CC1310 SimpleLink™
TI-15.4 Stack 2.x.x

Linux Developer's Guide

First Release – June 2016

Revision History

Rev	Author	Comments	Date
1.0	Texas Instruments	First Release	6.29.2016

Table Of Contents

1 Introduction

The purpose of this document is to give an overview of the Texas instruments TI15.4 Stack-2.0.0 Linux Software Development kit to help developers run the out of box example applications and create custom TI15.4 Stack 2.0.0 Applications running on Linux host, interfacing with CC13xx running MAC CoProcessor for their end products.

TI15.4 Stack 2.0.0 SDK consists of the software stack from Texas Instruments implementing the standard IEEE 802.15.4e/g specification; it also provides an implementation of Frequency hopping scheme derived from Wi-SUN FAN Specification, in addition it provides required tools, RTOS, example applications to help developers quickly get started to develop their own star topology based wireless network products.

Developers targeting products based on TI15.4 STACK-2.0.0 have two architecture choices for their product development. First option is to have entire TI15.4 Stack application and stack can run within the CC1310 MCU while the second option is to have the application running on a host (running Linux OS, etc) interfacing to the SimpleLink ULP CC1310 MCU running the MAC Co-Processor Application and the TI15.4 Stack-2.0.0 stack.

This document describes how to use the TI15.4 STACK-2.0.0 Linux SDK to develop applications running on a Linux host interfacing via a serial interface with the MAC CoProcessor running on the CC1310.

TI15.4 STACK-2.0.0 allows rapid development of low power and low cost communication networks with limited power and relaxed throughput requirements. It is primarily designed to create star topology wireless networks.

For more details on the IEEE 802.14.5 specification please refer to the specification documents or refer to the wiki page www.ti.com/ti-15.4-stack-wiki.

2 SW Architecture: MAC CoProcessor /Linux Host Block Diagram and Interface Model

This section describes the high level MAC Co-Processor based system architecture and the various SW components, as well as the overall system architecture. MAC Co-Processor is an entity which implements the MAC – IEEE 802.15.4e/g standard in a dedicated system on a chip and provides a serial interface to an external processor for control and processing of the co-processor operations. A description of the API is provided in the document in the /doc folder of the TI15.4 STACK-2.0.0 Linux SDK install.

The MAC Co-Processor approach is a scalable architecture split that fits perfectly for configurations where the host co-processor runs protocol stack layers over IEEE 802.15.4 e/g MAC/PHY (e.g. generic IP over 6LowPAN, ZigBee IP or ZigBee Pro) or even an application that wants to simply use the MAC/PHY as a data link.

With TI-15.4 Stack Linux Example Application the external host SW is running on a Linux-based platform. Though the high level SW partitioning for layers can be conceptually applied to non-Linux based hosts, the SW components developed and described in this document are specific to a Linux-host implementation for an x86 or ARM based platform.

The interface between the host and the MAC Co-Processor is defined at different logical layers in this split architecture: a physical layer (like USB or UART), a logical data-link layer and a presentation layer.

The physical layer interface is used to transport the serial frames over the physical link. Several physical interfaces can be used (e.g. USB or UART). This serial protocol is known as “MT” – the Management and Test protocol.

The frames transported over the physical serial link follow the format specified in the MAC Co-Processor Interface Guide in the /doc folder with the TI15.4 Stack Linux SDK install.

The Collector Example application delivered with this package illustrates an example of how a high-level application that wants to use the MAC Co-Processor services must be implemented

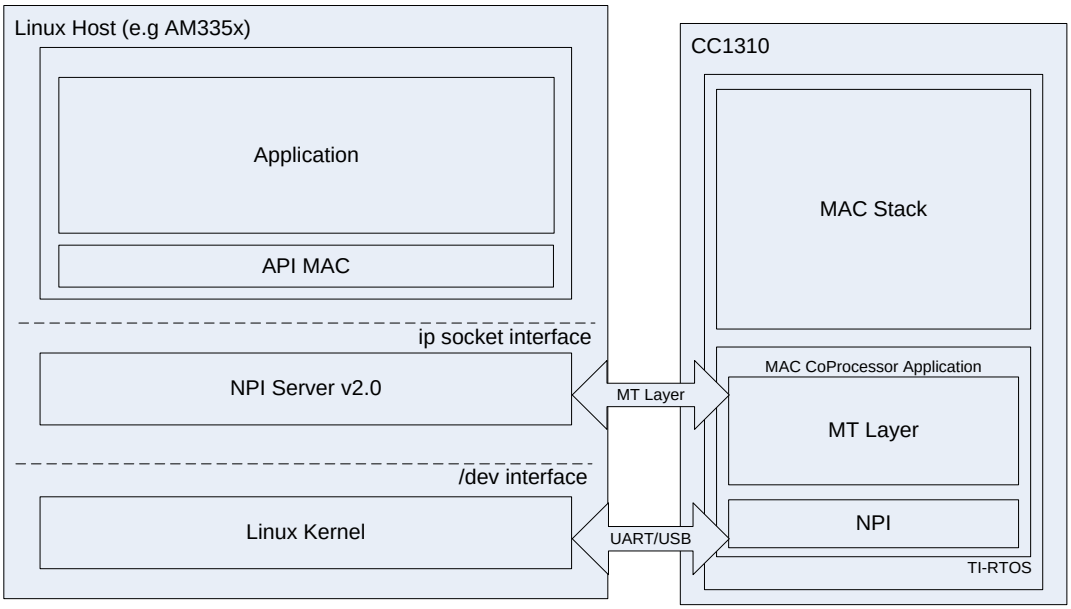


Figure 2-1: High Level SW Architecture of TI15.4 Stack-2.0.0 Linux Applications

SW System Components:

- **MAC Co-Processor Embedded Example Application**: This is embedded software running on a CC1310. This example application implements a 802.15.4e/g MAC/PHY layer and provides an {serial port} MT based interface to the applications running on the Linux host.
- **NPI Server v2:** The NPI (Network Processor Interface) Server provides a socket-based communication channel to the MAC Co-Processor that abstracts the physical serial communication mechanism, e.g. UART/USB. Note: This is an optional module, the API MAC can interface directly to the Linux Kernel via the dev interface as well.
- **Linux Kernel**: Provides a device driver model for the serial interface available as a character device over the physical port selected (e.g. USB). For instance, in USB case, the MAC Co-Processor implements a CDC USB driver class and therefore the kernel should provide a USB modem (ACM) service over /dev.
- **TI15.4 STACK Application**: Application module that implements a specific application using the 802.15.4e/g protocol and the MT structure-based model as application level interface.
    - **Application:** TI-15.4 Stack Linux SDK Sample Application provides a starting point to use the MAC Co-Processor by illustrating how to initialize the network, join a network, and perform data communication between the PAN Coordinator and the devices (i.e. implementing a star network topology).
    - **API MAC:** This is the application programming interface (API) for the Texas Instruments 802.15.4 MAC software. This API provides an interface to the management and data services of the 802.15.4 stack using the CoProcessor embedded example application. This module supports two communications methods. Method #1 is direct via a /dev/tty {serial-port} interface, or Method #2 via a TCP/IP stream socket to an NPI Server, which translates the TCP/IP connection into a /dev/tty {serial port} interface.

3 SDK Description

```
Notes:
1) Typographical convention used, the default SDK Installation directory is:
       ${HOME}/ti/simplelink/ti15.4stack_linux_64_02_00_00_xx
   References to the ${SDK_ROOT} in this document refer to this location.

2) The numbering convetion is:  Major, Minor, Patch, BuildID.  The Linux Build ID is
independent of the Embedded SDK Build ID and can be different number.
```

The Linux SDK is divided into several directories as shown below:

Figure 3-1: TI15.4 STACK-2.0.0 Linux SDK Out of Box

Figure above shows the ${SDK_ROOT} directory structure under your TI-15.4MAC-2.0.0 install directory, this section describes at a high level the contents in each folder.

- **components**: Contains the following libraries:
  - **common** – simple runtime OS features, File IO, and the "stream" interface
  - **nv** – emulates the Non-Volatile Memory as used in an embedded device
  - **api** – the API MAC and MT Msg interface for the embedded network processor
- **docs**: Various documents such as software developers guide, quick start guide, mac cop interface guide.
- **example**: example applications
  - **cc13xx-sbl** – flash update utility for the CC13xx
  - **collector** – Example application that demonstrates starting up the network, allowing network devices to join the network, collecting data from remote sensors
  - **gateway** – A Node JS based application that creates a local web-sever and presents network information and sensor data via a web application

    - **npi_server2 –** a socket/packet interface to the embedded co-processor.
- **firmware**: prebuilt hex files for the CC1310 Mac CoProcessor, sensor application
- **prebuilt**: pre-built linux x86_v64 and BBB binaries for the example applications, scripts to quickly run the out of box example applications
- **scripts**
    - Contains makefile fragments used to compile and link example applications.

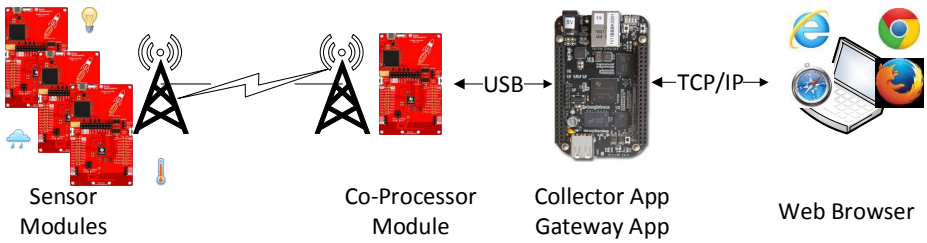
4 Development Environment

4.1 Hardware Requirements & Configurations

4.1.1 Beagle Bone Black + CC13xx-LP (CoProcessor) + Multiple CC13xx-LP (Sensors)

The TI-15.4 Stack SDK Linux example applications can be executed from Beaglebone as explained in this document. The following hardware is required:

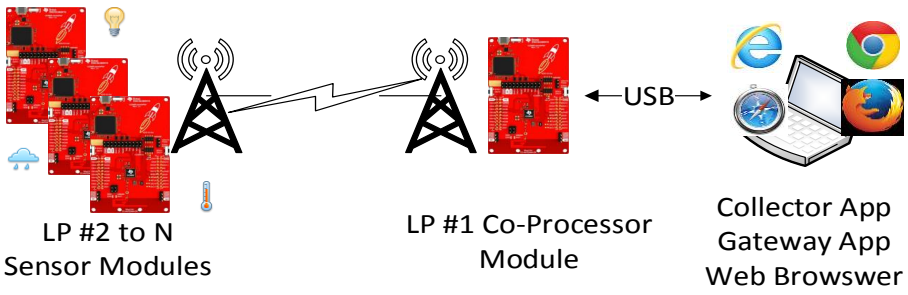
- Beagle Bone Black (https://beagleboard.org/black)
- Minimally: 8 GB micro-SD memory card (required to program the Processor SDK Image)
- Ethernet cable
- At least two or more CC13xx Launchpads
  - LaunchPad #1 – will act as the Co-Processor Module, the interface to the 802.15.4 network
  - LaunchPad #2 to N, for the network nodes that will join the TI-15.4 Stack based network.
- FTDI cable: http://elinux.org/Beagleboard:BeagleBone_Black_Accessories#Standard_FTDI_Cable
- One USB cable connecting the Launchpad to the Beagle Bone Black.
- 5Volt Power Supply for BeagleBone Black
  http://elinux.org/Beagleboard:BeagleBone_Black_Accessories#Power_Supplies
  or USB cable
  http://elinux.org/Beagleboard:BeagleBone_Black_Accessories#USB_Cables



4.1.2 Linux X86 PC + CC13xx-LP (Coprocessor) + Multiple CC13xx-LP (Sensors)

The TI-15.4 Stack SDK Linux example application can be executed from an x86 machine running Ubuntu connected via a USB to a CC13xx Launchpad running MAC CoProcessor application as explained in this document. The following hardware is required:

- A Linux x86 PC running Ubuntu OS 14.04 LTS (x86_64 Bit)
- USB Cable to connect to the launch pad
- At least two or more CC13xx Launchpads
  - LaunchPad #1 – will act as the Co-Processor Module, the interface to the 802.15.4 network
  - LaunchPad #2 to N, for the network nodes that will join the TI-15.4 Stack based network.



4.2 Required Software

This section explains how to setup the hardware with the desired software to be able to build and run the TI-15.4 Stack Linux SDK out-of-box example applications and setup to develop custom applications.

4.2.1 Linux Development Host

The x86 machine running Ubuntu can be used to develop and run the application or cross compile the applications for the BeagleBone Black platform. This section provides the instructions to setup to build, develop and run the applications on x86 machine and also on how to setup the x86 machine for cross compiling for the BBB platform.

Note: When developing for BBB platform on x86 machine, the C code is cross-compiled and linked as a Beaglebone application on your host x86 Linux_64 machine. The ARM executable is then copied to the Beaglebone to run the application.

When cross-compiling, executables are named: "host_<name>" – which means the compilation host. The prefix: "bbb_<name>" is used as a target prefix for Beaglebone black.

Install the following software on your x86_64 machine running Ubuntu 14.04LTS 64bit to run the TI-15.4 Stack Linux SDK out-of-box example applications.

1. Install the TI-15.4 Stack Linux SDK

   First run the TI 15.4 Stack installer on the windows PC. After installation is complete, the TI-15.4 Stack Linux SDK installer can be found at the default installation location:

   ```
   C:\ti\simplelink\ti-15.4-stack-sdk_2_00_00_xx\examples\linux
   ```

   Copy the file below to your Linux machine; the filename is:

   ```
   ti15.4stack_linux_x64_02_00_00_xx.run
   ```

   where, xx is the final build number for the installer.

   Note: The installer is a 64 bit Linux executable thus requires a 64 bit Linux machine.

   On the host (Linux x86_64) machine, go to the directory where you have the Linux installer and use the following commands to install:

```
bash$ chmod +x ti15.4stack_linux_x64_02_00_00_xx.run
bash$ ./ti15.4stack_linux_x64_02_00_00_xx.run
```

Note: The above ($) prompt indicates that this installer should be run as a normal user, and specifically not run as ROOT (with the # prompt).

The default TI-15.4 Stack install directory is:

```
${HOME}/ti/simplelink/ti15.4stack_linux_64_02_00_00_00_xx
```

2. Install the package: build essentials

```
bash$ sudo apt-get install build-essential
```

3. Install the package: NodeJS

```
bash$ sudo apt-get install nodejs
```

4. Your user name must be a member of the group “dialout”

```
bash$ sudo adduser $USER dialout
```

5. To cross compile for Beaglebone black install the TI processor-SDK-Linux-am335x from http://www.ti.com/tool/PROCESSOR-SDK-AM335X . The TI AM335x Linux SDK contains all of the cross compilation tools, headers, libraries and other required files for cross compiling to the Beagle Bone Black

---

Notes:

1) The TI-15.4 Stack Linux SDK assumes the processor SDK is installed in this location:

```
/home/${USER}/ti-processor-sdk-linux-am335x-evm-02.00.02.11
```

2) The TI-Processor SDK download is very large, and expands to an even larger installation (download: 3 gig, installed foot print: 4.5 gig) and is only required to build the example applications in via a cross compilation scheme.

If the TI Processor SDK is installed in a different location or is a different version, then two files listed below need to be updated:

- *File #1* `${SDK_ROOT}/scripts/front_matter.mak`
- *File #2* `${SDK_ROOT}/example/cc13xx-sbl/app/linux/Makefile`

Figure 4-1 *is a screen shot from* `front_matter.mak`*: (Adjust the version numbers as required)*

```
#=======================================
# These *MAY* need to be adjusted to fit your SDK install.
# NOTE: The Boot loader also has a macro that may need updating.
# SEE: ${root}/example/cc13xx-sbl/app/linux/Makefile
#---------------------------------------
# What is the Processor SDK Root directory
#---------------------------------------
bbb_TI_PROC_SDK_DIR=${HOME}/ti-processor-sdk-linux-am335x-evm-02.00.02.11
```

Figure 4-1: Screenshot from file front_matter.mak

Your x86 machine running Ubuntu should now be ready to run build and run the example applications or to cross compile applications for BBB platform.

6. To test and verify the setup, build the Linux application from source follow the instructions below:

   Step 1: Change to the TI-15.4 STACK ${SDK_ROOT} installation directory

   ```
   cd ${HOME}/ti/simplelink/ti15.4stack_linux_64_02_00_00_xx
   ```

   Step 2: At the "bash$" prompt, Type the following:

   ```
   bash ./build_all.sh          (builds the host version)
   bash ./build_all.sh bbb      (builds the BBB version)
   ```

   See Section 7.1 for more details.

   Step 3: The script will build the component libraries, and the example applications.

   Note: The script also creates: "*.log" files of the compilation process

The next step is to: Configure the Beagle Bone Black, (see Section 4.2.2) or skip ahead and flash program the CC13xxLP devices. See Section 4.2.3 (Using the CC13xx Linux Flash Programing tool), or Section 6 which discusses the Windows flash tool.

4.2.2 Beagle Bone Black

The Beagle Bone can be used in two different ways for development and running the applications:

Method #1: as a run time environment only, software is generally developed (edit, compile, and link) on a 'host X86 Linux machine' and then deployed {copied} to the Beagle Bone Black

Method #2: As a development environment, where the edit, compile, and link process is done directly on the Beagle Bone Black.

Steps below provide details on how to setup your Beaglebone for both methods described above.

1. Program the microSD memory card with the TI Linux Processor SDK v 0.2.00.02.11 or greater

---

Note: TI Linux Processor SDK Versions prior to April 2016 version (0.2.00.02.11) are missing the following required features

- *The NodeJS package is not present.*
  - *This is required to run the gateway example application which is based on the NodeJS framework.*
- *CONFIG_USB_ACM= (disabled, must be y, or m)*
  - *This is required as the Launchpad uses USB_ACM to emulate a serial interface with the Beagle Bone.*
- *Optional: Linux USB-Ethernet (RNDIS) driver was not present*
  - The Beagle Bone functions well using a standard Ethernet Cable

- *This feature provides "Ethernet over USB functionality". The NodeJS Webserver application can be accessed using either (a) the standard Ethernet cable, or (b) the "Ethernet over USB" solution*

- Download the prebuild processor sdk image "am335x-evm-linux-02.00.02.11.img.zip" from http://software-dl.ti.com/processor-sdk-linux/esd/AM335X/latest/index_FDS.html Follow the instructions at wiki-page to program the microSD memory card via Windows: http://processors.wiki.ti.com/index.php/Processor_SDK_Linux_Creating_a_SD_Card_with_Windows
  The Linux method is described here:
  http://processors.wiki.ti.com/index.php/Processor_SDK_Linux_create_SD_card_script

- To boot from the SD Card, do the following:

Step 1: Disconnect power and unplug USB cable from the Beagle Board
Step 2: Insert the SD Card
Step 3: Press (and hold) the "Boot Switch"
Step 3: Apply power to the Beagle Board (via USB or via barrel connector)
Step 4: Wait a few seconds – then Release the Boot Switch (Note: The boot switch is only detected at power up)
Step 5: In about 5 to 15 seconds the LEDs will start blinking

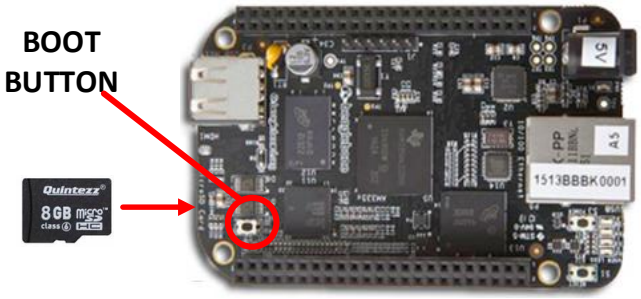
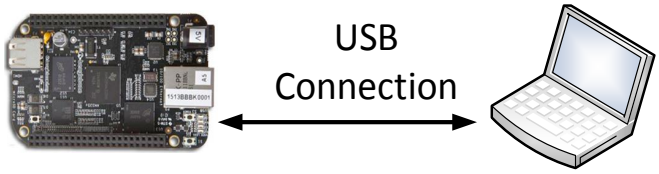
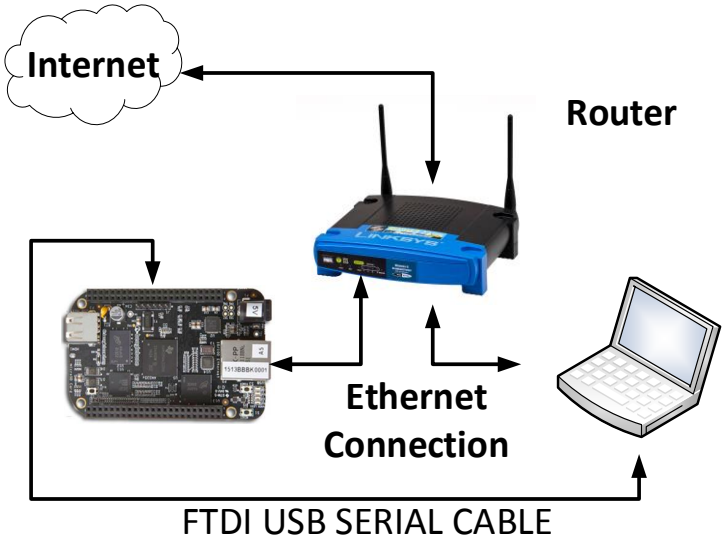


Figure 4-2 Boot Switch Location

2. *Determine the IP Address of the BBB*
   To monitor and control the TI-15.4 Stack Network when using the TI-15.4 Stack Linux SDK out of box gateway application, it is required to know the IP address of BBB.
   - Method 1: If connecting to the Linux Host PC/Virtual Machine via a USB cable
     - The USB IP address is hard coded 192.168.7.2 and is assigned during boot.



