Simple Media Access Controller (SMAC) for the HCS08

Reference Manual

Document Number: SMAC08RM
Rev. 2.0
03/2012

freescale™
semiconductor

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

***For Literature Requests Only*:**
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-521-6274 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Contents

About This Book

Audience v
Organization v
Revision History v
Definitions, Acronyms, and Abbreviations vi
References vi

Chapter 1
SMAC Introduction

1.1 Features 1-2
1.2 Hardware Configurations - Required MCU Resources 1-2
1.3 Freescale Integrated Development Environment (IDE) - BeeKit 1-2
1.4 Using BeeKit 1-3
1.4.1 BeeKit Concepts 1-3

Chapter 2
Software Architecture

2.1 Block Diagram 2-1
2.2 Hardware Support 2-2
2.3 SMAC Data Types 2-2
2.3.1 rxPacket_t 2-3
2.3.2 rxStatus_t 2-3
2.3.3 smacPdu_t 2-4
2.3.4 txPacket_t 2-4
2.3.5 txStatus_t 2-5
2.3.6 channels_t 2-5
2.3.7 clkoFrequency_t 2-6
2.3.8 scanModes_t 2-6
2.3.9 smacErrors_t 2-7
2.3.10 smacTestMode_t 2-7
2.3.11 timerTimeBase_t 2-8
2.3.12 versionedEntity_t 2-8

Chapter 3
Primitives

3.1 Application API 3-1
3.1.1 MCPSDataRequest 3-1
3.1.2 MLMELinkQuality 3-2
3.1.3 MLMERXDisableRequest 3-3
3.1.4 MLMERXEnableRequest 3-3
3.1.5 MLMEDozeRequest 3-4
3.1.6 MLMEEnergyDetect 3-6

3.1.7 MLMEFEGainAdjust 3-6
3.1.8 MLMEGetChannelRequest 3-7
3.1.9 MLMEHibernateRequest 3-8
3.1.10 MLMEPAOutputAdjust 3-8
3.1.11 MLMEPHYSoftReset 3-9
3.1.12 MLMERadioInit 3-10
3.1.13 MLMESetChannelRequest 3-10
3.1.14 MLMESetClockRate 3-11
3.1.15 MLMESetTmrPrescale 3-12
3.1.16 MLMEWakeRequest 3-13
3.1.17 MLMEPHYXtalAdjust 3-13
3.1.18 XCVRContReset 3-14
3.1.19 XCVRRestart 3-15
3.1.20 MLMEGetPromiscuousMode 3-15
3.1.21 MLMEGetRficVersion 3-16
3.1.22 MLMEScanRequest 3-16
3.1.23 MLMESetPromiscuousMode 3-17
3.1.24 MLMETestMode 3-18
3.1.25 SMACDisableInterrupts 3-19
3.1.26 SMACEnableInterrupts 3-20
3.1.27 MCPSDataComfirm 3-20
3.1.28 MCPSDataIndication 3-21
3.1.29 MLMEResetIndication 3-21
3.1.30 MLMEScanComfirm 3-22
3.1.31 MLMEWakeComfirm 3-22

About This Book

This guide provides a detailed description of the Freescale Simple Media Access Controller (SMAC) for the HCS08 MCU based devices.

This document replaces the SMAC User's Guide currently at Rev 1.5.

Audience

This document is intended for application developers 802.15.4 PHY wireless applications. The latest version of the Freescale SMAC for the HCS08 is incorporated into the Freescale BeeKit Wireless Connectivity Toolkit.

Organization

This document is organized into the following chapters.

Chapter 1 **SMAC Introduction** — This chapter introduces SMAC features and functionality.

Chapter 2 **Software Architecture** — This chapter describes SMAC software architecture.

Chapter 3 **Primitives** — This chapter provides a detailed description of SMAC primitives.

Revision History

The following table summarizes revisions to this document since the previous release (Rev. 1.0).

Revision History

Location	Revision
Throughout	Removed MC13219x references.
Throughout	Added information for MC13234 devices.
Chapter 1	Removed OTA, SNET, Low Power Bell, and Accelerometer references
Chapter 3	Removed UseExternalClock, UseMCUClock, and MCUInit sections.

Definitions, Acronyms, and Abbreviations

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

BDM debugger	A debugger using the BDM interface for communication with the MCU. An example is the P&E BDM Multilink debugger for HCS08.
BDM	Background Debug Module
EVB	Evaluation Boards - this term covers the 13192-EVB, 13192-SARD and 1320X-QE128-EVB boards.
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 -, address -, data - and checksum field. The 16 bit address field allows a memory space for up to 64 KB. 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).
HIWAVE	P&E HCS08 debugger GUI.
CPROG	P&E HCS08 flash programming tool called from HIWAVE. The tool is also available in a command line version where scripts can be made.
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] Zigbee.hlp (see Test Tool installation directory.\help)

Chapter 1
SMAC Introduction

Freescale supplies a number of IEEE 802.15.4 Standard compliant platforms for the 2.4 GHz ISM band. Applications software for a number of these devices is based on the HCS08 8-bit MCU architecture, and the Simple Media Access Controller (SMAC) is a simple ANSI C based code stack available as sample source code. The SMAC is used for evaluating new designs, doing RF testing, and developing simple proprietary IEEE 802.15.4 transceiver applications.

The SMAC can be used with the following device families:

1. MC1320x - this device is an IEEE 802.15.4 transceiver only, and is used typically with an MC9S08GB/GT MCU. The transceiver is controlled primarily through a SPI communication port. This device is a second generation to the MC1319x.
2. MC1321x - this device combines the MC1320x transceiver and an MC9S08GB60A MCU in a single package. This configuration is architecturally similar to an MC1320x application; the transceiver is still controlled primarily through a SPI communication port.
3. MC1323x - this device a full system on chip where the RF transceiver is integral to the IC and appears as a peripheral to the MCU bus. As a result, the SMAC applies differently as to how it communicates with and controls the transceiver, and also how communication services are provided. SMAC includes support for two devices of this platform:
   — MC13233
   — MC13234

NOTE

- The MC1320x devices can be used with other MCU architectures. In this situation, SMAC can be used as a template and adapted to the targeted architecture.
- The SMAC is intended as an evaluation/test utility and for simple applications. For more complex applications, the user is strongly recommended to consider Freescale's other IEEE 802.15.4 MAC-based stacks.
- When applying SMAC to any given device, it is recommended that the user become familiar with the device via the appropriate Reference Manual and Data Sheet.

