 0x75U

Alarm from device with long header (OMS Vol.2 Issue 3.0.0xxxx-xx-xx)

4.1.3.13 #define TPL_FIELD_CI_APL_ALARM_SHORT 0x74U

Alarm from device with short header (OMS Vol.2 Issue 3.0.0xxxx-xx-xx)

4.1.3.14 #define TPL_FIELD_CI_APL_ERROR 0x70U

Report of application errors (EN 13757-3)

4.1.3.15 #define TPL_FIELD_CI_APL_ERROR_LONG 0x6FU

Error from device with long header (OMS Vol.2 Issue 3.0.0xxxx-xx-xx)

4.1.3.16 #define TPL_FIELD_CI_APL_ERROR_SHORT 0x6EU

Error from device with short header (OMS Vol.2 Issue 3.0.0xxxx-xx-xx)

4.1.3.17 #define TPL_FIELD_CI_CMD_TO_DEVICE_LONG 0x5BU

CMD to device with long header (OMS Vol.2 Issue 2.0.0/2009-07-20) (DSMR 2.3.1)

4.1.3.18 #define TPL_FIELD_CI_CMD_TO_DEVICE_SHORT 0x5AU

CMD to device with short header (OMS Vol.2 Issue 2.0.0/2009-07-20) (DSMR 2.3.1)

4.1.3.19 #define TPL_FIELD_CI_DATA_SEND 0x51U

Data send (EN 13757-3)

4.1.3.20 #define TPL_FIELD_CI_EXT_DLL_I 0x8CU

Additional Link Layer may be applied for Radio messages with or without Application Layer.

4.1.3.21 #define TPL_FIELD_CI_EXT_DLL_II 0x8DU

Additional Link Layer may be applied for Radio messages with or without Application Layer.

4.1.3.22 #define TPL_FIELD_CI_HEADER_LONG 0x72U

12 byte header followed by variable format data (EN 13757-3)

4.1.3.23 #define TPL_FIELD_CI_HEADER_NO 0x78U

Variable data format respond without header (EN 13757-3)

4.1.3.24 #define TPL_FIELD_CI_HEADER_SHORT 0x7AU

4 byte header followed by variable data format respond (EN 13757-3)

4.1.3.25 #define TPL_FIELD_CI_LEN 0x01U

Length of the control information field.

4.1.3.26 #define TPL_FIELD_CI_LINK_FROM_DEVICE_LONG 0x8BU

Link extension from device with long header (OMS Vol.2 Issue 2.0.0/2009-07-20)

4.1.3.27 #define TPL_FIELD_CI_LINK_FROM_DEVICE_SHORT 0x8AU

Link extension from device with short header (OMS Vol.2 Issue 2.0.0/2009-07-20)

4.1.3.28 #define TPL_FIELD_CI_LINK_TO_DEVICE_LONG 0x80U

Link extension to device with long header (OMS Vol.2 Issue 2.0.02009-07-20)

4.1.3.29 #define TPL_FIELD_CI_MANU_LONG 0xA3U

Manufacture specific CI-Field with long header

4.1.3.30 #define TPL_FIELD_CI_MANU_NO 0xA1U

Manufacture specific CI-Field with no header

4.1.3.31 #define TPL_FIELD_CI_MANU_SHORT 0xA2U

Manufacture specific CI-Field with short header

4.1.3.32 #define TPL_FIELD_CI_MANU_SHORT_RF 0xA4U

Manufacture specific CI-Field with short header (for RF-Test)

4.1.3.33 #define TPL_FIELD_CI_NULL 0xFFU

No CI-field transmitted.

4.1.3.34 #define TPL_FIELD_CI_RESET 0x50U

Reset(EN 13757-3)

4.1.3.35 #define TPL_FIELD_CI_SLAVE_SELECT 0x52U

Slave select (EN 13757-3)

4.1.3.36 #define TPL_FIELD_CI_TIME_SYNC_1 0x6CU

Time synchronization (OMS Vol.2 Issue 3.0.0xxxx-xx-xx)

4.1.3.37 #define TPL_FIELD_CI_TIME_SYNC_2 0x6DU

Time synchronization (OMS Vol.2 Issue 3.0.0xxxx-xx-xx)

4.1.3.38 #define TPL_FIELD_CONF_WORD_ACCESSIBILITY 0x4000U

Receiver always on.

