Wireless M-Bus Documentation

Stack Reference Manual

STACKFORCE

embedded.connectivity.solutions

March 20, 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.

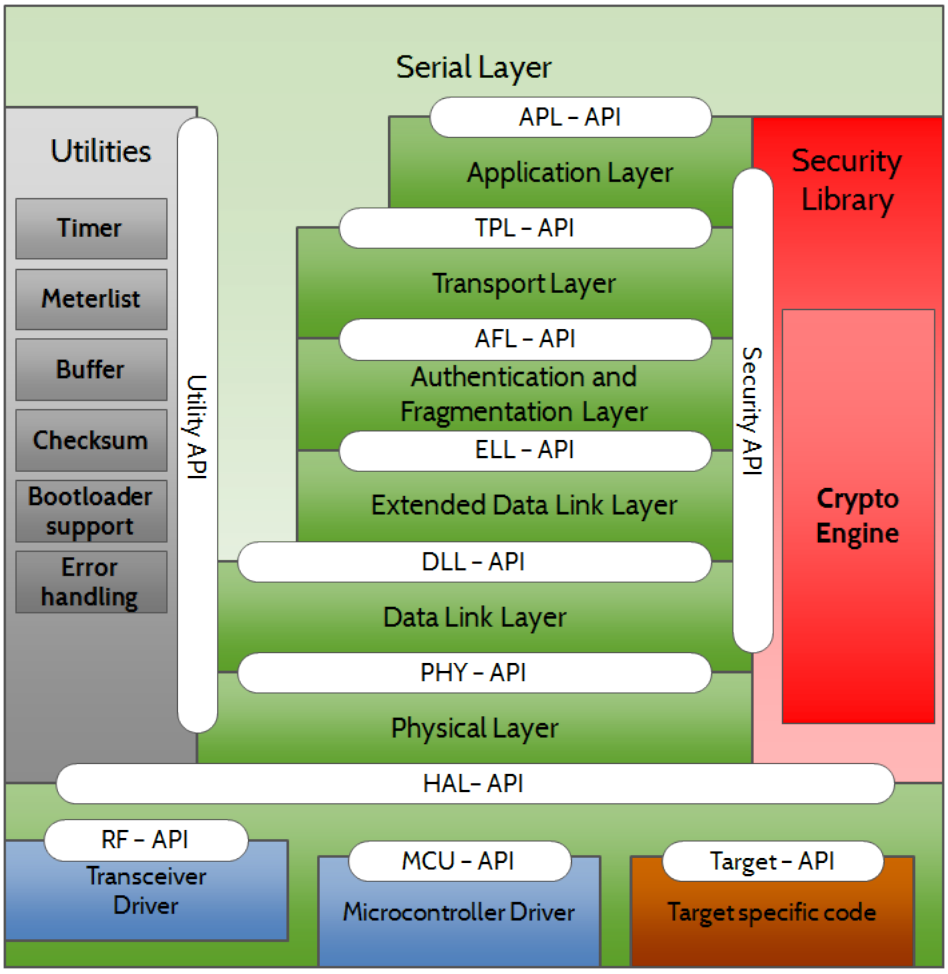

Figure 2.1: Architecture of the Wireless M-Bus Protocol Stack.

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 8 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 $N2(a-f)$.

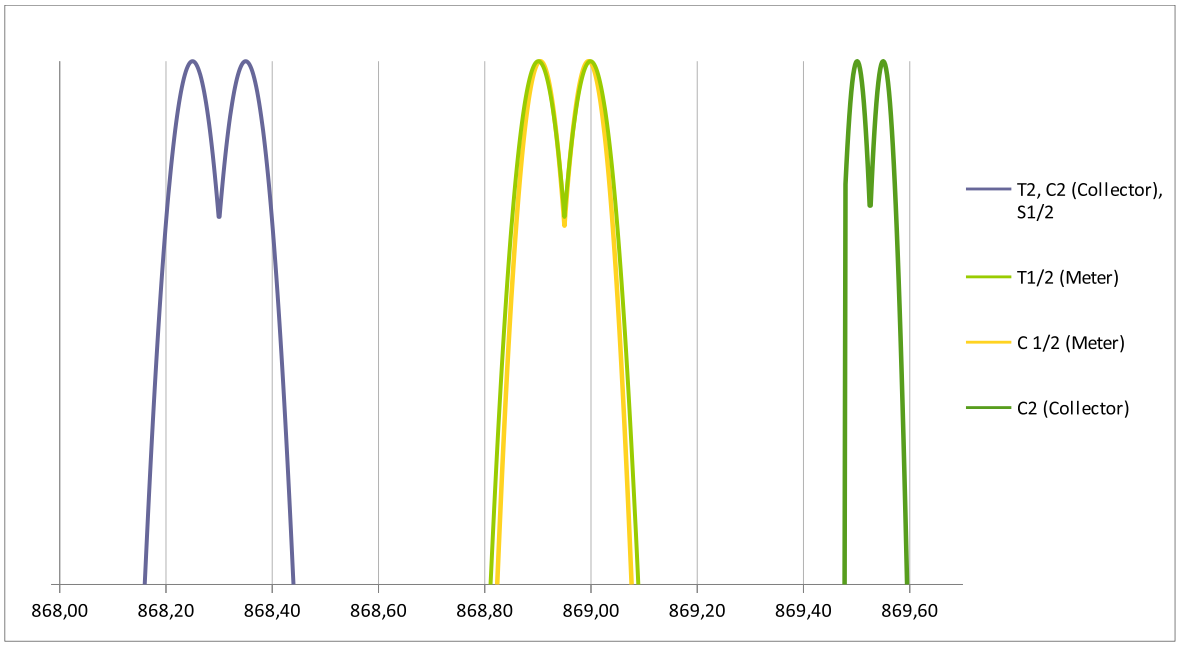

Figure 2.2: Frequencies of Wireless M-Bus modes S, T and C.

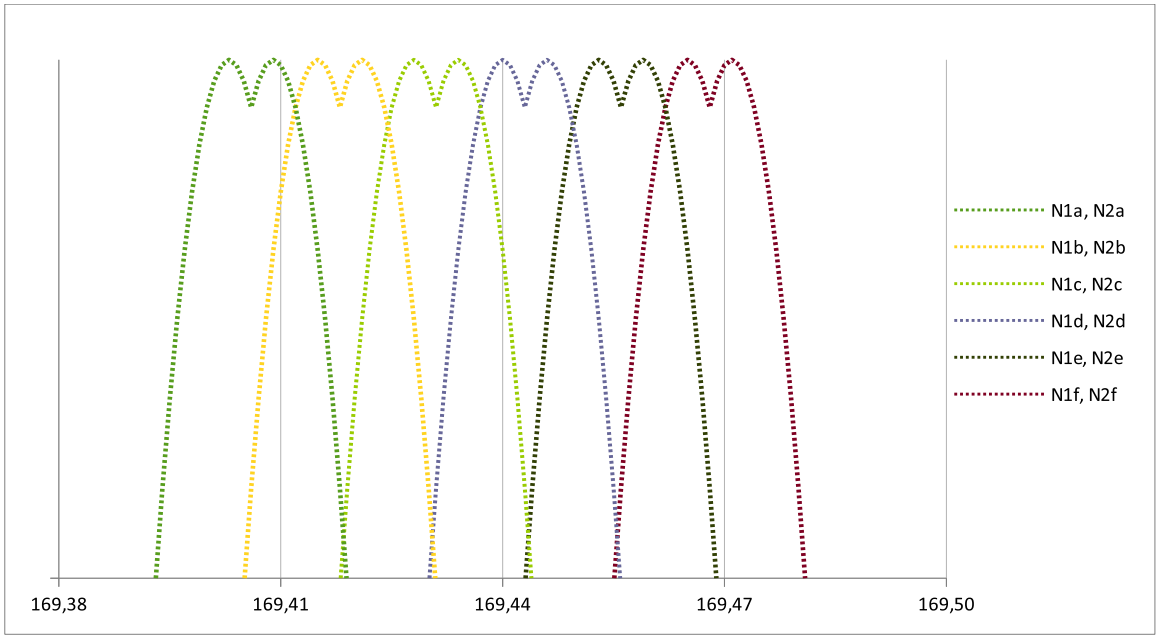

Figure 2.3: Frequencies of Wireless M-Bus mode N.

Table ?? 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]).

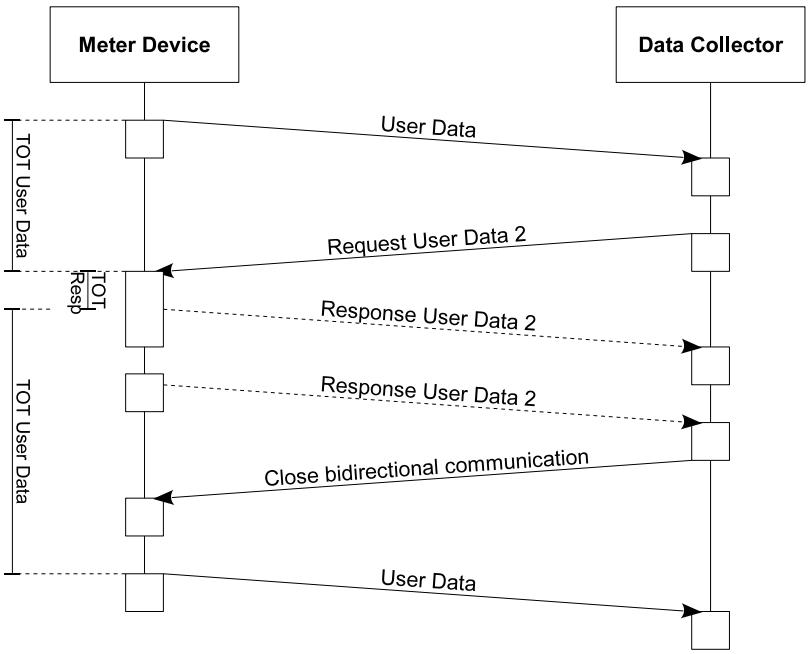

Figure 2.4: Bidirectional communication example for WM-Bus modes $S2$, $T2$, $C2$ and $N2(a-f)$.

Chapter 3

Module Index

3.1 Modules

Here is a list of all modules:

Chapter 4

File Index

4.1 File List

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

Chapter 5

Module Documentation

5.1 Application Layer Interface Description

Modules

5.1.1 Detailed Description

This section describes the API for the STACKFORCE application layer.

5.2 Compile time settings

This section lists compile time settings for the STACKFORCE application layer.

5.3 Define macros

Macros

- #define APL_STATUS_BUSY 0x80U
- #define APL_STATUS_INACTIVE 0x01U
- #define APL_STATUS_INSTALL 0x02U
- #define APL_STATUS_CONNECTED 0x04U
- #define APL_STATUS_SLEEP 0x08U
- #define APL_ERR_TLG_NOT_AVAILABLE 0xFFU
- #define APL_FIELD_CI_CMD_TO_DEVICE_NONE 0x51U
- #define APL_FIELD_CI_CMD_TO_DEVICE_SHORT 0x5AU
- #define APL_FIELD_CI_CMD_TO_DEVICE_LONG 0x5BU
- #define APL_FIELD_CI_TIME_SYNC_1 0x6CU
- #define APL_FIELD_CI_TIME_SYNC_2 0x6DU
- #define APL_FIELD_CI_APL_ERROR_SHORT 0x6EU
- #define APL_FIELD_CI_APL_ERROR_LONG 0x6FU
- #define APL_FIELD_CI_HEADER_LONG 0x72U
- #define APL_FIELD_CI_APL_ALARM_SHORT 0x74U
- #define APL_FIELD_CI_APL_ALARM_LONG 0x75U
- #define APL_FIELD_CI_HEADER_NONE 0x78U
- #define APL_FIELD_CI_HEADER_SHORT 0x7AU
- #define APL_FIELD_CI_LINK_TO_DEVICE_LONG 0x80U
- #define APL_FIELD_CI_RELAY 0x81U
- #define APL_FIELD_CI_NETWORK 0x82U
- #define APL_FIELD_CI_LINK_FROM_DEVICE_SHORT 0x8AU
- #define APL_FIELD_CI_LINK_FROM_DEVICE_LONG 0x8BU
- #define APL_FIELD_CI_BAUD_300 0xB8U
- #define APL_FIELD_CI_BAUD_600 0xB9U
- #define APL_FIELD_CI_BAUD_1200 0xBAU
- #define APL_FIELD_CI_BAUD_2400 0xBBU
- #define APL_FIELD_CI_BAUD_4800 0xBCU
- #define APL_FIELD_CI_BAUD_9600 0xBDU

- #define APL_FIELD_CI_BAUD_19200 0xBEU
- #define APL_FIELD_CI_BAUD_38400 0xBFU
- #define APL_FIELD_CI_NULL 0xFFU
- #define APL_FIELD_STATUS_NO_ERROR 0x00U
- #define APL_FIELD_STATUS_APPLICATION_BUSY 0x01U
- #define APL_FIELD_STATUS_ANY_APPLICATION_ERROR 0x02U
- #define APL_FIELD_STATUS_ALARM 0x03U
- #define APL_FIELD_STATUS_POWER_LOW 0x04U
- #define APL_FIELD_STATUS_PERMANENT_ERROR 0x08U
- #define APL_FIELD_STATUS_TEMPORARY_ERROR 0x10U
- #define APL_FIELD_CONF_WORD_CONTENT_OF_MESSAGE_MASK 0x000CU
- #define APL_FIELD_CONF_WORD_CONTENT_OF_MESSAGE_0 0x0004U
- #define APL_FIELD_CONF_WORD_CONTENT_OF_MESSAGE_1 0x0008U
- #define APL_FIELD_CONF_WORD_MODE_7_CONTENT_OF_MESSAGE_0 0x4000U
- #define APL_FIELD_CONF_WORD_MODE_7_CONTENT_OF_MESSAGE_1 0x8000U
- #define APL_FIELD_CONF_WORD_ENCRYPT_MODE_MASK 0x1F00U
- #define APL_FIELD_CONF_WORD_NO_ENCRYPTION 0x0000U
- #define APL_FIELD_CONF_WORD_DES_CBC 0x0200U
- #define APL_FIELD_CONF_WORD_DES_CBC_IV 0x0300U
- #define APL_FIELD_CONF_WORD_AES_CBC 0x0400U
- #define APL_FIELD_CONF_WORD_AES_CBC_IV 0x0500U
- #define APL_FIELD_CONF_WORD_AES_CBC_MODE_7 0x0700U
- #define APL_FIELD_CONF_WORD_AES_CBC_IV_DSMR 0x0F00U
- #define APL_DIF_DATA_FIELD_NO_DATA 0x00U
- #define APL_DIF_DATA_FIELD_8_INT 0x01U
- #define APL_DIF_DATA_FIELD_16_INT 0x02U
- #define APL_DIF_DATA_FIELD_24_INT 0x03U
- #define APL_DIF_DATA_FIELD_32_INT 0x04U
- #define APL_DIF_DATA_FIELD_32_REAL 0x05U
- #define APL_DIF_DATA_FIELD_48_INT 0x06U
- #define APL_DIF_DATA_FIELD_64_INT 0x07U
- #define APL_DIF_DATA_FIELD_SELECTION 0x08U
- #define APL_DIF_DATA_FIELD_2_BCD 0x09U

- #define APL_DIF_DATA_FIELD_4_BCD 0x0AU
- #define APL_DIF_DATA_FIELD_6_BCD 0x0BU
- #define APL_DIF_DATA_FIELD_8_BCD 0x0CU
- #define APL_DIF_DATA_FIELD_VARIABLE 0x0DU
- #define APL_DIF_DATA_FIELD_12_BCD 0x0EU
- #define APL_DIF_DATA_FIELD_SPECIAL_FUNC 0x0FU
- #define APL_DIF_DATA_FIELD_SPECIAL_FILLER 0x2FU
- #define APL_DIF_FUNC_INSTANTANEOUS 0x00U
- #define APL_DIF_FUNC_MAXIMUM 0x10U
- #define APL_DIF_FUNC_MINIMUM 0x20U
- #define APL_DIF_FUNC_ERROR 0x30U
- #define APL_DIF_LSB_STORAGE_NUMBER 0x40U
- #define APL_DIF_EXTENSION_BIT 0x80U
- #define APL_VIF_ENERGY_WH 0x00U
- #define APL_VIF_ENERGY_J 0x08U
- #define APL_VIF_VOLUME_M3 0x10U
- #define APL_VIF_MASS_KG 0x18U
- #define APL_VIF_ON_TIME 0x20U
- #define APL_VIF_OPERATING_TIME 0x24U
- #define APL_VIF_POWER_W 0x28U
- #define APL_VIF_POWER_JH 0x30U
- #define APL_VIF_VOLUME_FLOW_M3H 0x38U
- #define APL_VIF_VOLUME_FLOW_EXT_M3M 0x40U
- #define APL_VIF_VOLUME_FLOW_EXT_M3S 0x48U
- #define APL_VIF_MASS_FLOW_KGH 0x50U
- #define APL_VIF_FLOW_TEMPERATURE_C 0x58U
- #define APL_VIF_RETURN_TEMPERATURE_C 0x5CU
- #define APL_VIF_TEMPERATURE_DIF_K 0x60U
- #define APL_VIF_EXT_TEMPERATURE_C 0x64U
- #define APL_VIF_PRESSURE_BAR 0x68U
- #define APL_VIF_DATE 0x6CU
- #define APL_VIF_DATE_TIME 0x6DU
- #define APL_VIF_UNITS_FOR_HCA 0x6EU