1.1 Features

- Compact footprint:
  - Between 3 and 5K FLASH depending on configuration and platform used
  - From [(23 to 293) plus maximum packet length] bytes of RAM depending on configuration and platform used.
- MC1320x, MC1321x, and MC1323x compatible
- Proprietary, bi-directional RF communication link - custom frame format
- Low power features
- ANSI C source code targeted for the HCS08 core
- Low priority IRQ
- Easy-to-use sample applications included
- Programmable protocol function to allow automatically ignore standard IEEE 802.15.4 Standard MAC packets

1.2 Hardware Configurations - Required MCU Resources

There are two fundamental hardware configurations used with SMAC:

1. Separate transceiver and MCU - the MC1320x and MC1321x employ this configuration. The MC1321x is unique in that the system connections are on board a single package. MCU support requirements include:
   - System clock requirements - the transceiver requires a 16 MHz crystal for its reference oscillator. Common practice is to use a clock output (CLKO) from the transceiver to drive the MCU as an external clock source. It is still required, however, that the MCU have some internal/local clock source for startup.
   - Transceiver/MCU communication SPI port - four MCU GPIO pins are required for the SPI port (CLK, MISO, MOSI, and $\overline{CE}/\overline{SS}$). Although the SPI port could be provided by a “bit-banged” software driven interface, most common practice is a dedicated SPI peripheral module.
   - Addition control GPIO - additional MCU GPIO are required for transceiver control and status. At minimum, three IO are needed (Reset, RTXEN, and ATTN), however, as many as seven GPIOs can be used
   - One MCU pin capable of external interrupt request - this may be a dedicated IRQ pin, a KBI pin, or any GPIO capable of generating an IRQ based on an incoming signal transition.

1.3 Freescale Integrated Development Environment (IDE) - BeeKit

The SMAC is incorporated into the Freescale BeeKit Wireless Connectivity Toolkit. The incorporation of SMAC into BeeKit makes it easier for users to employ and customize SMAC. SMAC has evolved with the HCS08 platforms, and as a result, changes have occurred with later versions. The primary changes are as follows:

- No blocking functions are on the most recent SMAC implementation
- Modified arguments and return types

- Specific part number removed from the API function names

If migrating from a previous SMAC version, Freescale recommends comparing specific function details.

In addition to the SMAC codebase, Freescale includes a suite of demonstration applications for SMAC.

A list of SMAC-based applications includes:

- Generic Demo — Provides an empty template to start coding an SMAC application.
- Connectivity Test — Provides an easy way to test the RF performance of the transceiver for basic transmitter and receiver tests. It includes TX test modes (continuous tx, modulated, unmodulated, etc.), Packet Error Rate PER) tests, and Range tests (LQI measurements).
- Wireless UART — Application allows the Freescale IEEE 802.15.4 family of boards to communicate at standard serial baud rates from one board to another providing a wireless RS-232 or USB virtual COM port.
- Range Demonstration — Employs the Range Demonstration Plus application to determine /evaluate the maximum wireless node board RF range.
- Repeater Demonstration — Uses the Repeater application to repeat all the messages received in a specific channel to extend the range of a point-to-point link.
- Simple ZigBee Test Client (SZTC) — Application allows users to exercise the SMAC primitives by sending control frames through the Serial/USB port.

For more details about running the SMAC applications refer to the *SMAC for the HCS08 Demonstration Applications User's Guide* (SMAC08DAUG).

To use any of the existing applications available in SMAC, users must first generate the application as a project in a BeeKit solution. For more information about BeeKit, BeeKit Projects, and BeeKit Solutions, refer to the *BeeKit Wireless Connectivity Toolkit User's Guide* (BKWCTKUG) and the BeeKit on-line help.

1.4 Using BeeKit

SMAC is released in an independent codebase that is part of the Freescale BeeKit Wireless Connectivity Tool. To create a project for SMAC, users must employ the BeeKit Codebase that contains the SMAC code. For more information on BeeKit, refer to the *BeeKit Wireless Connectivity Toolkit User's Guide* (BKWCTKUG).

For more information on the Codebase as it applies to SMAC, see *SMAC for the HCS08 Demonstration Applications User's Guide* (SMAC08DAUG).

1.4.1 BeeKit Concepts

This section highlights some basic BeeKit terms and concepts. Again, for a more detailed description of BeeKit, refer to the *BeeKit Wireless Connectivity Toolkit User's Guide* (BKWCTKUG).

Codebase — A group of source files, configuration files, and generation rules that serve as a repository from which all BeeKit demos, templates, and other applications are generated.

Solution	A group of projects which are linked to a specific folder within the file structure of the computer and in this file structure, all other projects will generate their own folders.
BeeKit Project	A specific group of files that are exported from BeeKit to create a CodeWarrior project in the form of an XML file.
XML Project File	A BeeKit generated XML file ready for import into CodeWarrior. A CodeWarrior MCP file is generated from the BeeKit generated XML project file.

Figure 1-1 shows the folder structure of a typical project generated using SMAC Codebase for BeeKit.

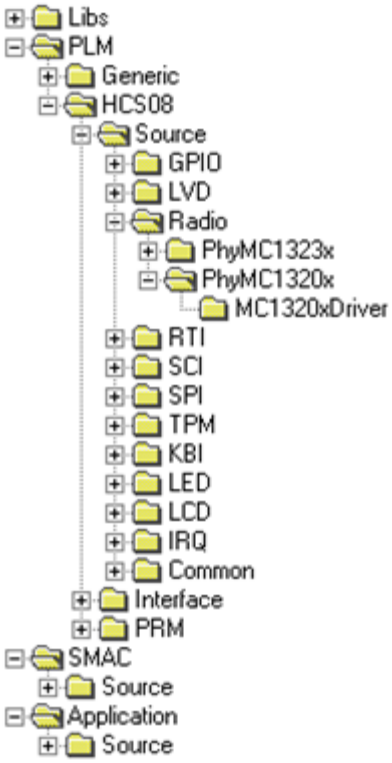

Figure 1-1. Example Project Folder Structure

Chapter 2
Software Architecture

This chapter describes the SMAC software architecture. In this version of the SMAC, all of the SMAC source code is included in the application. This represents a far less complex project that can be compiled using CodeWarrior Special Edition.

