MKW01 Simple Media Access Controller (SMAC)

Reference Manual

Document Number: MKW01SMACRM
Rev. 2
2/2014

freescale™
semiconductor

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

Contents

About This Book

This manual provides a detailed description of the Freescale MKW01 Simple Media Access Controller (MKW01 SMAC). This software is designed for use specifically with the MKW01 platform. The MKW01 is a highly-integrated, cost-effective, system-in-package (SiP), sub-1 GHz wireless node solution with an FSK, GFSK, MSK, or OOK modulation-capable transceiver and low-power, ARM Cortex M0+ 32-bit microcontroller. The highly integrated RF transceiver operates over a wide frequency range including 315 MHz, 433 MHz, 470 MHz, 868 MHz, 915 MHz, 928 MHz, and 955 MHz in the license-free Industrial, Scientific, and Medical (ISM) frequency bands.

The MKW01 SMAC software is pre-defined to operate in the 863–870 MHz, 902–928 MHz and 950–958 MHz bands.

Audience

This document is intended for application developers working on custom wireless applications that employ the MKW01. The latest version of the Freescale MKW01 SMAC is available in the Freescale website.

Organization

This document is organized into four chapters and one appendix.

Chapter 1	**MKW01 SMAC Introduction** — This chapter introduces MKW01 SMAC features and functionality.
Chapter 2	**Software Architecture** — This chapter describes MKW01 SMAC software architecture.
Chapter 3	**Primitives** — This chapter provides a detailed description of MKW01 SMAC primitives.

Revision History

The following table summarizes revisions to this document since the previous release (Rev. 0.0).

Revision History

Location	Revision
Appendix A	Added new appendix
Appendix B	Added new appendix

Conventions

This document uses the following notational conventions:

- `Courier monospaced type` indicate commands, command parameters, code examples, expressions, datatypes, and directives.
- *Italic type* indicates replaceable command parameters.
- All source code examples are in C.

Definitions, Acronyms, and Abbreviations

The following list defines the acronyms and abbreviations used in this document.

EVK	Evaluation Kit
GUI	Graphical User Interface
MAC	Medium Access Control
MCU	MicroController Unit
NVM	Non-Volatile Memory
PC	Personal Computer
PCB	Printed Circuit Board
S19	'S19' is the file extension used for the Freescale binary image format. The S19 file encapsulates the binary image as a list of ASCII records. Each record contains a length field, address field, data field, and checksum field. The S19 can be generated with CodeWarrior IDE and is the product from the linking process. S19 does not contain additional information to a debugger (where to look for source files).
OTA	Over the air.

References

The following sources were referenced to produce this book:

1. Freescale 802.15.4 MAC/PHY Software Reference Manual (802154MPSRM)
2. MKW01 Reference Manual (MKW01Z128RM.pdf)

Chapter 1
MKW01 SMAC Introduction

The Freescale MKW01 Simple Media Access Controller (MKW01 SMAC) is a simple ANSI C based codebase available as sample source code. The MKW01 SMAC is used for developing proprietary RF transceiver applications using Freescale's MKW01 sub-1 GHz transceiver plus microcontroller. The MKW01 is a system-in-package (SIP) device that includes an ARM Cortex M0+ based microcontroller and a sub-GHz ISM band radio front-end device in an LGA-56 package. Features of the MKW01 include:

- MCU has a 32-bit ARM Cortex M0+ CPU with a full set of peripheral functions
- MCU has 128KB flash and 16KB SRAM
- Full featured, programmable sub-1 GHz transceiver that supports FSK, GFSK, MSK, GMSK, and OOK modulations schemes.
- The MKW01 has internal and external connections between the MCU and transceiver:
  - The MCU communicates with the transceiver through an internally connected SPI port.
  - Several transceiver status bits are also internally or externally connected to MCU GPIO and are capable of generating interrupt requests.

NOTE

It is highly recommended the SMAC user be familiar with the MKW01 device. Additional details can be found in the device data sheet (MKW01Z128) and the MKW01 Reference Manual (MKW01Z128RM).

The MKW01 SMAC is a small codebase that provides simple communication and test applications based on drivers/PHY utilities available as source code. This environment is useful for hardware and RF debug, hardware standards certification, and developing proprietary applications. The MKW01 SMAC is provided as part of the Example Application Demos available for MKW01 and also as a standalone set of files.

To use any of the existing applications available in MKW01 SMAC, users must download and open the available Application Demos in the corresponding development environment (IDE).

SMAC features include:

- Compact footprint:
  - Between 3 to 5 KB of flash required, depending on configuration used
  - From 2 to 253 bytes (maximum packet length) RAM, depending on configuration used
- Very low power, proprietary, bidirectional RF communication link
- The MKW01 radio allows packet filtering by hardware checking the preamble and the synchronization word, which reduces software overhead and memory footprint.
- Broadcast communication

- Unicast communication — MKW01 SMAC includes a Node Address 8-bit field. This allows SMAC to perform unicast transmissions. To change the address of a node, modify this constant: gNodeAddress_c inside the `SMAC_config.h file.` It is set to 0xE0 by default.
- There are no blocking functions within the MKW01 SMAC.
- Easy-to-use sample applications included
- Flexible enough to configure packet header (preamble size, synchronization word size, and synchronization word value)
- Pre-defined settings at four different bands to initialize the SMAC protocol. The currently supported operating frequency bands are:
  - 863 – 870 MHz (Europe)
  - 902 – 928 MHz (US)
  - 920 – 928 MHz (Japan)
  - 950 – 958 MHz

1.1 MKW01 SMAC-based Demonstration Applications

The following is a list of MKW01 SMAC-based demonstration applications:

- PC-based GUI application is provided to easily and directly interface with the PHY layer:
  - Via a virtual COM port (VCOM)
  - At the “Common,” “Transmitter,” and “Receiver” tabs inside the Radio Utility section and SMAC functions (at the “PacketHandler” tab inside the Radio Utility section) without needing to write any code.
  - Allows the user to execute the rest of the demonstration applications included in SMAC, as explained below.
- PC-based Radio Utility GUI — Provides an easy way to test the RF performance of the transceiver for basic transmitter and receiver tests.
- PacketHandler — Users can make a point-to-point connection to send and receive OTA packets. This is useful for performing PER tests and configuring some OTA packet features such as preamble size and synchronization word, among others.
- Range Demonstration — Users can determine the maximum RF range between two development boards. One board (the local node) is connected to a PC and the other board (the remote node) is placed at varying distances from the local node. It performs a PER test and a RSSI test.
- Low Power Demonstration — Users have one local node connected to the PC and one remote node. The users can configure the remote node in different low-power modes of the MCU and the transceiver. The user selects the configuration on the local node, and it is sent to the remote note through over-the-air (OTA) packets.

1.2 Platform Requirements

The SMAC can be used with any customer target application or board, however, Freescale provides a development platform that consists of two boards to support the MKW01:

- MRB-KW01 daughter card — this platform mounts the MKW01Z128 device, required crystal, RF components, reset button, debug interface, power supply options, UART <> USB serial port connection, and full GPIO interface all on a small single printed circuit board (PCB). See *KW01 Development Hardware Reference Manual (KW01DHRM)* for more information.
- TWR-RF Module — the MRB-KW01 can be plugged into the TWR-RF module that provides additional power supply options, pushbutton switch, indicator LEDS, GPIO extension connections, and a UART <> USB serial port connection. See *TWR-RF Module Reference Manual* for more information.

Although the MRB-KW01 can be used alone, adding the TWR-RF Module as a motherboard provides user interface switches and LEDs.