- #define APL_VIF_AVERAGING_DURATION 0x70U
- #define APL_VIF_ACTUALITY_DURATION 0x74U
- #define APL_VIF_FABRICATION_NO 0x78U
- #define APL_VIF_ENHANCED_ID 0x79U
- #define APL_VIF_ADDRESS 0x7AU
- #define APL_VIF_FIRST_EXTENSION 0x7BU
- #define APL_VIF_SECOND_EXTENSION 0xFDU
- #define APL_VIF_MANUFACTURER 0x7FU
- #define APL_VIF_EXTENSION_BIT 0x80U
- #define APL_VIFE_SEC_KEY 0x19U
- #define APL_VIFE_TRANS_CTR 0x08U
- #define APL_VIFE_ERR_FLAG 0x17U
- #define APL_VIFE_SPEC_SUPP_INFO 0x67U
- #define APL_VIFE_MANUFR_ACCELERATION 0x01U
- #define APL_VIFE_MANUFR_ACTIVITY 0x02U
- #define APL_VIFE_MANUFR_FALL 0x03U
- #define APL_ALARM_UNSPECIFIED_BIT 1U
- #define APL_ALARM_NONE 0x00U
- #define APL_ALARM_ALL 0xFFU
- #define APL_ERROR_UNSPECIFIED 0x00U
- #define APL_ERROR_CI_UNIMPLEMENTED 0x01U
- #define APL_ERROR_BUFFER_TOO_LONG 0x02U
- #define APL_ERROR_TOO_MANY_RECORDS 0x03U
- #define APL_ERROR_PREMATURE_END_OF_RECORD 0x04U
- #define APL_ERROR_TOO_MANY_DIFES 0x05U
- #define APL_ERROR_TOO_MANY_VIFES 0x06U
- #define APL_ERROR_APPLICATION_BUSY 0x08U
- #define APL_ERROR_TOO_MANY_READOUTS 0x09U
- #define APL_ERROR_ACCESS_DENIED 0x10U
- #define APL_ERROR_CMD_UNKNOWN 0x11U
- #define APL_ERROR_PARAMETER 0x12U
- #define APL_ERROR_UNKNOWN_RECEIVER 0x13U
- #define APL_ERROR_DECRYPTION_FAILS 0x14U

- #define APL_ERROR_ENCRYPTION_NOT_SUPPORTED 0x15U
- #define APL_ERROR_SIGNATURE_NOT_SUPPORTED 0x16U
- #define APL_ERROR_DYNAMIC_APPLICATION_ERROR 0xF0U
- #define APL_ERROR_INVALID 0xFFU
- #define APL_TLG_FLAG_CLR 0x0000U

  The following flags are used within the stack to signal telegram status information to the user.
- #define APL_TLG_FLAG_FLOW_CONTROL 0x0001U
- #define APL_TLG_FLAG_ACCESS_DEMAND 0x0002U
- #define APL_TLG_FLAG_SIGNATURE 0x000CU
- #define APL_TLG_FLAG_SIGNATURE_UNKNOWN 0x0004U
- #define APL_TLG_FLAG_SIGNATURE_ERROR 0x0008U
- #define APL_TLG_FLAG_SIGNATURE_NO 0x000CU
- #define APL_TLG_FLAG_HOP_COUNTER 0x0010U
- #define APL_TLG_FLAG_REPEATER_ACCESS 0x0020U
- #define **APL_TLG_FLAG_COM_0** 0x0040U
- #define **APL_TLG_FLAG_COM_1** 0x0080U
- #define **APL_TLG_FLAG_ENCRYPTION_MODE** 0x1F00U
- #define **APL_TLG_FLAG_ENCRYPTION_MODE_5** 0x0500U
- #define **APL_TLG_FLAG_ENCRYPTION_MODE_7** 0x0700U
- #define **APL_TLG_FLAG_ENCRYPTION_MODE_13** 0x0D00U
- #define APL_TLG_FLAG_SYNC 0x2000U
- #define APL_TLG_FLAG_RX_ALWAYS_ON 0x4000U
- #define APL_TLG_FLAG_BIDIRECTIONAL 0x8000U

5.3.1 Detailed Description

Enable/disable APL clock module. Defined in wmbus_global.h by default

Enable/disable APL RX event. Defined in wmbus_global.h by default

Enable/disable APL TX event. Defined in wmbus_global.h by default

This describes general define macros for the APL.

5.3.2 Macro Definition Documentation

5.3.2.1 #define APL_ALARM_ALL 0xFFU

All alarms
Examples:

main_meter.c.

5.3.2.2 #define APL_ALARM_NONE 0x00U

No alarm: All alarm bits are 0
Examples:

main_meter.c.

5.3.2.3 #define APL_ALARM_UNSPECIFIED_BIT 1U

STACKFORCE Manufacturer Specific Alarms

Unspecified alarm: also if data field is missing

5.3.2.4 #define APL_DIF_DATA_FIELD_12_BCD 0x0EU

12 digit BCD

5.3.2.5 #define APL_DIF_DATA_FIELD_16_INT 0x02U

16 Bit IntegerBinary

5.3.2.6 #define APL_DIF_DATA_FIELD_24_INT 0x03U

24 Bit IntegerBinary

5.3.2.7 #define APL_DIF_DATA_FIELD_2_BCD 0x09U

2 digit BCD

5.3.2.8 #define APL_DIF_DATA_FIELD_32_INT 0x04U

32 Bit IntegerBinary

5.3.2.9 #define APL_DIF_DATA_FIELD_32_REAL 0x05U

32 Bit Real

5.3.2.10 #define APL_DIF_DATA_FIELD_48_INT 0x06U

48 Bit IntegerBinary

5.3.2.11 #define APL_DIF_DATA_FIELD_4_BCD 0x0AU

4 digit BCD

5.3.2.12 #define APL_DIF_DATA_FIELD_64_INT 0x07U

64 Bit IntegerBinary

5.3.2.13 #define APL_DIF_DATA_FIELD_6_BCD 0x0BU

6 digit BCD

5.3.2.14 #define APL_DIF_DATA_FIELD_8_BCD 0x0CU

8 digit BCD

5.3.2.15 #define APL_DIF_DATA_FIELD_8_INT 0x01U

8 Bit IntegerBinary

5.3.2.16 #define APL_DIF_DATA_FIELD_NO_DATA 0x00U

No data

5.3.2.17 #define APL_DIF_DATA_FIELD_SELECTION 0x08U

Selection for Readout

5.3.2.18 #define APL_DIF_DATA_FIELD_SPECIAL_FILLER 0x2FU

Special function: Idle Filler, following byte = DIF of next record.

5.3.2.19 #define APL_DIF_DATA_FIELD_SPECIAL_FUNC 0x0FU

Special Functions

5.3.2.20 #define APL_DIF_DATA_FIELD_VARIABLE 0x0DU

variable length

5.3.2.21 #define APL_DIF_EXTENSION_BIT 0x80U

DIF(E) extension bit.

5.3.2.22 #define APL_DIF_FUNC_ERROR 0x30U

Value during error state

5.3.2.23 #define APL_DIF_FUNC_INSTANTANEOUS 0x00U

Instantaneous value

5.3.2.24 #define APL_DIF_FUNC_MAXIMUM 0x10U

Maximum value

5.3.2.25 #define APL_DIF_FUNC_MINIMUM 0x20U

Minimum value

5.3.2.26 #define APL_DIF_LSB_STORAGE_NUMBER 0x40U

LSB of storage number

5.3.2.27 #define APL_ERR_TLG_NOT_AVAILABLE 0xFFU

The telegram is not available. This define has to have the same value as DLL_ERR_TLG_NOT_AVAILABLE
Examples:

main_collector.c.

5.3.2.28 #define APL_ERROR_ACCESS_DENIED 0x10U

Access denied (Login, Password or Authorisation level is wrong) (OMS Vol.2 Issue 2.0.0/2009-07-20)

5.3.2.29 #define APL_ERROR_APPLICATION_BUSY 0x08U

Application too busy for handling readout request

5.3.2.30 #define APL_ERROR_BUFFER_TOO_LONG 0x02U

Buffer too long, truncated

5.3.2.31 #define APL_ERROR_CI_UNIMPLEMENTED 0x01U

Unimplemented CI-Field

5.3.2.32 #define APL_ERROR_CMD_UNKNOWN 0x11U

Application/Command unknown or not supported (OMS Vol.2 Issue 2.0.0/2009-07-20)

5.3.2.33 #define APL_ERROR_DECRYPTION_FAILS 0x14U

Decryption key fails (OMS Vol.2 Issue 2.0.02009-07-20)

5.3.2.34 #define APL_ERROR_DYNAMIC_APPLICATION_ERROR 0xF0U

Dynamic Application Error (OMS Vol.2 Issue 2.0.02009-07-20)

5.3.2.35 #define APL_ERROR_ENCRYPTION_NOT_SUPPORTED 0x15U

Encryption method is not supported (OMS Vol.2 Issue 2.0.02009-07-20)

5.3.2.36 #define APL_ERROR_INVALID 0xFFU

Invalid error code.

5.3.2.37 #define APL_ERROR_PARAMETER 0x12U

Parameter is missing or wrong (OMS Vol.2 Issue 2.0.02009-07-20)

5.3.2.38 #define APL_ERROR_PREMATURE_END_OF_RECORD 0x04U

Premature end of record

5.3.2.39 #define APL_ERROR_SIGNATURE_NOT_SUPPORTED 0x16U

Signature method is not supported (OMS Vol.2 Issue 2.0.02009-07-20)

5.3.2.40 #define APL_ERROR_TOO_MANY_DIFES 0x05U

More than 10 DIFE's

5.3.2.41 #define APL_ERROR_TOO_MANY_READOUTS 0x09U

Too many readouts (for slaves with limited readouts per time)

5.3.2.42 #define APL_ERROR_TOO_MANY_RECORDS 0x03U

Too many records

5.3.2.43 #define APL_ERROR_TOO_MANY_VIFES 0x06U

More than 10 VIFE's

5.3.2.44 #define APL_ERROR_UNKNOWN_RECEIVER 0x13U

Unknown Receiver address (OMS Vol.2 Issue 2.0.02009-07-20)

5.3.2.45 #define APL_ERROR_UNSPECIFIED 0x00U

Unspecified error: also if data field is missing

5.3.2.46 #define APL_FIELD_CI_APL_ALARM_LONG 0x75U

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

5.3.2.47 #define APL_FIELD_CI_APL_ALARM_SHORT 0x74U

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

5.3.2.48 #define APL_FIELD_CI_APL_ERROR_LONG 0x6FU

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

5.3.2.49 #define APL_FIELD_CI_APL_ERROR_SHORT 0x6EU

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

5.3.2.50 #define APL_FIELD_CI_BAUD_1200 0xBAU

set baud rate to 1200 baud

5.3.2.51 #define APL_FIELD_CI_BAUD_19200 0xBEU

set baud rate to 19200 baud

5.3.2.52 #define APL_FIELD_CI_BAUD_2400 0xBBU

set baud rate to 2400 baud

5.3.2.53 #define APL_FIELD_CI_BAUD_300 0xB8U

set baud rate to 300 baud

5.3.2.54 #define APL_FIELD_CI_BAUD_38400 0xBFU

set baud rate to 38400 baud

5.3.2.55 #define APL_FIELD_CI_BAUD_4800 0xBCU

set baud rate to 4800 baud

5.3.2.56 #define APL_FIELD_CI_BAUD_600 0xB9U

set baud rate to 600 baud

5.3.2.57 #define APL_FIELD_CI_BAUD_9600 0xBDU

set baud rate to 9600 baud

5.3.2.58 #define APL_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)

5.3.2.59 #define APL_FIELD_CI_CMD_TO_DEVICE_NONE 0x51U

CMD to device with none header (DSMR v2.2)

5.3.2.60 #define APL_FIELD_CI_CMD_TO_DEVICE_SHORT 0x5AU

CMD to device with short header (DSMR 2.2)

5.3.2.61 #define APL_FIELD_CI_HEADER_LONG 0x72U

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

main_collector.c.

5.3.2.62 #define APL_FIELD_CI_HEADER_NONE 0x78U

no header followed by variable data format respond (EN 13757-3)

5.3.2.63 #define APL_FIELD_CI_HEADER_SHORT 0x7AU

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

main_collector.c.

5.3.2.64 #define APL_FIELD_CI_LINK_FROM_DEVICE_LONG 0x8BU

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

5.3.2.65 #define APL_FIELD_CI_LINK_FROM_DEVICE_SHORT 0x8AU

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

main_collector.c.

5.3.2.66 #define APL_FIELD_CI_LINK_TO_DEVICE_LONG 0x80U

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

5.3.2.67 #define APL_FIELD_CI_NETWORK 0x82U

networkapplication Layer (CENELEC-TC205)

5.3.2.68 #define APL_FIELD_CI_NULL 0xFFU

No CI-field transmitted. This define has to have the same value as TPL_FIELD_CI_NULL

5.3.2.69 #define APL_FIELD_CI_RELAY 0x81U

radio relaying and application layer (CEN-TC294)

5.3.2.70 #define APL_FIELD_CI_TIME_SYNC_1 0x6CU

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

5.3.2.71 #define APL_FIELD_CI_TIME_SYNC_2 0x6DU

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

5.3.2.72 #define APL_FIELD_CONF_WORD_AES_CBC 0x0400U

AES + CBC

5.3.2.73 #define APL_FIELD_CONF_WORD_AES_CBC_IV 0x0500U

AES + CBC with dynamic IV

5.3.2.74 #define APL_FIELD_CONF_WORD_AES_CBC_IV_DSMR 0x0F00U

AES + CBC with dynamic IV (DSMR Mode 15)

5.3.2.75 #define APL_FIELD_CONF_WORD_AES_CBC_MODE_7 0x0700U

AES + CBC with dynamic key

5.3.2.76 #define APL_FIELD_CONF_WORD_CONTENT_OF_MESSAGE_0 0x0004U

Content of message bit 0.

5.3.2.77 #define APL_FIELD_CONF_WORD_CONTENT_OF_MESSAGE_1 0x0008U

Content of message bit 1.

5.3.2.78 #define APL_FIELD_CONF_WORD_CONTENT_OF_MESSAGE_MASK 0x000CU

Content of message mask.

5.3.2.79 #define APL_FIELD_CONF_WORD_DES_CBC 0x0200U

DES + CBC

5.3.2.80 #define APL_FIELD_CONF_WORD_DES_CBC_IV 0x0300U

DES + CBC with dynamic IV

5.3.2.81 #define APL_FIELD_CONF_WORD_ENCRYPT_MODE_MASK 0x1F00U

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

5.3.2.82 #define APL_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)

5.3.2.83 #define APL_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)

5.3.2.84 #define APL_FIELD_CONF_WORD_NO_ENCRYPTION 0x0000U

Use no encryption.

5.3.2.85 #define APL_FIELD_STATUS_ALARM 0x03U

Abnormal condition / alarm
Examples:

main_meter.c.

5.3.2.86 #define APL_FIELD_STATUS_ANY_APPLICATION_ERROR 0x02U

Any application error

5.3.2.87 #define APL_FIELD_STATUS_APPLICATION_BUSY 0x01U

Application busy

5.3.2.88 #define APL_FIELD_STATUS_NO_ERROR 0x00U

No error

5.3.2.89 #define APL_FIELD_STATUS_PERMANENT_ERROR 0x08U

Permanent error

5.3.2.90 #define APL_FIELD_STATUS_POWER_LOW 0x04U

Power low

5.3.2.91 #define APL_FIELD_STATUS_TEMPORARY_ERROR 0x10U

Temporary error

5.3.2.92 #define APL_STATUS_BUSY 0x80U

The application layer is busy.

5.3.2.93 #define APL_STATUS_CONNECTED 0x04U

The meter device is connected to a data collector.

5.3.2.94 #define APL_STATUS_INACTIVE 0x01U

The application layer is inactive.

5.3.2.95 #define APL_STATUS_INSTALL 0x02U

The application layer is in installation mode.

5.3.2.96 #define APL_STATUS_SLEEP 0x08U

The device is ready to sleep

5.3.2.97 #define APL_TLG_FLAG_ACCESS_DEMAND 0x0002U

Set if user data class 1 is available.

5.3.2.98 #define APL_TLG_FLAG_BIDIRECTIONAL 0x8000U

Bidirectional communication.

5.3.2.99 #define APL_TLG_FLAG_CLR 0x0000U

The following flags are used within the stack to signal telegram status information to the user.

[@ Bit 0 Flow control DFC bit of the C-field. Defined in EN 60870-5-2 Bit 1 Access demand ACD bit of the C-field. Defined in EN 60870-5-2 Bit 2,3 Signature 00 No signature error 01 Signature unknown 10 Encryption error 11 No data encryption Bit 4 Hop-Counter Bit00 of the configuration word of Mode 5 EN13757-3 2013 Bit 5 Repeater Access Bit01 of the configuration word of Mode 5 EN13757-3 2013 Bit 6,7 Content of Message Bit02,03 of the configuration word of Mode 5 EN13757-3 2013 Bit 8,9,10,11,12 Encryption Mode Bit08,09,10,11,12 of the configuration word OM↩ S_VOL2_Primary_v401 Bit 13 Sync Bit13 of the configuration word of Mode 5 EN13757-3 2013 Bit 14 Accessibllity Bit14 of the configuration word of Mode 5 EN13757-3 2013 Bit 15 Bidirectional Bit15 of the configuration word of Mode 5 EN13757-3 2013

Clears all flags.

5.3.2.100 #define APL_TLG_FLAG_FLOW_CONTROL 0x0001U

Data collector only: Set if further requests may be sent.

5.3.2.101 #define APL_TLG_FLAG_HOP_COUNTER 0x0010U

Hop counter of the telegram

5.3.2.102 #define APL_TLG_FLAG_REPEATER_ACCESS 0x0020U

Hop counter of the telegram

5.3.2.103 #define APL_TLG_FLAG_RX_ALWAYS_ON 0x4000U

Receiver always on.