- Method 2: If using a 'network router' or Windows PC



  - The network router will assign a "random ip address" via DHCP to the BBB based on the order the devices "power up" for example if today – the BBB powers up first, it will receive BBB=xx.xx.xx.100, laptop = xx.xx.xx.101. However tomorrow – the IP address might reverse, or perhaps a new device is present on the router {your cell phone?}

To determine the assigned IP address there are 2 options:

Option 1: Use the Router HTTP Management page

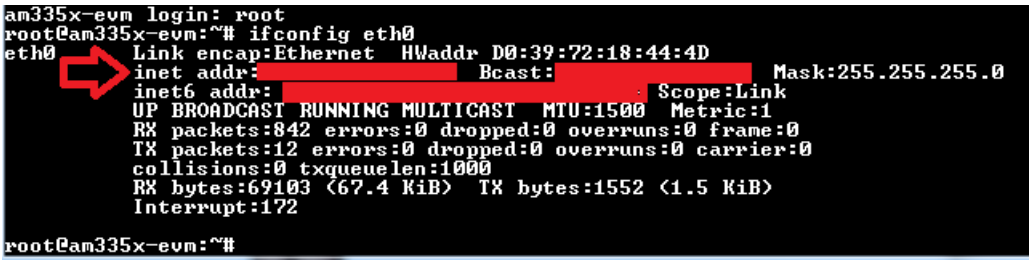
Each router brand is a bit different. See the examples below, the generic name is: "DHCP Client Table"

Brand	Example Link
LinkSys	http://www.linksys.com/us/support-article?articleNum=139502
NetGear	http://documentation.netgear.com/fvs336g/enu/202-10257-01/FVS336G_RM-11-07.html
Belkin	http://www.belkin.com/pyramid/AdvancedInfo/F5D8235-4/Advance/reserveIP.htm

Option 2: Using a FTDI USB Serial Cable and a terminal application

- Before booting the Beagle Board - connect the Ethernet cable, and connect an FTDI [USB Serial] cable. Open a terminal program using settings: 115200 8-N-1
- Power-cycle the BBB (remember to press & hold the boot button described above)
- Wait until the Beagle Board has finished booting

- Log into the Beagle Bone Black using the user name: 'root'
- Type the command: "ifconfig", the IP Address will be on the screen see below:
- Copy the "inet addr"



3. Prebuilt tar file location

   On the Linux PC, use the following commands to transfer the prebuilt tar file with built binaries to the BeagleBone Black. The TAR file can be found in 2 places:

   The Windows SDK install directory, under the installation directory as follows:

   ${WINDOWS_SDK}/examples/linux/ti15.4stack_linux_x64_2_00_00_xx-bbb_prebuilt.tar.gz

   Or in the Linux SDK, in the file: ${SDK_ROOT}/prebuilt/bbb_prebuitl.tar.gz

4. Copy the "bbb_prebuilt" file to the Beagle Bone Black.

   Substitute the appropriate address for the ${BBB_IP_ADDRESS} below:

```
bash$ cd ${SDK_ROOT}/prebuilt
bash$ scp bbb_prebuilt.tar.gz root@${BBB_IP_ADDRESS}:~/.
```

Notes:
1) Other tools can be used such as WinSCP or FileZilla.
2) The target directory "~/." is a short hand for "roots" ${HOME} directory, ie: /home/root
3) Later scp can be used to copy new binary files to the BBB during development.

5. Login to the BBB – (get a shell prompt) and unpack the prebuilt TAR file
   Type the commands

```
bash$ ssh root@${BBB_IP_ADDRESS}        (connect to the BBB)
root@am335x-evm# cd ${HOME}
root@am335x-evm# tar xf bbb_prebuilt.tar.gz
```

   The Prebuilt applications will be found in the "prebuilt" directory

6. OPTIONAL: To copy the Linux example SDK source code to the Beagle Bone to both build and run the application on BBB:

   Step 1: Create a TAR file of the entire ${SDK_ROOT} directory
   Step 2: Copy this new TAR file from your Linux host to the BBB via "scp"

(See the example above where the prebuilt files are copied for details.)

Step 3: Unpack the TAR file

See Section 7.1 for details about how to build on the BBB.

The BeagleBone black is now ready to run the TI-15.4 Stack SDK Linux example applications.

The next step is to flash program the CC13xxLP devices. See Section 4.2.3 (Using the CC13xx Linux Flash Programing tool), or Section 6 which discusses the Windows flash tool.

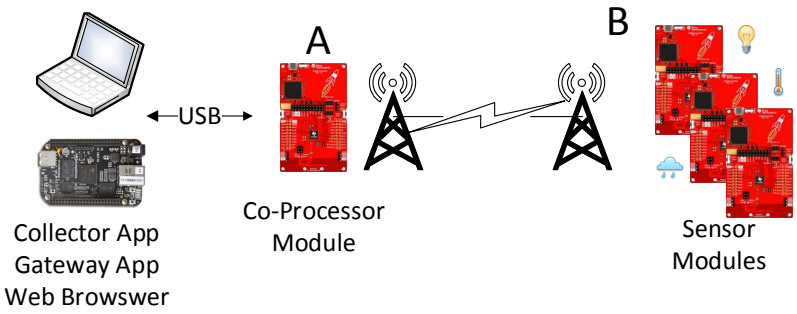
4.2.3 Program the CC13xx Launch Pads

This step provides instructions on how to program the CC1310 LaunchPads with the required hex files.

Prebuilt firmware files are in the ${SDK_ROOT}/firmware directory.

(A) Program one Launch pad with the ""${SDK_ROOT}/firmware/coprocessor_cc1310_lp.hex" image, label this CC13xx Launchpad as "the coprocessor"

(B) Program all other Launch pads with the "sensor" image "${SDK_ROOT}/firmware/default/sensor_default.hex", label these devices as "Sensor".



To program the launch pad devices there are two options:

- Option 1: Use the Linux application cc13xx-sbl. Please refer to Section 6.3 for details.
- Option 2: Use the Windows application Smart Flash Programmer 2. Please refer to section 4.2.3.1 for this method.

The CC13xx Launchpads are now ready. Section 6 describes how to run the out of box collector and gateway example applications.

4.2.3.1 Programming the MAC CoP Application on CC13xxLP using SmartRF Flash Programmer2

The TI-15.4 Stack Linux SDK includes a prebuilt hex file for the MAC Co-P. This hex file can be programmed via the SmartRF Flash Programmer 2 (see: http://www.ti.com/tool/flash-programmer) as explained below. In addition, the MAC Co-Processor CCS project workspace is included with the

windows installer. Please refer to the document TI-15.4MAC developers Guide.pdf on how to program the device using CCS.

Start the SmartRF Flash Programmer 2 on the windows machine, the figure below shows the steps to program the CC1310 with the desired hex file.

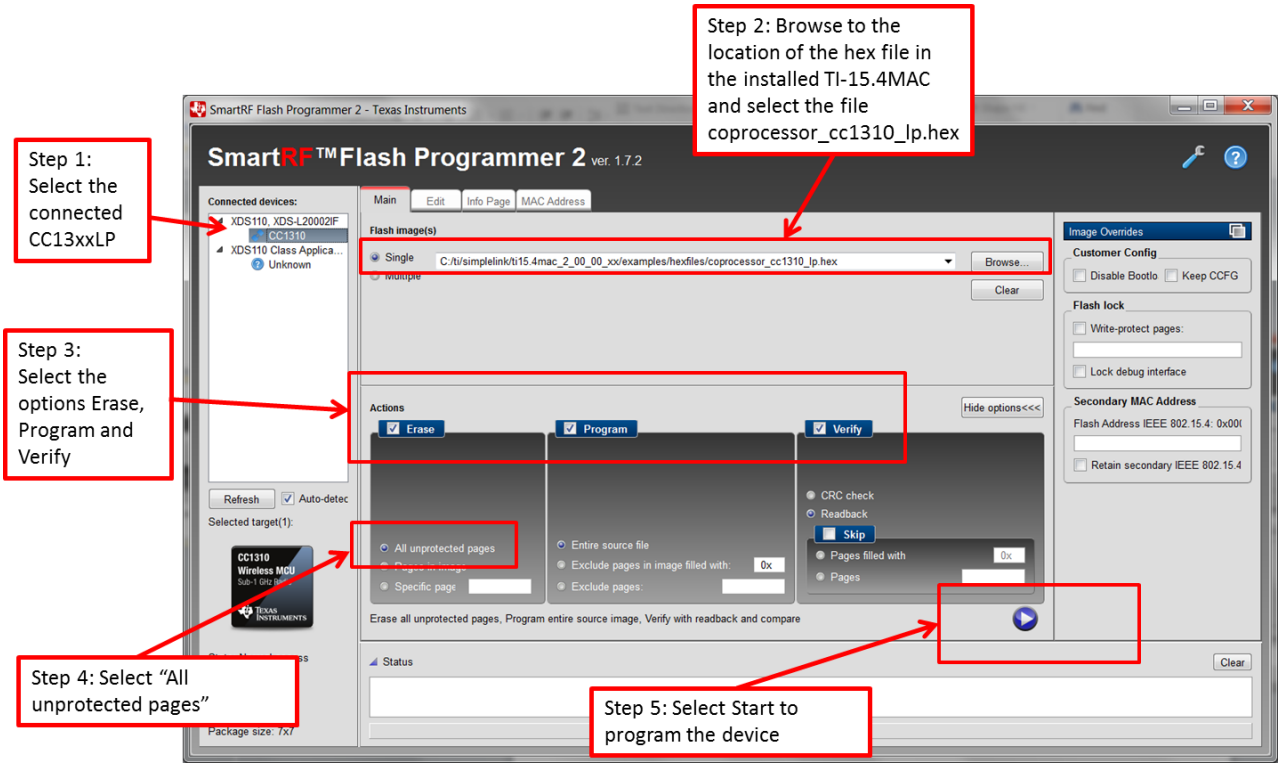


Figure 4-3: Steps to program the CC13xxLP using the SmartRF Flash Programmer 2.

Note:

1) *When connecting the Launch Pad – the SmartRF program might update the "debugger firmware" in the on-board XDS-110 debug feature.*

2) *Trouble shooting hint #1: It is easy to confuse various Launch Pads (ie: Sensor vs CoProcessor) be sure to *LABEL* the device properly.*

3) *Trouble shooting hint #2: To verify a device is programed correctly do the following:*
   Step 1: Uncheck "Erase"
   Step 2: Uncheck "Program"
   Step 3: CheckMark VERIFY
   Step 4: Choose "read-back" – this option will effectively "upload the hexfile" from the device and compare the hex file byte-for-byte against the selected HEX file.
   Step 5: Click the "Play/Start" button to begin the verify process.

To see this in action: First verify the Co-Processor hex file verses the sensor launch pad. The result should be “error”. Second: Unplug the sensor launch pad, and connect the co-processor launch pad, and press “Play/Start” the verification should be “Success”

5 Example Applications Overview

This section provides a high level overview of the out of box example applications.

5.1 Linux Collector and Gateway Example Application

The **Linux Collector and Gateway Application** are provided as part of the TI-15.4 Stack Linux SDK installer. The Linux Collector example application interfaces with the CC13xx running the MAC CoProcessor via a UART. Collector example application builds a full-function device that performs the functions of a network Coordinator (starting a network and permitting devices to join that network) and also provides an application to monitor and collect sensor data from one or more Sensor devices. In addition it provides a socket server interface to the Linux Gateway application.

The Linux Gateway application, implemented within the nodeJS framework, connects as a client to the socket server created by the Linux Collector. In addition, it establishes a local web-server to which the user can connect via a web-browser to monitor and control the network devices.

The collector and gateway application which provides IEEE 802.15.4 to IP Bridge is a great starting point for creating IOT applications with TI-15.4 Stack SDK.

Figure below shows the software architecture of the Linux Collector Example Application.

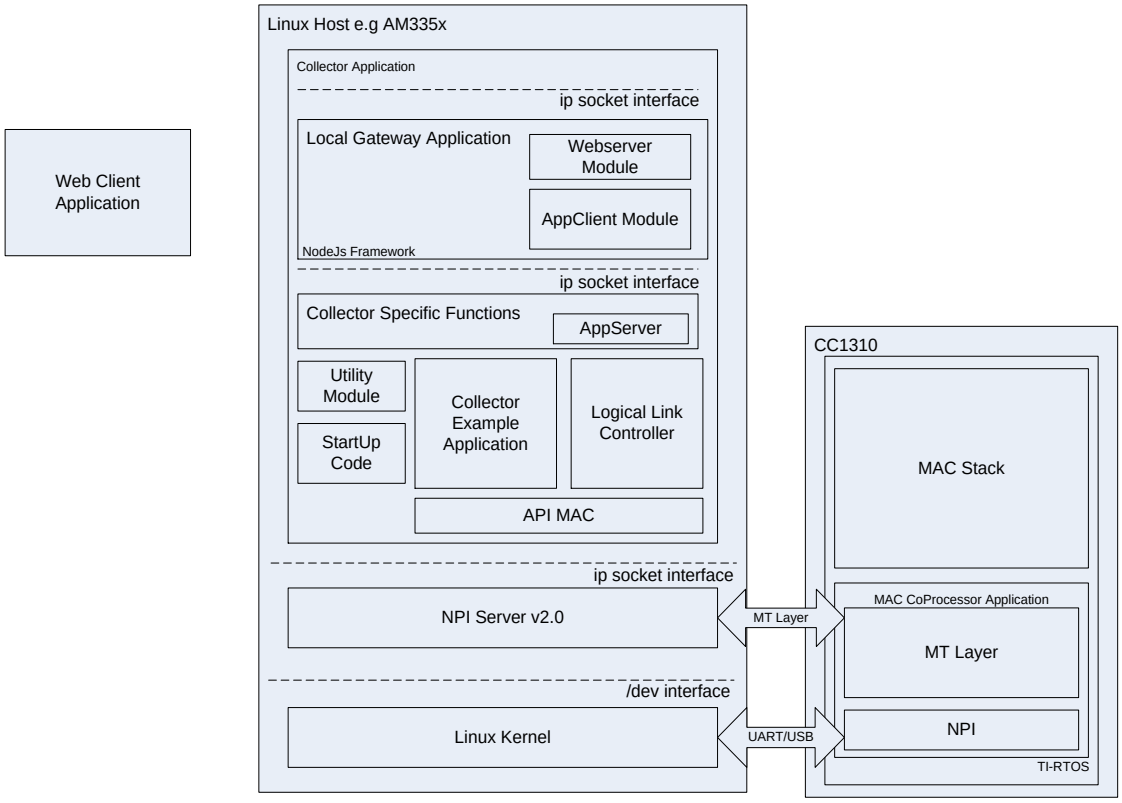


Figure 5-1: SW Block Diagram of the TI15.4 Stack -2.0.0 Linux Collector Example Application

Below is a high level description of each software block and where to find the relevant source code:

Linux Kernel: Has features and functionality provided by the Processor SDK

NPI Server v2.0: (directory: examples/npi_server2) Provides a serial communications link/interface to the application with the CC13xx Launchpad running MAC Co-Processor.

API MAC: (directory: components/api) This API provides an interface to the management and data services of the TI-15.4MAC protocol stack.

Collector: (file: example/collector/collector.c) Implements an example application which starts the network, allows new devices to join the network, configures the joining devices on how often to report the sensor data, for sleepy devices configures how often to poll for buffered messages in case of non-beacon and frequency hopping mode of network operation, and tracks if connected devices are active on the network or not by periodically sending the tracking request messages to which it expects the tracking response message.

Logical Link Controller: (file: example/collector/cllc.c) Abstracts the network management functionality from the application such as starting the network, associating new devices, responding to orphan notifications from the devices who have lost sync.

Collector Startup Code: (files: example/collector/linux_main.c and appsrv.c) Implements necessary logic to configure the application at startup. Note: A portion of the web server gateway code is also within appsrv.c.

NPI Server:

startup: example/npi_server2/linux_main.c
application: example/npi_server2/app_main.c

Common Library Component Module: (See directory: components/common) Provides various functionalities such as timer services, semaphores, etc. Please see section 9 for more details.

Collector Specific Functions: (file: example/collector/csf_linux.c) Implements the user interface specific functions, interfaces with the NV module to provide persistent data service to store information over power cycles, implements a server socket for user application to connect to the application to get useful network information and control network devices.

Local Gateway Application: (file: example/collector/appsrv.c) Implements an application client that connects to the application server implemented in the collector specific function, and also a webserver application.

Gateway Protocol files: These files define the message content/body between the “appsrv” component and the front end gateway application in the gateway folder. (Files: example/collector/*.proto, generated files: “*.pb-c.[ch]”, corresponding *_proto.c) . The NodeJS package uses a single ‘protobuf’ definition file: example/gateway/appClient/protofiles/appsrv.proto

Web-Application (The gateway application. see gateway directory and related NodeJS files) Allows users to view the network information, device information, information about reported data from the network devices and control the network devices.

MAC CoProcessor: (Prebuilt embedded hexfile) Running on CC13xx, implements an 802.15.4e/g MAC/PHY layer and provides an MT based interface to the applications running on the Linux host.

5.2 Linux Serial Bootloader Example Application

This is Linux flash update application which provides ability to upgrade the firmware on the CC13xx via the ROM serial bootloader.

5.2.1 Application Design Details

The serial bootloader example application has two main modules, cc13xxdnld module and the sblUart module. cc13xxdnld SW module has no dependencies on the OS or hosting HW and is intended to be portable across MPU's/MCU's and OS's. The sblUart files contain the OS/HW specific calls to the UART which are provided as callback functions to the cc13xxdnld SW module, which requires UART read/write access to communicate with the CC13xx/26xx ROM bootloader

5.2.1.1 CcDnld API

5.2.1.1.1 Overview

The CcDnld API should be used in application code. The CcDnld API is intended to ease the integration of the CC13xx bootloader functionality in a host processor connected to the CC13xx UART.

5.2.1.1.2 General Behavior

Before using the CcDnld API a binary/hex image should be prepared that is to be loaded into the cc13xx Flash.

To load data to the CC13xx Flash following needs to be performed:

1. Set the UART send/receive data callbacks with CcDnld_init().
2. Connect to the CC13xx ROM bootloader with CcDnld_connect().
3. Optionally Erase Flash with CcDnld_flashEraseRange().
4. Optionally Program Flash with CcDnld_startDownload() and CcDnld_sendData().
5. Optionally Erase Flash with CcDnld_flashEraseRange().
6. Optionally Verify Flash with CcDnld_verify().

5.2.1.1.3 Error handling

The CcDnld API will return CcDnld_Status containing success or error code. The CcDnld_Status codes are:

Status	Description
CcDnld_Status_Success	Success
CcDnld_Status_Cmd_Error	Invalid command

CcDnld_Status_State_Error	Invalid state
CcDnld_Status_Param_Error	Invalid Parameter

5.2.1.1.4 Supported Functions

Generic API function	Description
CcDnld_init()	Registers the UART read/write function points
CcDnld_connect()	Connects to the CC13xx/26xx ROM bootloader
CcDnld_flashEraseRange()	Erases the specified flash range
CcDnld_startDownload()	Sends the download command to the CC13xx/26xx ROM bootloader
CcDnld_sendData()	Sends program data to be program data to be programmed to the flash
CcDnld_verify()	Verify a flash range with data in a buffer

5.2.2 Important Linux host design considerations:

In addition to a standard 2-wire UART interface your final design should include two GPIO signals to control the CC13xx device.

- GPIO Signal A which will be used to reset or power cycle the CC13xx device.
- GPIO Signal B which controls the masked ROM boot loader entry into the flash update mode

At power up the CC13xx can configured to read a GPIO pin and either (a) proceed normally, or (b) enter bootloader/flash programmer mode. The configuration for this feature is stored in the CCFG (customer configuration) section of the on chip Flash memory. Please consult the CC1310 TRM (http://www.ti.com/lit/ug/swcu117f/swcu117f.pdf) Chapter 8 for details.

6 Running the Out of Box Example Applications

This section provides instructions on how to run the out of box example applications.

6.1 Running the Collector and Gateway Application

This section explains how to run the out of box collector application which creates the network and allows new devices to join to the network. Also, it explains how to run the gateway application which creates a local web-server to which users can connect using a web-browser to visualize the network information and reported sensor data from the sensor nodes. In addition, it also explains how to connect sensor nodes to the network and observe data communication between the Linux collector example application and the sensor nodes.

There are two ways to run the out of box example application:

- Option 1: Run the existing prebuilt Linux examples found in the ${SDK_ROOT}/prebuilt directory.
- Option 2: Build and run the Linux applications manually from source.

6.1.1 Embedded Prebuilt Hex Files & Frequency Selection

The “out of box” prebuilt HEX files use APIMAC_STD_US_915_PHY_1

```
/*! PHY IDs - 915MHz US Frequency band operating mode # 1 */
#define APIMAC_STD_US_915_PHY_1 1
/*! 863MHz ETSI Frequency band operating mode #1 */
#define APIMAC_STD_ETSI_863_PHY_3 3
```

To change the embedded devices the steps are:

Step 1: Modify the embedded “config.h” in the sensor example.
Step 2: Compile and create the new sensor hex file using TI Code Composer Studio
Step 3: Program the Sensor module with this new hex file.