4.1.3.39 #define TPL_FIELD_CONF_WORD_BI_ACCESSIBILITY_MASK 0xC000U

Bidirectional and accessibility mask.

4.1.3.40 #define TPL_FIELD_CONF_WORD_BIDIRECTIONAL 0x8000U

Bidirectional communication.

4.1.3.41 #define TPL_FIELD_CONF_WORD_CONTENT_OF_MESSAGE_0 0x0004U

Content of message bit 0.

4.1.3.42 #define TPL_FIELD_CONF_WORD_CONTENT_OF_MESSAGE_1 0x0008U

Content of message bit 1.

4.1.3.43 #define TPL_FIELD_CONF_WORD_CONTENT_OF_MESSAGE_MASK 0x000CU

Content of message mask.

4.1.3.44 #define TPL_FIELD_CONF_WORD_ENCRYPTION_BLOCK_MASK 0x00F0U

Number of encryption blocks mask.

4.1.3.45 #define TPL_FIELD_CONF_WORD_ENCRYPTION_MODE_13 0x0D00U

AES + CBC mode 13

4.1.3.46 #define TPL_FIELD_CONF_WORD_ENCRYPTION_MODE_15 0x0F00U

AES + CBC with dynamic IV (DSMR Mode 15)

4.1.3.47 #define TPL_FIELD_CONF_WORD_ENCRYPTION_MODE_2 0x0200U

DES + CBC

4.1.3.48 #define TPL_FIELD_CONF_WORD_ENCRYPTION_MODE_3 0x0300U

DES + CBC with dynamic IV

4.1.3.49 #define TPL_FIELD_CONF_WORD_ENCRYPTION_MODE_4 0x0400U

AES + CBC

4.1.3.50 #define TPL_FIELD_CONF_WORD_ENCRYPTION_MODE_5 0x0500U

AES + CBC with dynamic IV

4.1.3.51 #define TPL_FIELD_CONF_WORD_ENCRYPTION_MODE_7 0x0700U

AES + CBC mode 7

4.1.3.52 #define TPL_FIELD_CONF_WORD_ENCRYPTION_MODE_MASK 0x1F00U

Encryption mode mask. (5 Bits according to OMS v4.0.2)

4.1.3.53 #define TPL_FIELD_CONF_WORD_HOP_COUNTER 0x0001U

Hop counter of the telegram.

4.1.3.54 #define TPL_FIELD_CONF_WORD_MODE_7_CONTENT_OF_MESSAGE_0 0x4000U

Content of message bit 0 of Encryption Mode 7 Configuration Word (OMS v4.0.2)

4.1.3.55 #define TPL_FIELD_CONF_WORD_MODE_7_CONTENT_OF_MESSAGE_1 0x8000U

Content of message bit 1 of Encryption Mode 7 Configuration Word (OMS v4.0.2)

4.1.3.56 #define TPL_FIELD_CONF_WORD_NO_ENCRYPTION 0x0000U

Use no encryption.

4.1.3.57 #define TPL_FIELD_CONF_WORD_REPEATER_ACCESS 0x0002U

Repeater access.

4.1.3.58 #define TPL_FIELD_CONF_WORD_SYNC 0x2000U

Synchronous message.

4.1.3.59 #define TPL_METERLIST_ENTRY_LEN

Value:

```
(WMBUS_ADDR_LEN + \
                                                1U + \
                                                WMBUS_ADDR_LEN + \
                                                AES_SIZE_OF_KEY)
```

4.1.3.60 #define TPL_STATUS_BUSY 0x01U

Signals that stack is busy. Must be in sync with lower layer macros!

4.1.3.61 #define TPL_STATUS_CONNECTED 0x08U

Signals that the meter device is connected to a data collector. Must be in sync with lower layer macros!

4.1.3.62 #define TPL_TLG_FLAG_ACCESSIBILITY 0x4000U

Receiver accessibility.

4.1.3.63 #define TPL_TLG_FLAG_BIDIRECTIONAL 0x8000U

Bidirectional communication.

4.1.3.64 #define TPL_TLG_FLAG_CLR 0x0000U

Clears all flags.