5.3.2.104 #define APL_TLG_FLAG_SIGNATURE 0x000CU

Mask for signature field.

5.3.2.105 #define APL_TLG_FLAG_SIGNATURE_ERROR 0x0008U

5.3.2.106 #define APL_TLG_FLAG_SIGNATURE_NO 0x000CU

Data collector only: The data is unencrypted.

5.3.2.107 #define APL_TLG_FLAG_SIGNATURE_UNKNOWN 0x0004U

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

5.3.2.108 #define APL_TLG_FLAG_SYNC 0x2000U

Bidirectional communication.

5.3.2.109 #define APL_VIF_ACTUALITY_DURATION 0x74U

E111 01nn -> nn see APL_VIF_TIME_x

5.3.2.110 #define APL_VIF_ADDRESS 0x7AU

E111 1010

5.3.2.111 #define APL_VIF_AVERAGING_DURATION 0x70U

E111 00nn -> nn see APL_VIF_TIME_x

5.3.2.112 #define APL_VIF_DATE 0x6CU

E110 1100 -> data field 0010b, type G

5.3.2.113 #define APL_VIF_DATE_TIME 0x6DU

E110 1101 -> data field 0100b, type F E110 1101 -> data field 0011b, type J E110 1101 -> data field 0110b, type I

5.3.2.114 #define APL_VIF_ENERGY_J 0x08U

E000 1nnn -> 10^(nnn) J

5.3.2.115 #define APL_VIF_ENERGY_WH 0x00U

E000 0nnn -> 10^(nnn-3) Wh

5.3.2.116 #define APL_VIF_ENHANCED_ID 0x79U

E111 1001

5.3.2.117 #define APL_VIF_EXT_TEMPERATURE_C 0x64U

E110 01nn -> 10^(nn-3) °C

5.3.2.118 #define APL_VIF_EXTENSION_BIT 0x80U

VIF(E) extension bit.

5.3.2.119 #define APL_VIF_FABRICATION_NO 0x78U

E111 1000

5.3.2.120 #define APL_VIF_FIRST_EXTENSION 0x7BU

1111 1011

5.3.2.121 #define APL_VIF_FLOW_TEMPERATURE_C 0x58U

E101 10nn -> 10^(nn-3) °C

5.3.2.122 #define APL_VIF_MANUFACTURER 0x7FU

E111 1111

5.3.2.123 #define APL_VIF_MASS_FLOW_KGH 0x50U

E101 0nnn -> 10^(nnn-3) kgh

5.3.2.124 #define APL_VIF_MASS_KG 0x18U

E001 1nnn -> 10^(nnn-3) kg

5.3.2.125 #define APL_VIF_ON_TIME 0x20U

E010 00nn -> nn see APL_VIF_TIME_x

5.3.2.126 #define APL_VIF_OPERATING_TIME 0x24U

E010 01nn -> nn see APL_VIF_TIME_x

5.3.2.127 #define APL_VIF_POWER_JH 0x30U

E011 0nnn -> 10^(nnn) Jh

5.3.2.128 #define APL_VIF_POWER_W 0x28U

E010 1nnn -> 10^(nnn-3) W

5.3.2.129 #define APL_VIF_PRESSURE_BAR 0x68U

E110 10nn -> 10(nn-3) bar

5.3.2.130 #define APL_VIF_RETURN_TEMPERATURE_C 0x5CU

E101 11nn -> 10^(nn-3) °C

5.3.2.131 #define APL_VIF_SECOND_EXTENSION 0xFDU

1111 1101

5.3.2.132 #define APL_VIF_TEMPERATURE_DIF_K 0x60U

E110 00nn -> 10^(nn-3) K

5.3.2.133 #define APL_VIF_UNITS_FOR_HCA 0x6EU

E110 1110

5.3.2.134 #define APL_VIF_VOLUME_FLOW_EXT_M3M 0x40U

E100 0nnn -> 10^(nnn-7) m^3m

5.3.2.135 #define APL_VIF_VOLUME_FLOW_EXT_M3S 0x48U

E100 1nnn -> 10^(nnn-9) m^3s

5.3.2.136 #define APL_VIF_VOLUME_FLOW_M3H 0x38U

E011 1nnn -> 10^(nnn-6) m^3h

5.3.2.137 #define APL_VIF_VOLUME_M3 0x10U

E001 0nnn -> 10^(nnn-6) m^3

5.3.2.138 #define APL_VIFE_ERR_FLAG 0x17U

E001 0111: Error flags (binary) (device type specific)

5.3.2.139 #define APL_VIFE_MANUFR_ACCELERATION 0x01U

Data from an acceleration sensor [0] X [1] Y [2] Z

5.3.2.140 #define APL_VIFE_MANUFR_ACTIVITY 0x02U

Activity: boolean

5.3.2.141 #define APL_VIFE_MANUFR_FALL 0x03U

Fall: boolean

5.3.2.142 #define APL_VIFE_SEC_KEY 0x19U

E001 1001: By default this value is reserved.

5.3.2.143 #define APL_VIFE_SPEC_SUPP_INFO 0x67U

E110 0111: Specfial supplier information

5.3.2.144 #define APL_VIFE_TRANS_CTR 0x08U

E000 1000: Unique telegram identification (transmission counter)

5.4 Enumerations

Enumerations

- enum E_APL_STATUS_t { E_APL_STATUS_ERROR, E_APL_STATUS_INVALID_PARAM_ERROR, E_APL_STATUS_NOT_INIT_ERROR, E_APL_STATUS_SUCCESS }
- enum E_APL_RET_t { E_APL_RET_OK = 0U, E_APL_RET_ERR, E_APL_RET_NOT_READY, E_APL_RET_DUPLICATED }
- enum E_APL_COM_t {
  E_APL_COM_UNKNOWN, **E_APL_COM_METER_DEFAULT_DATA**, **E_APL_COM_METER_RESERVED_SIGNED**, **E_APL_COM_METER_STATIC**,
  E_APL_COM_METER_RESERVED, **E_APL_COM_OTHER_DEFAULT_COMMAND**, **E_APL_COM_OTHER_RESERVED_AUTH**, **E_APL_COM_OTHER_RESERVED**,
  E_APL_COM_OTHER_RESERVED_FUTURE }

5.4.1 Detailed Description

This section describes enumerations of the Stackforce application layer.

5.4.2 Enumeration Type Documentation

5.4.2.1 enum **E_APL_COM_t**

Enumeration for the "CC"-bit in the configuration word. Defined in the EN13757-3 2013. Enumeration is used by:

- wmbus_apl_col_setContentOfMessage()
- wmbus_apl_mtr_setContentOfMessage()
- wmbus_apl_col_getContentOfMessage()
- wmbus_apl_mtr_getContentOfMessage()

5.4.2.2 enum **E_APL_RET_t**

Enumeration of return values. Enumeration is used by wmbus_apl_col_meterRemove(), wmbus_apl_col_meterSetRfAdapter() and wmbus_apl_col_meterSetKey().

Enumerator

E_APL_RET_OK All OK

E_APL_RET_ERR Error.

E_APL_RET_NOT_READY Stack is not ready

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

5.4.2.3 enum **E_APL_STATUS_t**

Enumeration of the status of the apl functions. Enumeration is used by wmbus_apl_mtr_start() and wmbus_apl_↩ col_start().

Enumerator

E_APL_STATUS_ERROR Error occured for unknown reasons.

E_APL_STATUS_INVALID_PARAM_ERROR Invalid input parameter error

E_APL_STATUS_NOT_INIT_ERROR Stack not initialized error.

E_APL_STATUS_SUCCESS Operation success.

5.5 Structures

Data Structures

- struct s_apl_recordInfo_t
- struct s_apl_tlgAttr_t
- struct s_apl_startCommonAttr_t

5.5.1 Detailed Description

This section describes structures of the Stackforce application layer.

5.5.2 Data Structure Documentation

5.5.2.1 struct s_apl_recordInfo_t

Structure of a data record.

Data Fields

uint8_t	c_coding	Coding of the value. This field specifies the data length (number of bytes) and the coding (Binary/Real/BCD)
uint8_t	c_exponent	Exponent value. The number of bits is dependent of the chosen value coding.
uint8_t	c_extended	Extended value information.
uint8_t	c_type	Type of the value (InstaneousMaximumMinimum)
uint8_t	c_unit	Unit of the value (EnergyVolumeMass...)
uint16_t	i_dataLen	Lenght of the attached data.
uint8_t *	pc_data	Pointer to attached data.

5.5.2.2 struct s_apl_tlgAttr_t

Structure of telegram attributes. Structure is used by wmbus_apl_getTlgAttr() and wmbus_apl_evt_newMeter().
Examples:

main_collector.c, and main_meter.c.

Data Fields

uint8_t	c_accNo	Access number of the request telegram.
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	Link quality (RSSI). 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.
uint16_t	i_dataLen	Includes the number of data bytes (with filler bytes!)
uint16_t	i_flag	Flag register. See APL_TLG_FLAG_x for getting information about available flags.
uint16_t	i_meterId	Id of the meter device the telegram was received from. If the target is in meter device mode, this field is always NULL.
s_wmbus_addr_t	s_addr	Address of the sending device. If the target is in meter device mode, this field is has to be ignored
s_wmbus_addr_t	s_longHeaderAddr	Address information used for the long telegram header

5.5.2.3 struct s_apl_startCommonAttr_t

Start structure holding common attributes for any WMBus device Structure is referenced by device specific start structures s_apl_start_collector_attr_t and s_apl_start_meter_attr_t

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.

5.6 Event Callbacks

Functions

- void wmbus_apl_evt_rx (void)

  This function is called if a rf telegram is received to display a LED or a symbol on the display.

- void wmbus_apl_evt_tx (uint8_t c_tlgId)

  This function is called if a rf telegram is sent to display a LED or a symbol on the display.

- void wmbus_apl_evt_tlgAvailable (E_WMBUS_RX_t e_status, uint8_t c_tlgReqId, s_apl_tlgAttr_t *ps_tlgAttr)

  This function is called if a new telegram is available.

- E_APL_HEADER_TYPE_t wmbus_apl_evt_getCiHeader (uint8_t c_ci)

  Requests the type of header by the customer application.

5.6.1 Detailed Description

This section describes callback functions that must be implemented by the application that uses the Stackforce application layer.

5.6.2 Function Documentation

5.6.2.1 E_APL_HEADER_TYPE_t wmbus_apl_evt_getCiHeader (uint8_t c_ci)

Requests the type of header by the customer application.

To handle manufacturer specific CI fields this event functions has to be implemented by the customer application. If the stack has to send or receive a manufacturer specific or any unknown CI field this event is called. The application has to return the type of header that applies for this CI field in order to let the stack parse the header properly.

Parameters

c_ci	User specific control information field.

Returns

Header type that applies for the given CI field. Return value must be of type E_APL_HEADER_TYPE_t.

Examples:

main_collector.c, and main_meter.c.

5.6.2.2 void wmbus_apl_evt_tlgAvailable (E_WMBUS_RX_t e_status, uint8_t c_tlgReqId, **s_apl_tlgAttr_t** * ps_tlgAttr)

This function is called if a new telegram is available.

Parameters

e_status	Status of the received telegram. Type of E_WMBUS_RX_t.
c_tlgReqId	Id of the request telegram. If the recevied telegram is a response to a previously sent request, this parameter holds the ID of the request telegram. It is set to APL_ERR_TLG_NOT_AVAILABLE if there is no request telegram available.
ps_tlgAttr	Pointer to structure s_apl_tlgAttr_t containing attributes of the received telegram. In case of an error this value is NULL.

Examples:

main_collector.c, and main_meter.c.

5.6.2.3 void wmbus_apl_evt_tx (uint8_t c_tlgId)

This function is called if a rf telegram is sent to display a LED or a symbol on the display.

Parameters

c_tlgId	Id of the sent telegram.

Examples:

main_collector.c, and main_meter.c.

5.7 Public API functions

Functions

- uint8_t wmbus_apl_getStatus (void)

  Returns the current status of the application layer.

- bool_t wmbus_apl_destroyTlg (uint8_t c_tlgId)

  Destroys a telegram. This function may only be used in events which provide user data.

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

  Adds data bytes to the telegram.

- uint16_t wmbus_apl_readData (uint8_t c_tlgId, uint8_t *pc_data, uint16_t i_len, uint16_t i_offset)

  Reads data bytes from a telegram.

- void wmbus_apl_setDeviceAddr (s_wmbus_addr_t *ps_addr)

  Sets the own address of the device.

- void wmbus_apl_getDeviceAddr (s_wmbus_addr_t *ps_addr)

  Returns the address of the device.

- bool_t wmbus_apl_getLongHeaderAddr (uint8_t c_tlgId, s_wmbus_addr_t *ps_addr)

  Copies the long header address for the application layer of a received telegram to the specified memory.

- void wmbus_apl_setAccessNo (uint8_t c_accessNo)

  Sets the ACC counter of the device.

- uint8_t wmbus_apl_getAccessNo (void)

  Get the current ACC counter of the meter device.

- void wmbus_apl_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_apl_getTxPower (void)

  Reads the transmission power.

- E_WMBUS_SLEEP_RESULT_t wmbus_apl_sleep (E_WMBUS_SLEEP_MODE_t e_sleepMode)

  Sets the target in low power-mode.

- void wmbus_apl_bufCleanUp (void)

  Cleans up all buffers.

- bool_t wmbus_apl_getTlgAttr (uint8_t c_tlgId, s_apl_tlgAttr_t *ps_tlgAttrRecv)

  Reads the APL attributes of a received telegram.

- bool_t wmbus_apl_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_apl_getRfChannel (void)

  Reads the radio channel of the rf-module.

5.7.1 Detailed Description

This section describes callback public API functions of the STACKFORCE application layer.

5.7.2 Function Documentation

5.7.2.1 bool_t wmbus_apl_destroyTlg (uint8_t c_tlgId)

Destroys a telegram. This function may only be used in events which provide user data.

Parameters

c_tlgId	Id of the telegram to destroy.

Returns

TRUE if the telegram is deleted successfully, FALSE if the given telegram ID is out of range.

Examples:

main_collector.c, and main_meter.c.

5.7.2.2 uint8_t wmbus_apl_getAccessNo (void)

Get the current ACC counter of the meter device.

This function serves as counter part to wmbus_apl_setAccessNo().

Returns

Value of the internal access counter

5.7.2.3 void wmbus_apl_getDeviceAddr (s_wmbus_addr_t * ps_addr)

Returns the address of the device.

Parameters

ps_addr	Address of the storage to write the meter address into. Parameter must be of type s_wmbus↩ _addr_t.

5.7.2.4 bool_t wmbus_apl_getLongHeaderAddr (uint8_t c_tlgId, s_wmbus_addr_t * ps_addr)

Copies the long header address for the application layer of a received telegram to the specified memory.

Parameters

c_tlgId	ID of the incoming telegram.
ps_addr	Pointer to array for storing the address. Parameter must be of type s_wmbus_addr_t.

Returns

TRUE if address is copied successfully. FALSE otherwise, e.g. if the telegram does not exist or has no long header.

5.7.2.5 E_RADIO_CHANNEL_INDEX_t wmbus_apl_getRfChannel (void)

Reads the radio channel of the rf-module.
Returns

Current radio channel

5.7.2.6 uint8_t wmbus_apl_getStatus (void)

Returns the current status of the application layer.
Returns

Current status of the application layer. Returns APL_STATUS_BUSY if stack is currently processing some task. If device is a meter APL_STATUS_CONNECTED is returned if the meter device is connected to a data collector.

5.7.2.7 bool_t wmbus_apl_getTlgAttr (uint8_t c_tlgId, **s_apl_tlgAttr_t** * ps_tlgAttrRecv)

Reads the APL attributes of a received telegram.

Parameters

c_tlgId	ID of the telegram.
ps_tlgAttrRecv	Memory to save the telegram information into. Must be of type s_apl_tlgAttr_t.

Returns

TRUE if attributes were read successfully. FALSE otherwise.

5.7.2.8 uint8_t wmbus_apl_getTxPower (void)

Reads the transmission power.

Returns

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

5.7.2.9 uint16_t wmbus_apl_readData (uint8_t c_tlgId, uint8_t * pc_data, uint16_t i_len, uint16_t i_offset)

Reads data bytes from a telegram.

Parameters

c_tlgId	Id of the telegram to read the datam from.
pc_data	Memory to store the read data into.
i_len	Number of bytes to read.
i_offset	Offset to start reading at.

Returns

Number of read bytes.

Examples:

main_collector.c.

5.7.2.10 void wmbus_apl_setAccessNo (uint8_t c_accessNo)

Sets the ACC counter of the device.

The acc counter will be incremented by every synchronous transmission. With this function the user is able to set a defined acc. This is needed for example if the stack wakes up from mode E_WMBUS_SLEEP_MODE_DEEPSLEEP. In this case the user has to set the correct value for the acc counter using this function.

Parameters

c_accessNo	Value for the internal access number

5.7.2.11 void wmbus_apl_setDeviceAddr (s_wmbus_addr_t * ps_addr)

Sets the own address of the device.

Parameters

ps_addr	Address of the device. Parameter must be of type s_wmbus_addr_t.

5.7.2.12 bool_t wmbus_apl_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

e_channel	Radio channel or channel mask. 0xFFFF is reserved. E_RADIO_CHANNEL_INDEX_t .

Returns

TRUE if the channel was set. FALSE otherwise.

Examples:

main_collector.c, and main_meter.c.

5.7.2.13 void wmbus_apl_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.