2.1 Block Diagram

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

- Security Module

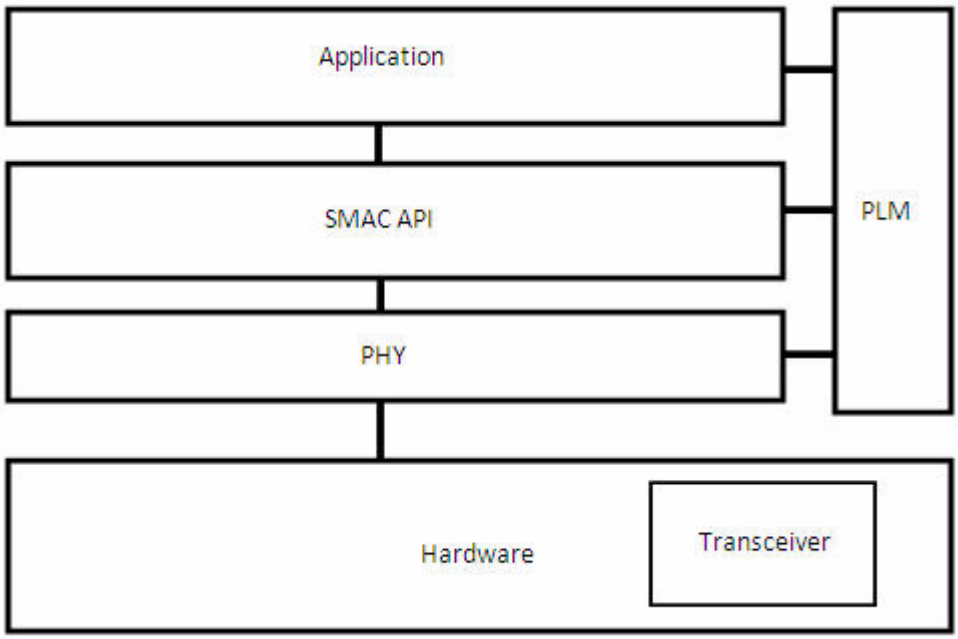

Figure 2-1. SMAC Layer Diagram

The Security modules and its API are included in the BeeKit project when the corresponding SMAC BeeKit property is set to “True”:

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

2.2 Hardware Support

This section describes the SMAC hardware support for the MCU, transceiver, antenna switch, and LNA. The SMAC can be easily ported to any MCU, as long as the system requirements are met. To port to another MCU, perform the following tasks:

1. Remove the `mc9s08gb60.c/mc9sqe128.c/mc1323c.c` file from the BeeKit project.
2. Add the specific c support file for the required MCU
3. Update the linker file (`.prm` file)
4. Update the `derivative.h` file to include the specific header file
5. Update the SMAC lower layers (MCU and SPI configuration files)
6. Update the application drivers (such as SCI, LED, KBI, etc.)

The SMAC supports the following Freescale transceiver families:

- MC1320x
- MC1321x
- MC1323x, including MC13233 and MC13234

The changes required in the software to support any of the Freescale transceivers are generated automatically by BeeKit after exporting a solution with the projects correctly configured. For more information on exporting projects, see the *BeeKit Wireless Connectivity Toolkit User's Guide* (BKWCTKUG), and the BeeKit on-line help.

SMAC projects support different target boards (those available in Freescale ZigBee kits) in the files. But it is easy to port an SMAC application from a standard target to a custom target by using the Platform Editor Wizard in Beekit.

The SMAC contains support for hardware that is controlled by GPIOs configured at the target board header file: These hardware options are as follows:

- External Antenna Switch
- LNA
- PA

For the External Antenna Switch, SMAC supports the following configurations:

- For the MC1320x and MC1321x transceivers, a single antenna using an external antenna switch controlled by an MCU GPIO pin.
- For MC1320x and MC1321x transceivers, a single antenna using an external antenna switch controlled by the BT_Bias pin of the transceiver.
- For MC1320x and MC1321x transceivers, a single antenna using the internal antenna switch.

2.3 SMAC Data Types

The following list shows the fundamental data types and the naming convention used in 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
int162_t	Signed 16 bit definition
int32_t	Signed 32 bit definition

These data types are used in the SMAC project as well as in the Applications projects and they are defined in the EmbeddedTypes.h file.

2.3.1 rxPacket_t

This structure defines the variable to be used for SMAC receiving 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`	Data reception buffer, see `smacPdu_t` data type for more detail.

Usage

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

1. Declare a pointer to `rxPacket_t` variable
2. Assign that variable the pointer to an array selected for receiving.
3. Use that variable as the argument when calling `MLMERXEnableRequest`

```
rxPacket_t *RxPacket;
uint8_t RxDataBuffer[130];
...
RxPacket = (rxPacket_t*)RxDataBuffer;
RxEnableResult = MLMERXEnableRequest(RxPacket, 0);
...
```

2.3.2 rxStatus_t

This enumeration list all the possible reception states:

```
typedef enum rxStatus_tag{
  rxInitStatus,
  rxProcessingReceptionStatus_c,
  rxSuccessStatus_c,
```

```
  rxTimeOutStatus_c,
  rxAbortedStatus_c,
  rxMaxStatus_c
}rxStatus_t;
```

Members

`rxInitStatus`	SMAC set this state when starting a reception.
`rxProcessingReceptionStatus_c`	This state is set when the SMAC is in the middle of receiving a packet.
`rxSuccessStatus_c`	This is one of the possible finish condition for a receiving packet that was successfully received and could be checked at the indication functions.
`rxTimeOutStatus_c`	This is one of the possible finish condition for a time out condition and could be checked at the indication functions..
`rxAbortedStatus_c`	This is one of the possible finish condition for a receiving packet that was not received due to sync lost or an reception disable has been commanded and could be checked at the indication functions.
`rxMaxStatus_c`	This element indicate the total number of possible reception states.

2.3.3 smacPdu_t

This structure defines the variable to be used for SMAC data buffer:

```
typedef struct smacPdu_tag {
  uint8_t  reserved[2];
  uint8_t  u8Data[1];
}smacPdu_t;
```

Members

reserved	Reserved.
u8Data	The data buffer to transmit.

2.3.4 txPacket_t

This structure defines the variable to be transmitted by 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 data buffer to transmit.

Usage

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

1. Declare a pointer to txPacket_t variable
2. Assign that variable the pointer to an array selected for transmitting.
3. Assign the appropriate value to the `txPacket_t` element of the `txStatus_t` structure
4. Use that variable as the argument when calling `MCPSDataRequest`

```
uint8_t TxDataBuffer[TX_DATA_BUFFER_SIZE];
txPacket_t *TxPacket;
...
TxPacket = (txPacket_t*)TxDataBuffer;
TxPacket->u8DataLength = MAX_DATA_LENGTH;
DataRequestResult = MCPSDataRequest(TxPacket);
...
```

2.3.5 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 can be validated at the confirm function.
`txFailureStatus_c`	Indicates there was an issue when transmitting and it can not be completed.
`txMaxStatus_c`	Indicates the total number of possible transmission states.

2.3.6 channels_t

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

```
typedef enum channels_tag{
  gChannel11_c = 0x0B,
  gChannel12_c,
  gChannel13_c,
  gChannel14_c,
  gChannel15_c,
  gChannel16_c,
  gChannel17_c,
  gChannel18_c,
  gChannel19_c,
  gChannel20_c,
  gChannel21_c,
  gChannel22_c,
  gChannel23_c,
```

```
  gChannel24_c,
  gChannel25_c,
  gChannel26_c,
  gTotalChannels_c
}channels_t;
```

Members

`gChannelXX_c`	Constant used to refer the IEEE 802.15.4 channel XX
`gTotalChannels_c`	Constant indicates the total number of channels.