1.3 MCU Resources Used by SMAC

As stated, the MKW01 contains an MCU and a transceiver in a single package with internal connections between these devices. The SMAC uses MCU resources as follows to support the transceiver use:

- The transceiver has a 32 MHz reference crystal oscillator (Frequency of the oscillator can vary depending on the region). In turn, the transceiver can supply an output clock signal that can be used to supply the MCU clock.
  - The MCU always starts on an internal clock exiting reset.
  - By default, the SMAC for boards with a 32 MHz crystal initiates a 2 MHz clock (crystal oscillator frequency/16) output from the transceiver that is routed to the EXTAL signal of the MCU.
  - By default, the SMAC for boards with a 30 MHz crystal initiates a 30 MHz clock (crystal oscillator frequency/1) output from the transceiver that is routed to the EXTAL xignal of the MCU.
  - For more detail on using the MKW01 clocks, see Chapter 3 of the MCU section of the MKW01 Reference Manual.
- The MCU Hardware SPI module is dedicated to communication to the transceiver and is connected internally.
- The MKW01 provides separate hardware reset signals, one for each of the two devices. Common practice is to control the transceiver reset by the MCU via a GPIO. SMAC assumes this approach.
- Several transceiver status bits are externally connected to MCU GPIO pins, which are capable of generating an interrupt request. SMAC supports this interrupt structure.

1.4 SMAC Basic Initialization

Before transmitting, receiving, or performing any other SMAC operation described in this manual, the system protocol must be initialized to configure the transceiver with correct functional settings and to set SMAC's state machines to known states. To initialize the SMAC, perform the following tasks in order:

NOTE

The transceiver has a crystal-based reference oscillator. In turn, the transceiver can generate a programmable clock output that can be used as the clock source for the MCU. The MKW01 SMAC assumes that this system clock configuration is in use (except during Sleep/Hibernate Low Power modes). For details on this configuration, refer to Section 3.4 of the MCU part of the *MKW01 Reference Manual* (MKW01Z128RM).

Changing the divide ratio of the clock on CLKOUT may cause a change in emissions from the product and require any compliance or certification to become invalid. Perform compliance testing with the MKW01 programmed to the final value of CLKOUT.

1. Initialize MCU interrupts and peripherals (transceiver interacts with the MCU via SPI). This initialization is included in every demo in the Platform_Init(void) function, available as source code. Use the following function as a reference for every application initialization for proper MCU and radio operation.

```
DisableInterrupts();
```

— Initialize GPIO ports and the SPI driver to start communication with the transceiver.

```
Platform_Init();       /* Initialization of MCU, ports and transceiver */
```

— As the MCU starts running with its internal clock, configure the clock output availability and frequency of the transceiver and when it is ready, change the MCU clock source.

```
dummy = MKW01Drv_ReadRegister(0x59);
dummy &= 0xEF;
dummy |= 0x10;
MKW01Drv_WriteRegister(0x59, dummy );

(void)MKW01Drv_Reset();// Reset Radio
MKW01Drv_ConfigureCLKO(ClkOutFxOsc_Div16); //32Mhz/16= 2MHz CLKOUT
/*Initialize clock based on CLKOUT*/
MCG_Setup();
```

2. Set the variable `smacStandalone` that is declared inside SMAC as TRUE.

```
smacStandalone = TRUE;
```

3. Initialize the basic operation settings for SMAC: frequency band, RF mode, and output power:

```
MLMERadioInit();
MLMESetFreqBand(g902__928MHz_c, gRFMode0_c);
MLMEPAOutputAdjust(gMaxOutputPower_c);

EnableInterrupts();
```

4. Reserve the RAM memory space needed by SMAC to allocate the received and transmitted OTA messages by declaring the buffers that must be of the size `gSmacBuffersSize_c`:

```
uint8_t RxDataBuffer[gSmacBuffersSize_c];
rxPacket_t *RxPacket;

uint8_t TxDataBuffer[gSmacBuffersSize_c];
txPacket_t *TxPacket;
```

Chapter 2
Software Architecture

This chapter describes the MKW01 SMAC software architecture. All of the SMAC source code is always included in the application. SMAC is primarily a set of utility functions or building blocks that users can use to build simple communications applications

2.1 Block Diagram

Figure 2-1 shows a simplified MKW01 SMAC block diagram. As shown in Figure 2-1, Note that the various MKW01 SMAC software components shown in Figure 2-1 also specify the implemented file names.

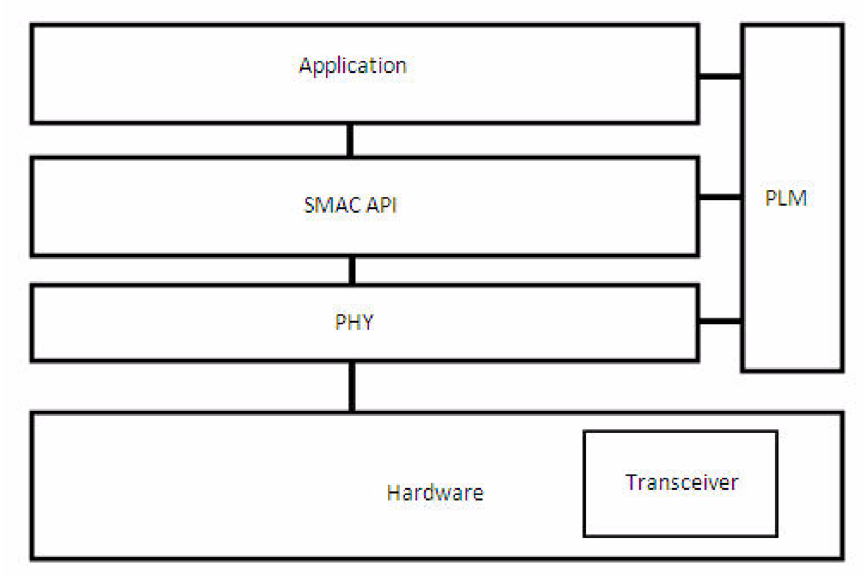

Figure 2-1. MKW01 SMAC Layer Diagram

An application programming interface (API) is implemented in the MKW01 SMAC as a C header file (`.h`) that allows access to the code. The code includes the API to specific functions. Thus, the application interface with the SMAC is accomplished by including the `SMAC_Interface.h` file, which makes reference to the required functions within the SMAC and provides the application with desired functionality.

2.2 RF Transceiver Configuration Support

The MKW01 transceiver is very configurable. The SMAC provides support for initializing the transceiver RF features including frequency of operation, modulation scheme, LNA gain, PA gain, RF port configuration, frame format, and others.

NOTE

MKW01 SMAC projects support only the MKW01-based target boards designated in the files.

2.3 MKW01 SMAC Data Types and Structures

The MKW01 SMAC fundamental data types and defined structures are discussed in the following sections.

2.3.1 Fundamental Data Types

The following list shows the fundamental data types and the naming convention used in the MKW01 SMAC:

uint8_t	Unsigned 8-bit definition
uint16_t	Unsigned 16-bit definition
uint32_t	Unsigned 32-bit definition
int8_t	Signed 8-bit definition
int16_t	Signed 16-bit definition
int32_t	Signed 32-bit definition

These data types are used in the MKW01 SMAC project as well as in the applications projects. They are defined in the `EmbeddedTypes.h` file.

2.3.2 smacPacket_t

This structure defines the characteristics of an SMAC packet that are not visible to the code. That is, the preamble length and synchronization word that are sent automatically by the transceiver as part of any packet transmission.

```
typedef struct smacPacket_tag{
      uint8_t u8SyncWordSize;
      uint8_t *u8SyncWordValue;
      uint16_t u16PreambleLength;
}smacPacket_t;
```

Members

u8SyncWordSize	Length of the synchronization word. It can go from 1 to 8 bytes.
*u8SyncWordValue	Pointer to an array that contains in each element the values for each of the bytes that correspond to the synchronization word.
u16PreambleLength	Size of the preamble to be sent. It can go from 0 to 65535 bytes. The preamble value is always a secession of alternating 1s and 0s.

