WIRELESS M-BUS STACK PROGRAMMER'S GUIDE

1. Overview

This specification describes in detail the Application Programmer Interface (API) for the Silicon Labs Wireless M-Bus stack. The Silicon Labs Wireless M-Bus stack uses a Silicon Labs C8051 MCU and EZRadioPRO®. Wireless M-Bus is a European standard for meter reading applications using the 868 MHz frequency band.

1.1. Stack Layers

Wireless M-Bus uses the 3-layer IEC model, which is a subset of the 7-layer OSI model. See Figure 1.

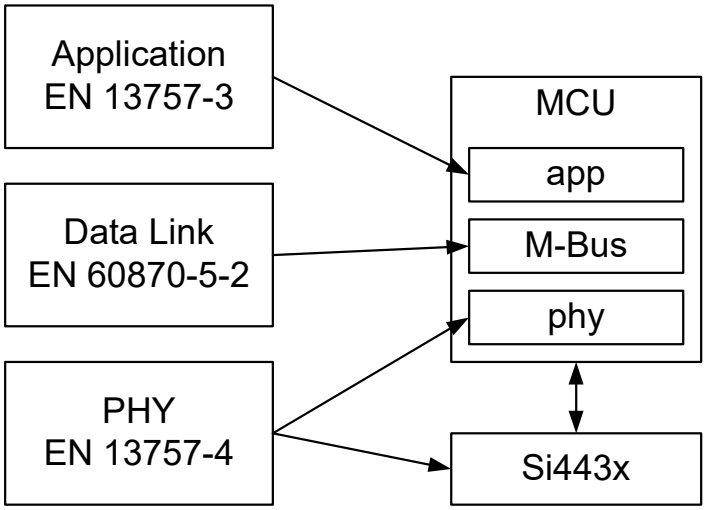


Figure 1. Stack Layers

The Physical Layer (PHY) is defined in EN 13757-4. The Physical Layer defines how the bits are encoded and transmitted, the RF modem characteristics (chip rate, preamble, and synchronization word), and RF parameters (modulation, center frequency, and frequency deviation).

The PHY layer is implemented using a combination of hardware and firmware. The EZRadioPRO® performs all of the RF and modem functions. The EZRadioPRO is used in FIFO mode with the packet handler disabled. The MbusPhy.c module provides SPI interface, encoding/decoding, block read/write, and packet handling and manages the transceiver states.

The M-Bus Data Link Layer is implemented in the MbusLink.c module. The M-Bus Application Programming interface consists of public functions that may be called from the Application Layer in the main thread. The MbusLink module also implements the Data Link Layer. The Data Link Layer will format and copy data from the application TX buffer to the MbusPhy TX buffer, adding the required headers and CRCs.

The Application layer itself is not part of the M-Bus firmware. The Application Layer defines how a wide variety of data is to be formatted for transmission. Most meters only need to transmit one or two types of data. Adding a large amount of code to accommodate any kind of data to the meter would add unnecessary code and cost to the meter. If might be feasible to implement a library or a header file with an exhaustive list of data types. However, most metering customers know exactly what kind of data they need to transmit and can refer to the standard for formatting details. A universal reader or sniffer might implement a complete set of application data types on the PC GUI. For these reasons, the Application Layer is implemented using example applications for a meter and reader.

1.2. Required Standards

1.2.1. EN 13757-4

EN 13757-4

Communication system 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

1.2.2. EN 13757-3

Communication system for meters and remote reading of meters

Part 3: Dedicated Application Layer

1.2.3. IEC 60870-2-1:1992

Telecontrol equipment and systems

Part 5: Transmission protocols

Section 1:Link transmission procedure

1.2.4. IEC 60870-1-1:1990

Telecontrol equipment and systems

Part 5: Transmission protocols

Section 1: Transmission frame formats

1.3. Definitions

- **M-Bus**—M-Bus is a wired standard for meter reading in Europe.
- **Wireless M-Bus**—Wireless M-Bus for meter reading applications in Europe.
- **PHY**—Physical Layer defines how data bits and bytes are encoded and transmitted.
- **API**—Application Programmer interface.
- **LINK**—Data Link Layer- defines how blocks and frames are transmitted.
- **CRC**—Cyclic Redundancy Check.
- **FSK**—Frequency Shift Keying.
- **Chip**—Smallest unit of transmitted data. One data bit is encoded as multiple chips.
- **Module**—A C code source .c file.

1.4. Coding Conventions

The following conventions were adopted for this project:

- The compiler_defs.h file is used for cross compiler support.
- Memory specific pointers are used for buffers and module registers.
- Each module has a short descriptive file name.
  - MbusLink.c
  - MbusPhy.c
- Public functions include a short module prefix:
  - MbusLink—mbus API
  - MbusPhy—PHY module
- Public variables also include the same module prefix, starting with a capital letter, since they are globals.
  - MbusLink—mbus API
  - MbusPhy—PHY module

- Each module includes an API header file with public function prototypes.
  - MbusLink.h
  - MbusPhy.h
- Module public bits and variables are declared external in the module header file.
- Public variables are bit or byte only, or qualified using a binary semaphore.
- Public variables are used sparingly.
- No module includes external variables from another module
- Public variable declarations and internal prototypes are located in the module source file.
- Hardware-specific macros and bit definitions are located in a hardware_defs.h header file.
- Compile time build options, memory specifiers macros, and buffer size macros are located in the module _defs.h files.
- Function parameters passed as inputs may be indicated using __in, and parameters passed as outputs may be indicated using __out.

2. PHY API

2.1. PHY Layer Modules and File Organization

Table 1. PHY Layer Modules and File Organization

File Name	Description	Comments
MbusPhy.h	Physical Layer header file	
MbusPhy.c	Physical Layer c file	
MbusPhy_const.h	Physical Layer constants header file	
MbusPhy_const.c	Physical Layer constants c file	Code constants used for Radio configuration
hardware_defs.h	Hardware definitions file	hardware specific macros
MbusPhy_def.h	M-Bus definitions file	M-Bus build and configuration options
si443x_B1.h	Si4432 header file	
si_toolchain.h	Compiler definitions file	cross compiler macros
C8051F930_defs.h	F930 header file	

2.2. Functions

2.2.1. MbusPhyInitRegisters

Prototype:

PHY_STATUS MbusPhyInitRegisters(void);

Parameters:

none

Returns:

PHY_STATUS (uint8_t) status:

PHY_STATUS_SUCCESS

PHY_STATUS_ERROR_INVALID_STATE

Description:

The PhyInitRegisters() function should be called after resetting the MCU, from the main() function, before using any other PHY functions. This function will initialize the internal PHY registers in the MCU memory to their default values. This function does not use the SPI or radio hardware.

It must be called while the radio is in shutdown mode. If the PHY is not in the shutdown state, this function will return the error code PHY_STATUS_ERROR_INVALID_STATE.

2.2.2. MbusPhyPowerUp

Prototype:

PHY_STATUS MbusPhyPowerUp (void);

Parameters:

none

Returns:

PHY_STATUS (uint8_t) status:

PHY_STATUS_SUCCESS

PHY_STATUS_ERROR_INVALID_STATE

Description:

This function will power up the radio from the shutdown state.

Note that the time to go into IDLE state depends on the current PHY state.