To change the Linux application,

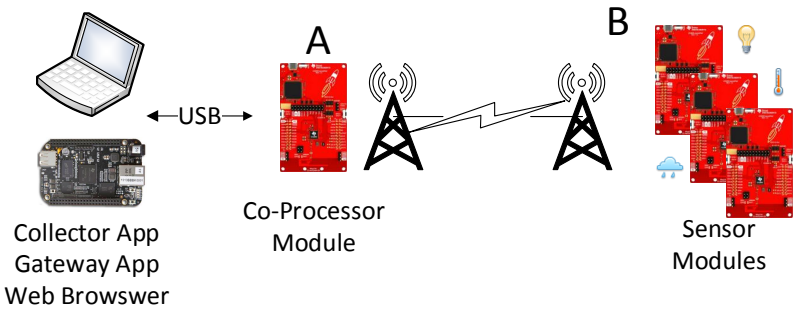
Step 1: Find the “collector.cfg” file (it is an ascii text file)
The example linux collector application reads this file at startup
Step 2: Modify the “config-phy-id” line as shown below.

```
config-percentfilter = 0x0ff
config-phy-id = 1
config-scan-duration = 5
```

6.1.2 Connect the Co-Processor LaunchPad

Program the Collector LaunchPad (A) and Sensor Launch Pads (B) with the HEX files as described in Section 4.2.3 (or as above)

Connect the CC13xxLP running the MAC Co-Processor as shown below, to either the Linux PC directly, or to the Beagle Bone Black.



6.1.3 About /dev/ttyACM0

Linux supports several types of USB Serial Ports. One type is known as /dev/ttyUSB<number>. Another type is known as /dev/ttyACM<number>.

Linux assigns the numbers in order, as each USB device is enumerated. For example: the first device is /dev/ttyACM0, the second device is: /dev/ttyACM1, etc.

Specifically the Launchpad presents as a /dev/ttyACM<number> device. The LaunchPad debug interface actually presents two serial interfaces – this example only uses the first interface aka: /dev/ttyACM0.

As a side note: FTDI serial cables present as /dev/ttyUSB<number>

The "out of box collector.cfg" file assumes the LaunchPad is exactly /dev/ttyACM0

As a software developer, it is common to have many devices connected to your development machine via the USB interface. These other devices may also present another /dev/ttyACM<some_number> interface (for example your mobile phone, or tablet).

Thus, if another serial device is already connected (or enumerated) when the LaunchPad is connected it may – or may not – appear as /dev/ttyACM0

To determine what is present, do this:

```
bash$
bash$ ls -l /dev/ttyACM*
crw-rw-rw- 1 root dialout 166, 0 Jun 22 15:55 /dev/ttyACM0
crw-rw-rw- 1 root dialout 166, 1 Jun 22 15:55 /dev/ttyACM1
bash$
```

Edit the collector.cfg accordingly.

Solutions:
Option 1: Unplug everything – leaving only the CoProcessor Launch Pad connected

Option 2: Sometimes "unplugging everything" is not a viable solution, the other alternative is to edit the "collector.cfg" file and change the "devname" line shown below:

```
; If collector app connects directly to a UART (no-npi-server) this is how to connect.
[uart-cfg]
        ;; Launchpads use USB and show up as: /dev/ttyACM0 and ACM1
        ;; Solutions using an FTDI or Prolific cable use /dev/ttyUSB0 or USB1
        ;; Hard serial ports are: /dev/ttyS0 to ttyS9
        ;devname = /dev/ttyUSB1
        devname = /dev/ttyACM0
        baudrate = 115200
        ; we use the default flags
        flag = default
```

6.1.4 Option1: Running Application using Prebuilt Binaries

To run the prebuilt example binaries follow the steps below:

Step 1: Program the Launch Pads as described above.
Step 2: Change to the prebuilt directory (either on the host, or the BeagleBone)
Step 3: Run the "run_demo.sh" script
Step 4: On the Linux x86 host, the web browser should launch automatically and connect to the desired ip address and port.
Step 5: On the Beagle Bone Black, launch your browser manually, then visit the specified website address (the shell script will display a link with the ip address and port id)
Step 5: Skip ahead to 6.1.6.

6.1.5 Option 2: Running the example Linux Applications after building from Source

This assumes you have:

a) Programmed the Launch Pads (see above) as required.
b) The Co-Processor Launch Pad is connected as /dev/ttyACM0

The steps below, manually perform the same steps the "run_demo.sh" script (above) performs.

6.1.5.1 Start the Collector Example Application

Step 1: Change to the collector directory: ${root}/example/collector
Note: You may need to visit & build the various library component directories first.
If required, type "make" to build the collector application
Step 2: Launch the collector application, and if desired a specific configuration file

The default configuration filename is: "collector.cfg"
The configuration file can be specified on the command line.

```
bash$ cd ${SDK_ROOT}/example/collector
bash$ make clean     (optional)
```

```
bash$ make host      (optional)
bash$ ./host_collector collector.cfg
```

or: Cross compile the BBB version and copy the executable to the BBB

```
bash$ make bbb
bash$ scp bbb_collector root@192.168.7.2:~/.
bash$ ssh root@192.168.7.2:~/.
root@am335x-evm# ./bbb_collector collector.cfg
```

Notes:

a) If you are using a VM, be sure to 'connect' the Launchpad USB device to the VM.

b) The device: /dev/ttyACM0 – is the debug serial port associated with the Launchpad. Details about this and the /dev/ttyACM1 device can be found in the launch pad documentation.

   By default, the configuration files assume the CC13xx CoProcessor application uses: /dev/ttyACM0 – if more than one device is present or if other "ACM" communications devices are present you may need to edit/change the configuration file.

   See the "devname" selection in the collector.cfg file for details.

c) Your username must be a member of the group "dialout" see section 4.2.1 for details.

6.1.5.2 Starting the gateway application

(When using the option 1, steps below are automatically performed by the 'run_demo.sh' script in the prebuilt directory)

In a separate shell window, do the following:

```
bash$ cd ${SDK_ROOT}/example/gateway
bash$ nodejs gateway.js            ← Use this for Ubuntu Linux
root@am335x-evm# node gateway.js ← Use this on the Beagle Bone Black
```

Note: There is a name conflict between various versions of Linux with respect to the application called "node" detailed here:
https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=614907#108
Currently: The Ubuntu distribution uses the name "nodejs"
Currently: The BeagleBone black uses the name "node"

Socket Configuration:

The gateway application is hard coded to use "localhost" port/service 5000 to communicate with the Linux collector application. The Collector socket configuration (port number or service number) is specified in the collector.cfg file.

For details see:

```
${SDK_ROOT}/example/collector/collector.cfg
${SDK_ROOT}/example/gateway/appClient/appclient.js
```

Start your web browser

(When using the option 1, steps below are automatically performed by the 'run_demo.sh' script in the prebuilt directory)

The Gateway (nodejs) web server operates on Port 1310
If you are running the gateway on your Linux host, use:

```
http://localhost:1310
```

Otherwise substitute your Beagle Bone Black IP address as required. You should now the see the screen as in Figure 6-1.

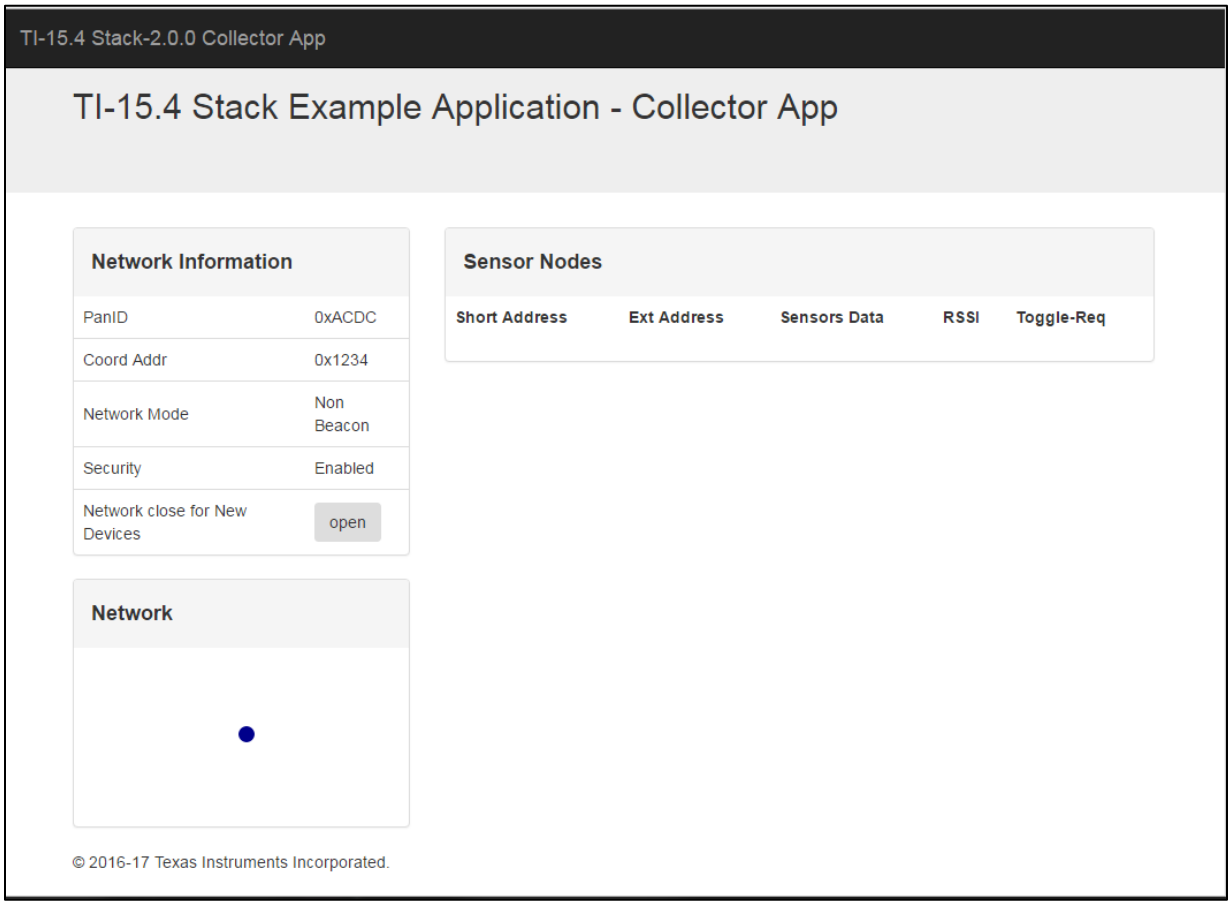


Figure 6-1: TI-15.4MAC gateway application web-application served by the local web-server after network startup

6.1.6 Joining the sensor nodes to the network

After starting the network the Collector application by default closes the network for new device joins. Once the network is open, new devices should be able to join the network. To open the network select the "Open" Button the on the web-browser.

After the network is open, power up the CC1310 LaunchPad programmed with the sensor example application. Once the device joins the network you should see “red” led solid “on” on the sensor launchpad. Also, you should see the new device, and then sensor data values appearing on the web page. After connecting several sensor nodes to the network to the network you should see a screen similar to the figure below.

Note: In frequency hopping configuration mode, the radio network is always open to new nodes.

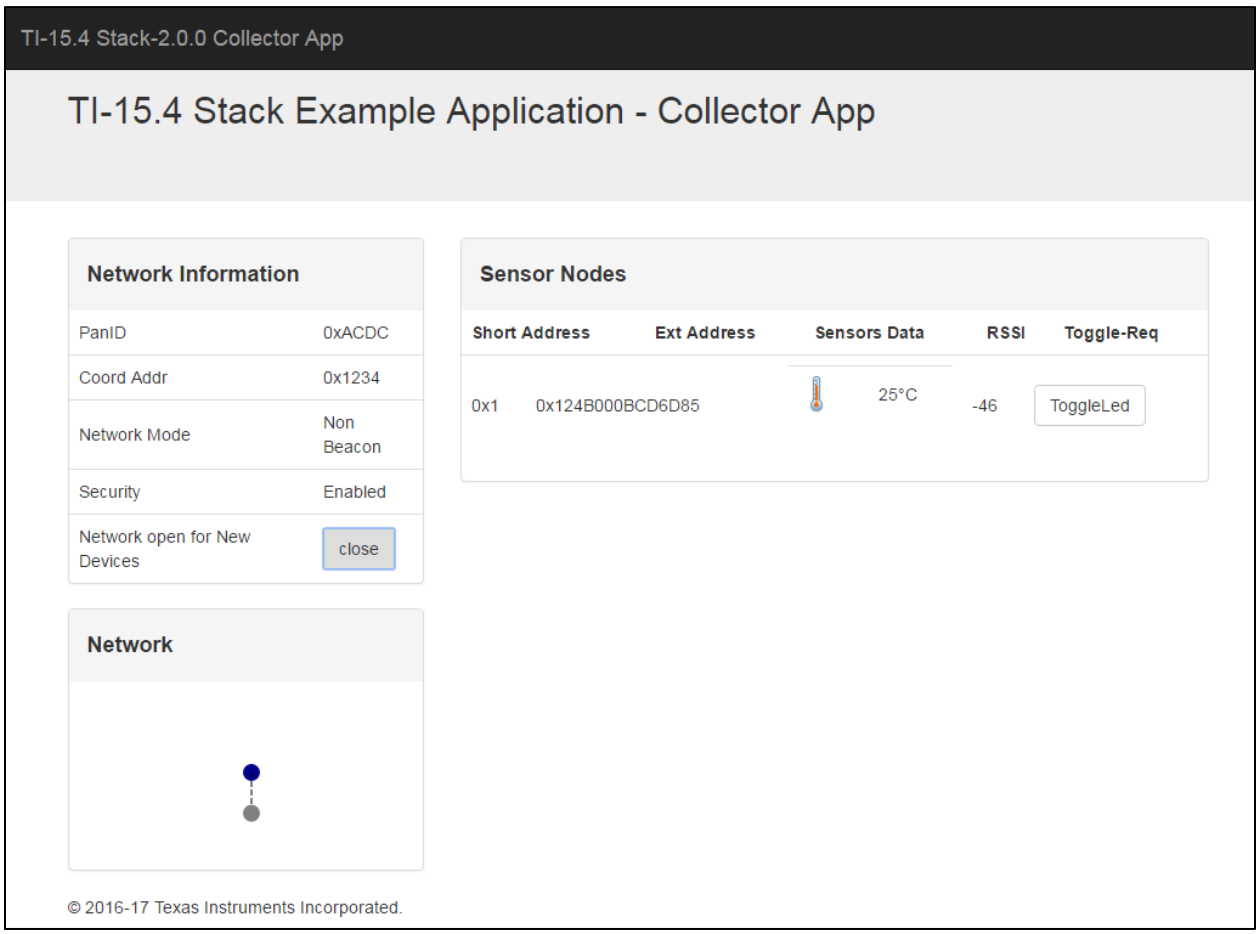


Figure 6-2: TI-15.4MAC gateway application web-application served by the local web-server after devices joined to the network

Using the front end you would then be able to send the toggle-led commands to the sensor nodes. After pressing the button you should see the “Red” Led toggle on the desired end node CC1310 Launchpad. Please note after pressing the toggleLed button there may be several second delay before the “Red” LED on the desired end node toggles. This is because the out of box sensor nodes are sleepy and wake up after sleep interval to retrieve the buffered, in this case toggle led request message, from the PAN-coordinator (collector).

6.2 Collector Application Configuration

The definitive description of the TI15.4 STACK network configuration items is described in the embedded developers guide. Below is a short summary of the key network configuration items.

6.2.1 Sensor and Collector Configuration

The embedded devices have a hard coded configuration – set by the file "config.h" in their respective prebuilt directories. There is a separate "config.h" for the Sensor and the Collector application, these settings must match otherwise they will not communicate.

6.2.2 Linux verses Embedded Config.h files

In the embedded device, various settings (discussed below) are compile time constants provided by the "config.h" file.

For example: CONFIG_SECURE - In the embedded device, this is a simple #define as shown below:

```
/*! Security Enable - set to true to turn on security */
#define CONFIG_SECURE                 true
```

In contrast, the Linux implementation uses a run time variable rather than a compile time constant, the Linux implementation has two parts

Part #1 – Is a global variable the CONFIG_SECURE macro refers to:

```
/*! Security Enable - set to true to turn on security */
extern bool linux_CONFIG_SECURE;
#define CONFIG_SECURE                 linux_CONFIG_SECURE
#define CONFIG_SECURE_DEFAULT         false
```

The Collector and other (ie CLLC and CSF) files still use the "CONFIG_SECURE" macro, the macro instead resolves to a global variable.

Part #2 – In the Linux case the global variables are in the file linux_main.c, along with their default values. At startup the file linux_main.c code reads and parses the application configuration file the contents of which may alter the value of the configuration values.

The default names of the configuration files are: collector.cfg and npi_server2.cfg

6.2.3 Setting the Channel of Operation

Configure the desired bit mask in the define CONFIG_CHANNEL_MASK in file config.h to select the desired channel(s).

```
/*!
 Channel mask - Each bit indicates if the corresponding channel is to be
 scanned First byte represents channel 0 to 7 and the last byte represents
 channel 128 to 135
 */
#define CONFIG_CHANNEL_MASK           { 0x0F, 0x00, 0x00, 0x00, 0x00, 0x00, \
                                        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \
                                        0x00, 0x00, 0x00, 0x00, 0x00 }
```

In the Linux configuration file you provide a list of channel numbers rather than a byte array of bits

6.2.4 Setting up Network Operation Mode

Network can be secure or non-secure, beacon enabled or disabled, or in frequency hopping mode. The embedded devices are configured via their config.h files, the important items are described below for desired mode of network operation

1. Frequency Selection
   The prebuilt HEX files use PHY_1, to use different frequency ranges modify the associated embedded config.h file and rebuild the firmware.

```
/*! PHY IDs - 915MHz US Frequency band operating mode # 1 */
#define APIMAC_STD_US_915_PHY_1 1
/*! 863MHz ETSI Frequency band operating mode #1 */
#define APIMAC_STD_ETSI_863_PHY_3 3
```

2. Non-Beacon Mode
   Set the defines for beacon order and superframe order as described below and set the frequency hopping mode define to false

```
#define CONFIG_MAC_BEACON_ORDER      15
#define CONFIG_MAC_SUPERFRAME_ORDER  15
#define CONFIG_FH_ENABLE             false
```

3. Beacon Enabled Mode
   Set the defines for beacon order and superframe order to desired value other than 15 such that superframe order is less than the beacon order. And set the CONFIG_FH_ENABLE to false as below

```
#define CONFIG_MAC_BEACON_ORDER      8
#define CONFIG_MAC_SUPERFRAME_ORDER  6
#define CONFIG_FH_ENABLE             false
```

4. Frequency Hopping mode
   Frequency hopping mode is selected if CONFIG_FH_ENABLE is set to true. In this configuration, values for beacon order and superframe order must be 15

```
#define CONFIG_MAC_BEACON_ORDER      15
#define CONFIG_MAC_SUPERFRAME_ORDER  15
#define CONFIG_FH_ENABLE             true
#define CONFIG_SECURE                true  /* frequency hopping must be SECURE*/
```

6.2.5 Setup MAC Data Frame Security

For Beacon or NonBeacon operation, set security to true/false by using the define listed below in the config.h file (note: Frequency Hopping requires Secure=True).

```
/*! Security Enable - set to true to turn on security */
#define CONFIG_SECURE                true
```

6.2.6 Setting up sensor reporting interval

Sensor reporting interval is defined in the file collector.c. Define the desired value in the milliseconds for the sensor reporting interval for the define listed below

```
/* Default configuration reporting interval, in milliseconds */
#define CONFIG_REPORTING_INTERVAL 30000
```

6.2.7 Setting up polling interval for sleepy sensor devices

Sleepy devices poll interval in case of non-beacon mode and frequency hopping mode can be configured by setting desired value in milliseconds for the define listed below in file collector.c:

```
/* Default configuration polling interval, in milliseconds */
#define CONFIG_POLLING_INTERVAL 3000
```

6.3 Serial Bootloader Application (Flash Update)

To use the Serial bootloader application, use the application executable with arguments described as below:

```
      ./host_cc13xx-sbl DEVICE      FILENAME   OPTIONS
Or    ./bbb_cc13xx-sbl   DEVICE     FILENAME   OPTIONS
```

Where:

DEVICE is the serial interface, for example: /dev/ttyACM0

FILENAME is the Intel hex or binary file to flash program into the device.

OPTIONS are:

- -e erase
- -p program
- -v verify

First program the CC1310 with an application that enables the boot loader feature [explained in the CC13xx TRM, Section 8]. The CoProcessor example embedded application demonstrate this feature. Specifically, the CoProcessor applications enables the bootloader feature via DIO13 Pin (Button 1 on CC1310 Launchpad). To force entry into the boot loader:

Step 1: Connect the LaunchPad

Step 2: Start the flash update tool, the tool will display “connecting…”

Step 3: Press and hold LaunchPad BTN-1

Step 4: Press and release the LaunchPad RESET button.