Usage

This data type does not need to be used from the application.

The variable `smacPacketConfig` of the type `smacPacket_t` is declared inside SMAC and it is updated every time the user changes the synchronization word or the preamble size by calling `MLMESetPreambleLength, MLMESetSyncWordSize, or MLMESetSyncWordValue` functions. It may be useful to monitor this variable while debugging to watch these radio configurations.

2.3.3 rxPacket_t

This structure defines the variable used for MKW01 SMAC received data buffer:

```
typedef struct rxPacket_tag{
        uint8_t    u8MaxDataLength;
        rxStatus_t rxStatus;
        uint8_t    u8DataLength;
        smacPdu_t  smacPdu;
}rxPacket_t;
```

Members

u8MaxDataLength	Max number of bytes to be received.
rxStatus	Indicates the reception state. See `rxStatus_t data` type for more detail.
u8DataLength	Number of received bytes.
smacPdu	The MKW01 SMAC protocol data unit.

Usage

This data type is used by an application in the following manner:

1. Declare a buffer to store a packet to be received OTA. Freescale recommends the size of this buffer to be at least as long as the biggest packet to be received by the application.
2. Declare a pointer of the type rxPacket_t.
3. Initialize the pointer to point to the buffer declared at the first step.
4. Initialize the `u8MaxDataLength` member of the packet structure. The SMAC will filter all the received packets having a payload size bigger than `u8MaxDataLength`.
5. Use the pointer as the argument when calling `MLMERXEnableRequest`:

```
uint8_t RxDataBuffer[gMaxSmacSDULength_c];
rxPacket_t *RxPacket;
RxPacket = (rxPacket_t*)RxDataBuffer;
RxPacket->u8MaxDataLength = gMaxSmacSDULength_c;
RxEnableResult = MLMERXEnableRequest(RxPacket, 0);
```

You can use a variable of the type `smacErrors_t` to store the result of executing `MLMERXEnableRequest` function.

2.3.4 rxStatus_t

This enumeration lists all the possible reception states:

```
typedef enum rxStatus_tag{
        rxInitStatus,
        rxProcessingReceptionStatus_c,
        rxSuccessStatus_c,
        rxTimeOutStatus_c,
        rxAbortedStatus_c,
        rxMaxStatus_c
}rxStatus_t;
```

Members

rxInitStatus	The MKW01 SMAC sets this state when starting a reception.
rxProcessingReceptionStatus_c	This state is set when the MKW01 SMAC is in the middle of receiving a packet.
rxSuccessStatus_c	This is one of the possible finish conditions for a received packet that was successfully received and could be checked by the indication functions.
rxTimeOutStatus_c	This is another of the possible finish conditions for a timeout condition and could be checked by the indication functions.
rxAbortedStatus_c	This is another of the possible finish conditions for a received packet that was not successfully received due to a PHY error or a reception disable having been commanded and checked by the indication functions.
rxMaxStatus_c	This element indicates the total number of possible reception states.

2.3.5 smacPdu_t

This type defines the SMAC's basic protocol data unit:

```
typedef struct smacPdu_tag{
      uint8_t u8DestAddress;
      uint8_t smacPdu[gPhyMaxDataLength_c];
}smacPdu_t;
```

Members

u8DestAddress	Address for the destination node. This member of the smacPdu_t type needs to be initialized only in a transmission packet. Use gBroadcastAddress_c (0xFF) for broadcast. This is not used for a received packet.
smacPdu[gPhyMaxDataLength_c].	Buffer to store the SMAC PDU content.

2.3.6 txPacket_t

This structure defines the type of variable to be transmitted by the MC12311 SMAC. It is located in the `SMAC_Interface.h` file and is defined as follows:

```
typedef struct txPacket_tag{
        uint8_t   u8DataLength;
        smacPdu_t smacPdu;
}txPacket_t;
```

Members

u8DataLength	The number of bytes to transmit.
smacPdu	The MKW01 SMAC protocol data unit.

Usage

This data type is used by an application in the following manner:

1. Declare a buffer to store the packet to be transmitted OTA. Freescale recommends the size of this buffer is at least as long as the biggest packet to be transmitted by the application.
2. Declare a pointer of the type rxPacket_t.
3. Initialize the pointer to point to the buffer declared at the first step.
4. Initialize the u8MaxDataLength member of the packet structure. SMAC will filter all the received packets having a payload size bigger than u8MaxDataLenth.
5. Use the pointer as the argument when calling MCPSDataRequest.

```
uint8_t TxDataBuffer[gMaxSmacSDULength_c];
txPacket_t *TxPacket;
...
TxPacket = (txPacket_t*)TxDataBuffer;
TxPacket->u8DataLength = gMaxSmacSDULength_c;
TxPacket->smacPdu.u8DestAddress = gBroadcastAddress_c /*For broadcast*/
DataRequestResult = MCPSDataRequest(TxPacket);
You can use a variable of the type smacErrors_t to store the result of executing
MCPSDataRequest function.
```

2.3.7 txStatus_t

This enumeration lists all the possible transmitting states. It is located in the `SMAC_Interface.h` file and is defined as follows:

```
typedef enum txStatus_tag{
        txSuccessStatus_c,
        txFailureStatus_c,
        txMaxStatus_c
}txStatus_t;
```

Members

txSuccessStatus_c	Indicates that the transmission was successfully executed.
txFailureStatus_c	Indicates there was an issue when transmitting and it cannot be completed.

txMaxStatus_c	Indicates the total number of possible transmission states.

2.3.8 smacRFConstants_t

This structure is used to store pre-determined parameters needed by SMAC to configure the PHY layer software.

```
typedef struct smacRFConstants_tag {
        uint32_t smacFirstChannelFrequency;
        uint32_t smacChannelSpacing;
        uint16_t smacTotalNumChannels;
        uint16_t smacBitRateReg;
        uint16_t smacFdevReg;
        uint16_t smacNumRssiMeasurements;
        uint8_t  smacRxBwReg;
        uint8_t  smacRxBwAfcReg;
        uint8_t  smacModulationParam;
        uint8_t  smacCcaThreshold;
} smacRFConstants_t;
```

Members

smacFirstChannelFrequency	Four byte field representing the RF frequency of the logical Channel 0 in Hz.
smacChannelSpacing	Four byte field representing the channel spacing in Hz.
smacTotalNumChannels	Two byte field representing the maximum number of channels
smacBitRateReg	Two byte field representing the value of RegBitrate register.
smacFdevReg	Two byte field representing the value of RegFdev registers.
smacNumRssiMeasurements	Two byte field representing the number of RSSI measurements needed by CCA/ED request primitives.
smacRxBwReg	One byte representing the value of RegRxBw register.
smacRxBwAfcReg	One byte representing the value of RegAfcBw register.
smacModulationParam	Field to specify the radio modulation parameters, including modulation type and also modulation shaping (filters).
smacCcaThreshold	One byte representing the value CCA threshold needed by CCA request primitive.

Usage

To change any of these parameters for radio operation, go of one of the three array declarations (one for each RF band) used in SMAC.c:

- `smacRFConstants_FreqBand_863__870MHz[]`
- `smacRFConstants_FreqBand_902__928MHz[]`
- `smacRFConstants_FreqBand_950__958MHz[]`

Freescale recommends using the default settings included in SMAC for optimal RF performance.