4.1.3.65 #define TPL_TLG_FLAG_HOP_COUNTER 0x0010U

Hop counter of the telegram

4.1.3.66 #define TPL_TLG_FLAG_REPEATER_ACCESS 0x0020U

Hop counter of the telegram

4.1.3.67 #define TPL_TLG_FLAG_SIGNATURE 0x000CU

Mask for signature field.

4.1.3.68 #define TPL_TLG_FLAG_SIGNATURE_ERROR 0x0008U

4.1.3.69 #define TPL_TLG_FLAG_SIGNATURE_NO 0x000CU

Data collector only: The data is unencrypted.

4.1.3.70 #define TPL_TLG_FLAG_SIGNATURE_UNKNOWN 0x0004U

Data collector only: Set if the data could not be encrpyted.

4.1.3.71 #define TPL_TLG_FLAG_SYNC 0x2000U

Bidirectional communication.

4.1.4 Enumeration Type Documentation

4.1.4.1 enum **E_TPL_AUTH_RESULT_t**

Result of authentication/verification of a message. Used by s_tpl_tlgAttr_t only in function wmbus_tpl_getTlgAttr . This information is gained by underlying Authentication and Fragmentation Layer (AFL) if present. If not present, s_tpl_tlgAttr_t will always signal E_AFL_AUTH_RESULT_NONE .

Enumerator

E_TPL_AUTH_RESULT_OK Authentication of received message was successfully verified.

E_TPL_AUTH_RESULT_NONE Received message is not authenticated.

E_TPL_AUTH_RESULT_ERROR An error occurred during verification of authentication of received message

4.1.4.2 enum **E_TPL_CRYPT_RET_t**

Enumeration of return values.

Enumerator

E_TPL_CRYPT_RET_OK The telegram is decrypted.

E_TPL_CRYPT_RET_ERR There was an error at telegram decryption.

E_TPL_CRYPT_RET_UNKNOWN Unknown encryption algorithm.

E_TPL_CRYPT_RET_TOO_FEW_BYTES Too few data bytes.

E_TPL_CRYPT_RET_INVALID The encrypted data is invalid.

4.1.4.3 enum **E_TPL_HEADER_TYPE_t**

Enumeration of header types.

Enumerator

E_TPL_HEADER_TYPE_NO No application header (0 bytes).

E_TPL_HEADER_TYPE_SHORT Short application header (4 bytes).

E_TPL_HEADER_TYPE_LONG Long application header (12 bytes).

E_TPL_HEADER_TYPE_EXT_I Extended link layer (2 bytes). (CI = 0x8C)

E_TPL_HEADER_TYPE_EXT_II Extended link layer (8 bytes). (CI = 0x8D)

E_TPL_HEADER_TYPE_INVALID Invalid header.

4.1.4.4 enum **E_TPL_RET_t**

Enumeration of return values.

Enumerator

E_TPL_RET_OK All OK

E_TPL_RET_ERR Error.

E_TPL_RET_NOT_READY Stack is not ready

E_TPL_RET_DUPLICATED Only used for the MeterAdd command. Indicates that the meter ist already in the list

4.1.4.5 enum **E_TPL_STATUS_t**

Enumeration of the status of the dll+ functions.
Enumerator

E_TPL_STATUS_ERROR Error occured for unknown reasons.

E_TPL_STATUS_INVALID_PARAM_ERROR Invalid input parameter error

E_TPL_STATUS_NOT_INIT_ERROR Stack not initialized error.

E_TPL_STATUS_SUCCESS Operation success.

4.1.5 Function Documentation

4.1.5.1 uint8_t wmbus_tpl_accessNoIncrementBy (uint8_t c_numberOfIncrements)

Increments the access number which is used from the tpl and the apl.

Parameters

c_numberOf↩ Increments	Number of incrementations.

Returns

Incremented access number

4.1.5.2 uint8_t wmbus_tpl_close (uint16_t i_meterId, bool_t b_sendNke)

Resets the flag to start a bidirectional communication. Function used by collector device only.

Parameters

i_meterId	Id of the meter device.
b_sendNke	TRUE if the collector should send an NKE message to the meter to terminate the fac window of the meter device. FALSE if the collector does not expect an answer. For example: After the collector sends the confimation of an Installation request dll_close() must be called with b_sendNKE = FALSE because the collector should not send an NKE message to end the communication.