Shutdown → Idle (17 ms)

Standby → Idle (1 ms)

While it is possible to call the MBusPhyIdle() function directly from shutdown, the MCU will be waiting a long time for the radio to complete the power on reset sequence. The MCU can perform tasks while the Radio is powering up by using the MbusPhyPowerUp() function first. The MBusPhyIdle() function should be called before using any PHY functions that require the PHY to be in the Idle or standby state.

This can be used to perform tasks while waiting for Idle mode.

MbusPhyPowerUp ();

// This code is performed while the crystal is starting.

MbusPhyRadioIdle ();

2.2.3. MbusPhyIdle

Prototype:

PHY_STATUS MbusPhyIdle (void);

Parameters:

none

Returns:

PHY_STATUS (uint8_t) status:

PHY_STATUS_SUCCESS

PHY_STATUS_ERROR_POR_TIMEOUT

PHY_STATUS_ERROR_XTAL_TIMEOUT

PHY_STATUS_ERROR_INVALID_STATE

Description:

This function will put the PHY into IDLE state from any other state.

Note that the time to go into IDLE state depends on the current radio state.

Shutdown → Idle (17 ms)

Standby → Idle (1 ms)

RX → Idle (does not wait)

The MbusPhyPowerUp function may be used to perform tasks while waiting for the radio to power up.

MbusPhyPowerUp ();

// This code is performed while the crystal is starting.

MbusPhyRadioIdle ();

The next higher layer does not need to know the current state of the Radio, but the execution time of this function will vary greatly depending on the current radio mode.

2.2.4. MbusPhyStandby

Prototype:

PHY_STATUS MbusPhyStandby (void);

Parameters:

Returns:

PHY_STATUS (uint8_t) status:

PHY_STATUS_SUCCESS

PHY_STATUS_ERROR_POR_TIMEOUT

Description:

This function will put the PHY (Radio) into Standby mode from any other mode.

Note that the time to go into Standby mode depends on the current radio mode.

Shutdown → Standby (16 ms)

RX → Standby – 0 (does not wait)

Idle → Standby – 0 (does not wait)

Normally, this function should not be used from ShutDown mode. The radio automatically goes from ShutDown to Idle mode following a POR. If this function is called when the Radio is in Shutdown mode, the MCU will wait until the POR is complete.

2.2.5. MbusPhyInit

Prototype:

PHY_STATUS MbusPhyInit(void);

Parameters:

none

Returns:

PHY_STATUS (uint8_t) status:

PHY_STATUS_SUCCESS

PHY_STATUS_ERROR_INVALID_STATE

PHY_STATUS_ERROR_UNSUPPORTED_RADIO

Description:

The MBusPhyInit() function will initialize the Si443x internal registers.

The PHY must be in the IDLE state before calling MBusPhyInit(). This may be accomplished by calling MbusPhyIdle () before MBusPhyInit().

The PHY_METER and PHY_MODE registers should be set using the MbusPhySet() command before calling the MbusPhyInit() function. Otherwise the default values will be used.

Since the Si4431 register values are lost, the PHY must be initialized every time after it is powered down.

When using the asymmetric T2 transfer mode, this function will not initialize the radio registers since the radio registers are initialized upon each MbusPhyTx() or MbusPhyRx() function.

The status return code need not be used. In most cases, the next higher layer should know not to call this function from an invalid state.

2.2.6. MbusPhyRadioShutDown

Prototype:

PHY_STATUS MbusPhyRadioShutDown (void);

Parameters:

none

Returns:

PHY_STATUS (uint8_t) status

PHY_STATUS_SUCCESS

This function will put the Radio into shutdown mode from any other mode.

The radio crystal is turned off manually before shutting down the chip.

This function does not use any timeouts.

All radio register values will be lost in shutdown mode.

2.2.7. MbusPhyGet

Prototype:

PHY_STATUS MbusPhyGet(uint8_t addr, uint8_t *value);

Parameters:

uint8_t addr – The PHY variable address (enumeration)

uint8_t *value – generic byte pointer to data

Returns:

PHY_STATUS (uint8_t) status:

PHY_STATUS _SUCCESS

PHY_STATUS_ERROR_INVALID_ADDRESS

Description:

The MbusPhyGet function can be used to read the PHY registers or Si4431 registers.

0x00-0x7F—Si443x registers (SPI read transfer)

0x80-0x84—MbusPhy module registers

0x85-0xFF—returns invalid address

The SPI transfers are blocking, and this function will block until two bytes are transferred. This will take at least 1.6 µs using a 10 MHz SPI clock.

The status return code will return an error if the address is invalid. The status return code need not be used. In most cases, the next higher layer should know not to pass an invalid address.

2.2.8. MbusPhySet

Prototype:

uint8_t MbusPhySet(uint8_t addr, uint8_t value);

Parameters:

uint8_t addr—The PHY variable address (enumeration)

uint8_t value

Returns:

PHY_STATUS (uint8_t) status:

PHY_STATUS_SUCCESS

PHY_STATUS_ERROR_INVALID_VALUE

PHY_STATUS_ERROR_READ_ONLY_ADDRESS

PHY_STATUS_ERROR_INVALID_ADDRESS

Description:

The MbusPhySet function can be used to set the PHY module registers or Si4431 registers.

0x00-0x7F – Si443x registers (SPI write transfer)

0x80-0x84 – MbusPhy module registers

0x85-0xFF – returns invalid address

The status return code will return an error if the value is out of range or read only. The status return code need not be used. In most cases, the next higher layer should know not to write an invalid value, read only address, or invalid address.

The SPI transfers are blocking, and this function will block until two bytes are transferred. This will take at least 1.6 µs using a 10 MHz SPI clock.

2.2.9. MbusPhyTx

Prototype:

PHY_STATUS MbusPhyTX(VARIABLE_SEGMENT_POINTER(, uint8_t, BUFFER_MSPACE));

Parameters:

VARIABLE_SEGMENT_POINTER(buffer, uint8_t, BUFFER_MSPACE));

pointer to TX buffer in BUFFER_MSPACE (typically xdata)

Returns:

PHY_STATUS (uint8_t) status:

PHY_STATUS_SUCCESS

PHY_STATUS_ERROR_INVALID_STATE—not in IDLE state

PHY_STATUS_ERROR_INVALID_LENGTH—invalid L field

PHY_STATUS_ERROR_TX_TIMEOUT—Radio failed to send message before timeout

Description:

This function will transmit the data in the transmit buffer. This function is blocking and will not return until transmission is complete.

If the asymmetric Mode T2 is being used, the radio will be initialized for TX.

The L field is read from the first byte of the buffer and need not be passed as a parameter. The packet length, number of blocks, and the number of encoded bytes is calculated from the L field. The number of encoded bytes is written to the radio TX packet length.

The packet in the TX buffer should be properly formatted (by the Data Link Layer) into blocks with CRCs between blocks.

The next higher layer should supply a buffer large enough to accommodate the entire packet as indicated by the L field. The TX function may send garbage data in the case of a buffer overrun. The TX function does not write to the TX buffer.

This function will copy the first 64 bytes into the radio FIFO. Each byte of data is encoded before writing to the SPI. If additional bytes are to be sent, the FIFO almost empty threshold will be set to 16 bytes and the FIFO almost empty interrupt will be enabled. If there are not additional bytes to be written, the FIFO almost empty will be cleared to 0, and the Packet Sent interrupt will be enabled.