2.3.9 smacFrequencyBands_t

This type defines the default frequency bands usable with SMAC. There are three supported frequency bands.

```
typedef enum smacFrequencyBands_tag{
        gSMAC_863_870MHz_c = g863__870MHz_c, /* 863-870   (Europe) */
        gSMAC_902_928MHz_c = g902__928MHz_c, /* 902-928   (US)     */
        gSMAC_950_958MHz_c = g950__958MHz_c, /* 950-958 (Japan)    */
}smacFrequencyBands_t;
```

Members

gSMAC_863_870MHz_c	To select transceiver operation at 868 MHz band (for Europe).
gSMAC_902_928MHz_c	To select transceiver operation at 915 MHz band (for US).
gSMAC_950_958MHz_c	To select transceiver operation at 950 MHz band (for Japan).

2.3.10 channels_t

Definition for RF channels. The number of channel varies in each defined operating band, depending on radio configurations. First logical channel in all bands is 0. It is defined as follows:

```
typedef uint8_t channels_t;
```

To know the total number of channels please refer to the `SMAC.c` file for the constants smacRFConstants_FreqBand_863__870MHz[], smacRFConstants_FreqBand_902__928MHz[], smacRFConstants_FreqBand_950__958MHz[]. These constants of the type smacRFConstants_t also include other information such as the bitrate, frequency deviation and modulation parameters among others.

Members

None

2.3.11 clkoFrequency_t

This enumeration lists all the possible transceiver clock output frequencies. It is located in the `SMAC_Interface.h` file. It is defined as follows:

```
typedef enum clkoFrequency_tag{
        gClko32MHz_c,
        gClko16MHz_c,
        gClko8MHz_c,
        gClko4MHz_c,
        gClko2MHz_c,
        gClko1MHz_c,
        gClkoOutOfRange_c
} clkoFrequency_t;
```

Members

gClko[XY]Hz_c	Constant used to indicate a [XY] Hz transceiver clock output frequency.
gClkoOutOfRange_c	Constant used to indicate the total number of possible clock output valid selections.

2.3.12 scanModes_t

This enumeration lists all the scanning methods available on the MC12311 SMAC. It is located in the `SMAC_Interface.h` file. It is defined as follows:

```
typedef enum scanModes_tag{
        gScanModeCCA_c,
        gScanModeED_c,
        gMaxScanMode_c
}scanModes_t;
```

Members

gScanModeCCA_c	Constant used to indicate a CCA scan mode that is comparing the actual energy level against a defined threshold and binary reporting it bit mapped on a 16-bit variable.
gScanModeED_c	Constant used to indicate an ED scan mode that is reading the actual energy level and storing the value on an 8-bit variable.
gMaxScanMode_c	Constant used to indicate the total number of possible scan methods available on the MKW01SMAC.

2.3.13 smacErrors_t

This enumeration is used as the set of possible return values on most of the MKW01 SMAC API functions and is located in the `SMAC_Interface.h`.

```
typedef enum smacErrors_tag{
        gErrorNoError_c = 0,
        gErrorBusy_c,
        gErrorOutOfRange_c,
        gErrorNoResourcesAvailable_c,
        gErrorNoValidCondition_c,
        gErrorCorrupted_c,
        gErrorMaxError_c
}smacErrors_t;
```

Members

gErrorNoError_c	The MKW01 SMAC accepts the request and processes it. This return value does not necessarily mean that the action requested was successfully executed. It means only that it was accepted for processing by the MKW01 SMAC.
gErrorBusy_c	This constant is returned when the MKW01 SMAC layer is not in an idle state, and it cannot perform the requested action.

gErrorOutOfRange_c	One or more of the parameters used when calling the function are out of the valid values range or one of the pointers is NULL.
gErrorNoResourcesAvailable_c	PHY or other lower layer is not able to process the MKW01 SMAC request, and as a result, the MKW01 SMAC cannot process it. Retries may be implemented at the application layer.
gErrorNoValidCondition_c	Returned when requesting an action on an invalid environment. Requesting MKW01 SMAC operations when MKW01 SMAC has not been initialized or requesting to wake the radio when it was not in low-power mode.
gErrorCorrupted_c	Data corruption was detected while performing the requested action. Currently SMAC implementation does not return this value, because corruption error is handled at the PHY software layer and the packet is discarded before it reaches SMAC layer.
gErrorMaxError_c	This constant indicates the total number of returned constants.

2.3.14 smacRFModes_t

This enumeration represents the possible radio operating modes. The MKW01 PHY supports only the 950–958 MHz, 902–928 MHz, and 863–870 frequency bands. It is defined as follows:

```
typedef enum smacRFModes_tag{
      gRFMode0_c = gPhyMode0_c,
      gRFMode1_c = gPhyMode1_c,
      gRFMode2_c = gPhyMode2_c,
      gRFMode3_c = gPhyMode3_c,
      gRFMaxMode_c
} smacRFModes_t;
```

Members

- In gSMAC_950_958MHz_c frequency band:
  - gRFMode0_c: 50 kbps; filtered FSK; mod index = 1.0; channel spacing (kHz) = 200
  - gRFMode2_c: 200 kbps; filtered FSK; mod index = 1.0; channel spacing (kHz) = 600
- In gSMAC_902_928MHz_c frequency band:
  - For Americas:
  - gRFMode0_c: 50 kbps; filtered FSK; mod index = 1.0; channel spacing (MHz) = 1
  - gRFMode1_c: 150 kbps; filtered FSK; mod index = 1.0; channel spacing (MHz) = 2
  - gRFMode2_c: 200 kbps; filtered FSK; mod index = 1.0; channel spacing (MHz) = 2
  - For Japan:
  - gRFMode0_c: 50 kbps; filtered FSK; mod index = 1.0; channel spacing (kHz) = 200
  - gRFMode1_c: 100 kbps; filtered FSK; mod index = 1.0; channel spacing (kHz) = 400
  - gRFMode2_c: 200 kbps; filtered FSK; mod index = 1.0; channel spacing (kHz) = 600
  -
  -

- In gSMAC_863_870MHz_c frequency band:
  - gRFMode0_c: 4.8 kbps; filtered FSK; mod index = 1.0; channel spacing (kHz) = 200
  - gRFMode1_c: 100 kbps; filtered FSK; mod index = 1.0; channel spacing (kHz) = 400

2.3.15 versionedEntity_t

This enumeration lists all the elements (hardware or software) that have a version attached. It is located in the `SMAC_Interface.h` file.

```
typedef enum versionedEntity_tag{
      gSwPhyVersion_c,
      gSwSmacVersion_c,
      gHwIcVersion_c,
      gMaxVersionedEntity_c
}versionedEntity_t;
```

Members

gSwPhyVersion_c	Use this to request the MKW01 PHY version.
gSwSmacVersion_c	Use this to request the MKW01 SMAC version.
gHwIcVersion_c	Use this to request the transceiver/SoC version number.
gMaxVersionedEntity_c	Defines the total number of versions entities.

2.3.16 packetConfig_t

This structure contains the fields to configure the packet sent by the transceiver. It is defined as follows:

```
typedef struct packetConfig_tag{
      uint16_t u16PreambleSize;
      uint8_t  u8SyncWordSize;
      uint8_t  *pu8SyncWord;
} packetConfig_t;
```

Members

u16PreambleSize	This is a 16-bit value that specifies the length in bytes of the preamble for the packet. Values from 0 to 65535 are required.
u8SyncWordSize	This is an 8-bit value that specifies the length in bytes of the synchronization word for the packet. Values from 0 to 8 are required.
*pu8SyncWord	This is a pointer to an 8-byte array that is used to store the value of the synchronization word.