2.3.7 clkoFrequency_t

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

```
typedef enum clkoFrequency_tag{
  gClko16MHz_c,
  gClko8MHz_c,
  gClko4MHz_c,
  gClko2MHz_c,
  gClko1MHz_c,
  gClko62p5kHz_c,
  gClko32p786kHz_c,
  gClko16p393kHz_c,
  gClkoOutOfRange_c
}clkoFrequency_t;
```

Members

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

2.3.8 scanModes_t

This enumeration list all the scanning methods available on SMAC and 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 actual energy level against a defined threshold and binary reporting it bit mapped on a 16 bit variable.
`gScanModeED_c`	Constant used to indicate a ED scan mode, that is reading the actual energy level and storing such value on a 8 bits variable.
`gMaxScanMode_c`	Constant indicates the total number of possible scan methods available at SMAC.

2.3.9 smacErrors_t

This enumeration is used as the set of possible return values on most of the 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	SMAC accept the request and will process it. NOTE that this return value does not necessary mean that the action requested was successful executed, it just means that is that accepted to be process by SMAC
gErrorBusy_c	This constant is returned when SMAC layer is not on a idle state, then it can not perform the requested action.
gErrorOutOfRange_c	One or more of the parameters used when calling the function are out of the valid values range.
gErrorNoResourcesAvailable_c	PHY or other lower layer is not able to process SMAC request then SMAC can not process it.
gErrorNoValidCondition_c	When requesting an action on an invalid environment. Requesting SMAC operations when SMAC has not been initialized or requesting awake the radio when it was not in low power mode.
gErrorCorrupted_c	Not used.
gErrorMaxError_c	This constant indicates the total number of return constants.

2.3.10 smacTestMode_t

This enumeration is used as the set of possible radio test modes that SMAC implements and is located in the SMAC_Interface.h:

```
typedef enum smacTestMode_tag{
  gTestModePRBS9_c,
  gTestModeForceIdle_c,
  gTestModeContinuousRx_c,
  gTestModeContinuousTxModulated_c,
  gTestModeContinuousTxUnmodulated_c,
  gMaxTestMode_c
}smacTestMode_t;
```

Members

gTestModePRBS9_c

gTestModeForceIdle_c

gTestModeContinuousRx_c

gTestModeContinuousTxModulated_c

gTestModeContinuousTxUnmodulated_c

gMaxTestMode_c

2.3.11 timerTimeBase_t

This enumeration is used as the set of possible transceiver timer base, is located in the SMAC_Interface.h. and is defined for the MC1323x platform as:

```
typedef enum timerTimeBase_tag {
    gTimeBase500kHz_c = 2,
    gTimeBase250kHz_c,
    gTimeBase125kHz_c,
    gTimeBase62p5kHz_c,
    gTimeBase31p25kHz_c,
    gTimeBase16p625kHz_c,
    gMaxTimeBase_c
}timerTimeBase_t;
```

And defined for the other supported platforms as:

```
typedef enum timerTimeBase_tag{
    gTimeBase2MHz_c = 0,
    gTimeBase1MHz_c,
    gTimeBase500kHz_c,
    gTimeBase250kHz_c,
    gTimeBase125kHz_c,
    gTimeBase62p5kHz_c,
    gTimeBase31p25kHz_c,
    gTimeBase16p625kHz_c,
    gMaxTimeBase_c
}timerTimeBase_t
```

Members

gTimeBaseXXHz_c	Use to set the transceiver timer base at XX Hz.
gMaxTimeBase_c	Used for validation purposes.

2.3.12 versionedEntity_t

This enumeration lists all the elements (HW or SW) that have a version attached and is located in the SMAC_Interface.h file.

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

Members

gSwSmacVersion_c	Use this to request the SMAC version.

`gHwIcVersion_c` Use this to request the Transceiver/SoC version number.

`gMaxVersionedEntity_c` Defines the total number of versions entities.

Chapter 3
Primitives

The following sections provide a detailed description of SMAC primitives associated with the SMAC API. The SMAC for S08 codebase supports the MC13202, MC13213, and MC1323x families.

NOTE

Some primitives may have different usage or affect as applied to the different device families. These are noted in the individual primitive descriptions.

3.1 Application API

This section highlights MCU resource requirements and details the application API primitives.

3.1.1 MCPSDataRequest

This data primitive sends an over-the-air packet. This is an asynchronous function, which means that the function does not run in the same thread as the caller (the application). It asks SMAC to transmit a packet and returns control to the application, when transmission is completed the result of the operation is reported by the MCPSDataConfirm callback which is called in an interrupt context.

Prototype

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

Arguments

```
txPacket_t *
```

Pointer to the packet to be transmitted

Returns

gErrorNoError_c	No errors and the transmission is confirmed
gErrorOutOfRange_c	One of the members in the txPacket_t structure is out of range (not a valid buffer size or data buffer pointer is NULL)
gErrorNoResourcesAvailable_c	The radio is performing another action and could not attend this request.
gErrorNoValidCondition_c	SMAC has not been initialized

Usage

- Declare an array where transmission data is stored

- Declare a pointer of the txPacket_t structure type
- Assign that pointer variable the pointer to the array selected for transmitting.
- Assign the appropriate value to the `u8DataLength` element of the txPacket_t structure
- Use the txPacket_t pointer as the argument when calling MCPSDataRequest

```
uint8_t TxDataBuffer[TX_DATA_BUFFER_SIZE];
txPacket_t *TxPacket;
...
TxPacket = (txPacket_t*)TxDataBuffer;
TxPacket->u8DataLength = MAX_DATA_LENGTH;
DataRequestResult = MCPSDataRequest(TxPacket);
...
```

3.1.2 MLMELinkQuality

Link quality is a relative measure of the strength or quality of a received packet. A value called a Link Quality Indicator (LQI) is returned by the transceiver as a result of a received packet.

This function returns an integer byte value for the link quality that is valid for the last received packet. This binary value can be translated into a measurement of received signal strength in dBm via the equation:

$$dBm = - (\text{Link Quality}[7:0]/2)_{dec}$$

Prototype

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

Arguments

None

Returns

`UINT8`

8 bit value representing the link quality value

Usage

- Call MLMERXEnableRequest to enable the receive function
- Wait for a MCPSDataIndication that a valid packet has been received
- ....
- Call MLMELinkQuality.

NOTE

- The value of the link quality is valid and preserved until the next received packet.
- This format does not meet IEEE 802.15.4 Standard requirement.