Returns

Returns the telegram ID of the closing telegram. If sending of NKE fails or no telegram can be created DLL_↩ ERR_TLG_NOT_AVAILABLE is returned.

4.1.5.3 void wmbus_tpl_clrHeaderStatusFlag (uint8_t c_flag)

Clears a flag in the current status byte which is used from the tpl and the apl.

Parameters

c_flag	Flag to be cleared.

4.1.5.4 uint16_t wmbus_tpl_cntDataBytes (uint8_t c_tlgId)

Counts the number of telegram data bytes.

Parameters

c_tlgId	Id of the telegram.

Returns

Number of data bytes.

4.1.5.5 E_WMBUS_COL_QUEUE_RET_t wmbus_tpl_col_clearMtrTlgQueue (uint16_t i_meterId)

Removes all telegrams from the queue for a specific meter.

This function only clears telegram references of a specific meter queue. The telegrams itself are not destroyed automatically!

Parameters

i_meterId	ID of the selected meter.

Returns

E_WMBUS_COL_QUEUE_RET_OK if all works fine.

4.1.5.6 void wmbus_tpl_col_clearTlgQueue (void)

This function resets the complete queue.

This function only clears the telegram references in the queue. The telegrams itself are not destroyed automatically!

Parameters

None.	

Returns

None.

4.1.5.7 uint8_t wmbus_tpl_createTlg (uint8_t c_type, s_wmbus_addr_t * ps_meterAddr, uint8_t c_ci)

Creates a new telegram.

Parameters

c_type	Telegram type.
ps_meterAddr	Address of the meter device.
c_ci	The value for the control information field.

Returns

Id of the telegram. DLL_ERR_TLG_NOT_AVAILABLE if there is not enough memory to create the telegram.

4.1.5.8 **E_TPL_CRYPT_RET_t** wmbus_tpl_decrypt (uint8_t c_tlgId)

Decrypts a complete telegrams.

Parameters

c_tlgId	Id of the telegram.

Returns

Status of telegram decryption.

4.1.5.9 bool_t wmbus_tpl_destroyTlg (uint8_t c_tlgId)

Destroys a telegram.

Parameters

c_tlgId	Id of the telegram to destroy.

Returns

TRUE if the telegram exists, FALSE if c_tlgId is out of range.

4.1.5.10 **E_TPL_CRYPT_RET_t** wmbus_tpl_encrypt (uint8_t c_tlgId)

Encrypts a complete telegrams.

Parameters

c_tlgId	Id of the telegram.

Returns

Status of telegram encryption.

4.1.5.11 bool_t wmbus_tpl_encryptPrepare (uint8_t c_tlgId)

Prepares data encryption.

Parameters

c_tlgId	Id of the telegram.

Returns

TRUE if the telegram is prepared for encryption.

4.1.5.12 void wmbus_tpl_evt_col_ACDBitSet (uint8_t c_tlgId)

This function is called if the ACD bit of a meter telegram is set. Function used by collector device only.

Parameters

c_tlgId	Id of the received telegram.

4.1.5.13 void wmbus_tpl_evt_col_opened (uint8_t c_tlgId)

This function is called if a bidirectional communication with a meter device is started. It is the acknowledgement to wmbus_dll_open().

Parameters

c_tlgId	Id of the request.

4.1.5.14 void wmbus_tpl_evt_col_tlgAvailable (E_WMBUS_RX_t e_status, uint8_t c_tlgReqId, uint8_t c_tlgId)

This function is called if a telegram is available. Collector device.

Parameters

e_status	Status of the telegram.
c_tlgReqId	Id of the request telegram. NULL if there is no request telegram available.
c_tlgId	Id of the telegram.

4.1.5.15 void wmbus_tpl_evt_col_tx (uint8_t c_tlgId)

This function is called if a telegram is sent successfully. Collector device.

Parameters

c_tlgId	Id of the sent telegram.

4.1.5.16 **E_TPL_HEADER_TYPE_t** wmbus_tpl_evt_getCiHeader (uint8_t c_ci)

Transmits a CI field and requires the type of headers.