2.3.17 lnaGainValues_t

Enumeration used to specify all the valid values for the MLMELnaGainAdjust function. It is defined as follows.

```
typedef enum lnaGainValues_tag{
       gLnaGain_0_c = 0,
       gLnaGain_1_c,
       gLnaGain_2_c,
       gLnaGain_3_c,
       gLnaGain_4_c,
       gLnaGain_5_c,
       gLnaGain_6_c,
       gLnaGain_7_c,
}lnaGainValues_t;
```

Members

gLnaGain_[X]_c	This is an enumeration member used to indicate the value of the LNA gain.

2.3.18 listenResolution_t

Enumeration used to specify an enumeration value that represents the possible three values (64 µs, 4.1 ms, or 262 ms) for the listen mode.

```
typedef enum listenResolution_tag{
      gListenRes_01_c = 0x10,
      gListenRes_02_c = 0x20,
      gListenRes_03_c = 0x30
}listenResolution_t;
```

Members

gListenRes_[XX]_c	This is an enumeration member used to indicate the value of the listen resolution.

Chapter 3
Primitives

The following sections provide a detailed description of MKW01 SMAC primitives associated with the MKW01 SMAC application API.

3.1 MCPSDataRequest

This data primitive is used to send an over-the-air (OTA) packet. This is an asynchronous function, which means it asks the MKW01 SMAC to transmit an OTA packet, but transmission could continue after the function returns.

Prototype

```
smacErrors_t MCPSDataRequest(txPacket_t *psTxPacket);
```

Arguments

txPacket_t *psTxPacket	Pointer to the packet to be transmitted.

Returns

gErrorNoError_c	Everything is ok and the transmission will be performed.
gErrorOutOfRange_c	One of the members in the pTxMessage structure is out of range (not valid buffer size or data buffer pointer is NULL).
gErrorBusy_c	The radio is performing another action and could not attend this request.
gErrorNoValidCondition_c	The MKW01 SMAC has not been initialized.
gErrorNoResourcesAvailable_c	The PHY or other lower layer cannot process an MKW01 SMAC request, so MKW01 SMAC cannot process it.

Usage

- SMAC must be initialized before calling this function.
- Declare a variable of the type smacErrors_t to save the result of the function execution.
- Prepare the txPacket_t parameter as explained in Chapter 2 txPacket_t declaration and usage.
- Call the MCPSDataRequest function.
- If the result of the call of the function is different than gErrorNoError_c, the application should handle the error returned. For instance, if the result is gErrorBusy_c the application should wait for the radio to finish a previous operation.

```
uint8_t TxDataBuffer[gMaxSmacSDULength_c];
```

```
txPacket_t *TxPacket;
smacErrors_t smacError;
...
TxPacket = (txPacket_t*)TxDataBuffer;
TxPacket->u8DataLength = gMaxSmacSDULength_c;
TxPacket->smacPdu.u8DestAddress = gBroadcastAddress_c /*For broadcast*/
smacError = MCPSDataRequest(TxPacket);
...
```

Implementation

This MCPSDataRequest primitive makes a call to the PHY function:
`PhyDataRequest(smacProccesPacketPtr.smacTxPacketPointer, gImmediateAction_c);`

which implements the setup for a Tx request and initiates data transmission. The PhyDataRequest function adjusts FillFifoBlock and FifoThreshold according to gFifoThreshold_c constant in the `Phy.h` file and to the packet's data length if it is less than gFifoThreshold_c (7 bytes). It configures the DIO pins that come from the transceiver as follows:

- DIO0 as TxPacketSent signaling
- DIO1 as TxFifoLevel signaling

3.2 MCPSDataConfirm

NOTE

A callback function is a function that is located within the user's application code and is needed by the MKW01 SMAC.

This callback function needs to be placed in the application. The MCPSDataConfirm primitive reports the results of a request to transfer data.

This function is called by the MKW01 SMAC in an interrupt context. The application code placed in the function should not take the CPU a long time to process.

This function should be declared in the upper layer (application layer) at least as an empty function.

Prototype

```
void MCPSDataConfirm(txStatus_t TransmissionResult);
```

Arguments

txStatus_t TransmissionResult: The result of the transmission attempt.

Returns

Nothing

Usage

This callback function will be executed after the transmission operation is completed, whether it is successful or not. Use `txStatus_t` argument to define the application execution path. Don't try to execute an SMAC primitive inside a callback function. Instead, set a flag to indicate at the application level that the callback was already executed.

NOTE

Consider that this function is executed during an interrupt execution and then follow all the recommendations for interrupts execution timing and context.

Implementation

The MCPSDataConfirm function is called by the PHY in the callback PhyPdDataConfirm(), which indicates the success of a PHY data request operation. The function is called after the last bit of the last data byte is sent over the air.

3.3 MCPSDataIndication

This callback function needs to be placed in the application. This allows the SMAC to call this function when a data packet is received by the radio to be processed by the application.

This function is called by SMAC in an interruption context. The application code placed in the function should not take the CPU long to process.

Prototype

```
void MCPSDataIndication(rxPacket_t *gsRxPacket);
```

Arguments

rxPacket_t *gsRxPacket: Pointer to the structure where the reception results have been stored.

Returns

Nothing

Usage

This callback function will be executed after the receive operation is completed, whether it is successful or not. Use the rxStatus element of rxPacket_t structure to define the application execution path. Do not try to execute an SMAC primitive inside a callback function. Instead, set a flag to indicate to the application level that the callback was already executed.

NOTE

This function is executed during an interrupt execution. Follow all the recommendations for interrupt execution timing and context.

Implementation

The MCPSDataIndication function is called by the PHY in the callback PhyPdDataIndication(), which indicates that a data packet has been received over the air. The function is called after the CRC of an Rx frame.

SMAC performs a basic packet filtering mechanism based on two parameters that are contained in the received message: data length and destination address. The criteria is as follows:

A received packet is passed to the upper layer only if:

- The data length is less than the maximum data length configured in the Rx structure (u8MaxDataLength).
- The destination address is a broadcast address (0xFF) or it matches the node address value (gNodeAddress_c == u8destAddress).

If a received packet is rejected by SMAC, the upper layer is not informed and SMAC will automatically set the transceiver to receive mode again.

3.4 MLMERXEnableRequest

Places the radio into receive mode on the channel pre-selected by MLMESetChannelRequest ().

Prototype

```
smacErrors_t MLMERXEnableRequest(rxPacket_t *gsRxPacket, uint32_t u32Timeout);
```

Arguments

rxPacket_t *gsRxPacket: Pointer to the structure where the reception results will be stored.

uint32_t u32Timeout: 32-bit timeout value, this is directly the value that is stored on the MCU’s timer configuration.

Returns

gErrorNoError_c	Everything is ok and the reception will be performed
gErrorOutOfRange_c	One of the members in the rxPacket_t structure is out of range (not valid buffer size or data buffer pointer is NULL).
gErrorBusy_c	The radio is performing another action and could not attend this request.
gErrorNoValidCondition_c	The MKW01 SMAC has not been initialized.
gErrorNoResourcesAvailable_c	The PHY or other lower layer cannot process a MKW01 SMAC request, so the MKW01 SMAC cannot process it.

Usage

- SMAC must be initialized before calling this function.
- Declare a variable of the type smacErrors_t to save the result of the function execution.
- Prepare the rxPacket_t parameter as explained in chapter 2 rxPacket_t declaration and usage.