5.7.2.14 E_WMBUS_SLEEP_RESULT_t wmbus_apl_sleep (E_WMBUS_SLEEP_MODE_t e_sleepMode)

Sets the target in low power-mode.

Parameters

e_sleepMode	Power mode for the target E_WMBUS_SLEEP_MODE_t.

Returns

Returns the status of the request as E_WMBUS_SLEEP_RESULT_t.

5.7.2.15 bool_t wmbus_apl_writeData (uint8_t c_tlgId, uint8_t * pc_data, uint16_t i_len, bool_t b_reverse)

Adds data bytes to the telegram.

Parameters

c_tlgId	Id of the telegram to add the data to.
pc_data	Pointer to the data to add to the telegram.
i_len	Number of bytes to add.
b_reverse	Set TRUE to add the data in reverse order.

Returns

TRUE if the data could be written.

Examples:

main_meter.c.

5.8 Application Layer Interface Description for a Collector Device.

Modules

- Collector compile time settings
- Collector structures
- Collector event callbacks
- Collector public API functions

5.8.1 Detailed Description

This section describes the API for the STACKFORCE application layer.

5.9 Collector compile time settings

Macros

- #define APL_ERR_METER_OUT_OF_RANGE 0xFFFFU
- #define APL_ERR_DEVICE_NOT_ADDED 0xFFFF
- #define APL_CONFIGURATION_WORD_SYNC 0x20U
- #define APL_AES_SIZE_OF_KEY 0x10U

5.9.1 Detailed Description

Device specific compile time settings for a collector device.

5.9.2 Macro Definition Documentation

5.9.2.1 #define APL_AES_SIZE_OF_KEY 0x10U

AES key length

5.9.2.2 #define APL_CONFIGURATION_WORD_SYNC 0x20U

Macro to mask the synchronous bit of configuration word

5.9.2.3 #define APL_ERR_DEVICE_NOT_ADDED 0xFFFF

The device is not available.

5.9.2.4 #define APL_ERR_METER_OUT_OF_RANGE 0xFFFFU

The data could not be read because the meter index is out of range. This define has to have the same value as DLL↩ _ERR_METER_OUT_OF_RANGE

5.10 Collector structures

Data Structures

- struct s_apl_meterEntry_t
- struct s_apl_meterList_t
- struct s_apl_addMeterRet_t
- struct s_apl_startCollectorAttr_t

5.10.1 Detailed Description

Device specific structures for a collector device.

5.10.2 Data Structure Documentation

5.10.2.1 struct s_apl_meterEntry_t

Structure of one meter entry in the list.

Examples:

main_collector.c.

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

5.10.2.2 struct s_apl_meterList_t

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

Examples:

main_collector.c.

Data Fields

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

5.10.2.3 struct s_apl_addMeterRet_t

Structure of the return value of the function wmbus_mem_mgmt_mlMeterAdd.

Data Fields

E_APL_RET_t	e_ret	Return status of the function.
uint16_t	i_meterId	Meter index.

5.10.2.4 struct s_apl_startCollectorAttr_t

Start structure to start a collector device
Examples:

main_collector.c.

Data Fields

s_apl_meter↩ List_t *	ps_meterList	Static meters to install at startup. If no static meters are available set the parameter to NULL.
s_apl_start↩ CommonAttr_t	s_apl_start↩ CommonAttr	Common start attributes for all WMBus device types

5.11 Collector event callbacks

Functions

- void wmbus_apl_evt_ACCDMDReceived (uint8_t c_tlgId)

  This function is called if the receives ACC-DMD telegram from the meter.

- void wmbus_apl_evt_ACDBitSet (uint8_t c_tlgId)

  This function is called if the ACD bit of a meter telegram is set. Triggered by the ACD-bit the MUC should request alarm data (class 1) with the next unsolicited trans-mission of the Meter.

- bool_t wmbus_apl_evt_newMeter (s_wmbus_addr_t *ps_meter, s_apl_tlgAttr_t *ps_tlgAttr)

  A new meter device is requesting for connection.

5.11.1 Detailed Description

Device specific callbacks for a collector device.

5.11.2 Function Documentation

5.11.2.1 void wmbus_apl_evt_ACCDMDReceived (uint8_t c_tlgId)

This function is called if the receives ACC-DMD telegram from the meter.

Parameters

c_tlgId	Id of the received telegram.

Examples:

main_collector.c.

5.11.2.2 void wmbus_apl_evt_ACDBitSet (uint8_t c_tlgId)

This function is called if the ACD bit of a meter telegram is set. Triggered by the ACD-bit the MUC should request alarm data (class 1) with the next unsolicited trans-mission of the Meter.

Parameters

c_tlgId	Id of the received telegram.

Examples:

main_collector.c.

5.11.2.3 bool_t wmbus_apl_evt_newMeter (s_wmbus_addr_t * ps_meter, **s_apl_tlgAttr_t** * ps_tlgAttr)

A new meter device is requesting for connection.

Parameters

ps_meter	Data of the meter.
ps_tlgAttr	Telegram attributes from the installation request of the meter device.

Returns

TRUE if the meter device has to be added to the meter list.

Examples:

main_collector.c.

5.12 Collector public API functions

Functions

- void wmbus_apl_col_init (void)

  Initializes the application layer for the device type collector.

- E_APL_STATUS_t wmbus_apl_col_start (s_apl_startCollectorAttr_t *s_startAttr)

  Starts the application layer for the device type collector.

- void wmbus_apl_col_run (void)

  Default running method of collector device application layer.

- uint8_t wmbus_apl_col_createCmdClkSync (uint16_t i_meterId)

  Creates a clock synchronization command for a meter device. Be careful: If APL_CLOCK_ENABLED is set to FALSE the command can not be created. Then always APL_ERR_TLG_NOT_AVAILABLE is returned.

- uint8_t wmbus_apl_col_createAlarmRequest (uint16_t i_meterId)

  Create an alarm request for a meter device.

- uint8_t wmbus_apl_col_createUserDataRequest (uint16_t i_meterId)

  Creates a user data request.

- uint8_t wmbus_apl_col_readError (uint8_t c_tlgId)

  Reads the optional error code of a telegram.

- bool_t wmbus_apl_col_open (void)

  Start a bidirectional communication the next time as possible.

- bool_t wmbus_apl_col_sendQueued (uint8_t c_tlgId, bool_t b_append)

  Adds a telegram to the collector queue.

- uint8_t wmbus_apl_col_close (uint16_t i_meterId, bool_t b_sendNke)

  Resets the flag to start a bidirectional communication.

- s_apl_addMeterRet_t wmbus_apl_col_meterAdd (s_apl_meterEntry_t *ps_meterEntry)

  Adds a meter device to the meter list.

- E_APL_RET_t wmbus_apl_col_meterRemove (uint16_t i_meterId)

  Removes a meter device from the meter list.

- E_APL_RET_t wmbus_apl_col_meterSetRfAdapter (uint16_t i_meterId, s_wmbus_addr_t *ps_rf)

  Updates the RF adapter address of a meter device.

- bool_t wmbus_apl_col_meterGetRfAdapter (uint16_t i_meterId, s_wmbus_addr_t *ps_rf)

  Reads the RF adapter address of a meter device.

- E_APL_RET_t wmbus_apl_col_meterSetKey (uint16_t i_meterId, uint8_t *pc_key, uint8_t c_len)

  Updates the meter specific key.

- bool_t wmbus_apl_col_meterGetKey (uint16_t i_meterId, uint8_t *pc_key, uint8_t c_len)

  Reads the key of the meter.

- uint16_t wmbus_apl_col_meterGetNum (void)

  Gets the number of connected meter devices.

- bool_t wmbus_apl_col_meterGetAddr (uint16_t i_meterId, s_wmbus_addr_t *ps_meterAddr)

  Reads the attributes of a meter device.

- uint16_t wmbus_apl_col_meterGetId (s_wmbus_addr_t *ps_meterAddr)

  Reads the id of a meter device.

- void wmbus_apl_col_clearTlgQueue (void)

  This function resets the complete queue.

- E_WMBUS_COL_QUEUE_RET_t wmbus_apl_col_clearMtrTlgQueue (uint16_t i_meterId)

  Removes all telegrams from the queue for a specific meter.

- uint8_t wmbus_apl_col_createTlg (uint8_t c_ci, uint16_t i_meterId, bool_t b_encrypt)

  Creates a telegram. On collector device.

- bool_t wmbus_apl_col_sendTlg (uint8_t c_tlgId)

  Sends a telegram. Collector device. Before the telegram has to be created with function wmbus_apl_col_createTlg().

- void wmbus_apl_col_wakeUp (E_WMBUS_SLEEP_MODE_t e_sleepMode)

  Wake up the target. Collector device.

- bool_t wmbus_apl_col_setContentOfMessage (E_APL_COM_t e_content, uint8_t c_tlgId)

  Set the content of message bits that are stored in the configuration word defined in the EN13757-3 2013.

- E_APL_COM_t wmbus_apl_col_getContentOfMessage (uint8_t c_tlgId)

  Get the content of message bits which are stored in the configuration word. This function should only be called for received telegrams.

5.12.1 Detailed Description

Device specific public APIs for a collector device.

5.12.2 Function Documentation

5.12.2.1 E_WMBUS_COL_QUEUE_RET_t wmbus_apl_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.

5.12.2.2 void wmbus_apl_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.

5.12.2.3 uint8_t wmbus_apl_col_close (uint16_t i_meterId, bool_t b_sendNke)

Resets the flag to start a bidirectional communication.

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 close telegram.

Examples:

main_collector.c.

5.12.2.4 uint8_t wmbus_apl_col_createAlarmRequest (uint16_t i_meterId)

Create an alarm request for a meter device.

Parameters

i_meterId	Id of the meter device to create the telegram.

Returns

Id of the request telegram. APL_ERR_TLG_NOT_AVAILABLE if the request could not be created.

5.12.2.5 uint8_t wmbus_apl_col_createCmdClkSync (uint16_t i_meterId)

Creates a clock synchronization command for a meter device. Be careful: If APL_CLOCK_ENABLED is set to FALSE the command can not be created. Then always APL_ERR_TLG_NOT_AVAILABLE is returned.

Parameters

i_meterId	Id of the meter device to create the telegram.

Returns

Id of the command telegram. APL_ERR_TLG_NOT_AVAILABLE if the request could not be created.

Examples:

main_collector.c.

5.12.2.6 uint8_t wmbus_apl_col_createTlg (uint8_t c_ci, uint16_t i_meterId, bool_t b_encrypt)

Creates a telegram. On collector device.

Parameters

c_ci	CI field to use.
i_meterId	Id of the meter device to send the telegram to. If the device is a meter this value is ignored.
b_encrypt	Set to enable telegram encryption.

Returns

Telegram id, APL_ERR_TLG_NOT_AVAILABLE if no telegram could be created.

5.12.2.7 uint8_t wmbus_apl_col_createUserDataRequest (uint16_t i_meterId)

Creates a user data request.

Parameters

i_meterId	Id of the meter device to create the telegram.
ps_record	Data of the request. NULL if no data has to be sent.

Returns

Id of the request telegram. APL_ERR_TLG_NOT_AVAILABLE if the request could not be created.

Examples:

main_collector.c.

5.12.2.8 **E_APL_COM_t** wmbus_apl_col_getContentOfMessage (uint8_t c_tlgId)

Get the content of message bits which are stored in the configuration word. This function should only be called for received telegrams.

Please note that this function is called by collector device, it returns the content of message of a received telegram from a meter.

Parameters

c_tlgId	ID of the telegram.

Returns

Content of the message. E_APL_COM_t .

5.12.2.9 **s_apl_addMeterRet_t** wmbus_apl_col_meterAdd (**s_apl_meterEntry_t** * ps_meterEntry)

Adds a meter device to the meter list.

Parameters

ps_meterEntry	Entry of a meter.

Returns

Returns the id of the meter device. Returns APL_ERR_DEVICE_NOT_ADDED in case of failure.

5.12.2.10 bool_t wmbus_apl_col_meterGetAddr (uint16_t i_meterId, s_wmbus_addr_t * ps_meterAddr)

Reads the attributes of a meter device.

Parameters

i_meterId	Id of the meter device.
ps_meterAddr	Memory to write the meter address into.

Returns

FALSE if the index is out of range.

5.12.2.11 uint16_t wmbus_apl_col_meterGetId (s_wmbus_addr_t * ps_meterAddr)

Reads the id of a meter device.

Parameters

ps_meterAddr	Address of the meter device.

Returns

id of the meter, APL_ERR_METER_OUT_OF_RANGE if there is no meter in the list with the address ps_meterAddr.

5.12.2.12 bool_t wmbus_apl_col_meterGetKey (uint16_t i_meterId, uint8_t * pc_key, uint8_t c_len)

Reads the key of the meter.

Parameters

i_meterId	Id 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.

5.12.2.13 uint16_t wmbus_apl_col_meterGetNum (void)

Gets the number of connected meter devices.

Returns

Number of connected meter devices.

5.12.2.14 bool_t wmbus_apl_col_meterGetRfAdapter (uint16_t i_meterId, s_wmbus_addr_t * ps_rf)

Reads the RF adapter address of a meter device.

Parameters

i_meterId	Id of the meter device.
ps_rf	Destination storage to write the address of the RF adapter to.

Returns

TRUE if the adapter address could be read.

5.12.2.15 **E_APL_RET_t** wmbus_apl_col_meterRemove (uint16_t i_meterId)

Removes a meter device from the meter list.

Parameters

i_meterId	Id of the meter to remove.

Returns

E_APL_RET_OK if deleting was successful.

5.12.2.16 **E_APL_RET_t** wmbus_apl_col_meterSetKey (uint16_t i_meterId, uint8_t * pc_key, uint8_t c_len)

Updates the meter specific key.

Parameters

i_meterId	Id of the meter device. APL_ERR_METER_OUT_OF_RANGE to set the same key for all devices.
pc_key	Key to set.
c_len	Size of the key.

Returns

E_APL_RET_OK if the key could be changed.

5.12.2.17 **E_APL_RET_t** wmbus_apl_col_meterSetRfAdapter (uint16_t i_meterId, s_wmbus_addr_t * ps_rf)

Updates the RF adapter address of a meter device.

Parameters

i_meterId	Id of the meter device.
ps_rf	Address of the RF adapter the meter device is connected to.

Returns

E_APL_RET_OK if the adapter address could be changed.

5.12.2.18 bool_t wmbus_apl_col_open (void)

Start a bidirectional communication the next time as possible.

Returns

TRUE if opperation was successful.

Examples:

main_collector.c.

5.12.2.19 uint8_t wmbus_apl_col_readError (uint8_t c_tlgId)

Reads the optional error code of a telegram.

Parameters

c_tlgId	Id of the telegram to read from.

Returns

Reset code of the telegram. If no error code is available, APL_ERROR_UNSPECIFIED is returned (cf. EN 13757-3).

5.12.2.20 void wmbus_apl_col_run (void)

Default running method of collector device application layer.

This method has to be called as often as possible in order to let the stack process its tasks in time.

Examples:

main_collector.c.

5.12.2.21 bool_t wmbus_apl_col_sendQueued (uint8_t c_tlgId, bool_t b_append)

Adds a telegram to the collector queue.

Parameters

c_tlgId	Id of the request.
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.

Examples:

main_collector.c.

5.12.2.22 bool_t wmbus_apl_col_sendTlg (uint8_t c_tlgId)

Sends a telegram. Collector device. Before the telegram has to be created with function wmbus_apl_col_createTlg().

Parameters

c_tlgId	Id of the telegram.

Returns

TRUE if the telegram could be sent.

5.12.2.23 bool_t wmbus_apl_col_setContentOfMessage (**E_APL_COM_t** e_content, uint8_t c_tlgId)

Set the content of message bits that are stored in the configuration word defined in the EN13757-3 2013.

This function should only be called for telegrams created by the collector device.

Parameters

e_content	Content of the message to set. Must be of type E_APL_COM_t.
c_tlgId	ID of the telegram.

Returns

TRUE if content of message was set. FALSE otherwise.

5.12.2.24 **E_APL_STATUS_t** wmbus_apl_col_start (**s_apl_startCollectorAttr_t** * s_startAttr)

Starts the application layer for the device type collector.

Parameters

s_startAttr	Start attributes of the collector device.

Returns

Status of the operation E_APL_STATUS_t

Examples:

main_collector.c.

5.12.2.25 void wmbus_apl_col_wakeUp (E_WMBUS_SLEEP_MODE_t e_sleepMode)

Wake up the target. Collector device.

Parameters

e_sleepMode	Current power-mode

5.13 Application Layer Interface Description for a Meter Device.

Modules

- Meter compile time settings
- Meter structures
- Meter event callbacks
- Meter public API functions

5.13.1 Detailed Description

This section describes the API for the STACKFORCE application layer.

5.14 Meter compile time settings

Macros

- #define APL_IGNORE_UNENCRYPTED_COLLECTOR_COMMANDS TRUE
- #define APL_INSTALL_RETRIES 6U
- #define APL_METER_SPONTANEOUS_CONF FALSE

5.14.1 Detailed Description

Device specific compile time settings for a meter device.

5.14.2 Macro Definition Documentation

5.14.2.1 #define APL_IGNORE_UNENCRYPTED_COLLECTOR_COMMANDS TRUE

Disables the reception of unencrypted commands (sent from collector to meter)

5.14.2.2 #define APL_INSTALL_RETRIES 6U

If the device is in installation mode, run installation maximal 3 times. Set this defined to 0 to repeat the telegram endless.

5.14.2.3 #define APL_METER_SPONTANEOUS_CONF FALSE

Enables sending spontaneous telegrams from meter device without reply.

5.15 Meter structures

Data Structures

- struct s_apl_startMeterAttr_t

5.15.1 Detailed Description

Device specific structures for a meter device.