Parameters

c_ci	User specific control information field.

Returns

Header type.

4.1.5.17 void wmbus_tpl_evt_mtr_tlgAvailable (E_WMBUS_RX_t e_status, uint8_t c_tlgReqId, uint8_t c_tlgId)

This function is called if a telegram is available. Meter device.

Parameters

e_status	Status of the telegram.
c_tlgReqId	Id of the request telegram. NULL if there is no request telegram available.
c_tlgId	Id of the telegram.

4.1.5.18 void wmbus_tpl_evt_mtr_tx (uint8_t c_tlgId)

This function is called if a telegram is sent successfully. Meter deivce.

Parameters

c_tlgId	Id of the sent telegram.

4.1.5.19 E_WMBUS_ACCESSIBILITY_t wmbus_tpl_getAccessibility (void)

Returns the global accessibility setting of the meter device. Default = E_WMBUS_ACCESSIBILITY_BIDIRECTIONAL_ACCESS Function used by bidirectional meter device only.

Returns

Displayed mode in the configuration word

4.1.5.20 uint8_t wmbus_tpl_getAccessNo (void)

Returns the current access number which is used from the tpl and the apl.

Returns

Current access number

4.1.5.21 void wmbus_tpl_getAddrOwn (s_wmbus_addr_t * ps_meterAddr)

Returns the address of the device.

Parameters

ps_meterAddr	Address of the storage to write the meter address into.

4.1.5.22 void wmbus_tpl_getCollectorAddr (s_wmbus_addr_t * ps_collectorAddr)

Reads the address of the data collector. (needed for OMS)

Parameters

ps_collectorAddr	Collector address

4.1.5.23 bool_t wmbus_tpl_getHeader (uint8_t c_tlgId, **s_tpl_header_t** * ps_header)

Reads the header of a telegram.

Parameters

c_tlgId	Id of the telegram to read the header from.
ps_header	Destination structure to write the header information to.

Returns

TRUE if the header could be read. Use ps_header->e_type to get the length of the header.

4.1.5.24 uint8_t wmbus_tpl_getHeaderStatus (void)

Returns the current status byte which is used from the tpl and the apl.

Returns

Current access number

4.1.5.25 **E_TPL_HEADER_TYPE_t** wmbus_tpl_getHeaderType (uint8_t c_ci)

Reads the header type of the CI field.

Parameters

c_ci	Control information field.

Returns

Type of the header.

4.1.5.26 uint32_t wmbus_tpl_getInterval (void)

Returns the periodical interval of a meter device. If the device is a collector always 0 is returned.

Returns

Periodical interval in milliseconds.

4.1.5.27 bool_t wmbus_tpl_getKey (uint8_t * pc_key, uint8_t c_len)

Reads the local encryption key.

Parameters

pc_key	Storage to save the key into.
c_len	Size of pc_key.

Returns

TRUE if the key is read.

4.1.5.28 E_RADIO_CHANNEL_INDEX_t wmbus_tpl_getRfChannel (void)

Reads the radio channel of the rf-module.

Returns

Current radio channel

4.1.5.29 uint8_t wmbus_tpl_getStatus (void)

Returns the current status of the transport layer.

Returns

Status of the transport layer.

4.1.5.30 bool_t wmbus_tpl_getTlgAttr (uint8_t c_tlgId, **s_tpl_tlgAttr_t** * ps_tlgAttrRecv)

Reads the attributes of a received telegram.

Parameters

c_tlgId	ID of the telegram.
ps_tlgAttrRecv	Memory to save the telegram information into.

Returns

Always TRUE.

4.1.5.31 uint8_t wmbus_tpl_getTxPower (void)

Reads the transmission power.

Returns

Tx power from -130dBm (0x0) to 125dBm (0xFE). 0xFF is reserved.

4.1.5.32 **s_tpl_addMeterRet_t** wmbus_tpl_meterAdd (**s_tpl_meterEntry_t** * ps_meterEntry)

Adds a meter device to the meter list. Function used by collector device only.

Parameters

ps_meter_entry	Entry of a meter.

Returns

Id of the added meter device. DLL_ERR_METER_OUT_OF_RANGE if there is no free space to add the meter device. If a device with the same address is already in the list, its id is returned.