- Call MLMERXEnableRequest function.
- If the result of the call of the function is different to gErrorNoError_c, the application may handle the error returned. For instance, if the result is gErrorBusy_c the application should wait for the radio to finish a previous operation.

```
uint8_t RxDataBuffer[gMaxSmacSDULength_c];
rxPacket_t *RxPacket;
smacErrors_t smacError;

RxPacket = (rxPacket_t*)RxDataBuffer;
RxPacket->u8MaxDataLength = gMaxSmacSDULength_c;
smacError = MLMERXEnableRequest(RxPacket, 0);

...
```

NOTE

- The return of anything different than gErrorNoError_c implies that the receiver did not go into receive mode.
- 32-bit timeout value of zero causes the receiver to never timeout and stay in receive mode until a valid data packet is received or the MLMERXDisableRequest function is called.
- To turn off the receiver before a valid packet is received, the MLMERXDisableRequest call can be used.

Implementation

This primitive makes a call to the PHY function:

```
PhyPlmeRxRequest(pAuxPacket, &smacLastDataRxParams, gImmediateAction_c);
```

This PHY function implements the setup for an Rx request action.

The PHY layer initiates the reception operation by performing the following actions:

- Configure DIO pins mapping registers:
  - DIO0 as SyncAddress
  - DIO1 as FifoLevel
  - DIO5 as ClkOut.
- Configure FillFifoCondition: the data received over the air is stored in FIFO only if the SyncAddress was received.
- Configure RegFifoThresh: set RegFifoThresh with FIFO threshold.
- Configure the RSSI threshold in RssiThreshold register. For correct operation of the AGC, RssiThreshold in RegRssiThresh must be set to the sensitivity of the receiver. The receiver will remain in wait mode until RssiThreshold is exceeded. SMAC configures the sensitivity of the transceiver at –127 dBm, which is the maximum sensitivity level.

3.5 MLMESetPreambleLength

Function used to change the preamble length of a transmitted packet.

Prototype

```
smacErrors_t MLMESetPreambleLength(uint16_t u16preambleLength);
```

Arguments

u16preambleLength: The preamble size in bytes that is set for the OTA packet.

Returns

gErrorNoValidCondition_c	The MKW01 SMAC has not been initialized.
gErrorBusy_c	The radio is performing another action and could not attend this request.
gErrorNoError_c	The preamble size was changed.

Usage

Call this function before sending a packet to configure the preamble size.

Implementation

This function calls the PHY function:

PhyPib_RFUpdatePreambleLength();

The transceiver's registers that are modified by this function are:

MKW01_Reg_PreambleMsb and MKW01_Reg_PreambleLsb.

To see more details about transceiver's registers, refer to the MKW01 reference manual (document number MKW01Z128RM), available from Freescale.com.

3.6 MLMESetSyncWordSize

Function used to change the synchronization word size in bytes. Values from 0 to 8 required.

Prototype

```
smacErrors_t MLMESetSyncWordSize(uint8_t u8syncWordSize);
```

Arguments

`u8syncWordSize: Value that represents the synchronization word size.`

Returns

gErrorNoValidCondition_c	The MKW01 SMAC has not been initialized.
gErrorBusy_c	The radio is performing another action and could not attend this request.
gErrorNoError_c	The synchronization word size was changed.

Usage

Call this function before sending a packet to configure the synchronization word size.

Implementation

This function calls the PHY function:

```
PhyPib_SetSyncWordSize(u8syncWordSize);
```

The transceiver’s register that is modified by this function is:

```
MKW01_Reg_SyncConfig
```

To see more details about transceiver’s registers refer to the MKW01 reference manual (document number MKW01Z128RM), available from Freescale.com.

3.7 MLMESetSyncWordValue

Function used to change the synchronization word value.

Prototype

```
smacErrors_t MLMESetSyncWordValue(uint8_t *u8syncWordValue);
```

Arguments

*u8syncWordValue	Pointer to the first element of an array that contains the values for the synchronization word.

Returns

gErrorNoValidCondition_c	The MKW01 SMAC has not been initialized.
gErrorBusy_c	The radio is performing another action and could not attend this request.
gErrorNoError_c	The synchronization word value was changed.

Usage

Call this function before sending a packet to configure the synchronization word value.

Implementation

This function calls the PHY function:

```
PhyPib_SetSyncWordValue(syncWordValueReg, syncValue);
```

The transceiver’s registers that are modified by this function are:

```
MKW01_Reg_SyncValue1-8
```

To see more details about the transceiver’s registers, refer to the MKW01 reference manual (document number MKW01Z128RM), available from Freescale.com.

3.8 MLMERXDisableRequest

Returns the radio to idle mode from receive mode.

Prototype

```
smacErrors_t MLMERXDisableRequest(void);
```

Arguments

None

Returns

gErrorNoError_c	The message was aborted or disabled
gErrorNoValidCondition_c	The radio is not in Rx state
gErrorBusy_c	The radio is performing another action and could not attend this request.

Usage

`Call MLMERXDisableRequest ()`

NOTE

This function can be used to turn off the receiver before a timeout occurs or when the receiver is in the always-on mode.

Implementation

This function calls the PHY function:

```
PhyPlmeForceTrxOffRequest();
```

It aborts the current requested action, puts the PHY in the idle state, and sets the transceiver in standby mode. It also disables any previous timeout programmed.

3.9 MLMELinkQuality

This function returns an integer value that is the RSSI value (received signal strength indicator) from the last received packet in the form:

```
      dBm = -(MLMELinkQuality());
```

Prototype

```
uint8_t MLMELinkQuality(void);
```

Arguments

None.

Returns

uint8_t	8-bit value representing the link quality value. Returns the result in smacLastDataRxParams.linkQuality.
Zero	The MKW01 SMAC has not been initialized.

Usage

Call the MLMELinkQuality()

Implementation

This function reads only the stored value in smacLastDataRxParams.linkQuality. This element contains the RSSI value calculated by the PHY layer during the last reception using the transceiver's RSSI block to measure the amount of energy available within the receiver channel bandwidth. The measurement time is equivalent to two symbol periods, and it starts only if the SFD has been successfully received.

The measurement is triggered by the SyncAddress interrupt. Afterwards:

- PHY layer sets RssiStart bit in RegRssiConfig register to start a new RSSI measurement.
- Waits until the RSSI measurement is finished by pooling RssiDone bit from RegRssiConfig register.
- Reads the RSSI value from RegRssiValue and adds this value to the one stored in gRssiAccumulator PHY global variable.

3.10 MLMERadioInit

The MLMERadioInit function initializes the radio parameters.

Prototype

```
smacErrors_t MLMERadioInit(void);
```

Arguments

None

Returns

gErrorNoError_c	Radio initialization was performed successfully.

Usage

Call the `MLMERadioInit`() function directly.

NOTE

Not executing this function could cause many MKW01 SMAC API functions to return gErrorNoValidCondition_c error code.

Implementation

This function initializes the transceiver and sets the registers with the default values for SMAC proper operation. The default values for each register can be found in the structure MKW01DefaultRegisterValues in file TransceiverDrv.c.

3.11 MLMEPacketConfig

This function sets the following parameters for OTA packets in the radio's registers:

- Preamble size
- Synchronization word size
- Synchronization word value

All the packets sent after the call of this function will have the configuration set by the last call of MLMEPacketConfig.

Prototype

```
smacErrors_t MLMEPacketConfig(packetConfig_t *pPacketCfg);
```

Arguments

packetConfig_t *pPacketCfg: Structure that contains the fields to configure the sent packet

Returns

gErrorNoError_c	The packet has been successfully configured.
gErrorBusy_c	The MKW01 SMAC is busy in other radio activity like transmitting/receiving data or performing a channel scan.
gErrorNoValidCondition_c	The MKW01 SMAC has not been initialized.