Once the FIFO has been initialized and interrupts have been enabled, the transmitter will be started by writing to the Function control 1 register. The MCU will then go to sleep until the next IRQ wakeup or timeout.

If the PHY is not in the IDLE state, the function will immediately return a PHY_STATUS_ERROR_INVALID_STATE error code.

If the first byte in the buffer is an invalid length for the L field, the function will immediately return a PHY_STATUS_ERROR_INVALID_LENGTH error code.

If for any reason the Radio does not successfully transmit all of the data in the TX buffer, the radio will be manually set into the IDLE mode, and this function will return the PHY_STATUS_ERROR_TX_TIMEOUT error code. This generally indicates a problem with the hardware.

2.2.10. MbusPhyRx

Prototype:

PHY_STATUS MbusPhyRX(uint32_t, SI_VARIABLE_SEGMENT_POINTER(buffer, uint8_t, BUFFER_MSPACE));

Parameters:

uint32_t timeout—ticks of 32.768 kHz clock.

SI_VARIABLE_SEGMENT_POINTER(buffer, uint8_t, BUFFER_MSPACE));

pointer to RX buffer in BUFFER_MSPACE (typically xdata)

Returns:

uint32_t status

PHY_STATUS_SUCCESS

PHY_STATUS_ERROR_NOTHING_RECEIVED—nothing receive in RX time

PHY_STATUS_ERROR_INVALID_STATE—not in IDLE state

PHY_STATUS_ERROR_INVALID_LENGTH—invalid received L field

PHY_STATUS_ERROR_DECODING—error decoding Manchester or 3 out of 6

PHY_STATUS_ERROR_RX_INCOMPLETE—incomplete message received

Description:

The MbusPhyRx() function is a blocking RX function with a timeouts. The receiver will remain on for the specified amount of time with the MCU in low-power sleep mode. The timeout or receiver interrupt will wake up the MCU.

The next higher layer should supply a buffer large enough to accommodate the entire packet for the largest possible L-field of interest.

The largest L-field of interest should be defined in Mbus_defs.h

#define USER_RX_MAX_L_FIELD 148

An example buffer declaration is shown below:

SI_SEGMENT_VARIABLE(userBuffer[USER_RX_BUFFER_SIZE], uint8_t, BUFFER_MSPACE)

The return status of this function will indicate if a packet has been successfully received. If a packet has been successfully received, this function will return a status of PHY_STATUS_SUCCESS. The PHY does not validate the CRCs; so, the packet still might contain a CRC error.

If the receiver timeout has expired, the function will return a status of PHY_STATUS_ERROR_NOTHING_RECEIVED. This may not be an error in many circumstances.

A decoding error at any point after the SYNC word results in the Receiver immediately returning a status of PHY_STATUS_ERROR_DECODING.

If the correct number of bytes is not received completely, a status of PHY_STATUS_ERROR_RX_INCOMPLETE will be returned.

2.3. PHY Module Registers

This it the structure typedef for the Mbus PHY

```
typedef struct MbusPhyRegisters
{
  uint8_t meter;
  uint8_t mode;
  uint8_t channel;
  uint8_t state;
  uint8_t RxPacketLength;
  uint8_t RxRSSI;
  } MbusPhyRegisters;
```

All PHY module registers can be read using the MbusPhySet() function.

3. Data Link Layer API

The Data Link Layer module implements a 13757-4:2005 compliant link layer. The Data Link Layer (LINK) provides an interface between the Physical Layer (PHY) and the Application Layer (AL).

Data Link Layer:

- Provides functions that transfer data between PHY and AL
- Generate CRCs for outgoing messages
- Detect CRC errors in incoming messages
- Provide physical addressing
- Acknowledge transfers for bidirectional communication modes
- Frame data bits
- Detect framing errors in incoming messages

3.1. Definitions

- **AL**—Application Layer
- **LINK**—Data Link Layer
- **NHL**—Next Higher Layer
- **PHY**—Physical Layer

3.2. Module and File Organization

Table 2. File Descriptions and Comments

File Name	Description	Comments
Common.h	Common typedefs and macros	
Crc.h	CRC-16-DNP header file	
Crc.c	CRC-16-DNP source file	
MbusLink.h	Data Link Layer header file	
MbusLink.c	Data Link Layer source file	
MbusLinkServices.h	Data Link Layer Services primitives header file	
MbusLinkServices.c	Data Link Layer Services primitives source file	Optional request, confirmation, indication, and response service primitives.
MbusLink_defs.h	Data Link Layer compile-time build options	
MbusLinkServices_defs.h	Data Link Layer Services compile-time build options	

3.3. Functions

3.3.1. MbusLinkInit

Prototype:

LINK_STATUS MbusLinkInit(void);

Parameters:

none

Returns:

LINK_STATUS_SUCCESS

LINK_STATUS_PHY_ERROR_INVALID_STATE

Description:

This function initializes the Data Link Layer module attributes to their default values, initializes the RTC at 32.768 kHz, and initializes the PHY registers. This function only needs to be called once at the beginning of code after an MCU reset.

3.3.2. MbusLinkRadioPowerOn

Prototype:

LINK_STATUS MbusLinkRadioPowerOn(void);

Parameters:

none

Returns:

LINK_STATUS_SUCCESS

LINK_STATUS_PHY_ERROR_INVALID_STATE

Description:

This function turns on the EZRadioPRO® but does not wait for the radio to enter IDLE mode. The NHL/AL can perform work while waiting for the radio oscillator to stabilize. Next, call MbusLinkRadioConfig() to wait for the radio to enter IDLE mode and configure the radio modes and channel.

3.3.3. MbusLinkRadioConfig

Prototype:

LINK_STATUS MbusLinkRadioConfig(void);

Parameters:

none

Returns:

LINK_STATUS_SUCCESS

LINK_STATUS_PHY_ERROR_INVALID_VALUE

LINK_STATUS_PHY_ERROR_INVALID_STATE

LINK_STATUS_PHY_ERROR_POR_TIMEOUT

LINK_STATUS_PHY_ERROR_XTAL_TIMEOUT

LINK_STATUS_PHY_ERROR_UNSUPPORTED_RADIO

Description:

This function waits for the EZRadioPRO to enter IDLE mode and stores the radio modes and channel settings before configuring the radio over SPI. Users must call this function after changing radio settings before the new settings will apply.

3.3.4. MbusLinkRadioPowerOff

Prototype:

LINK_STATUS MbusLinkRadioPowerOff(void);

Parameters:

none

Returns:

LINK_STATUS_SUCCESS

LINK_STATUS_PHY_ERROR_INVALID_STATE

Description:

Turns the EZRadioPRO off. All radio settings are lost when the radio is shut down. Call MbusLinkRadioPowerOn() and MbusLinkRadioConfig() before using the radio again.

3.3.5. MbusLinkRadioStandby

Prototype:

LINK_STATUS MbusLinkRadioStandby(void);

Parameters:

none

Returns:

LINK_STATUS_SUCCESS

LINK_STATUS_PHY_ERROR_POR_TIMEOUT

LINK_STATUS_PHY_ERROR_INVALID_STATE

Description:

Puts the EZRadioPRO into a low power standby mode. Radio settings are preserved in standby mode.
Call MbusLinkRadioResume() to resume from standby mode.