3.1.3 MLMERXDisableRequest

This function returns the radio to idle mode from receive mode.

Prototype

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

Arguments

None

Returns

gErrorNoError_c	When the message was aborted or disabled
gErrorOutOfRange_c	If the Radio was not in Rx state

Usage

Simply 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.

3.1.4 MLMERXEnableRequest

This function places the radio into receive mode in anticipation of receiving a data packet.

- The receiver is enabled on the channel previously selected by MLMESetChannelRequest ().
- The function call passes an argument that sets a timeout value for the receive mode -
  - The argument is a 32-bit value, however, the maximum value is 0x00FFFFFF
  - The total time period equals: value x (transceiver timer clock period). The timer clock period is determined by the frequency set by the MLMESetTmrPrescale function, see Section 3.1.15, “MLMESetTmrPrescale”.
  - If the timeout value is set to zero, the timeout function is disabled and once enabled, receive mode will continue until a valid packet is received or a MLMERXDisableRequest is executed.

Prototype

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

Arguments

`rxPacket_t *`

Location of the buffer to store the incoming data.

`uint32_t`

32-bit timeout value.

Returns

gErrorNoError_c	Everything is ok and the reception will be requested to the lower layers
gErrorOutOfRange_c	One of the members in the `rxPacket_t` structure is out of range (no valid buffer size or data buffer pointer is NULL).

gErrorNoResourcesAvailable_cRadio is performing another action and could not attend this request.

gErrorNoValidCondition_cSMAC has not been initialized.

Usage

- Declare a pointer to `rxPacket_t` variable
- Assign that variable the pointer to an array buffer to be used for the receive packet data.
- Use that variable as the argument when calling `MLMERXEnableRequest`

```
rxPacket_t *RxPacket;
uint8_t RxDataBuffer[130];
...
RxPacket = (rxPacket_t*)RxDataBuffer;
RxEnableResult = MLMERXEnableRequest(RxPacket, 0);
...
```

NOTE

- Return 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.

3.1.5 MLMEDozeRequest

The transceiver has low power modes with the reference oscillator disabled (Hibernate) or enabled (Doze). The Doze request function allows the user to put the radio either in Normal Doze Mode (without CLKO but with automatic timeout wake-up) or Acoma Doze Mode (with CLKO but without timeout):

- The reference oscillator is enabled during Doze mode and can be used in two ways -
  - — The reference oscillator must be enabled to drive the transceiver timer block if the timer is to be used for wake-up from Doze
  - — The transceiver provides the CLKO output clock signal which is used by the MCU as a clock source. While the transceiver is in low power mode, this output can be kept active to supply clock to the MCU
- The function call passes an argument that sets a timeout/wake-up value for Doze mode -
  - — The argument is a 32-bit value, however, the maximum value is 0x00FFFFFF
  - — The total time period equals: value x (transceiver timer clock period). The timer clock period is determined by the frequency set by the MLMESetTmrPrescale function, see Section 3.1.15, “MLMESetTmrPrescale”.

— If the wake-up value is set to zero, the wake-up function is disabled and once enabled, Doze mode will continue until a MLMEWakeRequest is executed. CLKO is enabled; this is known as Acoma.

- The transceiver can always be wakened by a MLMEWakeRequest function call or asserting the hardware reset to the transceiver

NOTE

This primitive can be used only on the MC1320x and MC1321x platforms. For the MC1323x platform, it is invalid.

Prototype

```
smacErrors_t MLMEDozeRequest(uint32_t u32Timeout);
```

Arguments

`u32Timeout`

Is the automatic wake-up time for this mode.

Returns

gErrorNoError_c	The Radio has been set in Doze mode
gErrorBusy_c	Radio is performing another action and could not attend this request.

Usage

To use Acoma Mode, users should call this function with “0” as parameter. Then, the SMAC sets the radio in Acoma Mode with CLKO Output MLMEWakeRequest. To exit from this state, users should call the MLMEWakeRequest function.

To use Normal Doze Mode, users should call this function with the desired timeout. When the SMAC places the radio in Doze Mode, the user is given 128 CLKO clock cycles more before disabling CLKO; if the MCU is using this signal as a clock source, the user should first switch to the MCU internal clock before making the MLMEDozeRequest call

Once the timeout is reached, the radio wakes-up and performs an IRQ Interrupt (attended by the SMAC).

NOTE

- The maximum allowed CLKO frequency during Doze Mode or Acoma Mode is equal to or less than 1MHz. The SMAC does no error checking for this when entering Doze Mode. The user application must handle this situation.
- Having CLKO available during Doze mode is optional and is controlled by setting transceiver Register 0x7, Bit 9. The reset value/default for this bit does not provide CLKO in Doze Mode. If the radio is reset, this bit needs to be set in order to provide the CLKO.

3.1.6 MLMEEnergyDetect

The transceiver can turn on the receiver and sample any energy present on the selected channel frequency. This function calls an energy detect (ED) cycle and returns detected the energy value via the return argument.

- The receiver is enabled on the channel previously selected by MLMESetChannelRequest ().
- Similar to the LQI, this function returns an integer byte value. This binary value can be translated into a measurement of received signal strength in dBm via the equation:

$$dBm = - (\text{Link Quality}[7:0]/2)_{dec}$$

Prototype

```
uint8_t MLMEEnergyDetect(void);
```

Arguments

None

Returns

```
UINT8
```

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

Usage

The function is simply called. The returned value is the energy detected on the presently selected channel.

NOTE

SMAC provides the MLMEScanRequest function as the default means to do energy detect or a CCA function across multiple channels. However, the transceiver is also capable doing a single CCA cycle instead of a simple ED; see the appropriate device reference manual. The MLMEEnergyDetect function source can be used as a prototype to provide a CCA function.

3.1.7 MLMEFEGainAdjust

The transceiver has an offset parameter that compensates the energy detection scheme used during an ED, CCA, or LQI measurement. This function allows users to adjust the results (plus or minus) as required for loss or gain from external components.

NOTE

This parameter is called POWER_COMP[7:0] on the MC13202 and MC13213 platforms. It is called CCA_OFFSET_CMP on the MC1323x platform

- The default value out of reset is 0x8D

- Either in the software initialization or via the MLMEFEGainAdjust function, it is recommended that this parameter be set to 0x9B. See appropriate family Reference Manual.
- The offset value must be INCREMENTED to LOWER the reported value. An increment of 4 to the offset value will affect a -1 dB change, and vice versa.

Prototype

```
smacErrors_t MLMEFEGainAdjust(uint8_t);
```

Arguments

`uint8_t`

8 bit binary value for the offset adjust.

Returns

gErrorNoError_c The power compensation value have been set on the radio.

Usage

Use a standard offset value of 0x9B. Use MLMEFEGainAdjust to increase or decrease the offset.