5.15.2 Data Structure Documentation

5.15.2.1 struct s_apl_startMeterAttr_t

Start structure to start a meter device
Examples:

main_meter.c.

Data Fields

bool_t	b_connected	Tell the device to bind to a data collector. Only possible, if the collector address is valid.
uint32_t	l_interval	The interval for periodical user data in ms.
uint8_t *	pc_ownKey	Key of the device
s_apl_start↩ CommonAttr_t	s_apl_start↩ CommonAttr	Common start attributes for all WMBus device types
s_wmbus_↩ addr_t *	s_collectorAddr	The address of the collector. If the collector is not known set it to zero.

5.16 Meter event callbacks

Functions

- uint8_t wmbus_apl_evt_alarmRequested (void)

  This function is called if user data class 1 (alarm) is requested. An alarm telegram (CI=74) is created. The content of the telegram has to be written with function wmbus_apl_writeApplicationError().

- bool_t wmbus_apl_evt_userDataRequested (uint8_t c_tlgId, bool_t b_periodical)

  This function is called if user data class 2 is requested. Now meter device data is requested. The respond telegram is created automatically, the data has to be added to the telegram by function wmbus_apl_writeData().

5.16.1 Detailed Description

Device specific callbacks for a meter device.

5.16.2 Function Documentation

5.16.2.1 uint8_t wmbus_apl_evt_alarmRequested (void)

This function is called if user data class 1 (alarm) is requested. An alarm telegram (CI=74) is created. The content of the telegram has to be written with function wmbus_apl_writeApplicationError().

Be careful: To signal, that there is an alarm available, before wmbus_apl_setErrorAvailable () should be used. This method sets a flag which is deleted if a user data class 1 request is received.

Returns

Alarm code to transmit.

Examples:

main_meter.c.

5.16.2.2 bool_t wmbus_apl_evt_userDataRequested (uint8_t c_tlgId, bool_t b_periodical)

This function is called if user data class 2 is requested. Now meter device data is requested. The respond telegram is created automatically, the data has to be added to the telegram by function wmbus_apl_writeData().

Parameters

c_tlgId	Id of the respond telegram.
b_periodical	TRUE if periodical user data is requested.

Returns

Set if a respond has to be sent. Not set if the request has to be ignored.

Examples:

main_meter.c.

5.17 Meter public API functions

Functions

- void wmbus_apl_mtr_init (void)

  Initializes the application layer for the device type meter.

- E_APL_STATUS_t wmbus_apl_mtr_start (s_apl_startMeterAttr_t *s_startAttr)

  Starts the application layer for the device type meter.

- void wmbus_apl_mtr_run (void)

  Default running method of meter device application layer.

- uint8_t wmbus_apl_mtr_getAlarmCode (void)

  Gets the alarm code from APL.

- void wmbus_apl_mtr_setAlarmCode (uint8_t c_alarmCode)

  Sets the alarm code in APL.

- void wmbus_apl_mtr_clrAlarmCode (uint8_t c_alarmCode)

  Clears the given alarm code in APL.

- uint8_t wmbus_apl_mtr_sendInstallationRequest (void)

  Creates an installation request.

- bool_t wmbus_apl_mtr_setLongHeaderAddr (uint8_t c_tlgId, s_wmbus_addr_t *ps_addr)

  Sets the long header address for the application layer of an outgoing telegram. By default, this is identical to the own address, but this function call permits to enter the address of e.g. a connected meter Before using this function, create the telegram, and pass the ID here.

- bool_t wmbus_apl_mtr_setErrorFlag (uint8_t c_error)

  Sets the error flag (aka. status byte) of the APL header of the meter telegram. The meter will afterwards send its telegram with the Status field according to value that has been set from this function. To clear status byte, call function wmbus_apl_mtr_clrErrorFlag().

- void wmbus_apl_mtr_clrErrorFlag (uint8_t c_error)

  Allowes to clear application error flags. The application error flags are never cleared automatically by the application layer.

- uint8_t wmbus_apl_mtr_getErrorFlag (void)

  Reads if there is a set application error.

- bool_t wmbus_apl_mtr_setKey (uint8_t *pc_key, uint8_t c_len)

  Writes the encryption key.

- bool_t wmbus_apl_mtr_getKey (uint8_t *pc_key, uint8_t c_len)

Reads the encryption key.

- void wmbus_apl_mtr_setCollectorAddress (s_wmbus_addr_t *ps_collectorAddr)

  Sets the address of the connected collector.

- void wmbus_apl_mtr_getCollectorAddress (s_wmbus_addr_t *ps_collectorAddr)

  Gets the address of the connected collector.

- void wmbus_apl_mtr_setInterval (uint32_t l_interval)

  Sets the periodical interval of the meter device.

- uint32_t wmbus_apl_mtr_getInterval (void)

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

- void wmbus_apl_mtr_setConnected (bool_t b_connect)

  Sets the meter as connected or disconnect.

- bool_t wmbus_apl_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.

- E_WMBUS_RESPONSE_DELAY_t wmbus_apl_mtr_getResponseDelay (void)

  Returns the response delay currently used by the meter device.

- uint8_t wmbus_apl_mtr_createTlg (uint8_t c_ci, uint16_t i_meterId, bool_t b_encrypt, bool_t b_synchronous)

  Creates a telegram. On meter device.

- bool_t wmbus_apl_mtr_sendTlg (uint8_t c_tlgId)

  Sends a telegram. Meter device. Before the telegram has to be created with function wmbus_apl_mtr_createTlg().

- void wmbus_apl_mtr_wakeUp (E_WMBUS_SLEEP_MODE_t e_sleepMode)

  Wake up the target. Meter device.

- bool_t wmbus_apl_mtr_setContentOfMessage (E_APL_COM_t e_content, uint8_t c_tlgId)

  Set the content of message bits that are stored in the configuration word defined in the EN13757-3 2013.

- E_APL_COM_t wmbus_apl_mtr_getContentOfMessage (uint8_t c_tlgId)

  Get the content of message bits which are stored in the configuration word. This function should only be called for received telegrams.

5.17.1 Detailed Description

Device specific public APIs for a meter device.

5.17.2 Function Documentation

5.17.2.1 void wmbus_apl_mtr_clrAlarmCode (uint8_t c_alarmCode)

Clears the given alarm code in APL.

Parameters

c_alarmCode	Alarm Code to be cleared.

Examples:

main_meter.c.

5.17.2.2 void wmbus_apl_mtr_clrErrorFlag (uint8_t c_error)

Allowes to clear application error flags. The application error flags are never cleared automatically by the application layer.

Parameters

c_error	Allowes to select one or more error to clear (cf. EN 13757-3 Table 5).

Examples:

main_meter.c.

5.17.2.3 uint8_t wmbus_apl_mtr_createTlg (uint8_t c_ci, uint16_t i_meterId, bool_t b_encrypt, bool_t b_synchronous)

Creates a telegram. On meter device.

Parameters

c_ci	CI field to use.
i_meterId	Id of the meter device to send the telegram to. If the device is a meter this value is ignored.
b_encrypt	Set to enable telegram encryption.
b_synchronous	Set to set the synchronous bit in the configuration word. Please only set the parameter if you know the rules for sending synchronous telegrams (EN13757-3&4) A collector will ignore this value.

Returns

Telegram id, APL_ERR_TLG_NOT_AVAILABLE if no telegram could be created.

5.17.2.4 uint8_t wmbus_apl_mtr_getAlarmCode (void)

Gets the alarm code from APL.
Returns

Alarm code.
Examples:

main_meter.c.

5.17.2.5 void wmbus_apl_mtr_getCollectorAddress (s_wmbus_addr_t * ps_collectorAddr)

Gets the address of the connected collector.

Parameters

ps_collectorAddr	Address of the collector.

5.17.2.6 **E_APL_COM_t** wmbus_apl_mtr_getContentOfMessage (uint8_t c_tlgId)

Get the content of message bits which are stored in the configuration word. This function should only be called for received telegrams.

Please note that this function is called by meter device, it returns the content of message of a received telegram from a collector.

Parameters

c_tlgId	ID of the telegram.

Returns

Content of the message. E_APL_COM_t .

5.17.2.7 uint8_t wmbus_apl_mtr_getErrorFlag (void)

Reads if there is a set application error.
Returns

Set error.

5.17.2.8 uint32_t wmbus_apl_mtr_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.

5.17.2.9 bool_t wmbus_apl_mtr_getKey (uint8_t * pc_key, uint8_t c_len)

Reads the encryption key.

Parameters

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

Returns

TRUE if the key is read successfully. FALSE otherwise.

5.17.2.10 E_WMBUS_RESPONSE_DELAY_t wmbus_apl_mtr_getResponseDelay (void)

Returns the response delay currently used by the meter device.

Parameters

None.	

Returns

E_WMBUS_RESPONSE_DELAY_SLOW - indicating slow response delay. E_WMBUS_RESPONSE_DELAY↩ _FAST - indicating fast response delay.

5.17.2.11 void wmbus_apl_mtr_run (void)

Default running method of meter device application layer.

This method has to be called as often as possible in order to let the stack process its tasks in time.

Examples:

main_meter.c.

5.17.2.12 uint8_t wmbus_apl_mtr_sendInstallationRequest (void)

Creates an installation request.

Returns

Id of the request telegram. APL_ERR_TLG_NOT_AVAILABLE if the request could not be sent.

5.17.2.13 bool_t wmbus_apl_mtr_sendTlg (uint8_t c_tlgId)

Sends a telegram. Meter device. Before the telegram has to be created with function wmbus_apl_mtr_createTlg().

Parameters

c_tlgId	Id of the telegram.

Returns

TRUE if the telegram could be sent.

5.17.2.14 void wmbus_apl_mtr_setAlarmCode (uint8_t c_alarmCode)

Sets the alarm code in APL.

Parameters

c_alarmCode	Alarm Code to be set.

5.17.2.15 void wmbus_apl_mtr_setCollectorAddress (s_wmbus_addr_t * ps_collectorAddr)

Sets the address of the connected collector.

Parameters

ps_collectorAddr	Address of the collector.

5.17.2.16 void wmbus_apl_mtr_setConnected (bool_t b_connect)

Sets the meter as connected or disconnect.

Parameters

b_connect	Set to connect the meter device.

5.17.2.17 bool_t wmbus_apl_mtr_setContentOfMessage (**E_APL_COM_t** e_content, uint8_t c_tlgId)

Set the content of message bits that are stored in the configuration word defined in the EN13757-3 2013.

This function should only be called for telegrams created by the meter device.

Parameters

e_content	Content of the message to set. Must be of type E_APL_COM_t.
c_tlgId	ID of the telegram.

Returns

TRUE if content of message was set. FALSE otherwise.

5.17.2.18 bool_t wmbus_apl_mtr_setErrorFlag (uint8_t c_error)

Sets the error flag (aka. status byte) of the APL header of the meter telegram. The meter will afterwards send its telegram with the Status field according to value that has been set from this function. To clear status byte, call function wmbus_apl_mtr_clrErrorFlag().

Parameters

c_error	This parameters is to signalize a temporary or permanent error (cf. EN 13757-3 Table 5).

Returns

TRUE if successful (when the function is called in meter device) FALSE if not successful

5.17.2.19 void wmbus_apl_mtr_setInterval (uint32_t l_interval)

Sets the periodical interval of the meter device.

Parameters

l_interval	Periodical interval in milliseconds.

5.17.2.20 bool_t wmbus_apl_mtr_setKey (uint8_t * pc_key, uint8_t c_len)

Writes the encryption key.

Parameters

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

Returns

TRUE if the key is written successfully. FALSE otherwise.

5.17.2.21 bool_t wmbus_apl_mtr_setLongHeaderAddr (uint8_t c_tlgId, s_wmbus_addr_t * ps_addr)

Sets the long header address for the application layer of an outgoing telegram. By default, this is identical to the own address, but this function call permits to enter the address of e.g. a connected meter Before using this function, create the telegram, and pass the ID here.

Parameters

c_tlgId	ID of the outgoing telegram.
ps_addr	Address to use for this telegram.

Returns

Success: FALSE if the telegram does not exist or has no long header

5.17.2.22 bool_t wmbus_apl_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.

5.17.2.23 **E_APL_STATUS_t** wmbus_apl_mtr_start (**s_apl_startMeterAttr_t** * s_startAttr)

Starts the application layer for the device type meter.

Parameters

s_startAttr	Start attributes for the meter device.

Returns

Status of the operation E_APL_STATUS_t

Examples:

main_meter.c.

5.17.2.24 void wmbus_apl_mtr_wakeUp (E_WMBUS_SLEEP_MODE_t e_sleepMode)

Wake up the target. Meter device.

Parameters

e_sleepMode	Current power-mode

5.18 Clock Interface Description

Modules

- Enumerations
- Structures
- Public API functions

5.18.1 Detailed Description

This section describes the API for the STACKFORCE clock module.

5.19 Enumerations

Enumerations

- enum E_CLOCK_t {
  E_CLOCK_SECONDS, **E_CLOCK_MINUTES**, **E_CLOCK_HOURS**, **E_CLOCK_DAYS**,
  E_CLOCK_MONTHS, **E_CLOCK_YEARS** }

5.19.1 Detailed Description

Enumerations for a common device.

5.19.2 Enumeration Type Documentation

5.19.2.1 enum **E_CLOCK_t**

Elements of the clock.

5.20 Structures

Data Structures

- struct s_clock_t

5.20.1 Detailed Description

Structures for a common device.

5.20.2 Data Structure Documentation

5.20.2.1 struct s_clock_t

Structure of a clock.

Data Fields

bool_t	b_running	Set if the clock is running.
uint8_t	c_days	Days of month (1...28.30.31)
uint8_t	c_hours	Hours (0...23)
uint8_t	c_minutes	Minutes (0...59)
uint8_t	c_months	Months of year (1...12)
uint8_t	c_seconds	Seconds (0...59)
uint16_t	i_years	Year (0...9999)
uint32_t	l_ms	System time in milliseconds.

5.21 Public API functions

Functions

- void wmbus_clock_init (s_clock_t *ps_clock)

  Initializes the clock.

- bool_t wmbus_clock_run (s_clock_t *ps_clock)

  Runs a clock.

- bool_t wmbus_clock_isLeap (uint16_t i_year)
- void wmbus_clock_start (s_clock_t *ps_clock)

  Starts the clock.

- void wmbus_clock_stop (s_clock_t *ps_clock)

  Stops the clock.

- void wmbus_clock_inc (s_clock_t *ps_clock, E_CLOCK_t e_element, bool_t b_single)

  Increments an element of the clock.

- void wmbus_clock_dec (s_clock_t *ps_clock, E_CLOCK_t e_element, bool_t b_single)

  Decrements an element of the clock.

- uint32_t wmbus_clock_calcMinutes (s_clock_t *ps_clock)

  Calc the minutes since the clock is running. (Max. 64 Years). This function is needed for meter devices which use the CI-field 0x8D.

5.21.1 Detailed Description

Public APIs for a common device.

5.21.2 Function Documentation

5.21.2.1 uint32_t wmbus_clock_calcMinutes (**s_clock_t** * ps_clock)

Calc the minutes since the clock is running. (Max. 64 Years). This function is needed for meter devices which use the CI-field 0x8D.

Parameters

ps_clock	pointer to the clock.

5.21.2.2 void wmbus_clock_dec (**s_clock_t** * ps_clock, **E_CLOCK_t** e_element, bool_t b_single)

Decrements an element of the clock.

Parameters

ps_clock	Clock to decrement.
e_element	Element to decrement.
b_single	Set if only the single element has to be changed. If not set all other elements are updated if necessary.

5.21.2.3 void wmbus_clock_inc (**s_clock_t** * ps_clock, **E_CLOCK_t** e_element, bool_t b_single)

Increments an element of the clock.

Parameters

ps_clock	Clock to increment.
e_element	Element to increment.
b_single	Set if only the single element has to be changed. If not set all other elements are updated if necessary.

5.21.2.4 void wmbus_clock_init (**s_clock_t** * ps_clock)

Initializes the clock.

Parameters

ps_clock	Clock to initialize.

5.21.2.5 bool_t wmbus_clock_isLeap (uint16_t i_year)

Checks if the year is a leap year.

Parameters

i_year	Year.

Returns

TRUE if the year is a leap year.

5.21.2.6 bool_t wmbus_clock_run (**s_clock_t** * ps_clock)

Runs a clock.

Parameters

ps_clock	Clock to run.

Returns

TRUE if one or more elements of the clock are changed.

5.21.2.7 void wmbus_clock_start (**s_clock_t** * ps_clock)

Starts the clock.

Parameters

ps_clock	Clock to start.

5.21.2.8 void wmbus_clock_stop (**s_clock_t** * ps_clock)

Stops the clock.

Parameters

ps_clock	Clock to stop.

Chapter 6

File Documentation

6.1 inc/pub/apl/wmbus_apl_api.h File Reference

Application Layer API of Wireless M-Bus.