Step 5: The application ‘connects’ and performs the flash update

See the next page for an example of how to use the application

Connection Diagram:

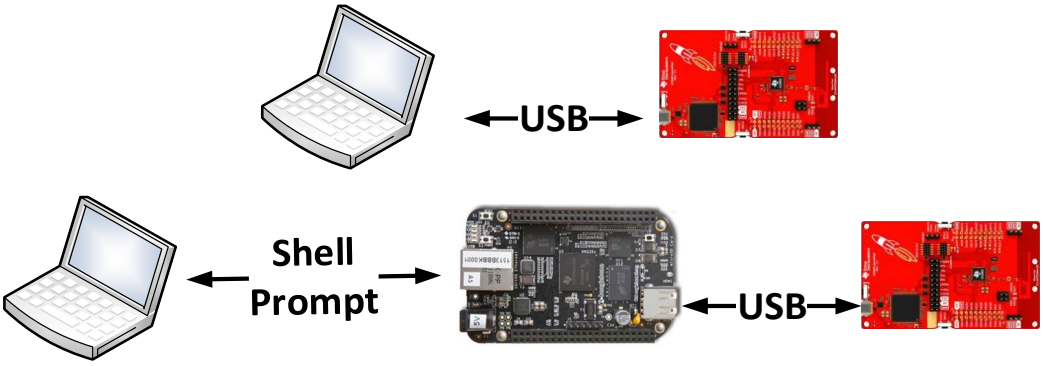

USB
Shell
Prompt
USB

Figure below shows an example of how to use the application, the steps demonstrated are:

Step 1: On the X86 Linux machine, clean & build the Beagle Bone Black (BBB) version
Step 2: Using SCP, copy the executable & hex files to the BBB
Step 3: Obtain a shell prompt on the BBB
Step 4: Execute the flash tool (see button press sequence described above)

```
bash$
bash$ make clean
/bin/rm -f host_* bbb_*
/bin/rm -rf objs
bash$
bash$ make bbb
mkdir -p objs/bbb
/home/duane/ti-processor-sdk-linux-am335x-evm-02.00.02.11/linux-devkit/sysroots/x86_64-arago-linux/usr/b
n/arm-linux-gnueabihf-gcc -c -Wall -g -std=gnu99 -I../../cc13xxdnld -I../../platform/linux   main.c  -o
bjs/bbb/main.o
/home/duane/ti-processor-sdk-linux-am335x-evm-02.00.02.11/linux-devkit/sysroots/x86_64-arago-linux/usr/b
n/arm-linux-gnueabihf-gcc -c -Wall -g -std=gnu99 -I../../cc13xxdnld -I../../platform/linux   ../../cc13x
dnld/cc13xxdnld.c  -o objs/bbb/cc13xxdnld.o
/home/duane/ti-processor-sdk-linux-am335x-evm-02.00.02.11/linux-devkit/sysroots/x86_64-arago-linux/usr/b
n/arm-linux-gnueabihf-gcc -c -Wall -g -std=gnu99 -I../../cc13xxdnld -I../../platform/linux   sblUart.c
o objs/bbb/sblUart.o
/home/duane/ti-processor-sdk-linux-am335x-evm-02.00.02.11/linux-devkit/sysroots/x86_64-arago-linux/usr/b
n/arm-linux-gnueabihf-gcc -o bbb_cc13xx-sbl objs/bbb/main.o objs/bbb/cc13xxdnld.o objs/bbb/sblUart.o
bash$
bash$ scp bbb_cc13xx-sbl root@192.168.7.2:~/.
bbb_cc13xx-sbl                                                100%   38KB  37.7KB/s   00:00
bash$
bash$ scp ../../../../firmware/coprocessor_cc1310_lp.hex root@192.168.7.2:~/.
coprocessor_cc1310_lp.hex                                     100%  310KB 310.1KB/s   00:00
bash$
bash$ ssh root@192.168.7.2
root@am335x-evm:~#
root@am335x-evm:~# ./bbb_cc13xx-sbl /dev/ttyACM0 ./coprocessor_cc1310_lp.hex -e -p -v
./bbb_cc13xx-sbl ccDnld-v1.00.00 -- Jun 23 2016 09:24:36

Opening serial port /dev/ttyACM0
Open binary file ./coprocessor_cc1310_lp.hex
Binary file size = 317514
Connecting:
Connected
Erasing:
[==================================================] 100%
Downloading:
[==================================================] 100%
Verifying:
[==================================================] 100%
Operation completed successfully
root@am335x-evm:~#
```

7 The Build System (how the makefiles work)

7.1 Building the applications from source

There are various ways to build the applications in the TI-15.4MAC Linux SDK installer.

7.1.1 Using Top Level Script

Script file "build_all.sh" can be used to build all the example applications. The table below explains various options available and their effect when using this script.

Options	Description
$ bash ./build_all.sh clean	Delete all compiled object and executable files
$ bash ./build_all.sh	Builds for host machine
$ bash ./build_all.sh host	Builds for host machine
$ bash ./build_all.sh bbb	Builds for Beagle Bone Black. (when building from the x86 machine requires the BBB cross compiler as mentioned in the Section 4.2.1)
$bash ./build_all.sh remake	Delete all compiles object and executable files and then build for the host machine

7.1.2 Use the Makefile within each each directory

Each component (library, or application) directory contains a Makefile that will build that component.

Step 1: Change to the specific directory (either application or library)

Step 2: Type "make"

The general targets supported by each Makefile are listed in the table below:

Options	Description
bash$ make	**builds the host**
bash$ make host	**builds the host**
bash$ make bbb	**builds the BBB variant**
bash$ make clean	**removes all generated files**
bash$ make remake	**make clean, followed by make**

7.2 Makefile System

Note:
The cc13xx-sbl {bootloader} is a simple self-contained application that uses a single self-contained Makefile and does not use the fragment based system described here.

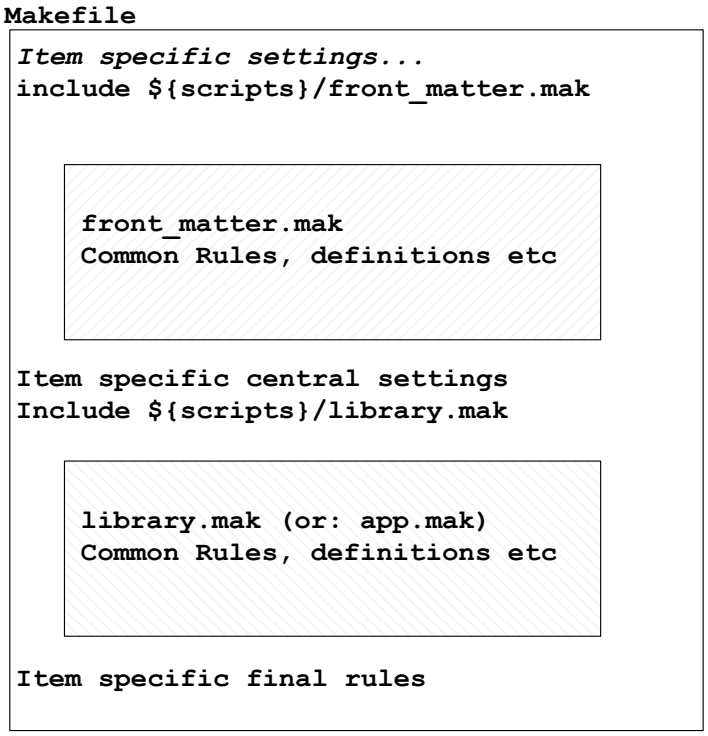
Generally there are two types of things built either (1) a static library, or (2) an application. The makefile method used in the example application is a 'makefile-fragment' scheme

There are 3 parts to this fragment based system:

- The primary Makefile – located with the source code (typically in a parent directory)

- The front_matter.mak make-fragment – in the scripts directory
- The app.mak or library.mak make-fragment – also in the scripts directory

The various components are shown graphically below, and are discussed in more detail further below, the outer box represents an example “Makefile” – the inner shaded boxes represent the fragments.



7.2.1 The primary Makefile

This file is found at the top level of each item, ie: the top directory of a library, or an application. The top portion of the Makefile does the following:

- Set a pseudo default target, typically named: _default
- Set various **${CFLAGS}**
- Include the **${scripts}/front_matter.mak**

7.2.2 The included ${scripts}/front_matter.mak

This file is located here: **${root}/scripts/front_matter.mak**
This file is generally “common-boiler-plate”, it this file determines:

- A key variable is **${ARCH}**, which specifies the type of build (host or bbb)
- The various compile (transformation) rules are listed
- The compiler generated dependency files are created and included.

7.2.3 Makefile [between the includes]

This is not a separate file; it is the portion of the original Makefile between the two include statements. In this section, the following items are listed:

- Source Files, ie: **${C_SOURCES}**
- Libraries that are used
- Any additional rules required (for example using the protobuf compiler)

7.2.4 The included ${scripts}/library.mak or ${scripts}/app.mak

This file is either: **${root}/scripts/library.mak**

Or, this file is: **${root}/scripts/app.mak**

- Creates the library
- Or creates the executable

7.2.5 Makefile [final portion]

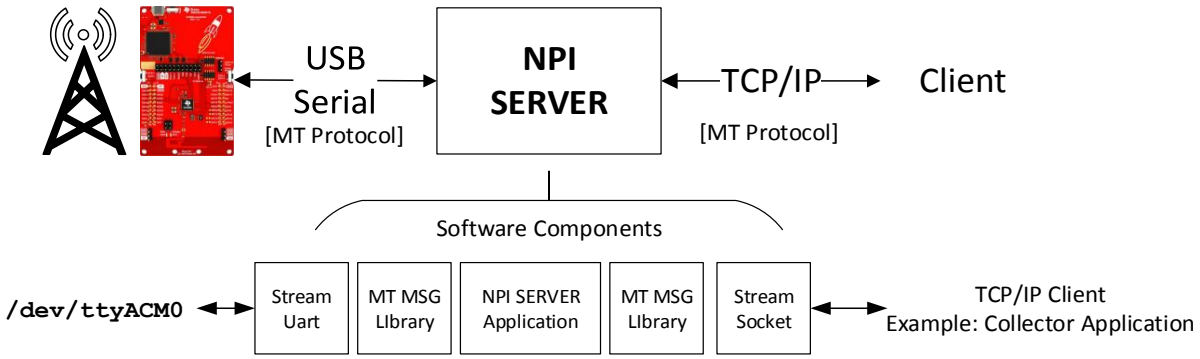
Again, this is not a separate file but is the final portion of the Makefile after the include statements.

- Contains directory, application, or library specific rules

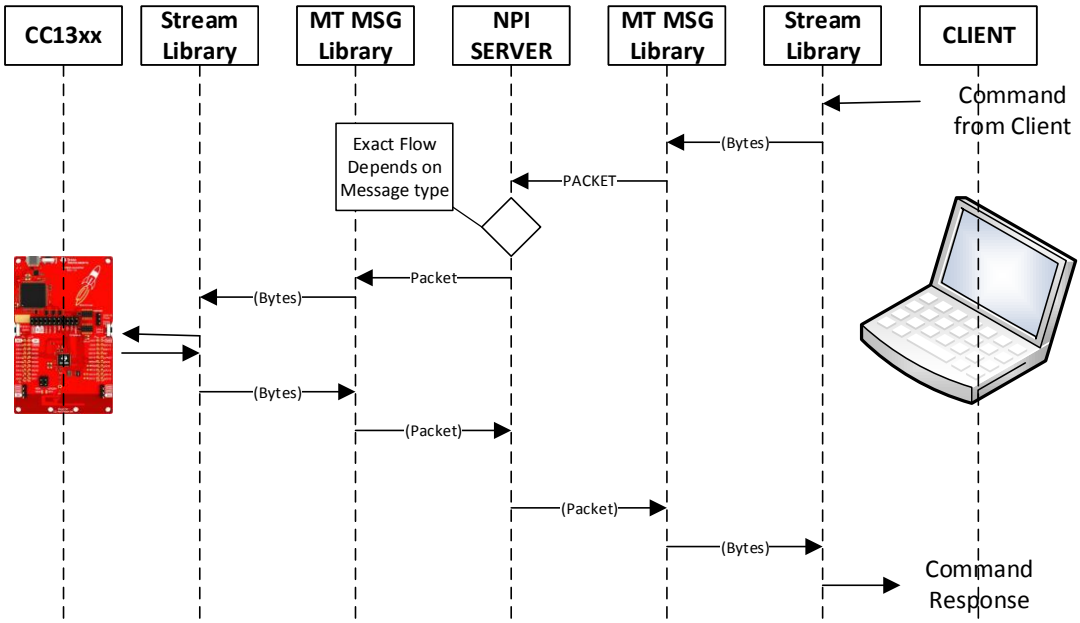
8 NPI Server In Detail

8.1 Overview

At a very high level - the NPI (Network Processor Interface) Server is a utility application that translates MT Packets to and from a TCP/IP connection and a serial port, a data flow diagram is shown below. The NPI Server itself is a rather small body of code, the bulk of the application comes from the Common Library (Supplying the Socket & Uart API), and the APIMAC library (Supplying the MT message handling layer).



Below is a UML style actor diagram showing the data flow diagram of the NPI Server, starting from the left hand side, the Linux client sends a message – the Stream (socket) library provides a byte stream interface to the MT MSG Library (part of the APIMAC library), the complete packet is presented to the application (the NPI SERVER), which depending upon the type of message decides how to forward, the message follows a similar route to the embedded device. Replies and/or asynchronous notifications from the device follow the same basic flow in reverse.



Description and data walk through:

1) The right side represents the client application on some PC. The connection is via a socket. The client application sends a message.
2) The STREAM library provides a byte stream interface for the MT Message Interface Layer.
3) The MT layer assembles the packet (if required, the MT layer will defragment the message) and presents a "packet" to the application layer.
4) The Application code (in the center) decides what to do with each message, and how or if the message should be forwarded to the other interface. For example – a Synchronous Request [SREQ] should produce a Synchronous Response (SRSP).
5) The message effectively follows the reverse procedure, through the MT layer and to the STREAM interface, in this case for a serial port.

8.2 Key Feature & Highlights:

- **Message Format (Message Geometry) Translation**
  In general, the embedded device (serial port) requires the MT Message Frame to have this format:

0xFE	LEN	CMD0	CMD1	(LEN) Payload bytes	FCS

  The byte sequence is defined as follows, the embedded column represents the CC13xx UART interface, the Socket column represents the Stream Socket interface between the NPI Server and the client.

Byte	Embedded	Socket	Description
0xfe	Required	Optional	Frame synchronization (message start) byte
LEN	1 Byte	2 Bytes	Length in bytes of the payload portion (LSB first)
CMD0	1 Byte	1 Byte	bits[7: 5] message type bits[4: 0] subsystem number
CMD1	1 Byte	1 Byte	Command byte for specific subsystem
(bytes)	Variable	Variable	Zero or more bytes, command specific
FCS	1 byte	Optional	XOR check value of the CMD0, CMD1 and (data)

- **Large Messages (Fragmentation):**
  Fragmentation can occur in either direction (HOST to CC13xx, or CC13xx to HOST)
  The embedded CC13xx firmware only supports a single byte for a length, to support large messages, fragmentation is required.
  The MT library provides this support, to transmit a large message; it is broken down and transmitted as a series of smaller fragments. Incoming fragmented messages are reassembled as needed.
- **Protocol Geometry is configurable via a configuration file**
  All of the above 'message format' geometry values are configurable via the NPI_SERVER2.CFG file. In addition, other geometry items {timeout, retry, etc} are also configurable. See the screen shot below, which is form the fi le: "npi_server2.cfg"

Note: The example collector application is configurable in the same way via ‘collector.cfg’

```
[uart-interface]
        include-chksum = true
        frame-sync = true
        fragmentation-size = 240
        retry-max = 3
        fragmentation-timeout-msecs = 1000
        intersymbol-timeout-msecs = 100
        srsp-timeout-msecs = 1000
        len-2bytes = false
        flush-timeout-msecs = 50

[socket-interface]
        include-chksum = false
        frame-sync = false
        fragmentation-size = 240
        retry-max = 3
        fragmentation-timeout-msecs = 1000
        intersymbol-timeout-msecs = 100
        srsp-timeout-msecs = 1000
        len-2bytes = true
        flush-timeout-msecs = 10
```

9 Linux SW Components

9.1 The Component NV (Non-Volatile) Library

Documentation can be found here: **`${root}/components/nv/inc/nvintf.h`**

9.1.1 Where used

The example Collector application uses this library component to simulate non-volatile storage that would be found on an embedded device. More detail about the NV module can be found in the embedded device documentation

9.1.2 Key Features & Highlights

- In the embedded device, two pages of FLASH memory are used as the data page. In the host (linux) environment, this is simulated via a simple file.
- Updating or creating an item: As items are written, or new items are created – the current NV page is scanned, and the old item is marked as invalid.
- Ping Pong Pages: Eventually the current page will fill, when that occurs the software will automatically compact the data into the other page, and erase the current page. A version (sequence number) is used to disambiguate the two pages if something goes wrong (ie: Power failure in the middle of compacting the page)

9.2 The Component MT (Layer) Library

Documentation can be found here: **`${root}/components/api/inc/mt_msg.h`**

Note: The MT Layer is a portion of the API Mac Library and is thus contained within the API mac library.

In general, the MT component provides a means to send and receive a message (mt_msg) through an interface (mt_interface). The interface could be a simple serial port, or a TCP/IP socket.

A message transmitted across an interface is a bytes stream in the following format:

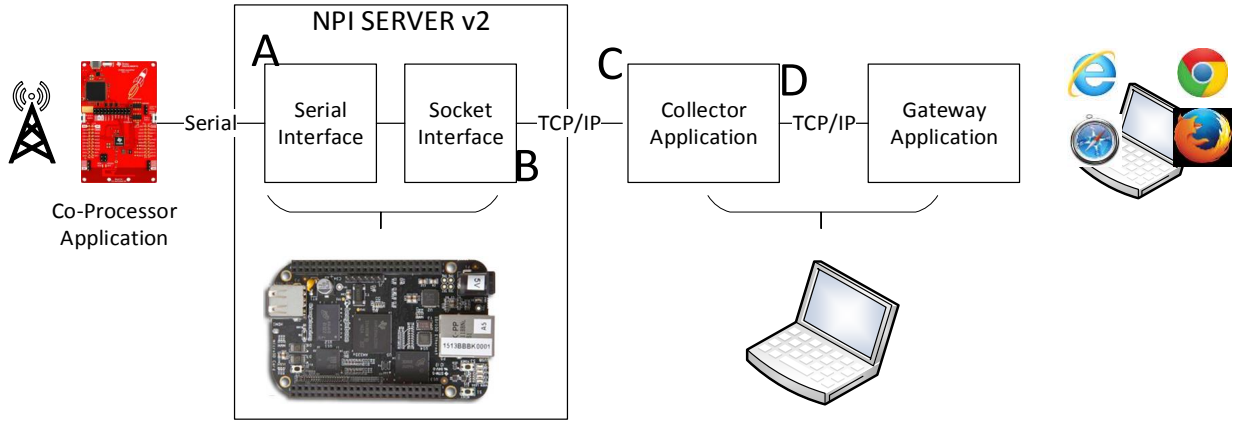
1) An optional frame synchronization byte: 0xfe? (Present or Not-Present)
2) A 1 or 2 byte length field? (1 or 2)
3) A command 0 byte (contains message type, and subsystem id number)
4) A command 1 byte (specific to the subsystem)
5) Zero or more payload bytes specific to the command bytes.
6) An optional Frame Check (xor-checksum) (Present or Not-Present)

9.2.1 Where used

The illustration below shows an configuration where the “gateway” application is running on your X86 Linux host machine, the NPI Server is running on a Beagle Bone Black – perhaps in some remote location far away.

In this example system, the “MT Message” component is used in 4 distinct places, see the diagram below for reference:

A) The NPI Server – providing a Serial (UART) MT interface to the Launch Pad.
B) The NPI Server – providing an MT TCP/IP Socket interface to the network.
C) The Collector application communicating with the remote NPI Server via TCP/IP
D) The Collector application uses the MT Protocol to encapsulate “Google Protobuf” messages to and from the Node-JS Gateway (web server) application



9.2.2 Important API functions:

The collector application, and NPI application configure the interface via their respective config files, this function handles the INI settings for an interface. For examples see: “linux_main.c” in the npi_server and collector applications.

```
int MT_MSG_INI_settings(struct ini_parser *pINI,
                        bool *handled,
                        struct mt_msg_interface *pIface);
```

The “settings” function handles a config file fragment like this:

```
[uart-interface]
        include-chksum = true
        frame-sync = true
        fragmentation-size = 240
        retry-max = 3
        fragmentation-timeout-msecs = 1000
        intersymbol-timeout-msecs = 100
        srsp-timeout-msecs = 1000
        len-2bytes = false
        flush-timeout-msecs = 50
```

Once the data structure has been configured, the interface is initialized or created via:

```
int MT_MSG_interfaceCreate(struct mt_msg_interface *pMI);
```

As an example of how to send and receive messages, review this function in the source code:

```
int MT_MSG_getVersion(struct mt_msg_interface *pDest,
                      struct mt_version_info *pInfo);
```

9.3 The Component API MAC Layer

The API MAC interface is documented here: **`${root}/components/api/inc/api_mac.h`**

Also see the Embedded Developers guide where “Doxygen Generated” content can be found.

10 The Component Common Library

This section describes at a high level some of the features in the **`${root}/components/common`** library. The goal of this document is a quick overview or introduction with pointers to where to find the detail. The various components are documented in detail in their respective header files.

10.1 Opaque types `intptr_t` verses `void *`

An important of portable code is that it runs across multiple platforms transparently. Another important item is the ability to create opaque types. In the past developers often use an integer as an opaque type, or void pointer as an opaque type.

On a 32bit ARM target (Beagle Bone Black) the compiler uses a 32bit pointer and a 32bit integer, casting a pointer to an integer is not a problem.

Specifically – an 64bit host is a problem, the sizeof(pointer) != sizeof(int) , improper handling of opaque types is often a source of bugs, **Solution:** The C99 standard introduced an **`intptr_t`** – which is a compiler supplied type that always works for both types. Hence, all opaque types in the common library use **`intptr_t`** for opaque values.