4.1.5.33 bool_t wmbus_tpl_meterGetAddr (uint16_t i_index, s_wmbus_addr_t * ps_meterAddr)

Reads the address of a meter device. Function used by collector device only.

Parameters

i_index	Index of the meter device to get the address from.
ps_meterAddr	Pointer to save the meter address into.

Returns

TRUE if the meter address could be read.

4.1.5.34 uint16_t wmbus_tpl_meterGetIndex (s_wmbus_addr_t * ps_meterAddr)

Reads the index of a meter device. Function used by collector device only.

Parameters

ps_meterAddr	Pointer to save the meter address into.

Returns

Index of the meter device. DLL_ERR_METER_OUT_OF_RANGE if the meter device is not in the list.

4.1.5.35 bool_t wmbus_tpl_meterGetKey (s_wmbus_addr_t * ps_meter, uint8_t * pc_key, uint8_t c_len)

Reads the key of the meter. Function used by collector device only.

Parameters

ps_meter	Address of the meter device.
pc_key	memory to write the key data to.

c_len	Size of pc_key.

Returns

TRUE if a key is available.

4.1.5.36 uint16_t wmbus_tpl_meterGetNum (void)

Gets the number of connected meter devices. Function used by collector device only.

Returns

Number of connected meter devices.

4.1.5.37 bool_t wmbus_tpl_meterGetRfAdapter (s_wmbus_addr_t * ps_meter, s_wmbus_addr_t * ps_rfadapter)

Reads the rf adapter address of the meter. Function used by collector device only.

Parameters

ps_meter	Address of the meter device.
ps_rfadapter	Destination memory to write the adapter address to.

Returns

TRUE if a rf adapter address is available.

4.1.5.38 **E_TPL_RET_t** wmbus_tpl_meterRemove (s_wmbus_addr_t * ps_meter)

Removes a meter device from the meter list. Function used by collector device only.

Parameters

ps_meter	Address of the meter device. Set to NULL to clear the complete meter list.

Returns

E_TPL_RET_OK if the meter is removed successfully.

4.1.5.39 **E_TPL_RET_t** wmbus_tpl_meterSetKey (s_wmbus_addr_t * ps_meter, uint8_t * pc_key, uint8_t c_len)

Updates the meter specific key. Function used by collector device only.

Parameters

ps_meter	Address of the meter device.
pc_key	Key to set.
c_len	Size of the key.

Returns

E_TPL_RET_OK if the key could be changed.

4.1.5.40 **E_TPL_RET_t** wmbus_tpl_meterSetRfAdapter (s_wmbus_addr_t * ps_meter, s_wmbus_addr_t * ps_rfadapter)

Updates the RF adapter address of the meter device. Function used by collector device only.

Parameters

ps_meter	Address of the meter device.
ps_rfadapter	Address of the RF adapter.

Returns

E_TPL_RET_OK if the adapter address could be changed.

4.1.5.41 bool_t wmbus_tpl_mtr_setResponseDelay (E_WMBUS_RESPONSE_DELAY_t e_respDelay)

Sets the response delay to be used by the bidirectional meter device of WMBus mode C or WMBus mode N. The user given input may be overwritten by the collector the meter is connected to.

According to the EN13757-4 (2013) meter devices of mode C and mode N respect the response delay as forced by the collector. This WMBus behaviour may lead to the effect that the user given input is overwritten (by the collector) once the meter is connected to a collector. Therefore this function must be used carefully. In order to verify the response delay that is currently used one can call wmbus_ell_mtr_getResponseDelay() .

Parameters

e_respDelay	Response delay to be used.

Returns

TRUE if set successfully. FALSE otherwise e.g. if not supported for the current device type.

4.1.5.42 bool_t wmbus_tpl_open (void)

Start a bidirectional communication the next time as possible. Function used by collector device only.

Returns

TRUE if opperation was successful.

4.1.5.43 uint16_t wmbus_tpl_receiveTlg (uint8_t c_tlgId, uint8_t * pc_data, uint16_t i_len, uint16_t i_offset, bool_t b_reverse)

Receives data from a telegram.