3.1.8 MLMEGetChannelRequest

The IEEE 802.15.4 Standard supports 16 set channels/frequencies. This function returns the current active channel. If the current frequency setting does not match a standard channel setting, an error is returned as `gChannelOutOfRange_c`

Prototype

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

Arguments

`None.`

Returns

{gChannel11_c to gChannel26_c}The current RF channel..

gChannelOutOfRange_cIf current channel could not be detected.

Usage

Simply call MLMEGetChannelRequest().

3.1.9 MLMEHibernateRequest

Similar to the MLMEDozeRequest, this call places the transceiver in a low power mode. This function places the transceiver into Hibernate mode where the reference oscillator is disabled (see below):

- For the MC13202, and MC1321x devices, the transceiver timer module and CLKO outputs are not functional.
- The MC1323x platform is unique in that the transceiver is not a separate device and the reference oscillator is not isolated to the transceiver. For this platform, this function only disables the clock to the transceiver/modem.
- The transceiver can be wakened by a MLMEWakeRequest function call or asserting the hardware reset to the transceiver. Note that MLMEWakeRequest function does not perform any operation for MC1323x platform devices. It only returns gErrorNoError_c

Prototype

```
smacErrors_t MLMEHibernateRequest(void);
```

Arguments

None

Returns

gErrorNoError_c	The Radio has been set in Hibernate mode.
gErrorBusy_c	Radio is performing another action and could not attend this request.

Usage

When function is called, the reference oscillator is disabled and CLKO (if enabled) is disabled after 128 clock cycles (similar to Doze). If the MCU is using this signal as a clock source, the user should first switch to internal clock and then make the MLMEHibernateRequest() call to place the radio into Hibernate mode.

3.1.10 MLMEPAOutputAdjust

This function is called to set the output power of the radio transmitter. A single parameter is passed allowing 16 different power settings:

- Power settings vary from 0 being the lowest to 15_{dec} the highest.
- Macros MIN_POWER, MAX_POWER, and NOMINAL_POWER are also acceptable

Table 3-1 shows the typical output power versus power settings for the MC13202/MC13213 families

Prototype

```
smacErrors_t MLMEPAOutputAdjust(uint8_t U8PaValue);
```

Arguments

```
uint8_t
```

An 8-bit value for the output power desired.

Returns

gErrorNoError_c If the action was performed.

gErrorOutOfRange_c If power exceeds the maximum power value.

Usage

This function can be called with an argument between 0 and 15 or using the MIN_POWER, MAX_POWER and NOMINAL_POWER macros.

Table 3-1. U8PaValue vs MC13202 and MC13213 TX Power Out

U8PaValue	Typical Differential Power at Output Contact (dBm)	U8PaValue	Typical Differential Power at Output Contact (dBm)
0b0000	-16.6	0b1000	-1.0
0b0001	-16.0	0b1001	-0.5
0b0010	-15.3	0b1010 (Nominal)	0.0
0b0011	-14.8	0b1011	0.4
0b0100	-8.8	0b1100	2.1
0b0101	-8.1	0b1101	2.8
0b0110	-7.5	0b1110	3.5
0b0111	-6.9	0b1111	3.6

3.1.11 MLMEPHYSoftReset

The MLMEPHYSoftReset function is called to perform a soft reset to the transceiver for all families. This function restarts PHY and SMAC states machines.

Prototype

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

Arguments

None

Returns

gErrorNoError_c If the action was performed.

Usage

Simply call the MLMEPHYSoftReset () function directly.

NOTE

When this function is called, a reset indication should be sent back to the application through the MLMEResetIndication(void) callback function. It is up to the application to handle this event.

3.1.12 MLMERadioInit

The MLMERadioInit function initializes the Radio parameters. This function must be called prior to any radio operation. The default parameters are set inside the function:

Initialize PHY using default parameters -

- Set Channel to channel 11
- Set max power level
- Set CCA Threshold to 129_{dec}

Prototype

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

Arguments

None

Returns

gErrorNoError_c Radio initialization was performed successfully.

gErrorNoResourcesAvailable_cRadio initialization couldn't be performed.

Usage

Simply call the `MLMERadioInit`() function directly.

NOTE

- Not executing this function could cause many SMAC API functions to return gErrorNoValidCondition_c error code.
- If user wishes to change initialization parameters, they must be changes within the function.

3.1.13 MLMESetChannelRequest

This function programs the frequency synthesizer for the radio and sets the actual frequency for which the radio transmits and receives. The function sets one of sixteen channels whose frequency corresponds to the IEEE 802.15.4 Standard designations. The channel argument sets Channel 11 through Channel 26 as determined by argument value “gChannel11_c” to “gChannel26_c”.

Prototype

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

Arguments

`channels_t`

An 8 bit value that represents the requested channels. Values gChannel11_c to gChannel26_c are expected.

Returns

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

Usage

Simply call MLMESetChannelRequest (channel requested).

3.1.14 MLMESetClockRate

For the MC13202 and MC13213 families, the transceiver can provide clock output CLKO that can in turn be used as the external clock source for the MCU. This function is called to set the desired CLKO output frequency. The CLKO frequency is divided down from the reference oscillator frequency of 16 MHz.

NOTE

This primitive can be used only on the MC1320x and MC1321x platforms. For the MC1323x platform it is invalid.

Prototype

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

Arguments

`clkoFrequency_t`

An 8-bit value that sets radio clock out frequency (CLKO).

Returns

gErrorNoError_c	If the action is performed
gErrorOutOfRange_c	If frequency exceeds the maximum CLKO frequency value

Usage

Call MLMESetClockRate ([*gClko16MHz_c - gClko16p393kHz_c*]). The CLKO frequency can be set from 16 MHz down to 16.393 kHz. The enumeration `clkoFrequency_t` that is used as an argument for MLMESetClockRate function is written to the Radio CLKO Control Register (0x0A). See specific device Reference Manual.

```
typedef enum clkoFrequency_tag
```

```
{
  gClko16MHz_c,
  gClko8MHz_c,
  gClko4MHz_c,
  gClko2MHz_c,
  gClko1MHz_c,
  gClko62p5kHz_c,
  gClko32p786kHz_c,
  gClko16p393kHz_c,
  gClkoOutOfRange_c
} clkoFrequency_t
```

3.1.15 MLMESetTmrPrescale

The transceiver on all three families has a dedicated timer block. This function changes the input clock rate to the timer counter and determines the base clock rate for all the timer comparators.

Prototype

```
smacErrors_t MLMESetTmrPrescale(timerTimeBase_t );
```

Arguments

```
timerTimeBase_t
```

Enumeration value that represents that represents timer prescale or timebase.