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

Data Structures

- struct s_apl_recordInfo_t
- struct s_apl_tlgAttr_t
- struct s_apl_startCommonAttr_t

Macros

- #define **__DECL_APPLICATIONLAYER_API_H__** extern
- #define APL_STATUS_BUSY 0x80U
- #define APL_STATUS_INACTIVE 0x01U
- #define APL_STATUS_INSTALL 0x02U
- #define APL_STATUS_CONNECTED 0x04U
- #define APL_STATUS_SLEEP 0x08U
- #define APL_ERR_TLG_NOT_AVAILABLE 0xFFU
- #define APL_FIELD_CI_CMD_TO_DEVICE_NONE 0x51U
- #define APL_FIELD_CI_CMD_TO_DEVICE_SHORT 0x5AU
- #define APL_FIELD_CI_CMD_TO_DEVICE_LONG 0x5BU
- #define APL_FIELD_CI_TIME_SYNC_1 0x6CU
- #define APL_FIELD_CI_TIME_SYNC_2 0x6DU
- #define APL_FIELD_CI_APL_ERROR_SHORT 0x6EU

- #define APL_FIELD_CI_APL_ERROR_LONG 0x6FU
- #define APL_FIELD_CI_HEADER_LONG 0x72U
- #define APL_FIELD_CI_APL_ALARM_SHORT 0x74U
- #define APL_FIELD_CI_APL_ALARM_LONG 0x75U
- #define APL_FIELD_CI_HEADER_NONE 0x78U
- #define APL_FIELD_CI_HEADER_SHORT 0x7AU
- #define APL_FIELD_CI_LINK_TO_DEVICE_LONG 0x80U
- #define APL_FIELD_CI_RELAY 0x81U
- #define APL_FIELD_CI_NETWORK 0x82U
- #define APL_FIELD_CI_LINK_FROM_DEVICE_SHORT 0x8AU
- #define APL_FIELD_CI_LINK_FROM_DEVICE_LONG 0x8BU
- #define APL_FIELD_CI_BAUD_300 0xB8U
- #define APL_FIELD_CI_BAUD_600 0xB9U
- #define APL_FIELD_CI_BAUD_1200 0xBAU
- #define APL_FIELD_CI_BAUD_2400 0xBBU
- #define APL_FIELD_CI_BAUD_4800 0xBCU
- #define APL_FIELD_CI_BAUD_9600 0xBDU
- #define APL_FIELD_CI_BAUD_19200 0xBEU
- #define APL_FIELD_CI_BAUD_38400 0xBFU
- #define APL_FIELD_CI_NULL 0xFFU
- #define APL_FIELD_STATUS_NO_ERROR 0x00U
- #define APL_FIELD_STATUS_APPLICATION_BUSY 0x01U
- #define APL_FIELD_STATUS_ANY_APPLICATION_ERROR 0x02U
- #define APL_FIELD_STATUS_ALARM 0x03U
- #define APL_FIELD_STATUS_POWER_LOW 0x04U
- #define APL_FIELD_STATUS_PERMANENT_ERROR 0x08U
- #define APL_FIELD_STATUS_TEMPORARY_ERROR 0x10U
- #define APL_FIELD_CONF_WORD_CONTENT_OF_MESSAGE_MASK 0x000CU
- #define APL_FIELD_CONF_WORD_CONTENT_OF_MESSAGE_0 0x0004U
- #define APL_FIELD_CONF_WORD_CONTENT_OF_MESSAGE_1 0x0008U
- #define APL_FIELD_CONF_WORD_MODE_7_CONTENT_OF_MESSAGE_0 0x4000U
- #define APL_FIELD_CONF_WORD_MODE_7_CONTENT_OF_MESSAGE_1 0x8000U
- #define APL_FIELD_CONF_WORD_ENCRYPT_MODE_MASK 0x1F00U

- #define APL_FIELD_CONF_WORD_NO_ENCRYPTION 0x0000U
- #define APL_FIELD_CONF_WORD_DES_CBC 0x0200U
- #define APL_FIELD_CONF_WORD_DES_CBC_IV 0x0300U
- #define APL_FIELD_CONF_WORD_AES_CBC 0x0400U
- #define APL_FIELD_CONF_WORD_AES_CBC_IV 0x0500U
- #define APL_FIELD_CONF_WORD_AES_CBC_MODE_7 0x0700U
- #define APL_FIELD_CONF_WORD_AES_CBC_IV_DSMR 0x0F00U
- #define APL_DIF_DATA_FIELD_NO_DATA 0x00U
- #define APL_DIF_DATA_FIELD_8_INT 0x01U
- #define APL_DIF_DATA_FIELD_16_INT 0x02U
- #define APL_DIF_DATA_FIELD_24_INT 0x03U
- #define APL_DIF_DATA_FIELD_32_INT 0x04U
- #define APL_DIF_DATA_FIELD_32_REAL 0x05U
- #define APL_DIF_DATA_FIELD_48_INT 0x06U
- #define APL_DIF_DATA_FIELD_64_INT 0x07U
- #define APL_DIF_DATA_FIELD_SELECTION 0x08U
- #define APL_DIF_DATA_FIELD_2_BCD 0x09U
- #define APL_DIF_DATA_FIELD_4_BCD 0x0AU
- #define APL_DIF_DATA_FIELD_6_BCD 0x0BU
- #define APL_DIF_DATA_FIELD_8_BCD 0x0CU
- #define APL_DIF_DATA_FIELD_VARIABLE 0x0DU
- #define APL_DIF_DATA_FIELD_12_BCD 0x0EU
- #define APL_DIF_DATA_FIELD_SPECIAL_FUNC 0x0FU
- #define APL_DIF_DATA_FIELD_SPECIAL_FILLER 0x2FU
- #define APL_DIF_FUNC_INSTANTANEOUS 0x00U
- #define APL_DIF_FUNC_MAXIMUM 0x10U
- #define APL_DIF_FUNC_MINIMUM 0x20U
- #define APL_DIF_FUNC_ERROR 0x30U
- #define APL_DIF_LSB_STORAGE_NUMBER 0x40U
- #define APL_DIF_EXTENSION_BIT 0x80U
- #define APL_VIF_ENERGY_WH 0x00U
- #define APL_VIF_ENERGY_J 0x08U
- #define APL_VIF_VOLUME_M3 0x10U

- #define APL_VIF_MASS_KG 0x18U
- #define APL_VIF_ON_TIME 0x20U
- #define APL_VIF_OPERATING_TIME 0x24U
- #define APL_VIF_POWER_W 0x28U
- #define APL_VIF_POWER_JH 0x30U
- #define APL_VIF_VOLUME_FLOW_M3H 0x38U
- #define APL_VIF_VOLUME_FLOW_EXT_M3M 0x40U
- #define APL_VIF_VOLUME_FLOW_EXT_M3S 0x48U
- #define APL_VIF_MASS_FLOW_KGH 0x50U
- #define APL_VIF_FLOW_TEMPERATURE_C 0x58U
- #define APL_VIF_RETURN_TEMPERATURE_C 0x5CU
- #define APL_VIF_TEMPERATURE_DIF_K 0x60U
- #define APL_VIF_EXT_TEMPERATURE_C 0x64U
- #define APL_VIF_PRESSURE_BAR 0x68U
- #define APL_VIF_DATE 0x6CU
- #define APL_VIF_DATE_TIME 0x6DU
- #define APL_VIF_UNITS_FOR_HCA 0x6EU
- #define APL_VIF_AVERAGING_DURATION 0x70U
- #define APL_VIF_ACTUALITY_DURATION 0x74U
- #define APL_VIF_FABRICATION_NO 0x78U
- #define APL_VIF_ENHANCED_ID 0x79U
- #define APL_VIF_ADDRESS 0x7AU
- #define APL_VIF_FIRST_EXTENSION 0x7BU
- #define APL_VIF_SECOND_EXTENSION 0xFDU
- #define APL_VIF_MANUFACTURER 0x7FU
- #define APL_VIF_EXTENSION_BIT 0x80U
- #define APL_VIFE_SEC_KEY 0x19U
- #define APL_VIFE_TRANS_CTR 0x08U
- #define APL_VIFE_ERR_FLAG 0x17U
- #define APL_VIFE_SPEC_SUPP_INFO 0x67U
- #define APL_VIFE_MANUFR_ACCELERATION 0x01U
- #define APL_VIFE_MANUFR_ACTIVITY 0x02U
- #define APL_VIFE_MANUFR_FALL 0x03U

- #define APL_ALARM_UNSPECIFIED_BIT 1U
- #define APL_ALARM_NONE 0x00U
- #define APL_ALARM_ALL 0xFFU
- #define APL_ERROR_UNSPECIFIED 0x00U
- #define APL_ERROR_CI_UNIMPLEMENTED 0x01U
- #define APL_ERROR_BUFFER_TOO_LONG 0x02U
- #define APL_ERROR_TOO_MANY_RECORDS 0x03U
- #define APL_ERROR_PREMATURE_END_OF_RECORD 0x04U
- #define APL_ERROR_TOO_MANY_DIFES 0x05U
- #define APL_ERROR_TOO_MANY_VIFES 0x06U
- #define APL_ERROR_APPLICATION_BUSY 0x08U
- #define APL_ERROR_TOO_MANY_READOUTS 0x09U
- #define APL_ERROR_ACCESS_DENIED 0x10U
- #define APL_ERROR_CMD_UNKNOWN 0x11U
- #define APL_ERROR_PARAMETER 0x12U
- #define APL_ERROR_UNKNOWN_RECEIVER 0x13U
- #define APL_ERROR_DECRYPTION_FAILS 0x14U
- #define APL_ERROR_ENCRYPTION_NOT_SUPPORTED 0x15U
- #define APL_ERROR_SIGNATURE_NOT_SUPPORTED 0x16U
- #define APL_ERROR_DYNAMIC_APPLICATION_ERROR 0xF0U
- #define APL_ERROR_INVALID 0xFFU
- #define APL_TLG_FLAG_CLR 0x0000U

  The following flags are used within the stack to signal telegram status information to the user.

- #define APL_TLG_FLAG_FLOW_CONTROL 0x0001U
- #define APL_TLG_FLAG_ACCESS_DEMAND 0x0002U
- #define APL_TLG_FLAG_SIGNATURE 0x000CU
- #define APL_TLG_FLAG_SIGNATURE_UNKNOWN 0x0004U
- #define APL_TLG_FLAG_SIGNATURE_ERROR 0x0008U
- #define APL_TLG_FLAG_SIGNATURE_NO 0x000CU
- #define APL_TLG_FLAG_HOP_COUNTER 0x0010U
- #define APL_TLG_FLAG_REPEATER_ACCESS 0x0020U
- #define **APL_TLG_FLAG_COM_0** 0x0040U
- #define **APL_TLG_FLAG_COM_1** 0x0080U

- #define **APL_TLG_FLAG_ENCRYPTION_MODE** 0x1F00U
- #define **APL_TLG_FLAG_ENCRYPTION_MODE_5** 0x0500U
- #define **APL_TLG_FLAG_ENCRYPTION_MODE_7** 0x0700U
- #define **APL_TLG_FLAG_ENCRYPTION_MODE_13** 0x0D00U
- #define APL_TLG_FLAG_SYNC 0x2000U
- #define APL_TLG_FLAG_RX_ALWAYS_ON 0x4000U
- #define APL_TLG_FLAG_BIDIRECTIONAL 0x8000U

Enumerations

- enum E_APL_STATUS_t { E_APL_STATUS_ERROR, E_APL_STATUS_INVALID_PARAM_ERROR, E_APL_↩ STATUS_NOT_INIT_ERROR, E_APL_STATUS_SUCCESS }
- enum E_APL_RET_t { E_APL_RET_OK = 0U, E_APL_RET_ERR, E_APL_RET_NOT_READY, E_APL_RET_D↩ UPLICATED }
- enum E_APL_COM_t {
  E_APL_COM_UNKNOWN, **E_APL_COM_METER_DEFAULT_DATA**, **E_APL_COM_METER_RESERVED_↩ SIGNED**, **E_APL_COM_METER_STATIC**,
  E_APL_COM_METER_RESERVED, **E_APL_COM_OTHER_DEFAULT_COMMAND**, **E_APL_COM_OTHER↩ _RESERVED_AUTH**, **E_APL_COM_OTHER_RESERVED**,
  E_APL_COM_OTHER_RESERVED_FUTURE }

Functions

- void wmbus_apl_evt_rx (void)

  This function is called if a rf telegram is received to display a LED or a symbol on the display.
- void wmbus_apl_evt_tx (uint8_t c_tlgId)

  This function is called if a rf telegram is sent to display a LED or a symbol on the display.
- void wmbus_apl_evt_tlgAvailable (E_WMBUS_RX_t e_status, uint8_t c_tlgReqId, s_apl_tlgAttr_t *ps_tlgAttr)

  This function is called if a new telegram is available.
- E_APL_HEADER_TYPE_t wmbus_apl_evt_getCiHeader (uint8_t c_ci)

  Requests the type of header by the customer application.
- uint8_t wmbus_apl_getStatus (void)

  Returns the current status of the application layer.
- bool_t wmbus_apl_destroyTlg (uint8_t c_tlgId)

Destroys a telegram. This function may only be used in events which provide user data.

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

  Adds data bytes to the telegram.

- uint16_t wmbus_apl_readData (uint8_t c_tlgId, uint8_t *pc_data, uint16_t i_len, uint16_t i_offset)

  Reads data bytes from a telegram.

- void wmbus_apl_setDeviceAddr (s_wmbus_addr_t *ps_addr)

  Sets the own address of the device.

- void wmbus_apl_getDeviceAddr (s_wmbus_addr_t *ps_addr)

  Returns the address of the device.

- bool_t wmbus_apl_getLongHeaderAddr (uint8_t c_tlgId, s_wmbus_addr_t *ps_addr)

  Copies the long header address for the application layer of a received telegram to the specified memory.

- void wmbus_apl_setAccessNo (uint8_t c_accessNo)

  Sets the ACC counter of the device.

- uint8_t wmbus_apl_getAccessNo (void)

  Get the current ACC counter of the meter device.

- void wmbus_apl_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_apl_getTxPower (void)

  Reads the transmission power.

- E_WMBUS_SLEEP_RESULT_t wmbus_apl_sleep (E_WMBUS_SLEEP_MODE_t e_sleepMode)

  Sets the target in low power-mode.

- void wmbus_apl_bufCleanUp (void)

  Cleans up all buffers.

- bool_t wmbus_apl_getTlgAttr (uint8_t c_tlgId, s_apl_tlgAttr_t *ps_tlgAttrRecv)

  Reads the APL attributes of a received telegram.

- bool_t wmbus_apl_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_apl_getRfChannel (void)

  Reads the radio channel of the rf-module.

6.1.1 Detailed Description

Application Layer API of Wireless M-Bus.
Copyright

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

STACKFORCE Include before:

- wmbus_typedefs.h
- wmbus_api.h
- stezdn_clock.h

This application layer provides the following control information values:

- 0x50: Application reset.
- 0x70: Slave to master: Report of application errors.
- 0x72: Slave to master: 12 byte header followed by variable format data. Telegrams with long headers may be received, but not sent.
- 0x71: Report of alarms.
- 0x78: Slave to master: Variable data format respond without header. Telegrams whithout header may be received but not sent.
- 0x7A: Slave to master: 4 byte header follwoed by variable data format respond. Each outgoing data telegram uses a short header.

6.2 inc/pub/apl/wmbus_apl_col_api.h File Reference

Application Layer API of Wireless M-Bus Collector Device.

Data Structures

- struct s_apl_meterEntry_t
- struct s_apl_meterList_t
- struct s_apl_addMeterRet_t
- struct s_apl_startCollectorAttr_t

Macros

- #define **__DECL_APPLICATIONLAYER_COL_API_H__** extern
- #define APL_ERR_METER_OUT_OF_RANGE 0xFFFFU
- #define APL_ERR_DEVICE_NOT_ADDED 0xFFFF
- #define APL_CONFIGURATION_WORD_SYNC 0x20U
- #define APL_AES_SIZE_OF_KEY 0x10U

Functions

- void wmbus_apl_evt_ACCDMDReceived (uint8_t c_tlgId)

  This function is called if the receives ACC-DMD telegram from the meter.

- void wmbus_apl_evt_ACDBitSet (uint8_t c_tlgId)

  This function is called if the ACD bit of a meter telegram is set. Triggered by the ACD-bit the MUC should request alarm data (class 1) with the next unsolicited trans-mission of the Meter.

- bool_t wmbus_apl_evt_newMeter (s_wmbus_addr_t *ps_meter, s_apl_tlgAttr_t *ps_tlgAttr)

  A new meter device is requesting for connection.

- void wmbus_apl_col_init (void)

  Initializes the application layer for the device type collector.

- E_APL_STATUS_t wmbus_apl_col_start (s_apl_startCollectorAttr_t *s_startAttr)

  Starts the application layer for the device type collector.

- void wmbus_apl_col_run (void)

  Default running method of collector device application layer.

- uint8_t wmbus_apl_col_createCmdClkSync (uint16_t i_meterId)

  Creates a clock synchronization command for a meter device. Be careful: If APL_CLOCK_ENABLED is set to FALSE the command can not be created. Then always APL_ERR_TLG_NOT_AVAILABLE is returned.

- uint8_t wmbus_apl_col_createAlarmRequest (uint16_t i_meterId)

  Create an alarm request for a meter device.

- uint8_t wmbus_apl_col_createUserDataRequest (uint16_t i_meterId)

  Creates a user data request.

- uint8_t wmbus_apl_col_readError (uint8_t c_tlgId)

  Reads the optional error code of a telegram.

- bool_t wmbus_apl_col_open (void)

Start a bidirectional communication the next time as possible.

- bool_t wmbus_apl_col_sendQueued (uint8_t c_tlgId, bool_t b_append)

  Adds a telegram to the collector queue.

- uint8_t wmbus_apl_col_close (uint16_t i_meterId, bool_t b_sendNke)

  Resets the flag to start a bidirectional communication.

- s_apl_addMeterRet_t wmbus_apl_col_meterAdd (s_apl_meterEntry_t *ps_meterEntry)

  Adds a meter device to the meter list.

- E_APL_RET_t wmbus_apl_col_meterRemove (uint16_t i_meterId)

  Removes a meter device from the meter list.

- E_APL_RET_t wmbus_apl_col_meterSetRfAdapter (uint16_t i_meterId, s_wmbus_addr_t *ps_rf)

  Updates the RF adapter address of a meter device.