Parameters

c_tlgId	Id of the telegram to read from.
pc_data	Memory to write the data into.
i_len	Number of bytes to read.
i_offset	Index to read the data from.
b_reverse	The data has to be received in reverse order.

Returns

Number of read bytes. DLL_ERR_INDEX_OUT_OF_RANGE if data could not be read.

4.1.5.44 uint8_t wmbus_tpl_sendAck (uint8_t c_accNo, uint16_t i_meterId)

Sends an Ack message.

Parameters

c_accNo	Access number to transmit.
i_meterId	Id of the connected meter. If the device is a meter this parameter will be ignored

Returns

Id of the sent telegram, DLL_ERR_TLG_NOT_AVAILABLE if no free buffer is available of the telegram could not be sent.

4.1.5.45 bool_t wmbus_tpl_sendQueued (uint8_t c_tlgId, uint16_t i_meterId, bool_t b_append)

Adds a telegram to the collector queue. Function used by collector device only.

Parameters

c_tlgId	Id of the request.
i_meterId	Id of the meter.
b_append	If TRUE the telegram will be appended to the device specific queue. If FALSE the telegram will set as first telegram of the queue. Append should be set to FALSE to add a request at the start of the queue if there was an problem while sending the request.

Returns

TRUE if the preparation was successful.

4.1.5.46 bool_t wmbus_tpl_sendTlg (uint8_t c_tlgId)

Sends a telegram. If the telegram is sent successfully, it is destroyed by the data link layer automatically.

Parameters

c_tlgId	Id of the telegram to send.

Returns

TRUE if sending was successful.

4.1.5.47 void wmbus_tpl_setAccessibility (E_WMBUS_ACCESSIBILITY_t e_access)

This function is only for meter devices. Set the acessibiolity type of the meter to save energy. Function used by bidirectional meter device only.

Parameters

e_access	Displayed mode in the configuration word

4.1.5.48 void wmbus_tpl_setAccessNo (uint8_t c_accessNo)

Sets the current access number which is used from the tpl and the apl.

Parameters

c_accessNo	Access number to be set.

4.1.5.49 void wmbus_tpl_setAddrOwn (s_wmbus_addr_t * ps_meterAddr)

Sets the address of the meter device.

Parameters

ps_meterAddr	Address of the storage to get the meter address from.

4.1.5.50 void wmbus_tpl_setCollectorAddr (s_wmbus_addr_t * ps_collectorAddr)

Sets the address of the data collector. (needed for OMS)

Parameters

ps_collectorAddr	Collector address

4.1.5.51 void wmbus_tpl_setConnected (bool_t b_connected)

Sets the device connected or disconnected.

Parameters

b_connected	TRUE if the device is connected.

4.1.5.52 bool_t wmbus_tpl_setHeader (uint8_t c_tlgId, **s_tpl_header_t** * ps_header)

Sets the header values of a telegram.

Parameters

c_tlgId	Id of the telegram.
ps_header	Destination structure to write the header information to.

Returns

TRUE if the header is added successfully.

4.1.5.53 void wmbus_tpl_setHeaderStatusFlag (uint8_t c_flag)

Sets a flag in the current status byte which is used from the tpl and the apl.

Parameters

c_flag	Flag to be set.

4.1.5.54 void wmbus_tpl_setInterval (uint32_t l_interval)

Sets the periodical interval of a meter device.

Parameters

l_interval	Interval to set in milliseconds.

4.1.5.55 bool_t wmbus_tpl_setKey (uint8_t * pc_key, uint8_t c_len)

Writes the local encryption key.

Parameters

pc_key	Storage to load the key from.
c_len	Size of pc_key.

Returns

TRUE if the key is written.

4.1.5.56 bool_t wmbus_tpl_setRfChannel (E_RADIO_CHANNEL_INDEX_t e_channel)

Sets the radio channel of the rf-module. This function should only be used in mode N to set the correct channel for the device.

Parameters

i_channel	Radio channel or channel mask. 0xFFFF is reserved.

Returns

TRUE if the channel was set.

4.1.5.57 void wmbus_tpl_setTlgFormat (uint8_t c_tlgId, E_WMBUS_FRAME_t e_frameType)

Sets the frame type of the telegram. This function should only be used in C-mode. Only the C-mode supports packets in format A and B.