3.3.6. MbusLinkRadioResume

Prototype:

LINK_STATUS MbusLinkRadioResume(void);

Parameters:

none

Returns:

LINK_STATUS_SUCCESS

LINK_STATUS_PHY_ERROR_INVALID_STATE

LINK_STATUS_PHY_ERROR_POR_TIMEOUT

LINK_STATUS_PHY_ERROR_XTAL_TIMEOUT

Description:

Resumes the EZRadioPRO from standby mode and places the radio into IDLE mode.

3.3.7. MbusLinkMcuSleep

Prototype:

LINK_STATUS MbusLinkMcuSleep(__in uint32_t duration);

Parameters:

duration—The amount of time for the MCU to sleep in RTC (32.768 kHz) ticks.

Returns:

LINK_STATUS_SUCCESS

LINK_STATUS_PHY_ERROR

Description:

This function places the MCU into a low power sleep mode and prepares the RTC to wake the device up after the specified duration. If MAINS_POWERED is defined in MbusPhy_defs.h, then the MCU will enter IDLE mode, allowing interrupts to occur.

3.3.8. MbusLinkSetAttribute

Prototype:

LINK_STATUS MbusLinkSetAttributes(__in LINK_ATTR_ID attr, __in uint8_t value);

Parameters:

attr—The attribute identifier for the attribute to modify

value—Sets the attribute value

Returns:

LINK_STATUS_SUCCESS

LINK_STATUS_PHY_ERROR_READ_ONLY_ADDRESS

LINK_STATUS_PHY_ERROR_INVALID_ADDRESS

LINK_STATUS_PHY_ERROR_INVALID_VALUE

LINK_STATUS_INVALID_ATTR

LINK_STATUS_NOT_PERMITTED

Description:

This function sets the specified attribute to the specified value.

Attribute IDs:

0x00–0x7F: Si443x registers (SPI write transfer)

0x80–0x8F: PHY variables

0x90–0xFF: LINK attributes (see "3.5. Data Link Layer Attributes" on page 20)

3.3.9. MbusLinkGetAttribute

Prototype:

LINK_STATUS MbusLinkGetAttributes(__in LINK_ATTR_ID attr, __out uint8_t* value);

Parameters:

attr—The attribute identifier for the attribute value to return

value—Returns the attribute value

Returns:

LINK_STATUS_SUCCESS

LINK_STATUS_PHY_ERROR_INVALID_ADDRESS

LINK_STATUS_INVALID_ATTR

LINK_STATUS_NOT_PERMITTED

Description:

This function returns the attribute value for the attribute specified.

Attribute IDs:

0x00–0x7F: Si443x registers (SPI write transfer)

0x8–0x8F: PHY variables

0x90–0xFF: LINK attributes

3.3.10. MbusLinkTransmitFrame

Prototype:

LINK_STATUS MbusLinkTransmitFrame(__in Frame* frame);

Parameters:

frame—A pointer to a LINK frame specifying the C, M, A, CI, and Data fields

Returns:

LINK_STATUS_SUCCESS
LINK_STATUS_PHY_ERROR_INVALID_VALUE
LINK_STATUS_PHY_ERROR_INVALID_STATE
LINK_STATUS_PHY_ERROR_INVALID_LENGTH
LINK_STATUS_PHY_ERROR_POR_TIMEOUT
LINK_STATUS_PHY_ERROR_XTAL_TIMEOUT
LINK_STATUS_PHY_ERROR_UNSUPPORTED_RADIO
LINK_STATUS_PHY_ERROR_TX_TIMEOUT
LINK_STATUS_INVALID_SIZE

Description:

This function takes a frame input and generates a frame buffer consisting of L, C, M, A, CI, Data, and CRC fields (stored in an internal TX frame buffer). The L and CRC fields are automatically calculated and added to the frame buffer. If the EZRadioPRO is not already in IDLE mode, this function will call MbusLinkRadioConfig(). This allows users to optimize power usage by generating a transmit frame while initializing the EZRadioPRO. Next, the function transmits the frame buffer via the PHY.

3.3.11. MbusLinkTransmitFrameAtTime

Prototype:

LINK_STATUS MbusLinkTransmitFrameAtTime(__in uint32_t txRtcTime, __in Frame* frame);

Parameters:

txRtcTime—The time in RTC (32.768 kHz) ticks at which to transmit the frame

frame—A pointer to a LINK frame specifying the C, M, A, CI, and Data fields

Returns:

LINK_STATUS_SUCCESS
LINK_STATUS_PHY_ERROR_INVALID_VALUE
LINK_STATUS_PHY_ERROR_INVALID_STATE
LINK_STATUS_PHY_ERROR_INVALID_LENGTH
LINK_STATUS_PHY_ERROR_POR_TIMEOUT
LINK_STATUS_PHY_ERROR_XTAL_TIMEOUT
LINK_STATUS_PHY_ERROR_UNSUPPORTED_RADIO
LINK_STATUS_PHY_ERROR_TX_TIMEOUT
LINK_STATUS_INVALID_SIZE

Description:

This function takes a frame input and generates a frame buffer consisting of L, C, M, A, CI, Data, and CRC fields (stored in an internal TX frame buffer). The L and CRC fields are automatically calculated and added to the frame buffer. If the EZRadioPRO® is not already in idle mode, then this function will call MbusLinkRadioConfig(). This allows users to optimize power usage by generating a transmit frame while initializing the EZRadioPRO®. Next, the function transmits the frame buffer via the PHY at the specified RTC time.

3.3.12. MbusLinkReceiveFrame

Prototype:

LINK_STATUS MbusLinkReceiveFrame(__in uint32_t timeout, __out Frame* frame);

Parameters:

timeout—The maximum number of RTC (32.768 kHz) ticks to wait to receive a valid preamble and sync word

frame—A pointer to a LINK frame that will return the C, M, A, CI, and Data fields

Returns:

LINK_STATUS_SUCCESS
LINK_STATUS_PHY_ERROR_INVALID_VALUE
LINK_STATUS_PHY_ERROR_INVALID_STATE
LINK_STATUS_PHY_ERROR_INVALID_LENGTH
LINK_STATUS_PHY_ERROR_NOTHING_RECEIVED
LINK_STATUS_PHY_ERROR_DECODING
LINK_STATUS_PHY_ERROR_POR_TIMEOUT
LINK_STATUS_PHY_ERROR_XTAL_TIMEOUT
LINK_STATUS_PHY_ERROR_UNSUPPORTED_RADIO
LINK_STATUS_PHY_ERROR_RX_PACKET_INCOMPLETE

Description:

This function first makes sure that the EZRadioPRO is in IDLE mode by calling MbusLinkRadioConfig() and then attempts to receive a frame using the PHY. If reception was successful, the function parses the received frame buffer by checking for valid data lengths and CRC and copies the data to a frame structure.

3.4. Service Primitive Functions

The Data Link Layer implements the service primitives defined in IEC 60870-5-2:1992. To include the service primitives module in a project, the user must include MbusLinkService.h and MbusLinkService.c in the project file and call MbusLinkServicesInit().