10.2 OS Abstraction: Semaphores, Mutexes, Threads, Timers

These features are documented here:

Description	Location
Mutex	**`${root}/components/common/inc/mutex.h`**
Semaphore	**`${root}/components/common/inc/semaphore.h`**
Threads	**`${root}/components/common/inc/threads.h`**
Timer	**`${root}/components/common/inc/timer`**

10.2.1 Where used

- **Threads**: In the collector application the primary “collector” application is a thread, there is also a timer thread. In the NPI Server a unique thread is created for each attached client.
- **Semaphores**: Are used to indicate when messages arrive and require service.
- **Mutexes**: Are used to lock access to shared resources across multiple threads

10.2.2 Important API elements

Timers:

- All time units are in milliseconds, and are relative to the request.
- The time value -1 (negative) means wait or block forever
- The time value 0 (zero) means: Test, do not block return fail/error immediately if not successful
- Positive time values represent the time in milliseconds.
- Long time outs (more than 20 days) are not supported

Mutexes, Semaphore, Threads:

- Have a “dbg_name” element for log identification purposes
- supports recursive locks and a timeout parameter
- Synchronization calls such as: Lock, Unlock, Get, Put and calls are present

10.3 Stream Interface

The stream interfaces are documented here:

Description	Location
Stream (generic)	**`${root}/components/common/inc/stream.h`**
Stream (uart)	**`${root}/components/common/inc/stream_uart.h`**
Stream (socket)	**`${root}/components/common/inc/stream_socket.h`**
Socket (common)	**`${root}/components/common/inc/stream_socket_private.h`**

10.3.1 Where used

The Stream module is used for the log files, configuration files, serial ports and sockets.

At a high level streams are simply a stream of bytes that can be read, or written.

The source (or destination) of a stream can be a file, a memory buffer, memory, a serial-uart, or a socket. All of these are abstracted as and can be treated as a “generic-stream-of-bytes”.

10.3.2 Important API elements

Opening or creating a stream, upon success returns a non-zero opaque `intptr_t` result

```
intptr_t STREAM_createUart(const struct uart_cfg *pCFG);
intptr_t SOCKET_CLIENT_create(struct socket_cfg *pCFG);
int SOCKET_CLIENT_connect(intptr_t h);
intptr_t STREAM_createWrFile(const char *filename);
intptr_t STREAM_createRdFile(const char *filename)
```

Closing:

```
void STREAM_close(intptr_t h);
```

Reading & Writing Bytes:

```
int STREAM_wrBytes(intptr_t h, const void *databytes, size_t nbytes, int
timeout_mSecs);
int STREAM_rdBytes(intptr_t h, void *databytes, size_t nbytes, int timeout_mSecs);
int STREAM_rxAvail(intptr_t h, int mSecs_timeout);
```

10.4 LOG file

The full LOG facility is documented here: **`${root}/components/common/inc/log.h`**

10.4.1 Where used

The library library components and applications create files using this log facility.

10.4.2 Important API Elements

10.4.2.1 Opening a log file

Logs are written to a file, specified by this function call:

```
void LOG_init(const char *filename);
```

10.4.2.2 Writing to the log printf() style

Logs are written using a **`printf()`** like function:

```
void LOG_printf(logflags_t whybits, const char *fmt, ...);
Example:
LOG_printf( LOG_ALWAYS, “the answer is %d\n”, 42 );
```

10.4.2.3 Controlling what is, or is not logged

What exactly is logged (enabled) or not logged (disabled) is controlled by the “whybits” parameter. To be clear this is not a “log level”, it is instead a (64bit) bitmask, and is managed by the LOG_test() function, which in pseudo code is shown below:

```
bool LOG_test(logflags_t whybits)
{
    // This is [pseudo-code], not actual code from the module
    // ie: LOG_ALWAYS, or LOG_ERROR
    if( special_case_true( whybits ) ){
        return true; // log the message
    }

    if (special_case_false(whybits)) {
        return false; // do not log the message
    }

    if( whybits & log_cfg.log_flags ){
        return true; // log the message
    } else {
        return false; // do nog log.
    }
}
```

10.4.2.4 Log Configuration

The application logging can be configured via the "ini-configuration" file, included in the **`${root}/components/common`** library is an log/ini-file callback function that can configure various aspects of the log feature, see the function: `LOG_INI_settings()`

Below is a small excerpt from a configuration file. The important items to note are:

(a) The name/value pair: 'filename' – specifies the log filename.
(b) The value "flag" enables a specific log (bit), for example "flag = foo" will use a look up table to determine the value of the "foo" bit, and set that bit in the **`log_cfg.log_flags`**
(c) The magic flag name, 'everything' which turns on (sets all) of the log bits enabling everything.
(d) Otherwise the 'named-bit' is set (equal 1)
(e) If the name exactly "not-", the bit is cleared (equal 0)

```
[log]
       filename = collector_log.txt
       ;---------------------------------------
       ; The flag name 'everything' is magic, it turns on everything.
       ; setting all of the log bits to 1.
       ;---------------------------------------
       flag = everything
       flag = not-sys_dbg_mutex
       flag = not-sys_dbg_thread
```

10.4.3 Example log output

The log file content looks like this:

```
1.012: uart: TX Msg (start) [pib-set-common]
1.012: pib-set-common msg(0002) nbytes=22 len=17 [ 0xfe 0x11 0x22 0x09 0x52 0x01 0x00 0x00]
1.012: uart: TX 22 bytes
1.012: 00000000: fe 11 22 09 52 01 00 00-00 00 00 00 00 00 00 00 |..".R...........|
1.012: 00000010: 00 00 00 00 00 69      -                       |.....i          |
1.012: uart: TX Msg (Complete) r=22 [pib-set-common]
```

The first column shows the time in seconds & milliseconds since the application was started.

10.5 INI Files

The INI component is documented here: **`${root}/components/common/inc/ini_file.h`**

10.5.1 Where used

The example applications: collector and the npiserver2 are configured via a text file in INI format. Both applications read the INI files via the INI common library component.

10.5.2 File Format

Comments start with a ";" (semi-colon), or a "#" (hash mark) and extend to the end of the line. There are sections, denoted by names in [square-brackets]. Values specific to that section title are indented and are in the form: "name = value", values may strings, Booleans, or numeric values.

For example, the figure below is portion of the collector configuration file.

```
; comment
[application]
    # Comment
    interface = uart
    config-secure = true
    config-pan-id = 0xacdc
    config-coord-short-addr = 1024
    config-mac-beacon-order = 15
    config-mac-superframe-order = 15
```

10.5.3 Important API elements

10.5.3.1 Reading the INI file

The primary means to read a configuration file is:

```
int INI_read(const char *filename,
             ini_rd_callback *callback_function_ptr,
             intptr_t client_cookie);
```

The actual content of the file is managed by the callback function, below is a code-snippet from the collector application. Highlights are: (1) An array of sub handlers to handle various sections (2) a simple iteration loop over the array.

```
/* Callback for parsing the INI file. */
static int cfg_callback(struct ini_parser *pINI, bool *handled)
```

```
{
    int x;
    int r;

    static ini_rd_callback * const ini_cb_table[] = {
        LOG_INI_settings,
        my_UART_INI_settings,
        my_SOCKET_INI_settings,
        my_MT_MSG_INI_settings,
        my_APP_settings,
        /* NOTE: Add More handlers here */
        /* Terminate list */
        NULL
    };

    for(x = 0 ; ini_cb_table[x] ; x++)
    {
        r = (*(ini_cb_table[x]))(pINI, handled);
        if(*handled)
        {
            return r;
        }
    }
    /* let the system handle it */
    return 0;
}
```

10.5.3.2 Handling Values – Callback example

Below is an example callback function that demonstrates how to

1. recognize the section & item-name
2. print an error message,
3. Obtain an integer value from the configuration file.

Other examples of this feature can be found in:

- **${root}/components/mt/src/mt_msg_ini.c**
- **${root}/components/common/src/stream_*_ini.c**

```
static int foobar_callback(struct ini_parser *pINI, bool *handled)
{
    // this check both the section name, and the item name.
    // To match only the [section], pass NULL as the 3rd parameter
    if (!INI_itemMatches(pINI, "foo", "bar")) {
        // This item is not section: [foo], item “bar”
        return 0;
    }

    // Print item value as a string...
    // Also see: INI_isValueBool(), or INI_valueAsInt()
    printf("BAR as a string is: %s\n", pINI->item_value);

    if ( ! INI_isValueInt(pINI) ) {
```

```
        // Also see isValueBool()
        INI_syntaxError(pINI, "[%s] %s = %s not an integer\n",
            pINI->cur_section, pINI->item_name, pINI->item_value);
        return -1;
    }
    // as an integer
    printf("BAR is: %d\n", INI_valueAsInt(pINI));
    *handled = true;
    return 0;
}
```

11 Trouble Shooting

Trouble shooting should be performed in layers:

Layer #1 – Embedded Devices

Use the Flash Programmer 2 and verify the embedded device is programed correctly.
See Section 4.2.3.1

Attach an LCD Booster Pack to the Embedded Sensor verify the screen.

Program a LaunchPad with a prebuilt embedded “Collector Hex file” (attach a Booster Pack LCD if available) and Join the embedded collector with the embedded sensor.

Layer #2 – Move to the Linux Collector Application

Is the /dev/ttyACM0 device present or it cannot be opened?

a) Verify the device: /dev/ttyACM* is present the Launchpad uses /dev/ttyACM[0..99]

   If not – examine the “dmesg” output to see if the USB device is being recognized
   If not – Determine if the USB ACM module is present in your Linux Kernel

b) If you are using a VM – did you disconnect (from the host) and connect the Launchpad to the Virtual Machine?

c) The collector.cfg file assumes the name: /dev/ttyACM0 is the Launchpad
   If other ACM devices are present, edit the configuration file

d) Are you a member of the group “dialout” – if not add yourself to this group.

e) Some VMs (or kernel drivers and features) take about 10 to 15 seconds to complete the USB process and release the serial port for use by an application. Whereas the BBB standalone Linux machine the process is complete in about 1 second

Enable Logging?

f) Change the application configuration file and enable logging See Section 10.4, enable everything (but not the SYS items) – within the first 2 to 3 seconds you should see a series of commands going to the CoProcessor and responses, the log is very verbose – and includes complete hex dumps of every message transfer.

g) It may be helpful to enable the ‘dup2stderr = true’ rather than review log files

h) In general, the collector application within the first few seconds should communicate with the coprocessor device, each message transferred is dumped as hex bytes – verify that the embedded device is responding to messages.

NV (overview) Settings – Perhaps an invalid configuration is stored in the NV simulation

Typically NV settings cause “joining” and “sensor connection” problems.

i) The Linux collector stores the simulated NV configuration in a file called: “nv-simulation.bin” – to reset the NV – simply remove this file, a new one will be created as needed.

j) The Collector configuration file supports a “load-nv-sim = false” option to force the NV settings to be reset each time the application starts.

k) Did the embedded sensor device “join” another network? The example sensor application at startup reads the buttons (right or BTN-2) if the button is pressed the sensor example will clear/reset the NV parameters.

Gateway Application

l) Node or NodeJS? See discussion in section 6.1.5

m) Enable logging in the collector application (See above, this section). Each message from the Gateway (nodejs application) is logged and hex dumped to the log file.

n) The Gateway application default is hard coded to connect to the localhost, on a specific port. See section 6.1.5 – Socket Configuration for details.

IP network issues?

o) Try to “ping” the other devices IP address?

p) Is your laptop (or company) “firewall” preventing access?

q) Use fixed IP address (not randomly assigned DHCP addresses)

SSH Connection Problems – aka: “Man in the middle”

Putty warning titled: “WARNING – POTENTIAL SECURITY BREACH”

Linux equivalent: “WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED”

To explain requires some background information:

IMPORTANT: Please understand this document is not a security document, network security is beyond the scope of this document – the information provided here is informational and is intended to help an engineer determine what might have happened.

Step 1 – Background – part 1

When first connecting to the BBB, your SSH client will ask to verify the new host SSH key. In the Linux example below, the user answered "yes" and accepted the new host key. (Putty uses a very windows dialog box in a similar style)

```
bash$ cd prebuilt
bash$ scp bbb_prebuilt.tar.gz  root@192.168.7.2:/home/root/.
The authenticity of host '192.168.7.2 (192.168.7.2)' can't be established.
RSA key fingerprint is [redacted]
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.7.2' (RSA) to the list of known hosts.
bbb_prebuilt.tar.gz                           100% 2246KB   2.2MB/s   00:00
bash$
```

Step 2 – Background – Part 2

When the user selected "yes" the SSH tool saves the remote host key (in this case, the key comes from the Beagle Bone Black). In Linux, this is saved in your ${HOME}/.ssh directory. Putty uses the windows registry.

Most often the key is associated with an IP address, or with a 'hostname'.

Where does this key come from? At first power up, the BBB creates a new random SSH HOST key and saves this key on its local SD Card – the next power up the BBB will reuse the saved key.

Important: *This host key is a RANDOM value, and there are now two copies of this key, copy #1 is on the BBB, and copy #2 is stored on your development laptop.*

Step 3 – The cause of the problem

Previously the connection was with the "old-beagle-bone-black" SD Card (with the old SSH key).

Most commonly the BBB has the same IP address – but the SD Card content is different.

More specifically that random SSH HOST KEY is different; by definition each key is random.

Example: Reformat the SD Card and a new random SSH key will be created. Or swap BBB devices and thus a different SD Card is in use, with a different random key.

Step 4 – The Error Message

From your laptop, you connect to the pseudo-new BBB, effectively this is a “different or new” SD Card. And thus the BBB will present a different random host key to your Laptop.

The security policy of SSH is to verify the remote host key with all known (and saved) host keys. Since a connection was made previously, the old key was saved. The correct security policy is to refuse the connection until the user affirmatively resolves the host key change.

Because most development environments are very closed, and very controlled updating the host key is often the correct action to take.

With the continual focus on product security it is very important to repeat the earlier warning: This document is not a security document – TI strongly encourages you to study and learn about various types of network security and make informed decisions about how your product security functions.

---

12 TIMAC 2.0 API

The following is the application programming interface (API) for the Texas Instruments 802.15.4 MAC software. This API provides an interface to the management and data services of the 802.15.4 stack.

12.1.1 Callback Functions

These functions must be implemented by the application and are used to pass events and data from the MAC to the application. Data accessed through callback function parameters (such as a pointer to data) are only valid for the execution of the function and should not be considered valid when the function returns. These functions execute in the context of the MAC. The callback function implementation should avoid using critical sections and CPU intensive operations. The callback table structure should be setup application then call ApiMac_registerCallbacks() to register the table.

12.1.2 Common Constants and Structures

- **Address Type** - the common address type used by the MAC is the ApiMac_sAddr_t.
- **Status Values** - the common MAC status type is ApiMac_status_t.
- **MAC Security Level** - The security level (ApiMac_secLevel_t) defines the encryption and/or authentication methods used on the message frame.
- **Key Identifier Modes** - The key identifier mode (ApiMac_keyIdMode_t) defines how the key is determined from the key index
- **Security Type** - MAC security structure (ApiMac_sec_t).

12.1.3 Initialization and Task Interfaces

- ApiMac_init()
- ApiMac_registerCallbacks()
- ApiMac_processIncoming()

12.1.4 Data Interfaces

- ApiMac_mcpsDataReq()
- ApiMac_mcpsPurgeReq()

12.1.5 Management Interfaces

- ApiMac_mlmeAssociateReq()
- ApiMac_mlmeAssociateRsp()
- ApiMac_mlmeDisassociateReq()
- ApiMac_mlmeOrphanRsp()
- ApiMac_mlmePollReq()
- ApiMac_mlmeResetReq()
- ApiMac_mlmeScanReq()
- ApiMac_mlmeStartReq()
- ApiMac_mlmeSyncReq()
- ApiMac_mlmeWSAsyncReq()

12.1.6 Management Attribute Interfaces

The MAC attributes can be read and written to by use of the following Get and Set functions, which are organized by the attributes data type:

- ApiMac_mlmeGetReqBool()
- ApiMac_mlmeGetReqUint8()
- ApiMac_mlmeGetReqUint16()
- ApiMac_mlmeGetReqUint32()
- ApiMac_mlmeGetReqArray()
- ApiMac_mlmeGetFhReqUint8()
- ApiMac_mlmeGetFhReqUint16()
- ApiMac_mlmeGetFhReqUint32()
- ApiMac_mlmeGetFhReqArray()
- ApiMac_mlmeGetSecurityReqUint8()
- ApiMac_mlmeGetSecurityReqUint16()
- ApiMac_mlmeGetSecurityReqArray()
- ApiMac_mlmeGetSecurityReqStruct()
- ApiMac_mlmeSetReqBool()
- ApiMac_mlmeSetReqUint8()
- ApiMac_mlmeSetReqUint16()
- ApiMac_mlmeSetReqUint32()
- ApiMac_mlmeSetReqArray()
- ApiMac_mlmeSetFhReqUint8()
- ApiMac_mlmeSetFhReqUint16()
- ApiMac_mlmeSetFhReqUint32()
- ApiMac_mlmeSetFhReqArray()
- ApiMac_mlmeSetSecurityReqUint8()
- ApiMac_mlmeSetSecurityReqUint16()
- ApiMac_mlmeSetSecurityReqArray()
- ApiMac_mlmeSetSecurityReqStruct()

12.1.7 Simplified Security Interfaces

- ApiMac_secAddDevice()
- ApiMac_secDeleteDevice()
- ApiMac_secDeleteKeyAndAssocDevices()
- ApiMac_secDeleteAllDevices()
- ApiMac_secGetDefaultSourceKey()
- ApiMac_secAddKeyInitFrameCounter()

12.1.8 Extension Interfaces

- ApiMac_randomByte()
- ApiMac_updatePanId()
- ApiMac_startFH()
- ApiMac_enableFH()
- ApiMac_parsePayloadGroupIEs()
- ApiMac_parsePayloadSubIEs()
- ApiMac_freeIEList()
- ApiMac_convertCapabilityInfo()
- ApiMac_buildMsgCapInfo()

12.2 File Documentation

12.3 api_mac.h File Reference

12.3.1 Data Structures

- struct ApiMac_sAddr_t
- struct ApiMac_sData_t
- struct ApiMac_MRFSKPHYDesc_t
- struct ApiMac_sec_t
- struct ApiMac_keyIdLookupDescriptor_t
- struct ApiMac_keyDeviceDescriptor_t
- struct ApiMac_keyUsageDescriptor_t
- struct ApiMac_keyDescriptor_t
- struct ApiMac_deviceDescriptor_t
- struct ApiMac_securityLevelDescriptor_t
- struct ApiMac_securityDeviceDescriptor_t
- struct ApiMac_securityKeyEntry_t
- struct ApiMac_securityPibKeyIdLookupEntry_t
- struct ApiMac_securityPibKeyDeviceEntry_t
- struct ApiMac_securityPibKeyUsageEntry_t
- struct ApiMac_securityPibKeyEntry_t
- struct ApiMac_securityPibDeviceEntry_t
- struct ApiMac_securityPibSecurityLevelEntry_t
- struct ApiMac_capabilityInfo_t
- struct ApiMac_txOptions_t
- struct ApiMac_mcpsDataReq_t
- struct ApiMac_payloadIeItem_t
- struct ApiMac_payloadIeRec_t
- struct ApiMac_mcpsDataInd_t
- struct ApiMac_mcpsDataCnf_t
- struct ApiMac_mcpsPurgeCnf_t
- struct ApiMac_panDesc_t
- struct ApiMac_mlmeAssociateReq_t
- struct ApiMac_mlmeAssociateRsp_t
- struct ApiMac_mlmeDisassociateReq_t
- struct ApiMac_mlmeOrphanRsp_t
- struct ApiMac_mlmePollReq_t
- struct ApiMac_mlmeScanReq_t
- struct ApiMac_mpmParams_t
- struct ApiMac_mlmeStartReq_t
- struct ApiMac_mlmeSyncReq_t
- struct ApiMac_mlmeWSAsyncReq_t
- struct ApiMac_secAddDevice_t
- struct ApiMac_secAddKeyInitFrameCounter_t

- struct ApiMac_mlmeAssociateInd_t
- struct ApiMac_mlmeAssociateCnf_t
- struct ApiMac_mlmeDisassociateInd_t
- struct ApiMac_mlmeDisassociateCnf_t
- struct ApiMac_beaconData_t
- struct ApiMac_coexist_t
- struct ApiMac_eBeaconData_t
- struct ApiMac_mlmeBeaconNotifyInd_t
- struct ApiMac_mlmeOrphanInd_t
- struct ApiMac_mlmeScanCnf_t
- struct ApiMac_mlmeStartCnf_t
- struct ApiMac_mlmeSyncLossInd_t
- struct ApiMac_mlmePollCnf_t
- struct ApiMac_mlmeCommStatusInd_t
- struct ApiMac_mlmePollInd_t
- struct ApiMac_mlmeWsAsyncCnf_t
- struct ApiMac_callbacks_t
- union ApiMac_sAddr_t.addr
- union ApiMac_mlmeBeaconNotifyInd_t.beaconData
- union ApiMac_mlmeScanCnf_t.result

12.3.2 Macros

- #define APIMAC_KEY_MAX_LEN 16
- #define APIMAC_SADDR_EXT_LEN 8
- #define APIMAC_MAX_KEY_TABLE_ENTRIES 2
- #define APIMAC_KEYID_IMPLICIT_LEN 0
- #define APIMAC_KEYID_MODE1_LEN 1
- #define APIMAC_KEYID_MODE4_LEN 5
- #define APIMAC_KEYID_MODE8_LEN 9
- #define APIMAC_KEY_SOURCE_MAX_LEN 8
- #define APIMAC_KEY_INDEX_LEN 1
- #define APIMAC_FRAME_COUNTER_LEN 4
- #define APIMAC_KEY_LOOKUP_SHORT_LEN 5
- #define APIMAC_KEY_LOOKUP_LONG_LEN 9
- #define APIMAC_MAX_KEY_LOOKUP_LEN APIMAC_KEY_LOOKUP_LONG_LEN
- #define APIMAC_DATA_OFFSET 24
- #define APIMAC_MAX_BEACON_PAYLOAD 16
- #define APIMAC_MIC_32_LEN 4
- #define APIMAC_MIC_64_LEN 8
- #define APIMAC_MIC_128_LEN 16
- #define APIMAC_MHR_LEN 37
- #define APIMAC_CHANNEL_PAGE_9 9
- #define APIMAC_CHANNEL_PAGE_10 10
- #define APIMAC_STANDARD_PHY_DESCRIPTOR_ENTRIES 3
- #define APIMAC_GENERIC_PHY_DESCRIPTOR_ENTRIES 3
- #define APIMAC_STD_US_915_PHY_1 1
- #define APIMAC_STD_US_915_PHY_2 2