Returns

gErrorNoError_c	If the action is performed
gErrorOutOfRange_c	If TimeBase exceeds the maximum valid value

Usage

Call MLMESetTmrPrescale ([*gTimeBase2MHz_c - gTimeBase16p625kHz_c*]).

NOTE

Valid values for this primitive depend on the platform been used. There are two sets of primitives:

- One set for the MC13202 and MC1321x families
- Alternate set for the MC1323x family

MC13202 and MC1321x primitives:

```
#if gTargetXcvr_c == gXcvrMc1321x_c || gTargetXcvr_c == gXcvrMc1320x_c || gTargetXcvr_c ==
gXcvrMc1319x_c
  typedef enum timerTimeBase_tag
  {
    gTimeBase2MHz_c = 0,
    gTimeBase1MHz_c,
    gTimeBase500kHz_c,
    gTimeBase250kHz_c,
    gTimeBase125kHz_c,
```

```
    gTimeBase62p5kHz_c,
    gTimeBase31p25kHz_c,
    gTimeBase16p625kHz_c,
    gMaxTimeBase_c
  } timerTimeBase_t;
```

MC1323x primitives:

```
#elif gTargetXcvr_c == gXcvrMc1323x_c
  typedef enum timerTimeBase_tag
  {
    gTimeBase500kHz_c = 2,
    gTimeBase250kHz_c,
    gTimeBase125kHz_c,
    gTimeBase62p5kHz_c,
    gTimeBase31p25kHz_c,
    gTimeBase16p625kHz_c,
    gMaxTimeBase_c
  } timerTimeBase_t;
#endif
```

3.1.16 MLMEWakeRequest

The device transceiver can be put into low power mode using the MLMEDozeRequest or MLMEHibernateRequest functions. The MLMEWakeRequest function can be called to wake-up the transceiver.

NOTE

This primitive can be used only on the MC1320x and MC1321x platforms. For the MC1323x platform, it is invalid.

Prototype

```
smacErrors_t MLMEWakeRequest(void);
```

Arguments

None

Returns

gErrorNoError_c If the action is performed

gErrorNoValidCondition_cIf the radio was not in low power mode

Usage

Call this function to waken the transceiver from Hibernate or Doze mode.

3.1.17 MLMEPHYXtalAdjust

All the devices have a reference oscillator that can be trimmed via register setting:

- The MC13202 and MC1321x devices have a 16 MHz reference oscillator. In the transceiver there is a register with an 8-bit “xtal_trim[7:0]” field that can be programmed to set the trim value
- The MC1323x devices have a 32 MHz reference oscillator. The 8-bit trim register is called XTAL1_TRIN[7:0] and can be programmed to set the trim value.

The use of the 8-bit trim field differs between device types. The user is referred to the appropriate family device Reference Manual for proper trim values.

Prototype

```
smacErrors_t MLMEPHYXtalAdjust(uint8_t );
```

Arguments

```
uint8_t
```

An 8-bit value representing the trim value to the oscillator.

Returns

gErrorNoError_c	If the action is performed
gErrorOutOfRange_c	If TrimValue exceeds the maximum trim value

Usage

Simply call the MLMEPHYXtalAdjust ([Desired trim value]) function directly.

3.1.18 XCVRContReset

For the MC1319x, MC1320x and MC1321x platforms the MCU controls and asserts the transceiver hardware reset. This function asserts and holds the reset line of the transceiver; shutting it down and holding it in its lowest power mode.

NOTE

This primitive can be used only on the MC1320x and MC1321x platforms. For the MC1323x platform, it is invalid.

Prototype

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

Arguments

None

Returns

Nothing

Usage

Call this function to hold the transceiver in reset.

3.1.19 XCVRRestart

For the MC1320x and MC1321x platforms, the MCU controls and asserts the transceiver hardware reset. This function de-asserts and releases the reset line of the transceiver. The function is called to release reset after calling XCVRContReset.

NOTE

This primitive can be used only on the MC1320x and MC1321x platforms. For the MC1323x platform, it is invalid.

Prototype

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

Arguments

None

Returns

Nothing

Usage

Call this function to release the transceiver hardware reset. The transceiver should be re-initialized after a reset.

3.1.20 MLMEGetPromiscuousMode

As standard practice SMAC allows the receiver function to filter incoming RX packets, allowing only SMAC-compatible packets to be recognized; see Section 3.1.23, “MLMESetPromiscuousMode. This mode can be turned off (i.e., promiscuous mode) and all received packets will be reported. The MLMEGetPromiscuousMode function is used to determine the current state of SMAC’s promiscuous mode.

Prototype

```
bool_t MLMEGetPromiscuousMode(void)
```

Arguments

None

Returns

TRUE	If SMAC’s promiscuous mode is active.

FALSE If SMAC's promiscuous mode is inactive.

Usage

Call this function to determine current SMAC's promiscuous mode state.

3.1.21 MLMEGetRficVersion

This function is used to read the version number of different hardware and software modules inside the SMAC platform including version number of the radio.

Prototype

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

Arguments

`versionedEntity_t`

The module for which the version is required. Possible values are:

- gSwSmacVersion_c
- gHwIcVersion_c - this argument returns the ID of the transceiver chip for the MC13202 and MC1323x. The MC1323x is a single IC and this returns its ID.

`uint8_t *`

A pointer to the buffer where the version will be written

Returns

gErrorNoError_c If the action is performed.

gFailOutOfRange_c If the requested Entity is not part of the stored ones.

Usage

Call this function to determine the current IC Hardware version or the SMAC Software stack version.

3.1.22 MLMEScanRequest

A useful function is to be able to scan the different frequency channels and report the status of reported energy. This function provides that capability:

- From one to all 16 channels can be scanned through this function
- Two different scan modes can be selected -
  - CCA mode - the channel energy is detected and compared to a programmed threshold. A busy channel is reported as TRUE and an idle channel is reported as FALSE
  - ED mode - the he channel energy is detected and simple reported as a binary value. The format is similar to the MLMEEnergyDetect function (see Section 3.1.6, "MLMEEnergyDetect)
- The reported values are stored into a 16-byte buffet (pointer is passed as a function argument)

Prototype

```
smacErrors_t MLMEScanRequest(uint16_t u16ChannelsToScan, scanModes_t, uint8_t
                  *pu8ChannelScan);
```

Arguments

`u16ChannelsToScan`

Bitmap of the Channels to be scanned. Any bit set to “1” corresponds to selected channel from Channel 11 to Channel 26.

`scanModes_t`

Determines scan mode to be used; gScanModeCCA_c or gScanModeED_c.

`uint8_t *pu8ChannelScan`

Pointer to 16-byte buffer where the scan results will be returned.

Returns

gErrorNoError_c	No errors.
gErrorOutOfRange_c	If there are no channels to scan in the bitmap, no valid technique passed or null pointer passed.
gErrorBusy_c	When SMAC is busy in other radio activity like transmitting/receiving data or performing a channel scan, among others.
gErrorNoValidCondition_c	If SMAC was not initialized.