3.4.1. MbusLinkServicesInit

Prototype:

void MbusLinkServicesInit(void);

Parameters:

none

Returns:

none

Description:

This function initializes the Mbus Link Services module.

3.4.2. MbusLinkServicesRequest

Prototype:

void MbusLinkServicesRequest(__in Request* request, __out Confirmation* confirmation);

Parameters:

request—A pointer to a Data Link Layer request primitive. Contains the frame to transmit and the number of retry attempts (for Send/Confirm and Request/Respond)

confirmation—A pointer to a Data Link Layer confirmation service primitive. Returns the status of the request, positive/negative acknowledge, the request service type, link state, and response frame (Request/Respond).

Returns:

none

Description:

This function processes a request service primitive, generates a valid C-field by setting the FCV, FCB, and PRM bits, transmits the request frame, checks for confirm/response if applicable, and retries data transmission when appropriate (Send/Confirm, Request/Respond when valid ack/nack is not received).

Flow Chart:

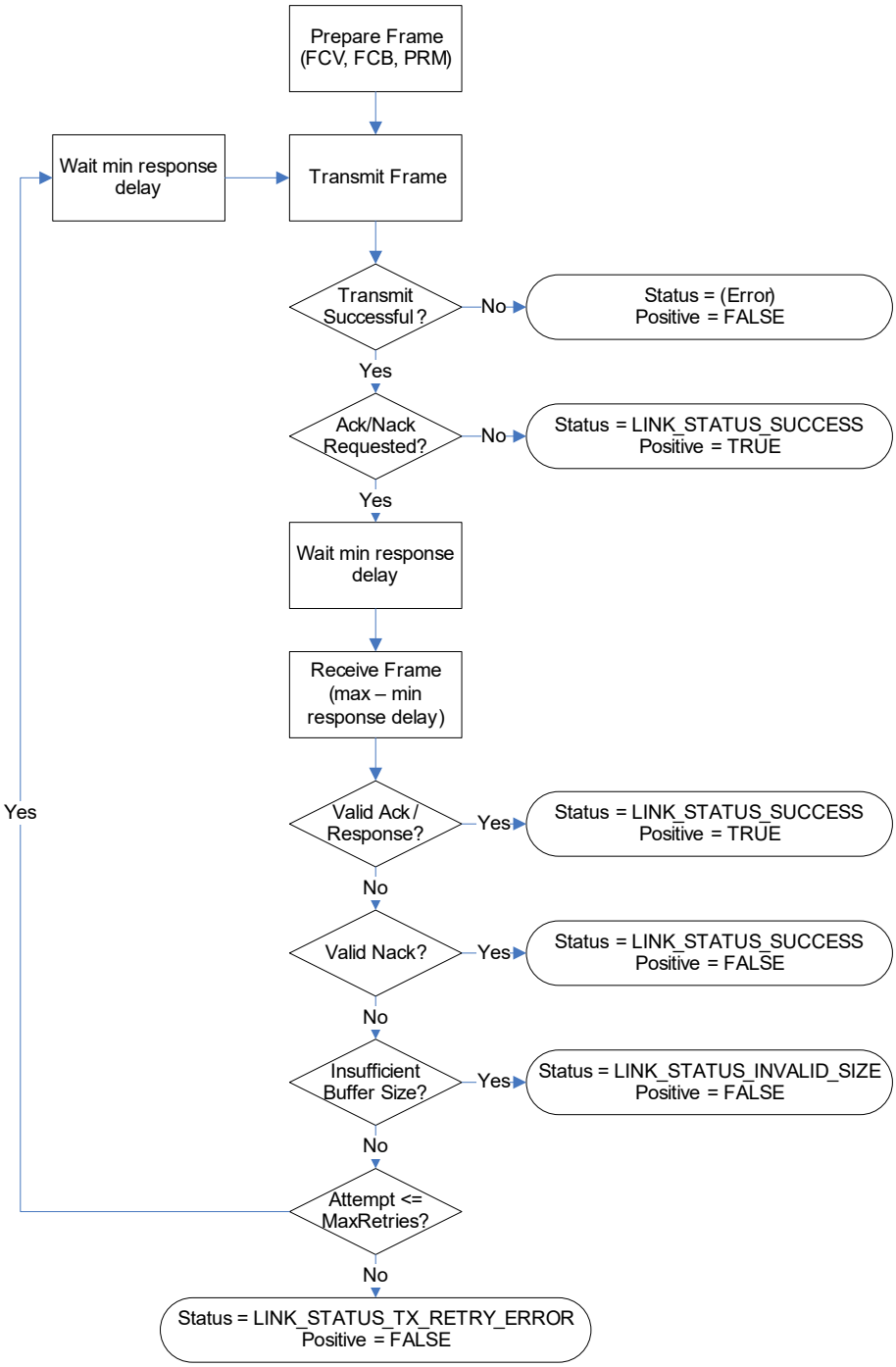


Figure 2. MbusLinkServicesRequest Flowchart

3.4.3. MbusLinkServicesIndication

Prototype:

void MbusLinkServicesIndication(__out Indication* indication);

Parameters:

indication—A pointer to a Data Link Layer indication service primitive. Specifies the read timeout and returns the status of the read operation, the received frame service type, and the received frame.

Returns:

none

Description:

This function attempts to receive a frame via the PHY within the timeout duration specified. Upon receipt of a frame, the indication service primitive returns the frame and service type.

Flow Chart:

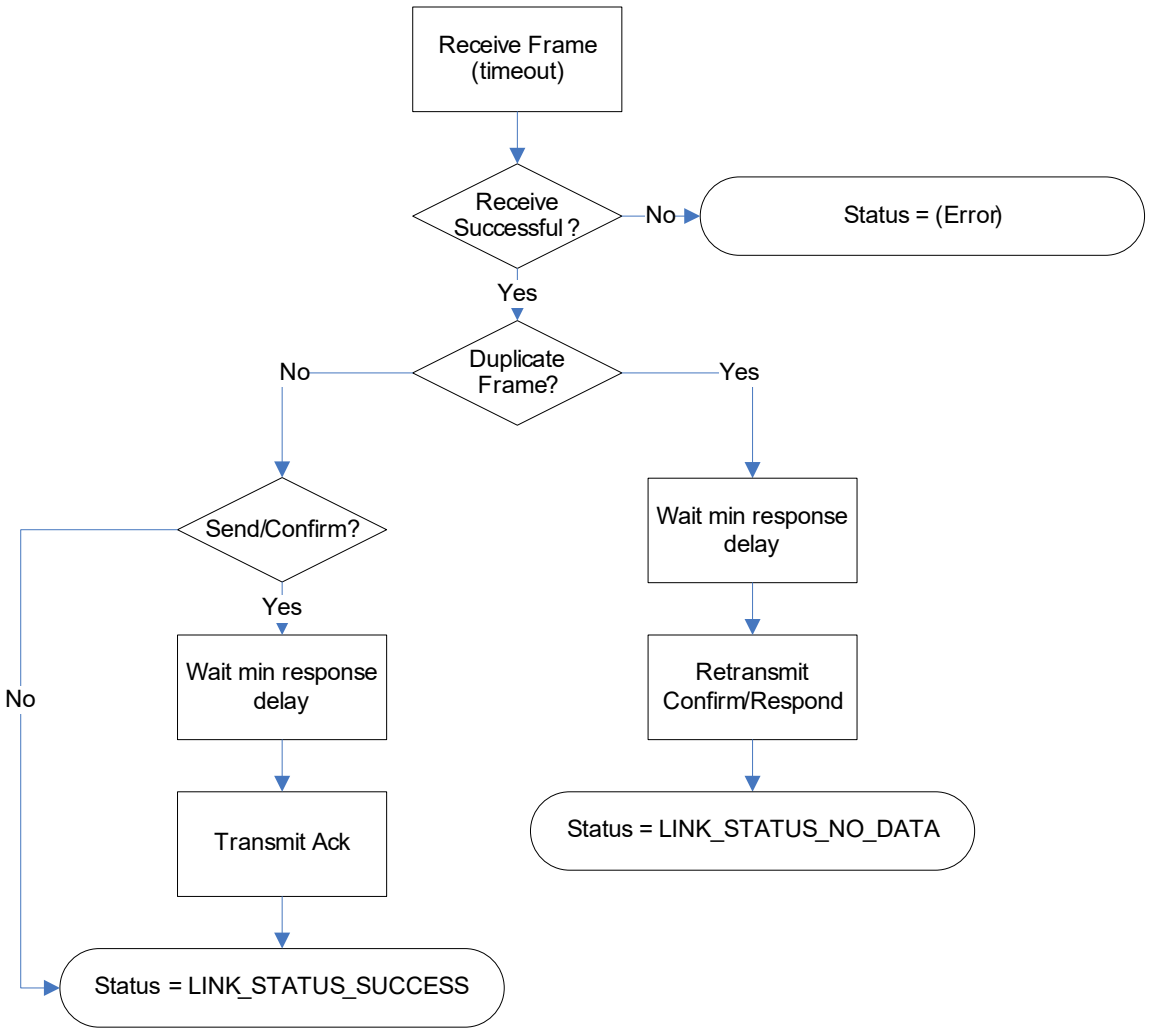


Figure 3. MbusLinkServicesIndication Flowchart

3.4.4. MbusLinkServicesResponse

Prototype:

LINK_STATUS MbusLinkServicesResponse(__in Indication* indication, __in Response* response);

Parameters:

indication—Indication returned from MbusLinkServicesIndication().

response—A pointer to a Data Link Layer response service primitive that specifies the response frame or a negative response.

Returns:

LINK_STATUS_SUCCESS
LINK_STATUS_PHY_ERROR_INVALID_VALUE
LINK_STATUS_PHY_ERROR_INVALID_STATE
LINK_STATUS_PHY_ERROR_INVALID_LENGTH
LINK_STATUS_PHY_ERROR_NOTHING_RECEIVED
LINK_STATUS_PHY_ERROR_DECODING
LINK_STATUS_PHY_ERROR_POR_TIMEOUT
LINK_STATUS_PHY_ERROR_XTAL_TIMEOUT
LINK_STATUS_PHY_ERROR_UNSUPPORTED_RADIO
LINK_STATUS_PHY_ERROR_RX_PACKET_INCOMPLETE
LINK_STATUS_INVALID_SIZE

Description:

This function attempts to receive a frame via the PHY within the timeout duration specified. Upon receipt of a frame, the indication service primitive returns the frame and service type.

Flow Chart:

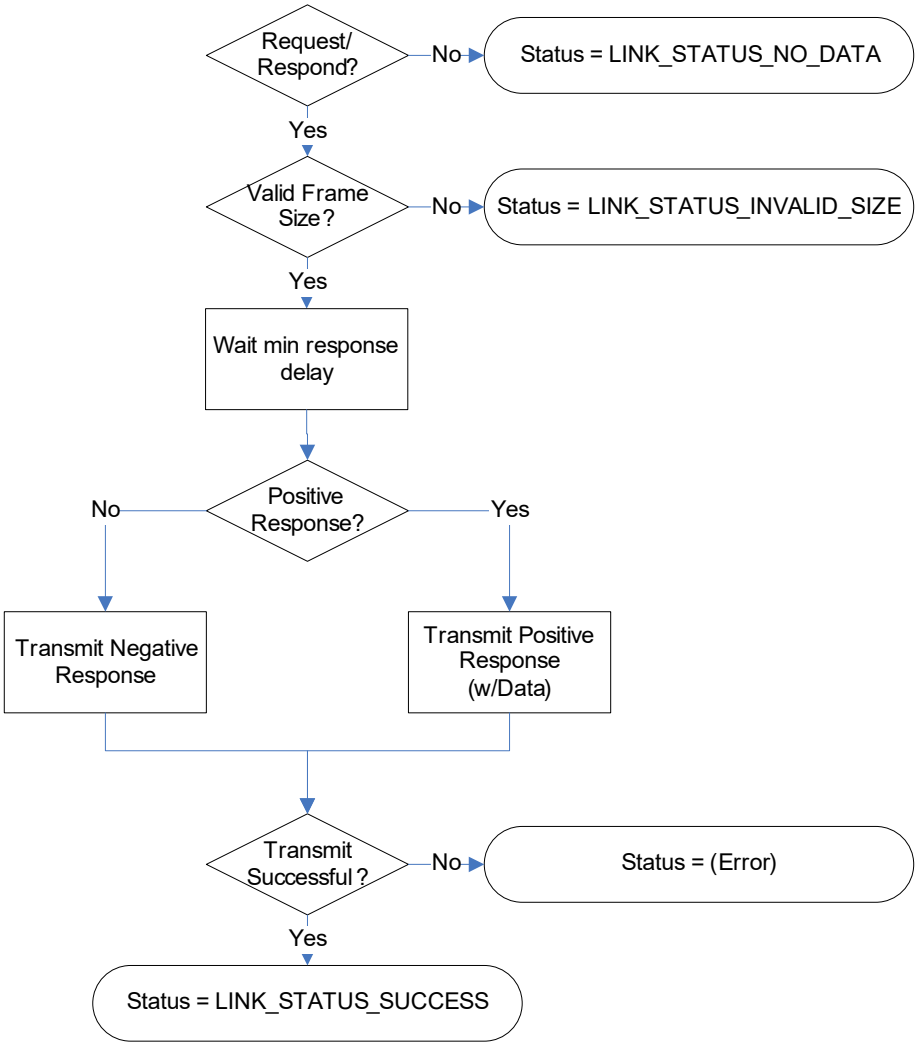


Figure 4. MbusLinkServicesResponse Flowchart

3.5. Data Link Layer Attributes

Table 3. Data Link Layer Attributes

Attribute	LINK_ATTR_ID Enumerations	ID	Description
LinkAttrMeterMode	LINK_ATTR_ID_METER_MODE	0x90	Meter/Other Mode: 0—LINK_MODE_OTHER 1—LINK_MODE_METER
LinkAttrRadioMode	LINK_ATTR_ID_RADIO_MODE	0x91	Wireless M-Bus radio mode: 0—LINK_MODE_S 1—LINK_MODE_SL 2—LINK_MODE_T 3—LINK_MODE_R
LinkAttrRadioChannel	LINK_ATTR_ID_RADIO_CHANNEL	0x92	R Mode Channel (0–9)
LinkAttrTxFcb	LINK_ATTR_ID_TX_FCB	0x93	unused
LinkAttrRxFcb	LINK_ATTR_ID_RX_FCB	0x94	FCB value from last frame received
LinkAttrModeSTroMin	LINK_ATTR_ID_MODE_S_TRO_MIN	0x95	Mode S: Minimum response delay (ms)
LinkAttrModeSTroMax	LINK_ATTR_ID_MODE_S_TRO_MAX	0x96	Mode S: Maximum response delay (ms)
LinkAttrModeTTackMin	LINK_ATTR_ID_MODE_T_TACK_MIN	0x97	Mode T: Minimum acknowledge delay (ms)
LinkAttrModeTTackMax	LINK_ATTR_ID_MODE_T_TACK_MAX	0x98	Mode T: Maximum acknowledge delay (ms)
LinkAttrModeRTroMin	LINK_ATTR_ID_MODE_R_TRO_MIN	0x99	Mode R: Minimum response delay (ms) (CI = 0x81)
LinkAttrModeRTroMax	LINK_ATTR_ID_MODE_R_TRO_MAX	0x9A	Mode R: Maximum response delay (ms) (CI = 0x81)
LinkAttrModeRTrmMin	LINK_ATTR_ID_MODE_R_TRM_MIN	0x9B	Mode R: Minimum response delay (ms)
LinkAttrModeRTrmMax	LINK_ATTR_ID_MODE_R_TRM_MAX	0x9C	Mode R: Maximum response delay (s)

3.5.1. Data Link Layer User-Modifiable Definitions

Table 4. User-Modifiable Definitions

Definition	Min	Default	Description
LINK_USER_MAX_TX_DATA_FIELD_SIZE	0	138	Specifies the max data-field size in bytes to be transmitted and is used to determine the size of the TX frame buffer
LINK_USER_MAX_RX_DATA_FIELD_SIZE	0	138	Specifies the max data-field size in bytes to be received and is used to determine the size of the RX frame buffer
LINK_USER_MSPACE		SEG_XDATA	The memory space used for all spaced pointers for frame data and service primitives

3.5.2. Data Link Layer Service Primitive User-Modifiable Definitions

Table 5. Service Primitive User-Modifiable Definitions

Definition	Min	Default	Description
LINK_AUTO_RESP_DATA_MAX	0	138	The size of the data-field buffer used for automatic responses to duplicate requests
LINK_TX_FCB_TABLE_ENTRIES_MAX	1	2	The number of destination addresses to record the frame control bit used for transmission
LINK_SYSCLK		20000UL	The system clock speed in kHz used to determine processing delays

3.6. Data Link Layer Structures

3.6.1. Frame

Typedef:

```
typedef uint8_t LINK_ATTR_MFR[LINK_ATTR_SIZE_MFR];
typedef uint8_t LINK_ATTR_ADDR[LINK_ATTR_SIZE_ADDR];

typedef struct DataField
{
        uint8_t size;
        uint8_t capacity;
        uint8_t* payload;
} DataField;
```

Members:

size - The number of valid bytes in the data-field
capacity - The size of the buffer pointed to by payload
payload - A pointer to a user space byte array

```
typedef struct Frame
{
        uint8_t c_field;
        LINK_ATTR_MFR m_field;
        LINK_ATTR_ADDR a_field;
        uint8_t ci_field;
        DataField data_field;
} Frame;
```

Members:

c_field - The Control-field
m_field - The manufacturer ID
a_field - The Address-field
ci_field - The Control Information-field
data_field - The Data-field

3.7. Data Link Layer Service Primitive Structures

3.7.1. Request

Typedef:

```
typedef struct Request
{
        uint8_t retries;
        Frame frame;
} Request;
```

Members:

retries - The number of times to retry an acknowledged transmission if a confirm/respond is not received.
frame - The frame to transmit. The C-field is used to determine the transmission procedure (Send/No Reply, Send/Confirm, Request/Respond). All fields must be specified when calling MbusLinkServicesRequest().

3.7.2. Confirmation

Typedef:

```
typedef struct Confirmation
{
    LINK_STATUS status;
    bool positive;
    LINK_SERVICE_TYPE serviceType;
    LINK_STATE link;
    Frame frame;
}; Confirmation;
```

Members:

status - returns the status of the MbusLinkServicesRequest().
positive - returns TRUE if the MbusLinkServicesRequest() was successful.
Send/NoReply: returns TRUE if the transmission was successful.
Send/Confirm: returns TRUE if a positive ack/confirm was received.
Request/Respond: returns TRUE if a response was received.; Returns FALSE if a respond nack is received.
serviceType - returns the service type of the request (Send/No Reply, Send/Confirm, Request/Respond).
link - returns the status of the Data Link Layer (operational/non operational)
frame - returns the frame received for Send/Confirm or Request/Respond.

3.7.2.1. Indication

Typedef:

```
typedef struct Indication
{
    uint32_t timeout;
    LINK_STATUS status;
    LINK_SERVICE_TYPE serviceType;
    Frame frame;
} Indication;
```

Members:

timeout - The number of milliseconds to wait to receive data for MbusLinkServicesIndication().
status - The status of MbusLinkServicesIndication().
serviceType - The service type of the received frame (Send/No Reply, Send/Confirm, Request/Respond).
frame - The received frame.

3.7.2.2. Response

Typedef:

```
typedef struct Response
{
    bool positive;
    Frame frame;
} Response;
```

Members:

positive - Set to TRUE if sending a response to a Send/Respond with data.
Set to FALSE if sending a respond nack.
frame - The frame response to send.