- #define APIMAC_STD_ETSI_863_PHY_3 3
- #define APIMAC_MRFSK_GENERIC_PHY_ID_BEGIN 128
- #define APIMAC_MRFSK_GENERIC_PHY_ID_END 143
- #define APIMAC_MRFSK_STD_PHY_ID_BEGIN APIMAC_STD_US_915_PHY_1
- #define APIMAC_MRFSK_STD_PHY_ID_END APIMAC_STD_ETSI_863_PHY_3
- #define APIMAC_PHY_DESCRIPTOR 0x01
- #define APIMAC_ADDR_USE_EXT 0xFFFE
- #define APIMAC_SHORT_ADDR_BROADCAST 0xFFFF
- #define APIMAC_SHORT_ADDR_NONE 0xFFFF
- #define APIMAC_RANDOM_SEED_LEN 32
- #define APIMAC_FH_UTT_IE 0x00000002
- #define APIMAC_FH_BT_IE 0x00000008
- #define APIMAC_FH_US_IE 0x00010000
- #define APIMAC_FH_BS_IE 0x00020000
- #define APIMAC_FH_HEADER_IE_MASK 0x000000FF
- #define APIMAC_FH_PROTO_DISPATCH_NONE 0x00
- #define APIMAC_FH_PROTO_DISPATCH_MHD_PDU 0x01
- #define APIMAC_FH_PROTO_DISPATCH_6LOWPAN 0x02
- #define APIMAC_154G_MAX_NUM_CHANNEL 129
- #define APIMAC_154G_CHANNEL_BITMAP_SIZ ((APIMAC_154G_MAX_NUM_CHANNEL + 7) / 8)
- #define APIMAC_HEADER_IE_MAX 2
- #define APIMAC_PAYLOAD_IE_MAX 2
- #define APIMAC_PAYLOAD_SUB_IE_MAX 4
- #define APIMAC_SFS_BEACON_ORDER(s) ((s) & 0x0F)
- #define APIMAC_SFS_SUPERFRAME_ORDER(s) (((s) >> 4) & 0x0F)
- #define APIMAC_SFS_FINAL_CAP_SLOT(s) (((s) >> 8) & 0x0F)
- #define APIMAC_SFS_BLE(s) (((s) >> 12) & 0x01)
- #define APIMAC_SFS_PAN_COORDINATOR(s) (((s) >> 14) & 0x01)
- #define APIMAC_SFS_ASSOCIATION_PERMIT(s) (((s) >> 15) & 0x01)
- #define APIMAC_FH_MAX_BIT_MAP_SIZE 32
- #define APIMAC_FH_NET_NAME_SIZE_MAX 32
- #define APIMAC_FH_GTK_HASH_SIZE 8

12.3.3 Typedefs

- typedef uint8_t ApiMac_sAddrExt_t[APIMAC_SADDR_EXT_LEN]
- typedef ApiMac_mcpsDataInd_t ApiMac_mlmeWsAsyncInd_t
- typedef void(* ApiMac_associateIndFp_t) (ApiMac_mlmeAssociateInd_t *pAssocInd)
- typedef void(* ApiMac_associateCnfFp_t) (ApiMac_mlmeAssociateCnf_t *pAssocCnf)
- typedef void(* ApiMac_disassociateIndFp_t) (ApiMac_mlmeDisassociateInd_t *pDisassociateInd)
- typedef void(* ApiMac_disassociateCnfFp_t) (ApiMac_mlmeDisassociateCnf_t *pDisassociateCnf)
- typedef void(* ApiMac_beaconNotifyIndFp_t) (ApiMac_mlmeBeaconNotifyInd_t *pBeaconNotifyInd)
- typedef void(* ApiMac_orphanIndFp_t) (ApiMac_mlmeOrphanInd_t *pOrphanInd)
- typedef void(* ApiMac_scanCnfFp_t) (ApiMac_mlmeScanCnf_t *pScanCnf)
- typedef void(* ApiMac_startCnfFp_t) (ApiMac_mlmeStartCnf_t *pStartCnf)
- typedef void(* ApiMac_syncLossIndFp_t) (ApiMac_mlmeSyncLossInd_t *pSyncLossInd)
- typedef void(* ApiMac_pollCnfFp_t) (ApiMac_mlmePollCnf_t *pPollCnf)
- typedef void(* ApiMac_commStatusIndFp_t) (ApiMac_mlmeCommStatusInd_t *pCommStatus)

- typedef void(* ApiMac_pollIndFp_t) (ApiMac_mlmePollInd_t *pPollInd)
- typedef void(* ApiMac_dataCnfFp_t) (ApiMac_mcpsDataCnf_t *pDataCnf)
- typedef void(* ApiMac_dataIndFp_t) (ApiMac_mcpsDataInd_t *pDataInd)
- typedef void(* ApiMac_purgeCnfFp_t) (ApiMac_mcpsPurgeCnf_t *pPurgeCnf)
- typedef void(* ApiMac_wsAsyncIndFp_t) (ApiMac_mlmeWsAsyncInd_t *pWsAsyncInd)
- typedef void(* ApiMac_wsAsyncCnfFp_t) (ApiMac_mlmeWsAsyncCnf_t *pWsAsyncCnf)
- typedef void(* ApiMac_unprocessedFp_t) (uint16_t param1, uint16_t param2, void *pMsg)

12.3.4 Enumerations

- enum ApiMac_assocStatus_t { ApiMac_assocStatus_success = 0, ApiMac_assocStatus_panAtCapacity = 1, ApiMac_assocStatus_panAccessDenied = 2 }
- enum ApiMac_addrType_t { ApiMac_addrType_none = 0, ApiMac_addrType_short = 2, ApiMac_addrType_extended = 3 }
- enum ApiMac_beaconType_t { ApiMac_beaconType_normal = 0, ApiMac_beaconType_enhanced = 1 }
- enum ApiMac_disassocateReason_t { ApiMac_disassocateReason_coord = 1, ApiMac_disassocateReason_device = 2 }
- enum ApiMac_commStatusReason_t { ApiMac_commStatusReason_assocRsp = 0, ApiMac_commStatusReason_orphanRsp = 1, ApiMac_commStatusReason_rxSecure = 2 }
- enum ApiMac_status_t { ApiMac_status_success = 0, ApiMac_status_subSystemError = 0x25, ApiMac_status_commandIDError = 0x26, ApiMac_status_lengthError = 0x27, ApiMac_status_unsupportedType = 0x28, ApiMac_status_autoAckPendingAllOn = 0xFE, ApiMac_status_autoAckPendingAllOff = 0xFF, ApiMac_status_beaconLoss = 0xE0, ApiMac_status_channelAccessFailure = 0xE1, ApiMac_status_counterError = 0xDB, ApiMac_status_denied = 0xE2, ApiMac_status_disabledTrxFailure = 0xE3, ApiMac_status_frameTooLong = 0xE5, ApiMac_status_improperKeyType = 0xDC, ApiMac_status_improperSecurityLevel = 0xDD, ApiMac_status_invalidAddress = 0xF5, ApiMac_status_invalidGts = 0xE6, ApiMac_status_invalidHandle = 0xE7, ApiMac_status_invalidIndex = 0xF9, ApiMac_status_invalidParameter = 0xE8, ApiMac_status_limitReached = 0xFA, ApiMac_status_noAck = 0xE9, ApiMac_status_noBeacon = 0xEA, ApiMac_status_noData = 0xEB, ApiMac_status_noShortAddress = 0xEC, ApiMac_status_onTimeTooLong = 0xF6, ApiMac_status_outOfCap = 0xED, ApiMac_status_panIdConflict = 0xEE, ApiMac_status_pastTime = 0xF7, ApiMac_status_readOnly = 0xFB, ApiMac_status_realignment = 0xEF, ApiMac_status_scanInProgress = 0xFC, ApiMac_status_securityError = 0xE4, ApiMac_status_superframeOverlap = 0xFD, ApiMac_status_trackingOff = 0xF8, ApiMac_status_transactionExpired = 0xF0, ApiMac_status_transactionOverflow = 0xF1, ApiMac_status_txActive = 0xF2, ApiMac_status_unavailableKey = 0xF3, ApiMac_status_unsupportedAttribute = 0xF4, ApiMac_status_unsupportedLegacy = 0xDE, ApiMac_status_unsupportedSecurity = 0xDF, ApiMac_status_unsupported = 0x18, ApiMac_status_badState = 0x19, ApiMac_status_noResources = 0x1A, ApiMac_status_ackPending = 0x1B, ApiMac_status_noTime = 0x1C, ApiMac_status_txAborted = 0x1D, ApiMac_status_duplicateEntry = 0x1E, ApiMac_status_fhError = 0x61, ApiMac_status_fhIeNotSupported = 0x62, ApiMac_status_fhNotInAsync = 0x63, ApiMac_status_fhNotInNeighborTable = 0x64, ApiMac_status_fhOutSlot = 0x65, ApiMac_status_fhInvalidAddress = 0x66, ApiMac_status_fhIeFormatInvalid = 0x67, ApiMac_status_fhPibNotSupported = 0x68, ApiMac_status_fhPibReadOnly = 0x69, ApiMac_status_fhPibInvalidParameter = 0x6A, ApiMac_status_fhInvalidFrameType = 0x6B, ApiMac_status_fhExpiredNode = 0x6C }
- enum ApiMac_secLevel_t { ApiMac_secLevel_none = 0, ApiMac_secLevel_mic32 = 1, ApiMac_secLevel_mic64 = 2, ApiMac_secLevel_mic128 = 3, ApiMac_secLevel_enc = 4, ApiMac_secLevel_encMic32 = 5, ApiMac_secLevel_encMic64 = 6, ApiMac_secLevel_encMic128 = 7 }
- enum ApiMac_keyIdMode_t { ApiMac_keyIdMode_implicit = 0, ApiMac_keyIdMode_1 = 1, ApiMac_keyIdMode_4 = 2, ApiMac_keyIdMode_8 = 3 }

- enum ApiMac_attribute_bool_t { ApiMac_attribute_associatePermit = 0x41, ApiMac_attribute_autoRequest = 0x42, ApiMac_attribute_battLifeExt = 0x43, ApiMac_attribute_gtsPermit = 0x4D, ApiMac_attribute_promiscuousMode = 0x51, ApiMac_attribute_RxOnWhenIdle = 0x52, ApiMac_attribute_associatedPanCoord = 0x56, ApiMac_attribute_timestampSupported = 0x5C, ApiMac_attribute_securityEnabled = 0x5D, ApiMac_attribute_includeMPMIE = 0x62, ApiMac_attribute_fcsType = 0xE9 }
- enum ApiMac_attribute_uint8_t { ApiMac_attribute_ackWaitDuration = 0x40, ApiMac_attribute_battLifeExtPeriods = 0x44, ApiMac_attribute_beaconPayloadLength = 0x46, ApiMac_attribute_beaconOrder = 0x47, ApiMac_attribute_bsn = 0x49, ApiMac_attribute_dsn = 0x4C, ApiMac_attribute_maxCsmaBackoffs = 0x4E, ApiMac_attribute_backoffExponent = 0x4F, ApiMac_attribute_superframeOrder = 0x54, ApiMac_attribute_maxBackoffExponent = 0x57, ApiMac_attribute_maxFrameRetries = 0x59, ApiMac_attribute_responseWaitTime = 0x5A, ApiMac_attribute_syncSymbolOffset = 0x5B, ApiMac_attribute_eBeaconSequenceNumber = 0x5E, ApiMac_attribute_eBeaconOrder = 0x5F, ApiMac_attribute_offsetTimeslot = 0x61, ApiMac_attribute_phyTransmitPowerSigned = 0xE0, ApiMac_attribute_logicalChannel = 0xE1, ApiMac_attribute_altBackoffExponent = 0xE3, ApiMac_attribute_deviceBeaconOrder = 0xE4, ApiMac_attribute_rf4cePowerSavings = 0xE5, ApiMac_attribute_frameVersionSupport = 0xE6, ApiMac_attribute_channelPage = 0xE7, ApiMac_attribute_phyCurrentDescriptorId = 0xE8 }
- enum ApiMac_attribute_uint16_t { ApiMac_attribute_coordShortAddress = 0x4B, ApiMac_attribute_panId = 0x50, ApiMac_attribute_shortAddress = 0x53, ApiMac_attribute_transactionPersistenceTime = 0x55, ApiMac_attribute_maxFrameTotalWaitTime = 0x58, ApiMac_attribute_eBeaconOrderNBPAN = 0x60 }
- enum ApiMac_attribute_uint32_t { ApiMac_attribute_beaconTxTime = 0x48, ApiMac_attribute_diagRxCrcPass = 0xEA, ApiMac_attribute_diagRxCrcFail = 0xEB, ApiMac_attribute_diagRxBroadcast = 0xEC, ApiMac_attribute_diagTxBroadcast = 0xED, ApiMac_attribute_diagRxUnicast = 0xEE, ApiMac_attribute_diagTxUnicast = 0xEF, ApiMac_attribute_diagTxUnicastRetry = 0xF0, ApiMac_attribute_diagTxUnicastFail = 0xF1, ApiMac_attribute_diagRxSecureFail = 0xF2, ApiMac_attribute_diagTxSecureFail = 0xF3 }
- enum ApiMac_attribute_array_t { ApiMac_attribute_beaconPayload = 0x45, ApiMac_attribute_coordExtendedAddress = 0x4A, ApiMac_attribute_extendedAddress = 0xE2 }
- enum ApiMac_securityAttribute_uint8_t { ApiMac_securityAttribute_keyTableEntries = 0x81, ApiMac_securityAttribute_deviceTableEntries = 0x82, ApiMac_securityAttribute_securityLevelTableEntries = 0x83, ApiMac_securityAttribute_autoRequestSecurityLevel = 0x85, ApiMac_securityAttribute_autoRequestKeyIdMode = 0x86, ApiMac_securityAttribute_autoRequestKeyIndex = 0x88 }
- enum ApiMac_securityAttribute_uint16_t { ApiMac_securityAttribute_panCoordShortAddress = 0x8B }
- enum ApiMac_securityAttribute_array_t { ApiMac_securityAttribute_autoRequestKeySource = 0x87, ApiMac_securityAttribute_defaultKeySource = 0x89, ApiMac_securityAttribute_panCoordExtendedAddress = 0x8A }
- enum ApiMac_securityAttribute_struct_t { ApiMac_securityAttribute_keyTable = 0x71, ApiMac_securityAttribute_keyIdLookupEntry = 0xD0, ApiMac_securityAttribute_keyDeviceEntry = 0xD1, ApiMac_securityAttribute_keyUsageEntry = 0xD2, ApiMac_securityAttribute_keyEntry = 0xD3, ApiMac_securityAttribute_deviceEntry = 0xD4, ApiMac_securityAttribute_securityLevelEntry = 0xD5 }
- enum ApiMac_FHAttribute_uint8_t { ApiMac_FHAttribute_unicastDwellInterval = 0x2004, ApiMac_FHAttribute_broadcastDwellInterval = 0x2005, ApiMac_FHAttribute_clockDrift = 0x2006, ApiMac_FHAttribute_timingAccuracy = 0x2007, ApiMac_FHAttribute_unicastChannelFunction = 0x2008, ApiMac_FHAttribute_broadcastChannelFunction = 0x2009, ApiMac_FHAttribute_useParentBSIE = 0x200A, ApiMac_FHAttribute_routingCost = 0x200F, ApiMac_FHAttribute_routingMethod = 0x2010, ApiMac_FHAttribute_eapolReady = 0x2011, ApiMac_FHAttribute_fanTPSVersion = 0x2012, ApiMac_FHAttribute_gtk0Hash = 0x2015, ApiMac_FHAttribute_gtk1Hash = 0x2016, ApiMac_FHAttribute_gtk2Hash = 0x2017, ApiMac_FHAttribute_gtk3Hash = 0x2018 }

- enum ApiMac_FHAttribute_uint16_t { ApiMac_FHAttribute_broadcastSchedId = 0x200B, ApiMac_FHAttribute_unicastFixedChannel = 0x200C, ApiMac_FHAttribute_broadcastFixedChannel = 0x200D, ApiMac_FHAttribute_panSize = 0x200E, ApiMac_FHAttribute_panVersion = 0x2014, ApiMac_FHAttribute_neighborValidTime = 0x2019 }
- enum ApiMac_FHAttribute_uint32_t { ApiMac_FHAttribute_BCInterval = 0x2001 }
- enum ApiMac_FHAttribute_array_t { ApiMac_FHAttribute_trackParentEUI = 0x2000, ApiMac_FHAttribute_unicastExcludedChannels = 0x2002, ApiMac_FHAttribute_broadcastExcludedChannels = 0x2003, ApiMac_FHAttribute_netName = 0x2013 }
- enum ApiMac_fhFrameType_t { ApiMac_fhFrameType_panAdvert = 0x00, ApiMac_fhFrameType_panAdvertSolicit = 0x01, ApiMac_fhFrameType_config = 0x02, ApiMac_fhFrameType_configSolicit = 0x03, ApiMac_fhFrameType_data = 0x04, ApiMac_fhFrameType_ack = 0x05, ApiMac_fhFrameType_eapol = 0x06, ApiMac_fhFrameType_invalid = 0xFF }
- enum ApiMac_payloadIEGroup_t { ApiMac_payloadIEGroup_ESDU = 0x00, ApiMac_payloadIEGroup_MLME = 0x01, ApiMac_payloadIEGroup_WiSUN = 0x04, ApiMac_payloadIEGroup_term = 0x0F }
- enum ApiMac_MLMESubIE_t { ApiMac_MLMESubIE_coexist = 0x21, ApiMac_MLMESubIE_sunDevCap = 0x22, ApiMac_MLMESubIE_sunFSKGenPhy = 0x23 }
- enum ApiMac_wisunSubIE_t { ApiMac_wisunSubIE_USIE = 1, ApiMac_wisunSubIE_BSIE = 2, ApiMac_wisunSubIE_PANIE = 4, ApiMac_wisunSubIE_netNameIE = 5, ApiMac_wisunSubIE_PANVersionIE = 6, ApiMac_wisunSubIE_GTKHashIE = 7 }
- enum ApiMac_scantype_t { ApiMac_scantype_energyDetect = 0, ApiMac_scantype_active = 1, ApiMac_scantype_passive = 2, ApiMac_scantype_orphan = 3, ApiMac_scantype_activeEnhanced = 5 }
- enum ApiMac_wisunAsycnOperation_t { ApiMac_wisunAsycnOperation_start = 0, ApiMac_wisunAsycnOperation_stop = 1 }
- enum ApiMac_wisunAsyncFrame_t { ApiMac_wisunAsyncFrame_advertisement = 0, ApiMac_wisunAsyncFrame_advertisementSolicit = 1, ApiMac_wisunAsyncFrame_config = 2, ApiMac_wisunAsyncFrame_configSolicit = 3 }
- enum ApiMac_fhDispatchType_t { ApiMac_fhDispatchType_none = 0, ApiMac_fhDispatchType_MHD_PDU = 1, ApiMac_fhDispatchType_6LowPAN = 2 }

12.3.5 Functions

- void * ApiMac_init (bool enableFH)
  Initialize this module.
- void ApiMac_registerCallbacks (ApiMac_callbacks_t *pCallbacks)
  Register for MAC callbacks.
- void ApiMac_processIncoming (void)
  Process incoming messages from the MAC stack.
- ApiMac_status_t ApiMac_mcpsDataReq (ApiMac_mcpsDataReq_t *pData)
  This function sends application data to the MAC for transmission in a MAC data frame.

  The MAC can only buffer a certain number of data request frames. When the MAC is congested and cannot accept the data request it will initiate a callback (ApiMac_dataCnfFp_t) with an overflow status (ApiMac_status_transactionOverflow) . Eventually the MAC will become uncongested and initiate the callback (ApiMac_dataCnfFp_t) for a buffered request. At this point the application can attempt another data request. Using this scheme, the application can send data whenever it wants but it must queue data to be resent if it receives an overflow status.
- ApiMac_status_t ApiMac_mcpsPurgeReq (uint8_t msduHandle)

This function purges and discards a data request from the MAC data queue. When the operation is complete the MAC sends a MCPS Purge Confirm which will initiate a callback (ApiMac_purgeCnfFp_t).