- bool_t wmbus_apl_col_meterGetRfAdapter (uint16_t i_meterId, s_wmbus_addr_t *ps_rf)

  Reads the RF adapter address of a meter device.

- E_APL_RET_t wmbus_apl_col_meterSetKey (uint16_t i_meterId, uint8_t *pc_key, uint8_t c_len)

  Updates the meter specific key.

- bool_t wmbus_apl_col_meterGetKey (uint16_t i_meterId, uint8_t *pc_key, uint8_t c_len)

  Reads the key of the meter.

- uint16_t wmbus_apl_col_meterGetNum (void)

  Gets the number of connected meter devices.

- bool_t wmbus_apl_col_meterGetAddr (uint16_t i_meterId, s_wmbus_addr_t *ps_meterAddr)

  Reads the attributes of a meter device.

- uint16_t wmbus_apl_col_meterGetId (s_wmbus_addr_t *ps_meterAddr)

  Reads the id of a meter device.

- void wmbus_apl_col_clearTlgQueue (void)

  This function resets the complete queue.

- E_WMBUS_COL_QUEUE_RET_t wmbus_apl_col_clearMtrTlgQueue (uint16_t i_meterId)

  Removes all telegrams from the queue for a specific meter.

- uint8_t wmbus_apl_col_createTlg (uint8_t c_ci, uint16_t i_meterId, bool_t b_encrypt)

  Creates a telegram. On collector device.

- bool_t wmbus_apl_col_sendTlg (uint8_t c_tlgId)

  Sends a telegram. Collector device. Before the telegram has to be created with function wmbus_apl_col_createTlg().

- void wmbus_apl_col_wakeUp (E_WMBUS_SLEEP_MODE_t e_sleepMode)

Wake up the target. Collector device.

- bool_t wmbus_apl_col_setContentOfMessage (E_APL_COM_t e_content, uint8_t c_tlgId)

  Set the content of message bits that are stored in the configuration word defined in the EN13757-3 2013.

- E_APL_COM_t wmbus_apl_col_getContentOfMessage (uint8_t c_tlgId)

  Get the content of message bits which are stored in the configuration word. This function should only be called for received telegrams.

6.2.1 Detailed Description

Application Layer API of Wireless M-Bus Collector Device.
Copyright

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

STACKFORCE Include before:

- wmbus_typedefs.h
  - wmbus_api.h
  - stezdn_clock.h
  - wmbus_global.h
  - wmbus_apl_api.h

6.3 inc/pub/apl/wmbus_apl_mtr_api.h File Reference

Application Layer API of Wireless M-Bus Meter Device.

Data Structures

- struct s_apl_startMeterAttr_t

Macros

- #define **__DECL_APPLICATIONLAYER_MTR_API_H__** extern
- #define APL_IGNORE_UNENCRYPTED_COLLECTOR_COMMANDS TRUE
- #define APL_INSTALL_RETRIES 6U
- #define APL_METER_SPONTANEOUS_CONF FALSE

Functions

- uint8_t wmbus_apl_evt_alarmRequested (void)

  This function is called if user data class 1 (alarm) is requested. An alarm telegram (CI=74) is created. The content of the telegram has to be written with function wmbus_apl_writeApplicationError().

- bool_t wmbus_apl_evt_userDataRequested (uint8_t c_tlgId, bool_t b_periodical)

  This function is called if user data class 2 is requested. Now meter device data is requested. The respond telegram is created automatically, the data has to be added to the telegram by function wmbus_apl_writeData().

- void wmbus_apl_mtr_init (void)

  Initializes the application layer for the device type meter.

- E_APL_STATUS_t wmbus_apl_mtr_start (s_apl_startMeterAttr_t *s_startAttr)

  Starts the application layer for the device type meter.

- void wmbus_apl_mtr_run (void)

  Default running method of meter device application layer.

- uint8_t wmbus_apl_mtr_getAlarmCode (void)

  Gets the alarm code from APL.

- void wmbus_apl_mtr_setAlarmCode (uint8_t c_alarmCode)

  Sets the alarm code in APL.

- void wmbus_apl_mtr_clrAlarmCode (uint8_t c_alarmCode)

  Clears the given alarm code in APL.

- uint8_t wmbus_apl_mtr_sendInstallationRequest (void)

  Creates an installation request.

- bool_t wmbus_apl_mtr_setLongHeaderAddr (uint8_t c_tlgId, s_wmbus_addr_t *ps_addr)

  Sets the long header address for the application layer of an outgoing telegram. By default, this is identical to the own address, but this function call permits to enter the address of e.g. a connected meter Before using this function, create the telegram, and pass the ID here.

- bool_t wmbus_apl_mtr_setErrorFlag (uint8_t c_error)

  Sets the error flag (aka. status byte) of the APL header of the meter telegram. The meter will afterwards send its telegram with the Status field according to value that has been set from this function. To clear status byte, call function wmbus_apl_mtr_clrErrorFlag().

- void wmbus_apl_mtr_clrErrorFlag (uint8_t c_error)

  Allowes to clear application error flags. The application error flags are never cleared automatically by the application layer.

- uint8_t wmbus_apl_mtr_getErrorFlag (void)

  Reads if there is a set application error.

- bool_t wmbus_apl_mtr_setKey (uint8_t *pc_key, uint8_t c_len)

  Writes the encryption key.

- bool_t wmbus_apl_mtr_getKey (uint8_t *pc_key, uint8_t c_len)

  Reads the encryption key.

- void wmbus_apl_mtr_setCollectorAddress (s_wmbus_addr_t *ps_collectorAddr)

  Sets the address of the connected collector.

- void wmbus_apl_mtr_getCollectorAddress (s_wmbus_addr_t *ps_collectorAddr)

  Gets the address of the connected collector.

- void wmbus_apl_mtr_setInterval (uint32_t l_interval)

  Sets the periodical interval of the meter device.

- uint32_t wmbus_apl_mtr_getInterval (void)

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

- void wmbus_apl_mtr_setConnected (bool_t b_connect)

  Sets the meter as connected or disconnect.

- bool_t wmbus_apl_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.

- E_WMBUS_RESPONSE_DELAY_t wmbus_apl_mtr_getResponseDelay (void)

  Returns the response delay currently used by the meter device.

- uint8_t wmbus_apl_mtr_createTlg (uint8_t c_ci, uint16_t i_meterId, bool_t b_encrypt, bool_t b_synchronous)

  Creates a telegram. On meter device.

- bool_t wmbus_apl_mtr_sendTlg (uint8_t c_tlgId)

  Sends a telegram. Meter device. Before the telegram has to be created with function wmbus_apl_mtr_createTlg().

- void wmbus_apl_mtr_wakeUp (E_WMBUS_SLEEP_MODE_t e_sleepMode)

  Wake up the target. Meter device.

- bool_t wmbus_apl_mtr_setContentOfMessage (E_APL_COM_t e_content, uint8_t c_tlgId)

  Set the content of message bits that are stored in the configuration word defined in the EN13757-3 2013.

- E_APL_COM_t wmbus_apl_mtr_getContentOfMessage (uint8_t c_tlgId)

  Get the content of message bits which are stored in the configuration word. This function should only be called for received telegrams.

6.3.1 Detailed Description

Application Layer API of Wireless M-Bus Meter Device.

Copyright

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

Author

STACKFORCE Include before:

- wmbus_typedefs.h
- wmbus_api.h
- stezdn_clock.h
- wmbus_global.h
- wmbus_apl_api.h

6.4 inc/pub/utils/wmbus_clock_api.h File Reference

Wireless M-Bus clock module Application Programming Interface.

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

Data Structures

- struct s_clock_t

Macros

- #define **__DECL_WMBUS_CLOCK_H__** extern

Enumerations

- enum E_CLOCK_t {
  E_CLOCK_SECONDS, **E_CLOCK_MINUTES**, **E_CLOCK_HOURS**, **E_CLOCK_DAYS**,
  E_CLOCK_MONTHS, **E_CLOCK_YEARS** }

Functions

- void wmbus_clock_init (s_clock_t *ps_clock)

Initializes the clock.

- bool_t wmbus_clock_run (s_clock_t *ps_clock)

  Runs a clock.

- bool_t wmbus_clock_isLeap (uint16_t i_year)
- void wmbus_clock_start (s_clock_t *ps_clock)

  Starts the clock.

- void wmbus_clock_stop (s_clock_t *ps_clock)

  Stops the clock.

- void wmbus_clock_inc (s_clock_t *ps_clock, E_CLOCK_t e_element, bool_t b_single)

  Increments an element of the clock.

- void wmbus_clock_dec (s_clock_t *ps_clock, E_CLOCK_t e_element, bool_t b_single)

  Decrements an element of the clock.

- uint32_t wmbus_clock_calcMinutes (s_clock_t *ps_clock)

  Calc the minutes since the clock is running. (Max. 64 Years). This function is needed for meter devices which use the CI-field 0x8D.

6.4.1 Detailed Description

Wireless M-Bus clock module Application Programming Interface.
Copyright

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

STACKFORCE Include before:

- wmbus_typedefs.h

Chapter 7

Example Documentation

7.1 main_collector.c

Collector application layer example.

```
/*============================================================================
                            INCLUDE FILES
============================================================================*/
#include "inc\pub\utils\wmbus_typedefs.h"
#include "inc\pub\utils\wmbus_api.h"
#include "inc\prv\cfg\wmbus_config.h"
#include "inc\pub\utils\wmbus_clock_api.h"
#include "inc\pub\hal\wmbus_hal.h"
/* Include common APL API */
#include "inc\pub\apl\wmbus_apl_api.h"
/* Include meter device specific APL API functions */
#include "inc\pub\apl\wmbus_apl_col_api.h"

/*============================================================================
                            DEFINES
============================================================================*/
#ifndef WMBUS_CFG_DEVICE
#error Please define the device configuration to a COLLECTOR device!
#elif WMBUS_CFG_DEVICE != WMBUS_CFG_DEVICE_COLLECTOR
#error Please define the device configuration to a COLLECTOR device!
#endif /* WMBUS_CFG_DEVICE */

#define DO_CLK_SYNC_TEST        TRUE

#if DO_CLK_SYNC_TEST
#define TLG_BEFORE_CLK_SYNC    5
#endif /* DO_CLK_SYNC_TEST */

#ifndef CUSTOMER_FREQ_OFFSET
#define CUSTOMER_FREQ_OFFSET 0x00
#endif /* CUSTOMER_FREQ_OFFSET */
/*============================================================================
                            ENUMS
============================================================================*/

/*============================================================================
                  data storage in RAM/FLASH
============================================================================*/
/*  for efficient memory usage, entry sizes of 16/32/64/128 bytes would be positive */
```

```
#define APP_FLASHPAGE_SIZE    512    /* page size in FLASH */
#define APP_ENTRY_SIZE        64     /* just an example */
#define APP_ENTRIES_PER_PAGE  (APP_FLASHPAGE_SIZE/APP_ENTRY_SIZE)
#define APP_METADATA_SIZE     4      /* 2 bytes meter ID, 1 byte RSSI, 1 byte */
#define APP_DATA_SIZE         (APP_ENTRY_SIZE - APP_METADATA_SIZE)

/* Start address fort the array in FLASH*/
#define APP_DATA_BASE_ADDRESS 0xB600
/* Upper limit: page containing the interrupt vectors */
#define MCU_VECTOR_PAGE_ADDRESS    0xFE00
/* number of pages before stop/roll-over --> to be decided */
/* in page 32, the interrupt vectors reside.*/
#define APP_DATA_PAGE_COUNT   (MCU_VECTOR_PAGE_ADDRESS - APP_DATA_BASE_ADDRESS)/APP_FLASHPAGE_SIZE

typedef struct S_APP_TLG_ENTRY_T
{
  uint16_t i_meterId;
  uint8_t c_quality;
  uint8_t c_accNo;
  uint8_t c_data[APP_DATA_SIZE];
} s_app_tlgEntry_t;

/* array in RAM to collect telegrams before writing to FLASH */
s_app_tlgEntry_t gs_dataArray[APP_ENTRIES_PER_PAGE];
/* next page to write */
uint8_t gc_nextFlashPage;
/* check if RAM array needs to be written to FLASH */
bool_t gb_tlgArrayComplete;
/* next array entry to write to RAM array */
uint8_t gc_nextRamEntry;

/*============================================================================
                            VARIABLES
============================================================================*/
/* Example collector address. */
s_wmbus_addr_t gs_collector = {{0xce,0x9a},   /* Manufacturer (here STZ) */
                    {0x80,0x00,0x00,0x02},    /* ident number            */
                     0x23,                    /* version                 */
                    WMBUS_DEV_TYPE_OTHER};    /* type, here other        */

/* Example meter entry for the meterlist of the collector. */
s_apl_meterEntry_t gs_meterEntry = {
                /* WMBus meter address */
                {{0xce,0x9a},{0x80,0x00,0x00,0x01},0x23,WMBUS_DEV_TYPE_WATER},
                /* WMBus mode of the meter */
                E_WMBUS_MODE_UNKNOWN,
                /* WMBus RF adapter address of the meter (unused) */
                {{0x0,0x0},{0x0,0x0,0x0,0x0},0x0,0x0},
                /* WMBus meter key */
                {0x00,0x11,0x22,0x33,0x44,0x55,0x66,0x77,
                 0x88,0x99,0xAA,0xBB,0xCC,0xDD,0xEE,0xFF}};

/* Example meterlist */
s_apl_meterList_t gs_meterList = {0x0001U, &gs_meterEntry};

s_apl_startCollectorAttr_t gs_startAttr =
```

```
{
  /* Init start strucutre */
  /* Frequency offset for the carrier. */
  CUSTOMER_FREQ_OFFSET,
  /* Device address. */
  &gs_collector,
  &gs_meterList,
};

#if DO_CLK_SYNC_TEST
/* Counter for received telegrams */
uint8_t gc_noOfReceivedTlg = 0x0U;

/* Requested telegram id REQ-UD */
uint8_t gc_tlgIdReqUD = APL_ERR_TLG_NOT_AVAILABLE;
/* Requested telegram id CLK-SYNC */
uint8_t gc_tlgIdClkSync = APL_ERR_TLG_NOT_AVAILABLE;
#endif /* DO_CLK_SYNC_TEST */

/*============================================================================
                            FUNCTION PROTOTYPES
============================================================================*/
#if DO_CLK_SYNC_TEST

static void loc_clockSyncTest(uint8_t c_tlgReqId);
#endif /* DO_CLK_SYNC_TEST */

/*============================================================================
                            FUNCTIONS
============================================================================*/

/*============================================================================*/
/*============================================================================*/
void main(void)
{
  /* Initialises the hal. */
  if(wmbus_hal_init() == E_HAL_STATUS_SUCCESS)
  {
    /* initialize APL */
    wmbus_apl_col_init();

    /* start the APL for device type collector */
    wmbus_apl_col_start(&gs_startAttr);

    /* Set the channel which should be used (for mode N-Devices only) */
    #if WMBUS_CHECK_MODE_or(WMBUS_MODE_N1 | WMBUS_MODE_N2)
    wmbus_apl_setRfChannel(E_RF_CFG_CHAN_169_1A);
    #endif /* WMBUS_CHECK_MODE_or(WMBUS_MODE_N1 | WMBUS_MODE_N2) */

    while(TRUE)
    {
      /* run the application layer */
      wmbus_apl_col_run();
    } /* while */
  } /* if */
} /* main() */
```

```
/*============================================================================*/
/*============================================================================*/
void wmbus_apl_evt_rx(void)
{
  /*
   * This function is used in particular if device is configured as: COLLECTOR
   * This event is called whenever a radio telegram is received (device
   * independent). You can use it to toggle LEDs or count received telegrams.
   * Function can be disabled using APL_EVT_RX_ENABLED in "wmbus_global.h".
   */

} /* wmbus_apl_evt_rx() */

/*============================================================================*/
/*============================================================================*/
void wmbus_apl_evt_tx(uint8_t c_tlgId)
{
  /*
   * This function is used in particular if device is configured as: METER
   * This event is called whenever a radio telegram is sent (device
   * independent). You can use it to toggle LEDs or count transmitted telegrams.
   * Function can be disabled using APL_EVT_TX_ENABLED in "wmbus_global.h".
   */

} /* wmbus_apl_evt_tx() */

/*============================================================================*/
/*============================================================================*/
E_APL_HEADER_TYPE_t wmbus_apl_evt_getCiHeader(uint8_t c_ci)
{
  /*
   * This function is used if the stack receives an unknown CI field (e.g. for
   * customer specific CI fields). In order to parse the header properly
   * the stack needs to distinguish between:
   *   - No Header
   *   - Short Header
   *   - Long Header
   *   - Invalid Header
   */
  E_APL_HEADER_TYPE_t e_return;

  switch(c_ci)
  {
    case 0xA1U:   /* STACKFORCE specific: Transmit string with no header. */
      e_return = E_APL_HEADER_TYPE_NO;
      break;
    case 0xA2U:   /* STACKFORCE specific: Transmit string with short header. */
      e_return = E_APL_HEADER_TYPE_SHORT;
      break;
    case 0xA3U:   /* STACKFORCE specific: Transmit string with long header. */
      e_return = E_APL_HEADER_TYPE_LONG;
      break;
    case 0xA4U:   /* STACKFORCE specific: Transmit string with short header. */
      e_return = E_APL_HEADER_TYPE_SHORT;
      break;
    default:      /* The application does not know the CI field. */
      e_return = E_APL_HEADER_TYPE_INVALID;
```

```
      break;

      /* please insert specific CI fields here */

  } /* switch(c_ci) */

  return e_return;
} /* wmbus_apl_evt_getCiHeader() */

/*============================================================================*/
/*============================================================================*/
void wmbus_apl_evt_tlgAvailable(E_WMBUS_RX_t e_status, uint8_t c_tlgReqId,
                          s_apl_tlgAttr_t *ps_tlgAttr)
{
  /* Current offset. */
  uint16_t i_offset = 0;
  /* current telegram buffer if data should be stored locally */
  uint8_t pc_dataBuf[25];

  /* check for the current status. */
  switch(e_status)
  {
    case E_WMBUS_RX_TLG_AVAILABLE:
    {
      /* new telegram available and the data may be read */
      switch(ps_tlgAttr->c_controlInfo)
      {
        case APL_FIELD_CI_HEADER_LONG:
        case APL_FIELD_CI_HEADER_SHORT:
        case APL_FIELD_CI_LINK_FROM_DEVICE_SHORT:
        {
          /* Read the whole telegram to local buffer */
          wmbus_apl_readData(ps_tlgAttr->c_tlgId, pc_dataBuf, ps_tlgAttr->
      i_dataLen, i_offset);

          /*
           * 11 bytes of data received with record 0 and record 1 in reverse order:
           * pc_dataBuf[18]...[11]: record 0 (default: timestamp)
           * pc_dataBuf[10]...[5] : record 1 (water volume)
           * pc_dataBuf[4]...[0] : record 2 (water volume)
           *
           * Record 0 (timestamp):
           * pc_dataBuf[18] is DIB: 0x6 = 48Bit Integer, Instantanous
           * pc_dataBuf[17] is VIB: 0x6D = special value used for timestamp
           * pc_dataBuf[16]...[11] is data: 48bit timestamp value
           *
           * Record 1 (water volume):
           * pc_dataBuf[10] is DIB: 0x0C = 8 digit BCD, Function=Instantaneous
           * pc_dataBuf[9] is VIB: 0x13 = Unit= Volume(m^3), Multiplier: 1m^3
           * pc_dataBuf[8] is data: 0x27 = Value LSB
           * pc_dataBuf[7] is DIB: 0x04 = Value: (= 2850427)
           * pc_dataBuf[6] is VIB: 0x85 = Value: (= 2850427)
           * pc_dataBuf[5] is data: 0x02 = Value MSB
           *
           * Record 2 (water volume):
           * pc_dataBuf[4] is DIB: 0x0B = 6 digit BCD, Function=Instantaneous
           * pc_dataBuf[3] is VIB: 0x3B = Unit=  Volume Flow(m^3/h), Multiplier= 1dm^3/h
```

```
           * pc_dataBuf[2] is DIB: 0x27 = Value LSB
           * pc_dataBuf[1] is VIB: 0x01 = Value   (= 127)
           * pc_dataBuf[0] is data: 0x00 = Value MSB
           */

          /*
           * Extract records here
           */

          #if DO_CLK_SYNC_TEST
          loc_clockSyncTest(c_tlgReqId);
          #endif /* DO_CLK_SYNC_TEST */
          break;
        }

        default: break;
     } /* switch(c_controlInfo) */

      break;
    } /* case E_WMBUS_RX_TLG_AVAILABLE */


    case E_WMBUS_RX_SIGNATURE_ERROR:
      /* signature was wrong */
      break;

    case E_WMBUS_RX_SIGNATURE_UNKNOWN:
      if(ps_tlgAttr != NULL)
      {
        /* The telegram is encrypted and can not be decrypted. */
      }
      break;

    case E_WMBUS_RX_REQUEST_TIMEOUT:
      /* A timeout occured while sending a request. The request telegram must be
         destroyed */
      wmbus_apl_destroyTlg(c_tlgReqId);
      break;

    default:
      break;

  } /* switch(e_status) */

  /* delete telegram after handling it */
  wmbus_apl_destroyTlg(ps_tlgAttr->c_tlgId);

} /* wmbus_apl_evt_tlgAvailable() */

/*============================================================================*/
/*============================================================================*/
bool_t wmbus_apl_evt_newMeter(s_wmbus_addr_t *ps_meter,
      s_apl_tlgAttr_t *ps_tlgAttr)
{
  /*
   * This function is used if device is configured as: COLLECTOR
   * Since we added the meter device for this demo manually to the collector and
```

```
   * we don't want any other meters in our list we don't add new meters here
   * by returning FALSE.
   */
  return FALSE;
} /* wmbus_apl_evt_newMeter() */

/*============================================================================*/
/*============================================================================*/
void wmbus_apl_evt_ACCDMDReceived(uint8_t c_tlgId)
{
  /*
   * This function is used if device is configured as: COLLECTOR
   * The event is triggered when the collector receives an ACC-DMD telegram
   * from connected meter. According to the EN-13757 and OMS, the collector
   * should perform REQ-UD and REQ-UD2. For DSMR v4.0.5, the collector shall
     do nothing.
   */
  return;
} /* wmbus_apl_evt_ACCDMDReceived() */

#if (DSMR_V405_ENABLED || DSMR_V22_PLUS_ENABLED)
/*============================================================================*/
/*============================================================================*/
void wmbus_apl_evt_keyExchangeCmdAcked_DSMR(void)
{
  /*
   * This function is used if device is configured as: COLLECTOR
   * The event is triggered when the meter acknowledes the DSMR key exchange
   * command. This is to let the collector know that the command is transmitted
   * successfully and the collector can start to use the new key.
   */
} /* wmbus_apl_evt_keyExchangeCmdAcked_DSMR() */
#endif /* DSMR_V405_ENABLED || DSMR_V22_PLUS_ENABLED */

#if DO_CLK_SYNC_TEST
/*============================================================================*/
/*============================================================================*/
static void loc_clockSyncTest(uint8_t c_tlgReqId)
{
  /* Error Flag */
  bool_t b_errorFlag = FALSE;

  /* Increment the number of received telegrams */
  gc_noOfReceivedTlg++;

  /* Start clock synchronization every received x telegrams */
  if(gc_noOfReceivedTlg == TLG_BEFORE_CLK_SYNC)
  {
    if(wmbus_apl_col_open() == TRUE)
    {
      /* Request User Data first */
      gc_tlgIdReqUD = wmbus_apl_col_createUserDataRequest(0x0U); /*
       Meter entry 0 */
      if(gc_tlgIdReqUD != APL_ERR_TLG_NOT_AVAILABLE)
        b_errorFlag = wmbus_apl_col_sendQueued(gc_tlgIdReqUD, TRUE);
      if(!b_errorFlag || gc_tlgIdReqUD == APL_ERR_TLG_NOT_AVAILABLE)
      {
```

```
        /* The telegram could not be sent */
        /* Reset parameters */
        gc_noOfReceivedTlg = 0;
        wmbus_apl_destroyTlg(gc_tlgIdReqUD);
        gc_tlgIdReqUD = APL_ERR_TLG_NOT_AVAILABLE;
      }/* if */
    }/* if */
  }
  else if(c_tlgReqId != APL_ERR_TLG_NOT_AVAILABLE)
  {
    if (c_tlgReqId == gc_tlgIdReqUD)
    {
       /* If the REQ-UD is successful then begin the actual clock sync */
      gc_tlgIdClkSync = wmbus_apl_col_createCmdClkSync(0x0U); /* Meter entry
       0 */

      if(gc_tlgIdClkSync != APL_ERR_TLG_NOT_AVAILABLE)
        b_errorFlag = wmbus_apl_col_sendQueued(gc_tlgIdClkSync, TRUE);
      if(!b_errorFlag || gc_tlgIdReqUD == APL_ERR_TLG_NOT_AVAILABLE)
      {
        /* The telegram could not be sent */
        /* Reset parameters */
        gc_noOfReceivedTlg = 0;
        wmbus_apl_destroyTlg(gc_tlgIdClkSync);
        gc_tlgIdClkSync = APL_ERR_TLG_NOT_AVAILABLE;
      }

      /* Resets the telegram ID */
      gc_tlgIdReqUD = APL_ERR_TLG_NOT_AVAILABLE;
    }
    else if(c_tlgReqId == gc_tlgIdClkSync)
    {
      /* Close the bidirectional communication */
      wmbus_apl_col_close(0x0U, TRUE);

      /* Reset parameters */
      gc_noOfReceivedTlg = 0;
      b_errorFlag = FALSE;
      gc_tlgIdClkSync = APL_ERR_TLG_NOT_AVAILABLE;
    }
  }
  else if(gc_noOfReceivedTlg >= 10)
  {
    /* Reset parameters */
    gc_noOfReceivedTlg = 0;
  }

  return;
} /* loc_clockSyncTest */
#endif /* DO_CLK_SYNC_TEST */

/*============================================================================*/
/*============================================================================*/
void wmbus_apl_evt_ACDBitSet(uint8_t c_tlgId)
{
  /* This function is called if the ACD bit of a meter telegram is set.
     Triggered by the ACD-bit the MUC should request alarm data (class 1)
```

```
     with the next unsolicited trans-mission of the Meter.*/
  return;
} /* wmbus_apl_evt_ACDBitSet() */
```

7.2 main_meter.c

Meter application layer example.

```
/*============================================================================
                            INCLUDE FILES
============================================================================*/
#include "inc\pub\utils\wmbus_typedefs.h"
#include "inc\pub\utils\wmbus_api.h"
#include "inc\prv\cfg\wmbus_config.h"
#include "inc\pub\utils\wmbus_clock_api.h"
#include "inc\pub\hal\wmbus_hal.h"
/* Include common APL API */
#include "inc\pub\apl\wmbus_apl_api.h"
/* Include meter device specific APL API functions */
#include "inc\pub\apl\wmbus_apl_mtr_api.h"


/*============================================================================
                            DEFINES
============================================================================*/
#ifndef WMBUS_CFG_DEVICE
#error Please define the device configuration to a METER device!
#elif WMBUS_CFG_DEVICE != WMBUS_CFG_DEVICE_METER
#error Please define the device configuration to a METER device!
#endif /* WMBUS_CFG_DEVICE */

#ifndef CUSTOMER_FREQ_OFFSET
#define CUSTOMER_FREQ_OFFSET 0x00
#endif /* CUSTOMER_FREQ_OFFSET */

/*============================================================================
                            VARIABLES
============================================================================*/
/* Example meter address (own address) */
s_wmbus_addr_t gs_addr = {{0xce,0x9a},        /* Manufacturer (here STZ) */
                    {0x80,0x00,0x00,0x01},    /* ident number            */
                    0x23,                     /* version                 */
                    WMBUS_DEV_TYPE_WATER};    /* type, here water        */

/* Example collector address */
s_wmbus_addr_t gs_collector = {{0xce,0x9a},   /* Manufacturer (here STZ) */
                    {0x80,0x00,0x00,0x02},    /* ident number            */
                     0x23,                    /* version                 */
                    WMBUS_DEV_TYPE_OTHER};    /* type, here other        */

/* Example meter key (own key)*/
uint8_t gpc_key[] = {0x00,0x11,0x22,0x33,0x44,0x55,0x66,0x77,
                    0x88,0x99,0xAA,0xBB,0xCC,0xDD,0xEE,0xFF};
```

```
/* Init start strucutre for meter device */
s_apl_startMeterAttr_t gs_startAttr =
{
  /* Frequency offset for the carrier. */
  CUSTOMER_FREQ_OFFSET,
  /* Device address */
  &gs_addr,
  /* Encryption key */
  gpc_key,
  /* Collector address. */
  &gs_collector,
  /* Set the device to connected. Only if the device is a meter device.
     Otherwise this field is ignored. */
  TRUE,
  /* Periodical interval for sending data. Only if the device is a meter
     device. [ms] */
  3000U
};

/*============================================================================
                            FUNCTIONS
============================================================================*/

/*============================================================================*/
/*============================================================================*/
void main(void)
{
  /* Initialises the hal. */
  if(wmbus_hal_init() == E_HAL_STATUS_SUCCESS)
  {
    /* initialize APL */
    wmbus_apl_mtr_init();

    /* start the APL for device type meter */
    wmbus_apl_mtr_start(&gs_startAttr);

    /* Set the channel which should be used (for mode N-Devices only) */
    #if WMBUS_CHECK_MODE_or(WMBUS_MODE_N1 | WMBUS_MODE_N2)
    wmbus_apl_setRfChannel(E_RF_CFG_CHAN_169_1A);
    #endif /* WMBUS_CHECK_MODE_or(WMBUS_MODE_N1 | WMBUS_MODE_N2) */

    while(TRUE)
    {
      /* run the application layer */
      wmbus_apl_mtr_run();
    } /* while */
  }/* if */
} /* main() */

/*============================================================================*/
/*============================================================================*/
void wmbus_apl_evt_rx(void)
{
  /*
   * This function is used in particular if device is configured as: COLLECTOR
   * This event is called whenever a radio telegram is received (device
   * independent). You can use it to toggle LEDs or count received telegrams.
```

```
   * Function can be disabled using APL_EVT_RX_ENABLED in "wmbus_global.h".
   */

} /* wmbus_apl_evt_rx() */

/*============================================================================*/
/*============================================================================*/
void wmbus_apl_evt_tx(uint8_t c_tlgId)
{
  /*
   * This function is used in particular if device is configured as: METER
   * This event is called whenever a radio telegram is sent (device
   * independent). You can use it to toggle LEDs or count transmitted telegrams.
   * Function can be disabled using APL_EVT_TX_ENABLED in "wmbus_global.h".
   */

} /* wmbus_apl_evt_tx() */

/*============================================================================*/
/*============================================================================*/
bool_t wmbus_apl_evt_userDataRequested(uint8_t c_tlgId, bool_t b_periodical)
{
  /*
   * This function is used if device is configured as: METER
   * The periodical data includes the time. Further data can be added here
   * to the telegram:
   * Record 0: time information (already set automatically by the stack!)
   * Record 1: our example data (pc_staticData[])
   */
  uint8_t pc_staticData[]={ 0x0C,  /* Record 1: DIB: Data=8 digit BCD, Function=Instantaneous*/
                            0x13,  /* Record 1: VIB: Unit= Volume(m^3), Multiplier= 1dm^3 */
                            0x27,  /* Record 1: Value LSB */
                            0x04,  /* Record 1: Value: (= 2850427) */
                            0x85,  /* Record 1: Value: (= 2850427) */
                            0x02,  /* Record 1: Value MSB */
                            0x0B,  /* Record 2: DIB: Data=6 digit BCD, Function=Instantaneous*/
                            0x3B,  /* Record 2: VIB: Unit=  Volume Flow(m^3/h), Multiplier= 1dm^3/h */
                            0x27,  /* Record 2: Value LSB */
                            0x01,  /* Record 2: Value   (= 127) */
                            0x00}; /* Record 2: Value MSB */

  /* Write the data to the telegram */
  wmbus_apl_writeData(c_tlgId, pc_staticData, sizeof(pc_staticData), FALSE);

  /* Tell the stack to send out the telegram by returning TRUE. If set to FALSE
   * the stack will ignore the request and no telegram is sent. */
  return TRUE;
} /* wmbus_apl_evt_userDataRequested() */

/*============================================================================*/
/*============================================================================*/
E_APL_HEADER_TYPE_t wmbus_apl_evt_getCiHeader(uint8_t c_ci)
{
  /*
   * This function is used if the stack receives an unknown CI field (e.g. for
   * customer specific CI fields). In order to parse the header properly
   * the stack needs to distinguish between:
```

```
   *   - No Header
   *   - Short Header
   *   - Long Header
   *   - Invalid Header
   */
  E_APL_HEADER_TYPE_t e_return = E_APL_HEADER_TYPE_INVALID;

  switch(c_ci)
  {
    case 0xA1U:   /* STACKFORCE specific: Transmit string with no header. */
      e_return = E_APL_HEADER_TYPE_NO;
      break;
    case 0xA2U:   /* STACKFORCE specific: Transmit string with short header. */
      e_return = E_APL_HEADER_TYPE_SHORT;
      break;
    case 0xA3U:   /* STACKFORCE specific: Transmit string with long header. */
      e_return = E_APL_HEADER_TYPE_LONG;
      break;
    case 0xA4U:   /* STACKFORCE specific: Transmit string with short header. */
      e_return = E_APL_HEADER_TYPE_SHORT;
      break;
    default:      /* The application does not know the CI field. */
      e_return = E_APL_HEADER_TYPE_INVALID;
      break;

      /* please insert specific CI fields here */

  } /* switch(c_ci) */

  return e_return;
} /* wmbus_apl_evt_getCiHeader() */

/*============================================================================*/
/*============================================================================*/
void wmbus_apl_evt_tlgAvailable(E_WMBUS_RX_t e_status, uint8_t c_tlgReqId,
                          s_apl_tlgAttr_t *ps_tlgAttr)
{
  /*
   * This function is used to signal the event that a telegram was received
   * successfully. The application is now able to handle the telegram.
   */

  /* check for the current status. */
  switch(e_status)
  {
    case E_WMBUS_RX_REQUEST_TIMEOUT:
      /* A timeout occured while sending a request. The request telegram must be
         destroyed */
      wmbus_apl_destroyTlg(c_tlgReqId);
      break;
  }/* switch */
  /* delete telegram after handling it */
  wmbus_apl_destroyTlg(ps_tlgAttr->c_tlgId);
} /* wmbus_apl_evt_tlgAvailable() */

/*============================================================================*/
/*============================================================================*/
```

```
uint8_t wmbus_apl_evt_alarmRequested(void)
{
  uint8_t c_alarm = APL_ALARM_NONE;

  c_alarm = wmbus_apl_mtr_getAlarmCode();
  wmbus_apl_mtr_clrErrorFlag(APL_FIELD_STATUS_ALARM);
  wmbus_apl_mtr_clrAlarmCode(APL_ALARM_ALL);

  return c_alarm;
} /* wmbus_apl_evt_alarmRequested() */
```

Chapter 8

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.