3.8. LINK_STATUS

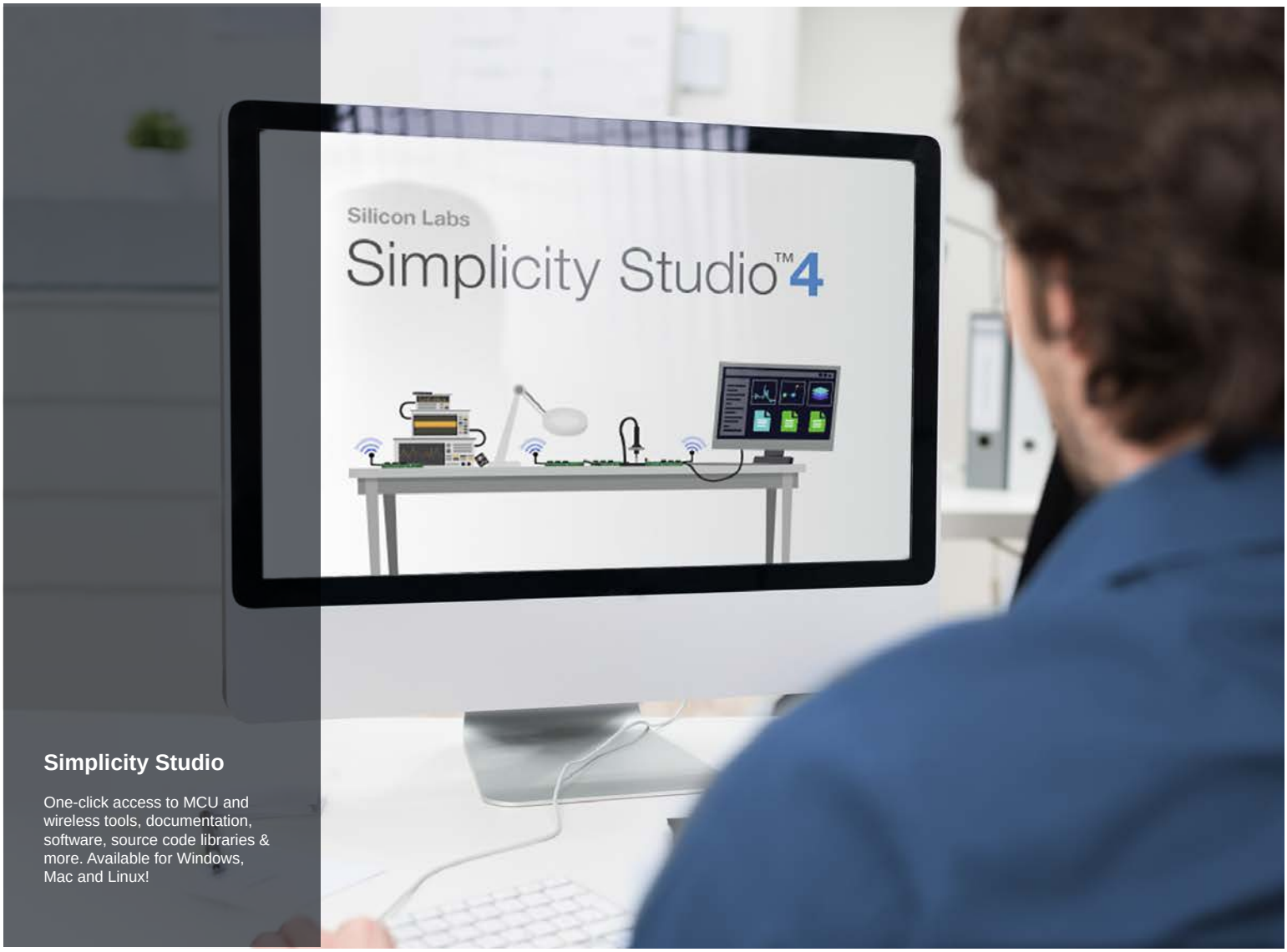
```
// LINK_STATUS
typedef enum LINK_STATUS
{
// PHY layer status codes
  LINK_STATUS_SUCCESS = 0x00,
  LINK_STATUS_PHY_ERROR,                // unspecified
  LINK_STATUS_PHY_ERROR_TIMEOUT,        // unspecified timeout
  LINK_STATUS_PHY_ERROR_READ_ONLY_ADDRESS,
  LINK_STATUS_PHY_ERROR_INVALID_ADDRESS,
  LINK_STATUS_PHY_ERROR_INVALID_VALUE,
  LINK_STATUS_PHY_ERROR_INVALID_STATE,
  LINK_STATUS_PHY_ERROR_INVALID_LENGTH,
  LINK_STATUS_PHY_ERROR_NOTHING_RECEIVED,
  LINK_STATUS_PHY_ERROR_DECODING,
  LINK_STATUS_PHY_ERROR_POR_TIMEOUT,
  LINK_STATUS_PHY_ERROR_XTAL_TIMEOUT,
  LINK_STATUS_PHY_ERROR_UNSUPPORTED_RADIO,
  LINK_STATUS_PHY_ERROR_TX_TIMEOUT,
  LINK_STATUS_PHY_ERROR_RX_INVALID_LENGTH,
  LINK_STATUS_PHY_ERROR_RX_PACKET_INCOMPLETE,

// LINK layer status codes
  LINK_STATUS_INVALID_SIZE = 0x40,
  LINK_STATUS_INVALID_ATTR,
  LINK_STATUS_NOT_PERMITTED,
  LINK_STATUS_CRC_ERROR,
  LINK_STATUS_FRAME_ERROR,
  LINK_STATUS_TX_RETRY_ERROR,
  LINK_STATUS_UNKNOWN_ERROR = 0xFF

} LINK_STATUS;
```

Table 6. LINK_STATUS Enumeration Descriptions

Enumeration	Value	Description
LINK_STATUS_SUCCESS	0x00	Success
LINK_STATUS_PHY_ERROR	0x01	PHY_STATUS_ERROR
LINK_STATUS_PHY_ERROR_TIMEOUT	0x02	PHY_STATUS_ERROR_TIMEOUT
LINK_STATUS_PHY_ERROR_READ_ONLY_ADDRESS	0x03	PHY_STATUS_ERROR_READ_ONLY_ADDRESS
LINK_STATUS_PHY_ERROR_INVALID_ADDRESS	0x04	PHY_STATUS_ERROR_INVALID_ADDRESS
LINK_STATUS_PHY_ERROR_INVALID_VALUE	0x05	PHY_STATUS_ERROR_INVALID_VALUE
LINK_STATUS_PHY_ERROR_INVALID_STATE	0x06	PHY_STATUS_ERROR_INVALID_STATE
LINK_STATUS_PHY_ERROR_INVALID_LENGTH	0x07	PHY_STATUS_ERROR_INVALID_LENGTH
LINK_STATUS_PHY_ERROR_NOTHING_RECEIVED	0x08	PHY_STATUS_ERROR_NOTHING_RECEIVED
LINK_STATUS_PHY_ERROR_DECODING	0x09	PHY_STATUS_ERROR_DECODING
LINK_STATUS_PHY_ERROR_POR_TIMEOUT	0x0A	PHY_STATUS_ERROR_POR_TIMEOUT
LINK_STATUS_PHY_ERROR_XTAL_TIMEOUT,	0x0B	PHY_STATUS_ERROR_XTAL_TIMEOUT
LINK_STATUS_PHY_ERROR_UNSUPPORTED_RADIO	0x0C	PHY_STATUS_ERROR_UNSUPPORTED_RADIO
LINK_STATUS_PHY_ERROR_TX_TIMEOUT,	0x0D	PHY_STATUS_ERROR_TX_TIMEOUT
LINK_STATUS_PHY_ERROR_RX_INVALID_LENGTH	0x0E	PHY_STATUS_ERROR_RX_INVALID_LENGTH
LINK_STATUS_PHY_ERROR_RX_PACKET_INCOMPLETE	0x0F	PHY_STATUS_ERROR_RX_PACKET_INCOMPLETE
LINK_STATUS_INVALID_SIZE	0x40	Buffer is not large enough
LINK_STATUS_INVALID_ATTR	0x41	Invalid attribute ID
LINK_STATUS_NOT_PERMITTED	0x42	Attribute is read only or write only
LINK_STATUS_CRC_ERROR	0x43	Frame is invalid due to CRC mismatches
LINK_STATUS_FRAME_ERROR	0x44	Frame is invalid due to incorrect block format
LINK_STATUS_TX_RETRY_ERROR	0x45	Could not transmit service request (retries failed)
LINK_STATUS_UNKNOWN_ERROR	0xFF	Unspecified error



Simplicity Studio

One-click access to MCU and wireless tools, documentation, software, source code libraries & more. Available for Windows, Mac and Linux!

IoT Portfolio
www.silabs.com/IoT

SW/HW
www.silabs.com/simplicity

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

SILICON LABS

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

http://www.silabs.com