- ApiMac_status_t ApiMac_mlmeAssociateReq (ApiMac_mlmeAssociateReq_t *pData)
  This function sends an associate request to a coordinator device. The application shall attempt to associate only with a PAN that is currently allowing association, as indicated in the results of the scanning procedure. In a beacon-enabled PAN the beacon order must be set by using ApiMac_mlmeSetReq() before making the call to ApiMac_mlmeAssociateReq().
  When the associate request is complete the appliction will receive the ApiMac_associateCnfFp_t callback.
- ApiMac_status_t ApiMac_mlmeAssociateRsp (ApiMac_mlmeAssociateRsp_t *pData)
  This function sends an associate response to a device requesting to associate. This function must be called after the ApiMac_associateIndFp_t callback. When the associate response is complete the callback ApiMac_commStatusIndFp_t is called to indicate the success or failure of the operation.
- ApiMac_status_t ApiMac_mlmeDisassociateReq (ApiMac_mlmeDisassociateReq_t *pData)
  This function is used by an associated device to notify the coordinator of its intent to leave the PAN. It is also used by the coordinator to instruct an associated device to leave the PAN. When the disassociate procedure is complete the applications callback ApiMac_disassociateCnfFp_t is called.
- ApiMac_status_t ApiMac_mlmeGetReqBool (ApiMac_attribute_bool_t pibAttribute, bool *pValue)
  This direct execute function retrieves an attribute value from the MAC PIB.
- ApiMac_status_t ApiMac_mlmeGetReqUint8 (ApiMac_attribute_uint8_t pibAttribute, uint8_t *pValue)
  This direct execute function retrieves an attribute value from the MAC PIB.
- ApiMac_status_t ApiMac_mlmeGetReqUint16 (ApiMac_attribute_uint16_t pibAttribute, uint16_t *pValue)
  This direct execute function retrieves an attribute value from the MAC PIB.
- ApiMac_status_t ApiMac_mlmeGetReqUint32 (ApiMac_attribute_uint32_t pibAttribute, uint32_t *pValue)
  This direct execute function retrieves an attribute value from the MAC PIB.
- ApiMac_status_t ApiMac_mlmeGetReqArray (ApiMac_attribute_array_t pibAttribute, uint8_t *pValue)
  This direct execute function retrieves an attribute value from the MAC PIB.
- ApiMac_status_t ApiMac_mlmeGetFhReqUint8 (ApiMac_FHAttribute_uint8_t pibAttribute, uint8_t *pValue)
  This direct execute function retrieves an attribute value from the MAC Frequency Hopping PIB.
- ApiMac_status_t ApiMac_mlmeGetFhReqUint16 (ApiMac_FHAttribute_uint16_t pibAttribute, uint16_t *pValue)
  This direct execute function retrieves an attribute value from the MAC Frequency Hopping PIB.
- ApiMac_status_t ApiMac_mlmeGetFhReqUint32 (ApiMac_FHAttribute_uint32_t pibAttribute, uint32_t *pValue)
  This direct execute function retrieves an attribute value from the MAC Frequency Hopping PIB.
- ApiMac_status_t ApiMac_mlmeGetFhReqArray (ApiMac_FHAttribute_array_t pibAttribute, uint8_t *pValue)
  This direct execute function retrieves an attribute value from the MAC Frequency Hopping PIB.
- ApiMac_status_t ApiMac_mlmeGetSecurityReqUint8 (ApiMac_securityAttribute_uint8_t pibAttribute, uint8_t *pValue)
  This direct execute function retrieves an attribute value from the MAC Secutity PIB.
- ApiMac_status_t ApiMac_mlmeGetSecurityReqUint16 (ApiMac_securityAttribute_uint16_t pibAttribute, uint16_t *pValue)
  This direct execute function retrieves an attribute value from the MAC Secutity PIB.

- ApiMac_status_t ApiMac_mlmeGetSecurityReqArray (ApiMac_securityAttribute_array_t pibAttribute, uint8_t *pValue)
  This direct execute function retrieves an attribute value from the MAC Secutity PIB.
- ApiMac_status_t ApiMac_mlmeGetSecurityReqStruct (ApiMac_securityAttribute_struct_t pibAttribute, void *pValue)
  This direct execute function retrieves an attribute value from the MAC Secutity PIB.
- ApiMac_status_t ApiMac_mlmeOrphanRsp (ApiMac_mlmeOrphanRsp_t *pData)
  This function is called in response to an orphan notification from a peer device. This function must be called after receiving an Orphan Indication Callback. When the orphan response is complete the Comm Status Indication Callback is called to indicate the success or failure of the operation.
- ApiMac_status_t ApiMac_mlmePollReq (ApiMac_mlmePollReq_t *pData)
  This function is used to request pending data from the coordinator. When the poll request is complete the Poll Confirm Callback is called. If a data frame of nonzero length is received from the coordinator the Poll Confirm Callback has a status ApiMac_status_success and then calls the Data Indication Callback for the received data.
- ApiMac_status_t ApiMac_mlmeResetReq (bool setDefaultPib)
  This direct execute function resets the MAC. This function must be called once at system startup before any other function in the management API is called.
- ApiMac_status_t ApiMac_mlmeScanReq (ApiMac_mlmeScanReq_t *pData)
  This function initiates an energy detect, active, passive, or orphan scan on one or more channels. An energy detect scan measures the peak energy on each requested channel. An active scan sends a beacon request on each channel and then listening for beacons. A passive scan is a receive-only operation that listens for beacons on each channel. An orphan scan is used to locate the coordinator with which the scanning device had previously associated. When a scan operation is complete the Scan Confirm callback is called.

  *For active or passive scans the application sets the maxResults parameter the maximum number of PAN descriptors to return. If maxResults is greater than zero then the application must also set result.panDescriptor to point to a buffer of size maxResults * sizeof(ApiMac_panDesc_t) to store the results of the scan. The application must not access or deallocate this buffer until the Scan Confirm Callback is called. The MAC will store up to maxResults PAN descriptors and ignore duplicate beacons.*

  An alternative way to get results for an active or passive scan is to set maxResults to zero or set PIB attribute ApiMac_attribute_autoRequest to FALSE. Then the MAC will not store results but rather call the Beacon Notify Indication Callback" for each beacon received. The application will not need to supply any memory to store the scan results but the MAC will not filter out duplicate beacons.

  For energy detect scans the application must set result.energyDetect to point to a buffer of size 18 bytes to store the results of the scan. The application must not access or deallocate this buffer until the Scan Confirm Callback is called.

  An energy detect, active or passive scan may be performed at any time if a scan is not already in progress. However a device cannot perform any other MAC management operation or send or receive MAC data until the scan is complete.
- ApiMac_status_t ApiMac_mlmeSetReqBool (ApiMac_attribute_bool_t pibAttribute, bool value)
  This direct execute function sets an attribute value in the MAC PIB.
- ApiMac_status_t ApiMac_mlmeSetReqUint8 (ApiMac_attribute_uint8_t pibAttribute, uint8_t value)
  This direct execute function sets an attribute value in the MAC PIB.

- ApiMac_status_t ApiMac_mlmeSetReqUint16 (ApiMac_attribute_uint16_t pibAttribute, uint16_t value)
  This direct execute function sets an attribute value in the MAC PIB.
- ApiMac_status_t ApiMac_mlmeSetReqUint32 (ApiMac_attribute_uint32_t pibAttribute, uint32_t value)
  This direct execute function sets an attribute value in the MAC PIB.
- ApiMac_status_t ApiMac_mlmeSetReqArray (ApiMac_attribute_array_t pibAttribute, uint8_t *pValue)
  This direct execute function sets an attribute value in the MAC PIB.
- ApiMac_status_t ApiMac_mlmeSetFhReqUint8 (ApiMac_FHAttribute_uint8_t pibAttribute, uint8_t value)
  This direct execute function sets an attribute value in the MAC Frequency Hopping PIB.
- ApiMac_status_t ApiMac_mlmeSetFhReqUint16 (ApiMac_FHAttribute_uint16_t pibAttribute, uint16_t value)
  This direct execute function sets an attribute value in the MAC Frequency Hopping PIB.
- ApiMac_status_t ApiMac_mlmeSetFhReqUint32 (ApiMac_FHAttribute_uint32_t pibAttribute, uint32_t value)
  This direct execute function sets an attribute value in the MAC Frequency Hopping PIB.
- ApiMac_status_t ApiMac_mlmeSetFhReqArray (ApiMac_FHAttribute_array_t pibAttribute, uint8_t *pValue)
  This direct execute function sets an attribute value in the MAC Frequency Hopping PIB.
- ApiMac_status_t ApiMac_mlmeSetSecurityReqUint8 (ApiMac_securityAttribute_uint8_t pibAttribute, uint8_t value)
  This direct execute function sets an attribute value in the MAC Security PIB.
- ApiMac_status_t ApiMac_mlmeSetSecurityReqUint16 (ApiMac_securityAttribute_uint16_t pibAttribute, uint16_t value)
  This direct execute function sets an attribute value in the MAC Security PIB.
- ApiMac_status_t ApiMac_mlmeSetSecurityReqArray (ApiMac_securityAttribute_array_t pibAttribute, uint8_t *pValue)
  This direct execute function sets an attribute value in the MAC Security PIB.
- ApiMac_status_t ApiMac_mlmeSetSecurityReqStruct (ApiMac_securityAttribute_struct_t pibAttribute, void *pValue)
  This direct execute function sets an attribute value in the MAC Security PIB.
- ApiMac_status_t ApiMac_mlmeStartReq (ApiMac_mlmeStartReq_t *pData)
  This function is called by a coordinator or PAN coordinator to start or reconfigure a network. Before starting a network the device must have set its short address. A PAN coordinator sets the short address by setting the attribute ApiMac_attribute_shortAddress. A coordinator sets the short address through association.

  When parameter panCoordinator is TRUE, the MAC automatically sets attributes ApiMac_attribute_panID and ApiMac_attribute_logicalChannel to the panId and logicalChannel parameters. If panCoordinator is FALSE, these parameters are ignored (they would already be set through association).

  The parameter beaconOrder controls whether the network is beacon-enabled or non beacon-enabled. For a beacon-enabled network this parameter also controls the beacon transmission interval.

  When the operation is complete the Start Confirm Callback is called.
- ApiMac_status_t ApiMac_mlmeSyncReq (ApiMac_mlmeSyncReq_t *pData)
  This function requests the MAC to synchronize with the coordinator by acquiring and optionally tracking its beacons. Synchronizing with the coordinator is recommended before associating in a

beacon-enabled network. If the beacon could not be located on its initial search or during tracking, the MAC calls the Sync Loss Indication Callback with ApiMac_status_beaconLoss as the reason.

Before calling this function the application must set PIB attributes ApiMac_attribute_beaconOrder, ApiMac_attribute_panId and either ApiMac_attribute_coordShortAddress or ApiMac_attribute_coordExtendedAddress to the address of the coordinator with which to synchronize.

The application may wish to set PIB attribute ApiMac_attribute_autoRequest to FALSE before calling this function. Then when the MAC successfully synchronizes with the coordinator it will call the Beacon Notify Indication Callback". After receiving the callback the application may set ApiMac_attribute_autoRequest to TRUE to stop receiving beacon notifications.

This function is only applicable to beacon-enabled networks.

- uint8_t ApiMac_randomByte (void)
  This function returns a random byte from the MAC random number generator.
- ApiMac_status_t ApiMac_updatePanId (uint16_t panId)
  Update Device Table entry and PIB with new Pan Id.
- ApiMac_status_t ApiMac_mlmeWSAsyncReq (ApiMac_mlmeWSAsyncReq_t *pData)
  This functions handles a WiSUN async request. The possible operation is Async Start or Async Stop. For the async start operation, the caller of this function can indicate which WiSUN async frame type to be sent on the specified channels.
- ApiMac_status_t ApiMac_startFH (void)
  This function starts the frequency hopping. Frequency hopping operation should have been enabled using ApiMac_enableFH() before calling this API. No need to call this API if you have called ApiMac_mlmeStartReq() with the startFH field set to true.
- ApiMac_status_t ApiMac_parsePayloadGroupIEs (uint8_t *pPayload, uint16_t payloadLen, ApiMac_payloadIeRec_t **pList)
  Parses the Group payload information element. This function creates a linked list (plist) from the Payload IE (pPayload). Each item in the linked list is a seperate Group IE with its own content.

  If no IEs are found pList will be set to NULL.

  The caller is responsible to release the memory for the linked list by calling ApiMac_freeIEList().

  Call this function to create the list of Group IEs, then call ApiMac_parsePayloadSubIEs() to parse each of the group IE's content into sub IEs.
- ApiMac_status_t ApiMac_parsePayloadSubIEs (uint8_t *pContent, uint16_t contentLen, ApiMac_payloadIeRec_t **pList)
  Parses the payload sub information element. This function creates a linked list (pList) of sub IEs from the Group IE content (pContent). Each item in the linked list is a seperate sub IE with its own content.

  If no IEs are found pList will be set to NULL.

  The caller is responsible to release the memory for the linked list by calling ApiMac_freeIEList().

  Call this function after calling ApiMac_parsePayloadGroupIEs().
- void ApiMac_freeIEList (ApiMac_payloadIeRec_t *pList)
  Free the linked list allocated by ApiMac_parsePayloadGroupIEs() or ApiMac_parsePayloadSubIEs().
- ApiMac_status_t ApiMac_enableFH (void)

Enables the Frequency hopping operation. Make sure you call this function before setting any FH parameters or before calling ApiMac_mlmeStartReq() or ApiMac_startFH(), if you're using FH.

- uint8_t ApiMac_convertCapabilityInfo (ApiMac_capabilityInfo_t *pMsgcapInfo)
  Convert ApiMac_capabilityInfo_t data type to uint8 capInfo.
- void ApiMac_buildMsgCapInfo (uint8_t cInfo, ApiMac_capabilityInfo_t *pPBcapInfo)
  Convert from bitmask byte to API MAC capInfo.
- ApiMac_status_t ApiMac_secAddDevice (ApiMac_secAddDevice_t *pAddDevice)
  Adds a new MAC device table entry.
- ApiMac_status_t ApiMac_secDeleteDevice (ApiMac_sAddrExt_t *pExtAddr)
  Removes MAC device table entries.
- ApiMac_status_t ApiMac_secDeleteKeyAndAssocDevices (uint8_t keyIndex)
  Removes the key at the specified key Index and removes all MAC device table enteries associated with this key. Also removes(initializes) the key lookup list associated with this key.
- ApiMac_status_t ApiMac_secDeleteAllDevices (void)
  Removes all MAC device table entries.
- ApiMac_status_t ApiMac_secGetDefaultSourceKey (uint8_t keyId, uint32_t *pFrameCounter)
  Reads the frame counter value associated with a MAC security key indexed by the designated key identifier and the default key source.
- ApiMac_status_t ApiMac_secAddKeyInitFrameCounter (ApiMac_secAddKeyInitFrameCounter_t *pInfo)
  Adds the MAC security key, adds the associated lookup list for the key, initializes the frame counter to the value provided. It also duplicates the device table enteries (associated with the previous key if any) if available based on the flag dupDevFlag value and associates the device descriptor with this key.

12.3.6 Data Structure Documentation

12.3.6.1 struct ApiMac_sAddr_t

MAC address type field structure

12.3.6.1.1 Data Fields:

union ApiMac_sAddr_t	addr	The address can be either a long address or a short address depending the addrMode field.
ApiMac_addrType_t	addrMode	Address type/mode

12.3.6.2 struct ApiMac_sData_t

Data buffer structure

12.3.6.2.1 Data Fields:

uint8_t *	p	pointer to the data buffer
uint16_t	len	length of the data buffer

12.3.6.3 struct ApiMac_MRFSKPHYDesc_t

Generic PHY Descriptor. We are using this structure for both Channel Page 9 and Channel Page 10.

12.3.6.3.1 Data Fields:

uint32_t	firstChCentrFreq	First Channel Center frequency
uint16_t	numChannels	Number of channels defined for the particular PHY mode
uint32_t	channelSpacing	Distance between Adjacent center channel frequencies
uint8_t	fskModScheme	2-FSK/2-GFSK/4-FSK/4-GFSK
uint8_t	symbolRate	Symbol rate selection
uint8_t	fskModIndex	Modulation index as a value encoded in MR-FSK Generic PHY Descriptor IE (IEEE802.15.4g section 5.2.4.20c). 2FSK MI = 0.25 + Modulation Index * 0.05 4FSK MI is a third of 2FSK MI
uint8_t	ccaType	Channel clearance algorithm selection

12.3.6.4 struct ApiMac_sec_t

Common security type

12.3.6.4.1 Data Fields:

uint8_t	keySource[APIMAC_KEY_SOURCE_MAX_LEN]	Key source
uint8_t	securityLevel	Security Level
uint8_t	keyIdMode	Key identifier mode
uint8_t	keyIndex	Key index

12.3.6.5 struct ApiMac_keyIdLookupDescriptor_t

Key ID Lookup Descriptor

12.3.6.5.1 Data Fields:

uint8_t	lookupData[APIMAC_MAX_KEY_LOOKUP_LEN]	Data used to identify the key
uint8_t	lookupDataSize	0x00 indicates 5 octets; 0x01 indicates 9 octets

12.3.6.6 struct ApiMac_keyDeviceDescriptor_t

Key Device Descriptor

12.3.6.6.1 Data Fields:

uint8_t	deviceDescriptorHandle	Handle to the DeviceDescriptor

bool	uniqueDevice	True if the device is unique
bool	blackListed	This key exhausted the frame counter.

12.3.6.7 struct ApiMac_keyUsageDescriptor_t

Key Usage Descriptor

12.3.6.7.1 Data Fields:

uint8_t	frameType	Frame Type
uint8_t	cmdFrameId	Command Frame Identifier

12.3.6.8 struct ApiMac_keyDescriptor_t

Key Descriptor

12.3.6.8.1 Data Fields:

ApiMac_keyIdLookupDescriptor_t *	keyIdLookupList	A list identifying this KeyDescriptor
uint8_t	keyIdLookupEntries	The number of entries in KeyIdLookupList
ApiMac_keyDeviceDescriptor_t *	keyDeviceList	A list indicating which devices are currently using this key, including their blacklist status.
uint8_t	keyDeviceListEntries	The number of entries in KeyDeviceList
ApiMac_keyUsageDescriptor_t *	keyUsageList	A list indicating which frame types this key may be used with.

uint8_t	keyUsageListEntries	The number of entries in KeyUsageList
uint8_t	key[APIMAC_KEY_MAX_LEN]	The actual value of the key
uint32_t	frameCounter	PIB frame counter in 802.15.4 is universal across key, but it makes more sense to associate a frame counter with a key.

12.3.6.9 struct ApiMac_deviceDescriptor_t

Device Descriptor

12.3.6.9.1 Data Fields:

uint16_t	panID	The 16-bit PAN identifier of the device
uint16_t	shortAddress	The 16-bit short address of the device
ApiMac_sAddrExt_t	extAddress	The 64-bit IEEE extended address of the device. This element is also used inunsecuring operations on incoming frames.

12.3.6.10 struct ApiMac_securityLevelDescriptor_t

Security Level Descriptor

12.3.6.10.1 Data Fields:

uint8_t	frameType	Frame Type
uint8_t	commandFrameIdent	Command Frame ID

	ifier	
uint8_t	securityMinimum	The minimal required/expected security level for incoming MAC frames.
bool	securityOverrideSecurityMinimum	Indication of whether originating devices for which the Exempt flag is set may override the minimum security level indicated by the Security Minimum element. If TRUE, this indicates that for originating devices with Exempt status, the incoming security level zero is acceptable.

12.3.6.11 struct ApiMac_securityDeviceDescriptor_t

Security Device Descriptor

12.3.6.11.1 Data Fields:

ApiMac_deviceDescriptor_t	devInfo	Device information
uint32_t	frameCounter[APIMAC_MAX_KEY_TABLE_ENTRIES]	The incoming frame counter of the device. This value is used to ensure sequential freshness of frames.
bool	exempt	Device may override the minimum security level settings.

12.3.6.12 struct ApiMac_securityKeyEntry_t

MAC key entry structure

12.3.6.12.1 Data Fields:

uint8_t	keyEntry[APIMAC_KEY_MAX_LEN]	The 128-bit key

uint8_t	keyIndex	the key's index - unique
uint32_t	frameCounter	The key's frame counter

12.3.6.13 struct ApiMac_securityPibKeyIdLookupEntry_t

Security PIB Key ID lookup entry for a Get/Set ApiMac_securityAttribute_keyIdLookupEntry

12.3.6.13.1 Data Fields:

uint8_t	keyIndex	index into the macKeyIdLookupList
uint8_t	keyIdLookupIndex	index into macKeyIdLookupList[keyIndex]
ApiMac_keyIdLookupDescriptor_t	lookupEntry	Place to put the requested data

12.3.6.14 struct ApiMac_securityPibKeyDeviceEntry_t

Security PIB Key ID device entry for a Get/Set ApiMac_securityAttribute_keyDeviceEntry

12.3.6.14.1 Data Fields:

uint8_t	keyIndex	index into the macKeyDeviceList
uint8_t	keyDeviceIndex	index into macKeyDeviceList[keyIndex]
ApiMac_keyDeviceDescriptor_t	deviceEntry	Place to put the requested data

12.3.6.15 struct ApiMac_securityPibKeyUsageEntry_t

Security PIB Key ID usage entry for a Get/Set ApiMac_securityAttribute_keyUsageEntry

12.3.6.15.1 Data Fields:

uint8_t	keyIndex	index into the macKeyUsageList
uint8_t	keyUsageIndex	index into macKeyUsageList[keyIndex]
ApiMac_keyUsageDescriptor_t	usageEntry	Place to put the requested data

12.3.6.16 struct ApiMac_securityPibKeyEntry_t

Security PIB Key entry for a Get/Set ApiMac_securityAttribute_keyEntry

12.3.6.16.1 Data Fields:

uint8_t	keyIndex	index into the macKeyTable
uint8_t	keyEntry[APIMAC_KEY_MAX_LEN]	key entry
uint32_t	frameCounter	frame counter

12.3.6.17 struct ApiMac_securityPibDeviceEntry_t

Security PIB device entry for a Get/Set ApiMac_securityAttribute_deviceEntry

12.3.6.17.1 Data Fields:

uint8_t	deviceIndex	index into the macDeviceTable
ApiMac_securityDeviceDescriptor_t	deviceEntry	Place to put the requested data

12.3.6.18 struct ApiMac_securityPibSecurityLevelEntry_t

Security PIB level entry for a Get/Set ApiMac_securityAttribute_securityLevelEntry

12.3.6.18.1 Data Fields:

uint8_t	levelIndex	index into the macSecurityLevelTable
ApiMac_securityLevelDescriptor_t	levelEntry	Place to put the requested data

12.3.6.19 struct ApiMac_capabilityInfo_t

Structure defines the Capabilities Information bit field.