Usage

- Declare a pointer to `packetConfig_t` variable
- Set the parameters of the packetConfig_t variable (PreambleSize, SyncWordSize and SyncWord).
- Use that variable as the argument when calling MLMEPacketConfig

```
packetConfig_t pPacketCfg;
uint8_t u8SyncWord[8];
pPacketCfg.pu8SyncWord = & u8SyncWord[0];
pPacketCfg.u8SyncWordSize=SyncConfig_SyncSize_3;
pPacketCfg.u16PreambleSize=0x0004;
MLMEPacketConfig(&pPacketCfg);
```

3.12 MLMESetInterpacketRxDelay

This sets the inter-packet delay for the packet handler.

Prototype

```
smacErrors_t MLMESetInterPacketRxDelay(uint8_t u8InterPacketRxDelay);
```

Arguments

uint8_t u8InterPacketRxDelay: 8 bit variable that represents the interpacket delay in ms.

Returns

gErrorNoError_c	The action is performed.
gErrorNoValidCondition_c	The MKW01 SMAC is not initialized.
gErrorBusy_c	The MKW01 SMAC is performing a reception, transmission or scan sequence.
gErrorOutOfRange_c	The requested delay is not valid.

Usage

Call the function with the selected delay, which is in ms.

```
uint8_t u8InterPacketRxDelay;
u8InterPacketRxDelay=10;
MLMESetInterPacketRxDelay(u8InterPacketRxDelay);
```

Implementation

This function calls the PHY function:

```
PhyPib_SetInterPacketRxDelay(u8InterPacketRxDelay);
```

The transceiver's register that is modified by this function is:

```
MKW01_Reg_PacketConfig2
```

More information about the InterPacket Rx Delay can be found in the MKW01 reference manual (document number MKW01Z128RM), available from Freescale.com.

3.13 MLMESleepRequest

The call of this function causes the transceiver to go into sleep mode.

Prototype

```
smacErrors_t MLMESleepRequest();
```

Arguments

None.

Returns

gErrorNoError_c	The radio has been set in sleep mode
gErrorBusy_c	When MKW01 SMAC is busy in other radio activity like transmitting/receiving data or performing a channel scan.

gErrorNoValidCondition_c	The MKW01 SMAC is not initialized.

Usage

```
Call MLMESleepRequest()
```

Implementation

This function calls the PHY function:

```
MKW01Drv_RadioSleepReq();
```

3.14 MLMERssi

This call starts an RSSI read and returns the amount of energy present at the preset frequency band at the time of the reading. This is similar to an energy detect (ED) cycle.

The MLMERssi function returns an 8-bit value that represents:

RSSI = –(returned value / 2) dBm

Prototype

```
uint8_t MLMERssi(void);
```

Arguments

None

Returns

uint8_t	An unsigned 8-bit value representing the energy on the current channel.

Usage

Call `MLMERssi`()

Implementation

This function calls the PHY function:

`PhyPib_GetRssi`();

which triggers an RSSI read and reads the value of the `MKW01_Reg_RssiValue` register.

3.15 MLMELnaGainAdjust

This function adjusts the LNA gain. For more info about LNA gain, refer to the MKW01 reference manual (document number MKW01Z128RM), available from Freescale.com.

Prototype

```
smacErrors_t MLMELnaGainAdjust(lnaGainValues_t gainValue);
```

Arguments

lnaGainValues_t gainValue: LNA valid gain values.

Returns

gErrorNoError_c	If the action was performed correctly.
gErrorBusy_c	When MKW01 SMAC is busy in other radio activity like transmitting/receiving data or performing a channel scan.
gErrorNoValidCondition_c	If the MKW01 SMAC is not initialized.

Usage

1. Declare a lnaGainValues_t variable.
2. Assign a valid gain value to the variable.
3. Call the function with the previous variable and its parameters.

```
lnaGainValues gainValue;
gainValue=gLnaGain_4_c;
MLMELnaGainAdjust(gainValue);
```

3.16 MLMESetChannelRequest

This sets the frequency on which the radio will transmit or receive.

Prototype

```
smacErrors_t MLMESetChannelRequest(channels_t newChannel);
```

Arguments

channels_t newChannel: An 8-bit value that represents the requested channel.

Returns

gErrorNoError_c	The channel set has been performed
gErrorBusy_c	The MKW01 SMAC is busy in other radio activity like transmitting/receiving data or performing a channel scan.
gErrorOutOfRange_c	The requested channel is not valid
gErrorNoValidCondition_c	The MKW01 SMAC is not initialized.

Usage

Call the function `MLMESetChannelRequest(newChannel);`

NOTE

Each band and mode have a different valid channel range. Be sure to enter a valid channel for the band and mode in which the radio is transmitting.

3.17 MLMEGetChannelRequest

This function returns the current channel.

Prototype

```
channels_t MLMEGetChannelRequest(void);
```

Arguments

None

Returns

channels_t (uint8_t)	The current RF channel.
gErrorBusy_c	The MKW01 SMAC is busy in other radio activity like transmitting/receiving data or performing a channel scan.
gErrorNoValidCondition_c	The MKW01 SMAC is not initialized.

Usage

Call `MLMEGetChannelRequest();`

3.18 MLMESetFreqBand

This function sets the frequency band selected and the PHY mode in which the radio will be working/transmitting.

Prototype

```
smacErrors_t MLMESetFreqBand(smacFrequencyBands_t freqBand, smacRFModes_t phyMode);
```

Arguments

uint8_t freqBand: Frequency band in which the radio will be working.

uint8_t phyMode: PHY mode of the frequency band to work.

Returns

gErrorNoError_c	The channel set has been performed.

gErrorBusy_c	The MKW01 SMAC is busy in other radio activity like transmitting/receiving data or performing a channel scan.
gErrorNoValidCondition_c	The MKW01 SMAC is not initialized.

Usage

Call the function with the valid parameters.

Valid parameters for the frequency band are:

- g902__928MHz_c
- g863__870MHz_c
- g902__928MHz_c

Valid parameters for the PHY mode are:

- gRFMode0_c
- gRFMode1_c
- gRFMode2_c
- gRFMode3_c
-

Usage:

- SMAC must be initialized before calling this function.
- Call this function with the right parameters:

```
MLMESetFreqBand (g902__928MHz_c, gRFMode0_c);
```

NOTE

Be sure to enter a valid value for the frequency band and the PHY mode in which it will be working.

3.19 MLMEPAOutputAdjust

This function adjusts the output power of the transmitter.

Prototype

```
smacErrors_t MLMEPAOutputAdjust(uint8_t u8PaValue);
```

Arguments

uint8_t u8PaValue	8-bit value for the output power desired. Values 0 – 31 are required.

Returns

gErrorOutOfRange_c	u8Power exceeds the maximum power value gMaxOutputPower_c (0x1F).
gErrorBusy_c	The MKW01 SMAC is performing a reception, transmission or scan sequence.
gErrorNoError_c	The action is performed.

gErrorNoValidCondition_c	The MKW01 SMAC is not initialized.

Usage

Call `MLMEPAOutputAdjust(u8PaValue);`

NOTE

Be sure to enter a valid value for the PA output adjust.

Implementation

This function calls the PHY function:

```
PhyPib_SetCurrentTransmitPower(u8PaValue);
```

3.20 MLMEEnablePABoost

This function enables the high sensitivity mode to reduce the noise floor in the receiver. This function enables the PA1 and PA2 or both.

Prototype

```
smacErrors_t MLMEEnablePABoost(uint8_t u8PABoostCfg);
```

Arguments

uint8_t u8PABoostCfg: parameter to enable PA1 and/or PA2.

Returns

gErrorNoError_c	If the action is performed.
gErrorBusy_c	If the MKW01 SMAC is performing a reception, transmission or scan sequence.
gErrorNoValidCondition_c	If the MKW01 SMAC is not initialized.