Usage

Call this function to initiate an energy scan with the selected technique and a buffer of 16 elements.

- When gScanModeCCA_c is used, a CCA 802.15.4 scan is performed. CCA Mode 1 is implemented by default, which means that it only checks for energy above a threshold. The default CCA Threshold is 0x80. This threshold is initialized at PHY level. CCA returns FALSE if the channel was found idle and TRUE if the channel was busy.
- For gScanModeED_c the individual channel energy is observed. In ED Scan Mode, the output results are the same format as the MLMEEnergyDetect function and are stored in each byte of the output buffer from channel 11 to 26.

Channels that are not scanned have no returned entry in the buffer.

3.1.23 MLMESetPromiscuousMode

SMAC appends 0xFF7E as the first two bytes of the packet to identify its own non-standard packets from other IEE802.15.4 Standard MAC packets in the same PHY layer. When promiscuous mode is off, SMAC filters the received packets and just allows/reports those beginning with 0xFF7E; when promiscuous mode is on SMAC lets all messages pass, whether beginning with 0xFF7E or not. MLMESetPromiscuousMode function allows enabling or clearing the promiscuous mode.

If the device is not in promiscuous mode, and a “standard” packet is received, the packet is discarded without notifying the application that a packet has been received, and the receiver is re-enabled.

This function also affects the TX function; the effect of NOT setting the promiscuous mode (default) is that transmission adds the 0xFF7E to the header packet and thus the maximum data length is limited minus 2 bytes. Setting promiscuous mode disables this appending of the header.

Prototype

```
void MLMESetPromiscuousMode(bool_t );
```

Arguments

`bool_t:`

This is a boolean value that indicates if the promiscuous mode is on (TRUE) or off (FALSE).

Returns

Nothing

Usage

Call this function to set SMAC’s promiscuous mode.

3.1.24 MLMETestMode

For evaluating RF circuitry and doing certification testing, radio test modes are required. The MLMETestMode function enables a number of radio tests. These include the following:

- PRBS9 Mode — Repeatedly sends out a 64-byte packet using a PRBS9 algorithm. All packets that are transmitted in this mode are identical. The data is produced by the function call.
- Force_idle — Places the radio back into idle mode
- Continuous RX — Places the radio into receive mode and allows developers to look for any spectral issues related to the RX section of the radio. Also, this mode can be used to measure the static RX current for the radio.
- Continuous TX without modulation — CW mode that allows characterization of the TX spectral output with no modulation
- Continuous TX with modulation — CW mode that allows characterization of the TX spectral output with modulation

Prototype

```
smacErrors_t MLMETestMode(smacTestMode_t);
```

Arguments

`smacTestMode_t`

The test mode selected.

Returns

gErrorNoError_c	If the radio has been set in the selected test mode.
gErrorBusy_c	When SMAC is busy in other radio activity like transmitting/receiving data or performing a channel scan, among others.

Usage

Call the MLMETestMode() function with the packet to be transmitted and the mode to be executed. See the notes for a list of modes implemented.

This following is a list of the modes implemented (defined as Macros):

gTestModePRBS9_c	Repeated transmits a PRBS9 pattern as long as enabled
gTestModeForceIdle_c	Forces the radio to back to the original IDLE mode.
gTestModeContinuousRx_c	Sets the radio into continuous RX mode.

gTestModeContinuousTxModulated_cSets the device to continuously transmit a 10101010 pattern.

gTestModeContinuousTxUnmodulated_cSets the device to continuously transmit an unmodulated CW

3.1.25 SMACDisableInterrupts

This function is called to disable the interrupts that are used by SMAC concerning the radio/transceiver:

- For MC1323x platform devices - radio interrupts disabled in CNTRL2 register
- For MC1320x and MC1321x platforms: SPI interrupts and the IRQ pin from radio are disabled

Prototype

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

Arguments

None

Returns

Nothing

Usage

Call this function to prevent SMAC from generating radio-based interrupts.

NOTE

Be aware that interrupts must be enabled for certain SMAC functions to operate properly.

3.1.26 SMACEnableInterrupts

This function enables the interrupts that are used by SMAC (see Section 3.1.25, "SMACDisableInterrupts").

Prototype

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

Arguments

None

Returns

Nothing

Usage

Call this function to allow SMAC to enable interrupts.

3.1.27 MCPSDataComfirm

This function is executed when a transmission operation finishes, this is a callback function that the application should implement.

Prototype

```
void MCPSDataComfirm(txStatus_t);
```

Arguments

`txStatus_t`

This indicates the status of the transmission operation

Returns

Nothing

Usage

This callback function must be executed once a transmission operation is completed; successful or not. Use `txStatus_t` argument to define the application execution path.

NOTE

This function is executed as part of an interrupt service routine, as a result follow all the recommendations for interrupt execution timing and context.

3.1.28 MCPSDataIndication

This function is executed when a reception operation finishes, this is a callback function that the application should implement.

Prototype

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

Arguments

`rxPacket_t*`

A pointer to the received packet

Returns

Nothing

Usage

This callback function must be executed once the reception operation is completed; successful or not. Use the `rxStatus` element of `rxPacket_t` structure to define the application execution path.

NOTE

This function is executed as part of an interrupt service routine, as a result follow all the recommendations for interrupt execution timing and context.

3.1.29 MLMEResetIndication

This function is executed after a reset has been commanded and when the transceiver indicates the reset has finished.

Prototype

```
void MLMEResetIndication(void);
```

Arguments

`Nothing`

Returns

Nothing

Usage

Use this function to acknowledge that the transceiver has been reset.

NOTE

This function is executed as part of an interrupt service routine, as a result follow all the recommendations for interrupt execution timing and context.

3.1.30 MLMEScanComfirm

This function is executed after scan operation has finished.

Prototype

```
void MLMEScanComfirm(channels_t);
```

Arguments

`channels_t`

If an ED scan was requested, it represents the cleanest channel or the one with less energy detected. If a CCA scan was requested this argument is meaningless. Format is the `channels_t` type.

Returns

Nothing

Usage

Use this function to acknowledge that the scan operation has finished and the scan stored values are valid.

NOTE

This function is executed as part of an interrupt service routine, as a result follow all the recommendations for interrupt execution timing and context.

3.1.31 MLMEWakeComfirm

This function is executed after Wake operation has finished.

Prototype

```
void MLMEWakeComfirm(void);
```

Arguments

Nothing

Returns

Nothing

Usage

Use this function to acknowledge that the transceiver has awake.

NOTE

This function is executed as part of an interrupt service routine, as a result follow all the recommendations for interrupt execution timing and context.