Parameters

c_tlgId	Telegram Id
e_frameType	Frame type.

4.1.5.58 void wmbus_tpl_setTlgMode (uint8_t c_tlgId, E_WMBUS_MODE_t e_mode)

Sets the mode of the telegram.

Parameters

c_tlgId	Telegram Id
e_mode	Mode of the received telegram (S, S sync, T, C or N).

4.1.5.59 void wmbus_tpl_setTxPower (uint8_t c_txPower)

Sets the transmission power of the rf-module. Please have a look in the corresponding data sheet of the selected transceiver to choose a supported transmission power.

Parameters

c_txPower	Tx power from -130dBm (0x0) to 125dBm (0xFE). 0xFF is reserved.

4.1.5.60 void wmbus_tpl_setTxWait (uint32_t l_timeout)

Sets a timeout to wait before sending the next telegram.

Parameters

l_timeout	Time to wait in milliseconds.

4.1.5.61 E_WMBUS_SLEEP_RESULT_t wmbus_tpl_sleep (E_WMBUS_SLEEP_MODE_t e_sleepMode)

Checks if the dll is able to go in low power-mode.

Parameters

e_sleepMode	Power mode for the target

Returns

Returns the status of the request as E_PHY_SLEEP_RESULT_t.

4.1.5.62 void wmbus_tpl_wakeUp (E_WMBUS_SLEEP_MODE_t e_sleepMode)

Wake up the target.

Parameters

e_apl_sleep_↩ mode	Current power-mode

4.1.5.63 bool_t wmbus_tpl_writeTlg (uint8_t c_tlgId, uint8_t * pc_data, uint16_t i_len, uint16_t i_offset, bool_t b_reverse)

Writes data to a telegram.

Parameters

c_tlgId	Id of the telegram to write the data into.
pc_data	Data to write.
i_len	Number of bytes to write.
i_offset	Offset to start writing into the telegram. DLL_TLG_WRITE_APPEND if the data has to be appended.
b_reverse	TRUE if the data has to be written reverse.

Returns

TRUE if writing was successful.

4.2 inc/pub/tpl/wmbus_tpl_col_api.h File Reference

Transport Layer Application Programming Interface of Wireless M-Bus Collector Device.

Macros

- #define **__DECL_TRANSPORTLAYER_COL_API_H__** extern

4.2.1 Detailed Description

Transport Layer Application Programming Interface of Wireless M-Bus Collector Device.
Copyright

STACKFORCE GmbH, Heitersheim, Germany, http://www.stackforce.de
Author

STACKFORCE

4.3 inc/pub/tpl/wmbus_tpl_mtr_api.h File Reference

Transport Layer Application Programming Interface of Wireless M-Bus Meter Device.

Macros

- #define **__DECL_TRANSPORTLAYER_MTR_API_H__** extern

4.3.1 Detailed Description

Transport Layer Application Programming Interface of Wireless M-Bus Meter Device.

Copyright

STACKFORCE GmbH, Heitersheim, Germany, http://www.stackforce.de

Author

STACKFORCE

Chapter 5

Contact information

In case of questions, requests for quotations or ideas for further improvements, please contact:

STACKFORCE GmbH
Poststrasse 35
79423 Heitersheim
Germany

Freiburg HRB 711613
Geschäftsführer: David Rahusen

Tel: +49 7634-69960-20
Fax: +49 7634-69960-30
Url: http://www.stackforce.de
E-mail: metering@stackforce.de

Bibliography

[1] "Communication systems for meters and remote reading of meters." Part 1: Data exchange; English version EN 13757-1, 2002.

[2] "Communication systems for meters and remote reading of meters." Part 2: Physical and link layer; English version EN 137572, 2004.

[3] "Communication systems for meters and remote reading of meters." Part 3: Dedicated application layer; English version EN 137573, 2011.

[4] "Communication systems for meters and remote reading of meters." Part 4: Wireless meter readout (Radio meter reading for operation in the 868 MHz to 870 MHz SRD band); German version EN 13757-4, 2011.

[5] "Communication systems for meters and remote reading of meters." Part 5: Wireless Relaying; English version prEN 13757-5, 2009.

[6] "Telecontrol equipment and systems." Part 5: Transmission protocols; EN 870-5, 2002.