Usage

1. Define the PA to enable, valid values are:
- gDisablePA_Boost_c -> Use this option for output in RFIO only.
- gEnablePA1_Boost_c
- gEnablePA2_Boost_c
- gEnablePABoth_Boost_c -> Use this option for output in PA1 + PA2
2. Call the function with the selected parameter.
```
MLMEEnablePABoost(gEnablePABoth_Boost_c);
```

Implementation

This function calls the PHY function:

PhyPib_PABoostCfg (u8PABoostCfg);

3.21 MLMEDisablePABoost

This function disables the high-sensitivity mode and sets the sensitivity to normal.

Prototype

```
smacErrors_t MLMEDisablePABoost(void);
```

Arguments

None.

Returns

gErrorBusy_c	The MKW01 SMAC is performing a reception, transmission or scan sequence
gErrorNoError_c	The action is performed.
gErrorNoValidCondition_c	The MKW01 SMAC is not initialized.

Usage

Call `MLMEDisablePABoost()`

Implementation

This function calls the PHY function:

```
PhyPib_PABoostCfg (gDisablePA_Boost_c);
```

With gDisablePA_Boost_c as the parameter.

3.22 MLMEPHYSoftReset

The MLMEPHYSoftReset function is called to perform a software reset to the radio, PHY, and MKW01 SMAC state machines.

Prototype

```
smacErrors_t MLMEPHYSoftReset(void);
```

Arguments

None

Returns

gErrorNoError_c	If the action is performed.
gErrorNoValidCondition_c	If the MKW01 SMAC is not initialized.
gErrorBusy_c	If the MKW01 SMAC is performing a reception, transmission or scan sequence.

Usage

Call `MLMEPHYSoftReset();`

Implementation

This function calls the PHY function:

```
PhyPlmeForceTrxOffRequest();
```

3.23 MLMESetClockRate

This function is called to set the desired clock out from the radio.

Prototype

```
smacErrors_t MLMESetClockRate(clkoFrequency_t clkFreq);
```

Arguments

clkoFrequency_t	An enumeration value that represents radio clock out frequency (CLKO).

Returns

gErrorNoError_c	The action is performed
gErrorBusy_c	The MKW01 SMAC is busy in other radio activity like transmitting/receiving data or performing a channel scan.
gErrorNoValidCondition_c	The MKW01 SMAC is not initialized.

Usage

Call the function with its valid parameters `MLMESetClockRate(clkFreq);`. Valid parameters for clkFreq are:

```
gClko32MHz_c
gClko16MHz_c
gClko8MHz_c
gClko4MHz_c
gClko2MHz_c
gClko1MHz_c
```

3.24 MLMEStandbyRequest

This primitive puts the radio in standby mode.

Prototype

```
smacErrors_t MLMEStandbyRequest(void);
```

Arguments

None.

Returns

gErrorNoError_c	The action is performed.
gErrorNoValidCondition_c	The MKW01 is not initialized.
gErrorBusy_c	The MKW01 SMAC is performing a reception, transmission, or scan sequence.

Usage

Call `MLMEStandbyRequest();`

Implementation

This function calls the PHY function:

```
MKW01Drv_RadioStandByReq(0);
```

3.25 MLMEListenRequest

This primitive gets the radio in listen mode. It is useful to get lower power consumption.

Prototype

```
smacErrors_t MLMEListenRequest(listenResolution_t listenResolution, uint8_t listenCoef);
```

Arguments

listenResolution_t listenResolution: An enumeration value that represents the possible three values (64 µs, 4.1 ms or 262 ms).

uint8_t listenCoef: 8 bit value to define the duration of the listen mode.

Returns

gErrorNoError_c	If the action is performed.
gErrorBusy_c	When MKW01 SMAC is busy in other radio activity like transmitting/receiving data or performing a channel scan.
gErrorNoValidCondition_c	If the MKW01 SMAC is not initialized.

Usage

Declare the listenCoef variable.

Define the value of the variable according to the time in listen mode expected. The time in the listen mode is calculated with the next equation:

$$t_{ListenX} = ListenCoefX \cdot ListenResolX$$

Call the function with the listen resolution mode and the listen coefficient calculated with the previous equation.

```
uint8_t listenCoef;
```

```
listenCoef=0x1F;
MLMEListenRequest(listenResolutionX, listenCoef);
```

Implementation

This function calls various PHY functions to configure the radio properly to enter listen mode in the next sequence:

```
PhyPib_SetListenResRx((uint8_t)listenResolution);
 PhyPib_SetListenCoefRx(listenCoef);


 PhyPib_SetOperationMode(OpMode_StandBy);
 PhyPib_SetOperationMode(OpMode_Listen_On);
```

3.26 XCVRContReset

This function asserts the reset line of the transceiver, shutting it down and holding it in its lowest power mode.

Prototype

```
void XCVRContReset(void);
```

Arguments

None

Returns

Nothing

Usage

Call `XCVRContReset();`

Implementation

This function calls the PHY function:

```
MKW01Drv_AssertReset();
```

3.27 XCVRRestart

This function deasserts the reset line, thus it powers the transceiver up. The function is called to release reset after calling XCVRContReset.

Prototype

```
void XCVRRestart(void);
```

Arguments

None

Returns

Nothing

Usage

Call `XCVRRestart();`

Implementation

This function calls the PHY function:

```
MKW01Drv_DeassertReset();
```

3.28 MLMEGetRficVersion

This function is used to read the version number of the software modules inside the MKW01 SMAC platform.

Prototype

```
macErrors_t MLMEGetRficVersion(versionedEntity_t Entity, uint8_t *Buffer);
```

Arguments

versionedEntity_t entity: PHY or MKW01 SMAC version

uint8_t *Buffer: to return either MKW01 SMAC or PHYversion

Returns

gErrorNoError_c	If the action is performed.
gErrorBusy_c	If the MKW01 SMAC is busy.
gErrorOutOfRange_c	If the requested Entity is not part of the stored ones.

Usage

1. Create a pointer used as buffer where the information will be stored.
2. Call the function with the version entity that is requested and the pointer to the buffer as parameters.

```
uint8_t bufferVersion[];
MLMEGetRficVersion(gSwSmacVersion_c, bufferVersion);
```

3.29 MLMEScanRequest

This function scans a selected channel and returns the RSSI in the selected channel.

Prototype

```
smacErrors_t MLMEScanRequest(channels_t u8ChannelToScan, uint8_t *u8ChannelScanResult);
```

Arguments

channels_t u8ChannelToScan: Channel to be scanned.

uint8_t *u8ChannelScanResult: Pointer to where the result of the scan will be stored.

Returns

gErrorNoError_c	Everything is normal and the scan will be performed.
gErrorBusy_c	The radio is performing another action.
gErrorNoValidCondition_c	The MKW01 SMAC has not been initialized.

Usage

Declare a pointer where the result will be stored.

Call the function with the selected channel to be scanned and the pointer where the information will be stored.

```
uint8_t *u8ChannelScanResult;
MLMEScanRequest(u8ChannelToScan, u8ChannelScanResult);
```

NOTE

Be sure to enter a valid channel according to the frequency band and the PHY mode in which the radio is working.

3.30 SMACDisableInterrupts

This function disables the interrupts that are used by MKW01 SMAC.

Prototype

```
void SMACDisableInterrupts(void);
```

Arguments

None

Returns

Nothing

Usage

Call `SMACDisableInterrupts();`

3.31 SMACEnableInterrupts

This function enables the interrupts that are used by MKW01 SMAC.

Prototype

```
void SMACEnableInterrupts(void);
```

Arguments

None

Returns

Nothing

Usage

Call `SMACEnableInterrupts();`