12.3.6.19.1 Data Fields:

bool	panCoord	True if the device is a PAN Coordinator
bool	ffd	True if the device is a full function device (FFD)
bool	mainsPower	True if the device is mains powered
bool	rxOnWhenIdle	True if the device's RX is on when the device is idle
bool	security	True if the device is capable of sending and receiving secured frames
bool	allocAddr	True if allocation of a short address in the associate procedure is needed.

12.3.6.20 struct ApiMac_txOptions_t

Data Request Transmit Options

12.3.6.20.1 Data Fields:

bool	ack	Acknowledged transmission. The MAC will attempt to retransmit the frame until it is acknowledged
bool	indirect	Indirect transmission. The MAC will queue the data and wait for the destination device to poll for it. This can only be used by a coordinator device
bool	pendingBit	This proprietary option forces the pending bit set for direct transmission
bool	noRetransmits	This proprietary option prevents the frame from being retransmitted
bool	noConfirm	This proprietary option prevents a MAC_MCPS_DATA_CNF event from being sent for this frame
bool	useAltBE	Use PIB value MAC_ALT_BE for the minimum backoff exponent
bool	usePowerAndChannel	Use the power and channel values in macDataReq_t instead of the PIB values

12.3.6.21 struct ApiMac_mcpsDataReq_t

MCPS data request type

12.3.6.21.1 Data Fields:

ApiMac_sAddr_t	dstAddr	The address of the destination device
uint16_t	dstPanId	The PAN ID of the destination device
ApiMac_addrType_t	srcAddrMode	The source address mode
uint8_t	msduHandle	Application-defined handle value associated with this data request
ApiMac_txOptions_t	txOptions	TX options bit mask
uint8_t	channel	Transmit the data frame on this channel
uint8_t	power	Transmit the data frame at this power level
uint8_t *	pIEList	pointer to the payload IE list, excluding termination IEs
uint16_t	payloadIELen	length of the payload IE
ApiMac_fhDispatchType_t	fhProtoDispatch	Freq hopping Protocol Dispatch - RESERVED for future use, should be cleared.
uint32_t	includeFhIEs	Bitmap indicates which FH IE's need to be included
ApiMac_sData_t	msdu	Data buffer

ApiMac_sec_t	sec	Security Parameters

12.3.6.22 struct ApiMac_payloadIeItem_t

Structure a Payload information Item

12.3.6.22.1 Data Fields:

bool	ieTypeLong	True if payload IE type is long
uint8_t	ieId	IE ID
uint16_t	ieContentLen	IE Content Length - max size 2047 bytes
uint8_t *	pIEContent	Pointer to the IE's content

12.3.6.23 struct ApiMac_payloadIeRec_t

A Payload IE Link List record

12.3.6.23.1 Data Fields:

void *	pNext	Pointer to the next element in the linked list, NULL if no more
ApiMac_payloadIeItem_t	item	Payload IE information item

12.3.6.24 struct ApiMac_mcpsDataInd_t

MCPS data indication type

12.3.6.24.1 Data Fields:

ApiMac_sAddr_t	srcAddr	The address of the sending device
ApiMac_sAddr_t	dstAddr	The address of the destination device
uint32_t	timestamp	The time, in backoffs, at which the data were received
uint16_t	timestamp2	The time, in internal MAC timer units, at which the data were received
uint16_t	srcPanId	The PAN ID of the sending device
uint16_t	dstPanId	The PAN ID of the destination device
uint8_t	mpduLinkQuality	The link quality of the received data frame
uint8_t	correlation	The raw correlation value of the received data frame
int8_t	rssi	The received RF power in units dBm
uint8_t	dsn	The data sequence number of the received frame
uint16_t	payloadIeLen	length of the payload IE buffer (pPayloadIE)
uint8_t *	pPayloadIE	Pointer to the start of payload IEs

ApiMac_fhFrameType_t	fhFrameType	Frequency Hopping Frame Type
ApiMac_fhDispatchType_t	fhProtoDispatch	Frequency hopping protocol dispatch - RESERVED for future use.
uint32_t	frameCntr	Frame counter value of the received data frame (if used)
ApiMac_sec_t	sec	Security Parameters
ApiMac_sData_t	msdu	Data Buffer

12.3.6.25 struct ApiMac_mcpsDataCnf_t

MCPS data confirm type

12.3.6.25.1 Data Fields:

ApiMac_status_t	status	Contains the status of the data request operation
uint8_t	msduHandle	Application-defined handle value associated with the data request
uint32_t	timestamp	The time, in backoffs, at which the frame was transmitted
uint16_t	timestamp2	The time, in internal MAC timer units, at which the frame was transmitted
uint8_t	retries	The number of retries required to transmit the data frame

uint8_t	mpduLinkQuality	The link quality of the received ack frame
uint8_t	correlation	The raw correlation value of the received ack frame
int8_t	rssi	The RF power of the received ack frame in units dBm
uint32_t	frameCntr	Frame counter value used (if any) for the transmitted frame

12.3.6.26 struct ApiMac_mcpsPurgeCnf_t

MCPS purge confirm type

12.3.6.26.1 Data Fields:

ApiMac_status_t	status	The status of the purge request operation
uint8_t	msduHandle	Application-defined handle value associated with the data request

12.3.6.27 struct ApiMac_panDesc_t

PAN descriptor type

12.3.6.27.1 Data Fields:

ApiMac_sAddr_t	coordAddress	The address of the coordinator sending the beacon

uint16_t	coordPanId	The PAN ID of the network
uint16_t	superframeSpec	The superframe specification of the network, this field contains the beacon order, superframe order, final CAP slot, battery life extension, PAN coordinator bit, and association permit flag. Use the following macros to parse this field: APIMAC_SFS_BEACON_ORDER(), APIMAC_SFS_SUPERFRAME_ORDER(), APIMAC_SFS_FINAL_CAP_SLOT() APIMAC_SFS_BLE(), APIMAC_SFS_PAN_COORDINATOR(), and APIMAC_SFS_ASSOCIATION_PERMIT().
uint8_t	logicalChannel	The logical channel of the network
uint8_t	channelPage	The current channel page occupied by the network
bool	gtsPermit	TRUE if coordinator accepts GTS requests. This field is not used for enhanced beacons.
uint8_t	linkQuality	The link quality of the received beacon
uint32_t	timestamp	The time at which the beacon was received, in backoffs
bool	securityFailure	TRUE if there was an error in the security processing
ApiMac_sec_t	sec	The security parameters for the received beacon frame

12.3.6.28 struct ApiMac_mlmeAssociateReq_t

MLME associate request type

12.3.6.28.1 Data Fields:

ApiMac_sec_t	sec	The security parameters for this message
uint8_t	logicalChannel	The channel on which to attempt association
uint8_t	channelPage	The channel page on which to attempt association
uint8_t	phyID	Identifier for the PHY descriptor
ApiMac_sAddr_t	coordAddress	Address of the coordinator with which to associate
uint16_t	coordPanId	The identifier of the PAN with which to associate
ApiMac_capabilityInfo_t	capabilityInformation	The operational capabilities of this device

12.3.6.29 struct ApiMac_mlmeAssociateRsp_t

MLME associate response type

12.3.6.29.1 Data Fields:

ApiMac_sec_t	sec	The security parameters for this message

ApiMac_sAddrExt_t	deviceAddress	The address of the device requesting association
uint16_t	assocShortAddress	The short address allocated to the device
ApiMac_assocStatus_t	status	The status of the association attempt

12.3.6.30 struct ApiMac_mlmeDisassociateReq_t

MLME disassociate request type

12.3.6.30.1 Data Fields:

ApiMac_sec_t	sec	The security parameters for this message
ApiMac_sAddr_t	deviceAddress	The address of the device with which to disassociate
uint16_t	devicePanId	The PAN ID of the device
ApiMac_disassociateReason_t	disassociateReason	The disassociate reason
bool	txIndirect	Transmit Indirect

12.3.6.31 struct ApiMac_mlmeOrphanRsp_t

MLME orphan response type

12.3.6.31.1 Data Fields:

ApiMac_sec_t	sec	The security parameters for this message

ApiMac_sAddrExt_t	orphanAddress	The extended address of the device sending the orphan notification
uint16_t	shortAddress	The short address of the orphaned device
bool	associatedMember	TRUE if the orphaned device is associated with this coordinator

12.3.6.32 struct ApiMac_mlmePollReq_t

MLME poll request type

12.3.6.32.1 Data Fields:

ApiMac_sAddr_t	coordAddress	The address of the coordinator device to poll
uint16_t	coordPanId	The PAN ID of the coordinator
ApiMac_sec_t	sec	The security parameters for this message

12.3.6.33 struct ApiMac_mlmeScanReq_t

MLME scan request type

12.3.6.33.1 Data Fields:

uint8_t	scanChannels[APIMAC_154G_CHANNEL_BITMAP_SIZ]	Bit mask indicating which channels to scan
ApiMac_scantype_t	scanType	The type of scan
uint8_t	scanDuration	The exponent used in the scan duration calculation

uint8_t	channelPage	The channel page on which to perform the scan
uint8_t	phyID	PHY ID corresponding to the PHY descriptor to use
uint8_t	maxResults	The maximum number of PAN descriptor results, these results will be returned in the scan confirm.
bool	permitJoining	Only devices with permit joining enabled respond to the enhanced beacon request
uint8_t	linkQuality	The device will respond to the enhanced beacon request if mpduLinkQuality is equal or higher than this value
uint8_t	percentFilter	The device will then randomly determine if it is to respond to the enhanced beacon request based on meeting this probability (0 to 100%).
ApiMac_sec_t	sec	The security parameters for this message
bool	MPMScan	When TRUE, scanDuration is ignored. When FALSE, scan duration shall be set to scanDuration; MPMScanDuration is ignored
uint8_t	MPMScanType	BPAN or NBPAN
uint16_t	MPMScanDuration	If MPMScanType is BPAN, MPMScanDuration values are 0-14. It is used

		in determining the max time spent scanning for an EB in a beacon enabled PAN on the channel. [aBaseSuperframeDuration * 2^n symbols], where n is the MPMScanDuration. If MPMScanType is NBPAN, valid values are 1 - 16383. It is used in determining the max time spent scanning for an EB in nonbeacon-enabled PAN on the channel. [aBaseSlotDuration * n] symbols, where n is MPMScanDuration.

12.3.6.34 struct ApiMac_mpmParams_t

MPM(Multi-PHY layer management) parameters

12.3.6.34.1 Data Fields:

uint8_t	eBeaconOrder	The exponent used to calculate the enhanced beacon interval. A value of 15 indicates no EB in a beacon enabled PAN
uint8_t	offsetTimeSlot	Indicates the time diff between the EB and the preceeding periodic Beacon. The valid range for this field is 10 - 15.
uint16_t	NBPANEBeaconOrder	Indicates how often the EB to tx in a non-beacon enabled PAN. A value of 16383 indicates no EB in a non-beacon enabled PAN
uint8_t *	pIEIDs	pointer to the buffer containing the Information element IDs which needs to be sent in Enhanced Beacon. This field is reserved for future use and should be set to NULL.
uint8_t	numIEs	The number of Information Elements in the buffer (size of buffer at pIEIDs. This field is reserved for future use and should be set to 0.

12.3.6.35 struct ApiMac_mlmeStartReq_t

MLME start request type

12.3.6.35.1 Data Fields:

uint32_t	startTime	The time to begin transmitting beacons relative to the received beacon
uint16_t	panId	The PAN ID to use. This parameter is ignored if panCoordinator is FALSE
uint8_t	logicalChannel	The logical channel to use. This parameter is ignored if panCoordinator is FALSE
uint8_t	channelPage	The channel page to use. This parameter is ignored if panCoordinator is FALSE
uint8_t	phyID	PHY ID corresponding to the PHY descriptor to use
uint8_t	beaconOrder	The exponent used to calculate the beacon interval
uint8_t	superframeOrder	The exponent used to calculate the superframe duration
bool	panCoordinator	Set to TRUE to start a network as PAN coordinator
bool	batteryLifeExt	If this value is TRUE, the receiver is disabled after MAC_BATT_LIFE_EXT_PERIODS full

		backoff periods following the interframe spacing period of the beacon frame
bool	coordRealignment	Set to TRUE to transmit a coordinator realignment prior to changing the superframe configuration
ApiMac_sec_t	realignSec	Security parameters for the coordinator realignment frame
ApiMac_sec_t	beaconSec	Security parameters for the beacon frame
ApiMac_mpmParams_t	mpmParams	MPM (multi-PHY layer management) parameters
bool	startFH	Indicates whether frequency hopping needs to be enabled

12.3.6.36 struct ApiMac_mlmeSyncReq_t

MAC_MlmeSyncReq type

12.3.6.36.1 Data Fields:

uint8_t	logicalChannel	The logical channel to use
uint8_t	channelPage	The channel page to use
uint8_t	phyID	PHY ID corresponding to the PHY descriptor to use

uint8_t	trackBeacon	Set to TRUE to continue tracking beacons after synchronizing with the first beacon. Set to FALSE to only synchronize with the first beacon

12.3.6.37 struct ApiMac_mlmeWSAsyncReq_t

MLME WiSUN Async request type

12.3.6.37.1 Data Fields:

ApiMac_wisunAsyncOperation_t	operation	Start or Stop Async operation
ApiMac_wisunAsyncFrame_t	frameType	Async frame type
uint8_t	channels[APIMAC_154G_CHANNEL_BITMAP_SIZ]	Bit Mask indicating which channels to send the Async frames for the start operation
ApiMac_sec_t	sec	The security parameters for this message

12.3.6.38 struct ApiMac_secAddDevice_t

Structure to pass information to the ApiMac_secAddDevice().

12.3.6.38.1 Data Fields:

uint16_t	panID	PAN ID of the new device
uint16_t	shortAddr	short address of the new device
ApiMac_sAddrExt_t	extAddr	extended address of the new device

bool	exempt	Device descriptor exempt field value (true or false), setting this field to true means that this device can override the minimum security level setting.
uint8_t	keyIdLookupDataSize	key ID lookup data size as it is stored in PIB, (i.e., 0 for 5 bytes, 1 for 9 bytes).
uint8_t	keyIdLookupData[APIMAC_MAX_KEY_LOOKUP_LEN]	key ID lookup data, to look for the key table entry and create proper key device descriptor for this device.
uint32_t	frameCounter	Frame Counter
bool	uniqueDevice	key device descriptor uniqueDevice field value (true or false)
bool	duplicateDevFlag	A flag (true or false) to indicate whether the device entry should be duplicated even for the keys that do not match the key ID lookup data. The device descriptors that are pointed by the key device descriptors that do not match the key ID lookup data shall not update the frame counter based on the frameCounter argument to this function or shall set the frame counter to zero when the entry is newly created.

12.3.6.39 struct ApiMac_secAddKeyInitFrameCounter_t

Structure to pass information to the ApiMac_secAddKeyInitFrameCounter().

12.3.6.39.1 Data Fields:

uint8_t	key[APIMAC_KEY_MAX_LEN]	Key

uint32_t	frameCounter	Frame Counter
uint8_t	replaceKeyIndex	Key index of the mac security key table where the key needs to be written
bool	newKeyFlag	If set to true, the function will duplicate the device table enteries associated with the previous key, and associate it with the key. If set to false, the function will not alter device table entries associated with whatever key that was stored in the key table location as designated by replaceKeyIndex.
uint8_t	lookupDataSize	Key ID lookup data size as it is stored in PIB, i.e., 0 for 5 bytes, 1 for 9 bytes.
uint8_t	lookupData[APIMAC_MAX_KEY_LOOKUP_LEN]	Key ID lookup data, to look for the key table entry and create proper key device descriptor for this device.

12.3.6.40 struct ApiMac_mlmeAssociateInd_t

MAC_MLME_ASSOCIATE_IND type

12.3.6.40.1 Data Fields:

ApiMac_sAddrExt_t	deviceAddress	The address of the device requesting association
ApiMac_capabilityInfo_t	capabilityInformation	The operational capabilities of the device requesting association
ApiMac_sec_t	sec	The security parameters for this message

12.3.6.41 struct ApiMac_mlmeAssociateCnf_t

MAC_MLME_ASSOCIATE_CNF type

12.3.6.41.1 Data Fields:

ApiMac_assocStatus_t	status	Status of associate attempt
uint16_t	assocShortAddress	If successful, the short address allocated to this device
ApiMac_sec_t	sec	The security parameters for this message

12.3.6.42 struct ApiMac_mlmeDisassociateInd_t

MAC_MLME_DISASSOCIATE_IND type

12.3.6.42.1 Data Fields:

ApiMac_sAddrExt_t	deviceAddress	The address of the device sending the disassociate command
ApiMac_disassociateReason_t	disassociateReason	The disassociate reason
ApiMac_sec_t	sec	The security parameters for this message

12.3.6.43 struct ApiMac_mlmeDisassociateCnf_t

MAC_MLME_DISASSOCIATE_CNF type

12.3.6.43.1 Data Fields:

ApiMac_status_t	status	status of the disassociate attempt

ApiMac_sAddr_t	deviceAddress	The address of the device that has either requested disassociation or been instructed to disassociate by its coordinator
uint16_t	panId	The pan ID of the device that has either requested disassociation or been instructed to disassociate by its coordinator

12.3.6.44 struct ApiMac_beaconData_t

MAC Beacon data type

12.3.6.44.1 Data Fields:

uint8_t	numPendShortAddr	The number of pending short addresses
uint16_t *	pShortAddrList	The list of device short addresses for which the sender of the beacon has data
uint8_t	numPendExtAddr	The number of pending extended addresses
uint8_t *	pExtAddrList	The list of device short addresses for which the sender of the beacon has data
uint8_t	sduLength	The number of bytes in the beacon payload of the beacon frame
uint8_t *	pSdu	The beacon payload

12.3.6.45 struct ApiMac_coexist_t

Coexistence Information element content type

12.3.6.45.1 Data Fields:

uint8_t	beaconOrder	Beacon Order field shall specify the transmission interval of the beacon
uint8_t	superFrameOrder	Superframe Order field shall specify the length of time during which the superframe is active (i.e., receiver enabled), including the Beacon frametransmission time
uint8_t	finalCapSlot	Final CAP slot
uint8_t	eBeaconOrder	Enhanced Beacon Order field specifies the transmission interval of the Enhanced Beacon frames in a beacon enabled network
uint8_t	offsetTimeSlot	Time offset between periodic beacon and the Enhanced Beacon.
uint8_t	capBackOff	Actual slot position in which the Enhanced Beacon frame is transmitted due to the backoff procedure in the CAP
uint16_t	eBeaconOrderNBPAN	NBPAN Enhanced Beacon Order field specifies the transmission interval between consecutive Enhanced Beacon frames in the nonbeacon-enabled mode

12.3.6.46 struct ApiMac_eBeaconData_t

MAC Enhanced beacon data type

12.3.6.46.1 Data Fields:

ApiMac_coexist_t	coexist	Beacon Coexist data

t		

12.3.6.47 struct ApiMac_mlmeBeaconNotifyInd_t

MAC_MLME_BEACON_NOTIFY_IND type

12.3.6.47.1 Data Fields:

ApiMac_beaconType_t	beaconType	Indicates the beacon type: beacon or enhanced beacon
uint8_t	bsn	The beacon sequence number or enhanced beacon sequence number
ApiMac_panDesc_t	panDesc	The PAN Descriptor for the received beacon
union ApiMac_mlmeBeaconNotifyInd_t	beaconData	Beacon data union depending on beaconType, select beaconData or or eBeaconData.

12.3.6.48 struct ApiMac_mlmeOrphanInd_t

MAC_MLME_ORPHAN_IND type

12.3.6.48.1 Data Fields:

ApiMac_sAddrExt_t	orphanAddress	The address of the orphaned device
ApiMac_sec_t	sec	The security parameters for this message

12.3.6.49 struct ApiMac_mlmeScanCnf_t

MAC_MLME_SCAN_CNF type

12.3.6.49.1 Data Fields:

ApiMac_status_t	status	status of the scan request
ApiMac_scantype_t	scanType	The type of scan requested
uint8_t	channelPage	The channel page of the scan
uint8_t	phyId	PHY ID corresponding to the PHY descriptor used during scan
uint8_t	unscannedChannels[APIMAC_154G_CHANNEL_BITMAP_SIZ]	Bit mask of channels that were not scanned
uint8_t	resultListSize	The number of PAN descriptors returned in the results list
union ApiMac_mlmeScanCnf_t	result	Depending on the scanType the results are in this union

12.3.6.50 struct ApiMac_mlmeStartCnf_t

MAC_MLME_START_CNF type

12.3.6.50.1 Data Fields:

ApiMac_status_t	status	status of the start request

12.3.6.51 struct ApiMac_mlmeSyncLossInd_t

MAC_MLME_SYNC_LOSS_IND type

12.3.6.51.1 Data Fields:

ApiMac_status_t	reason	Reason that the synchronization was lost
uint16_t	panId	The PAN ID of the realignment
uint8_t	logicalChannel	The logical channel of the realignment
uint8_t	channelPage	The channel page of the realignment
uint8_t	phyID	PHY ID corresponding to the PHY descriptor of the realignment
ApiMac_sec_t	sec	The security parameters for this message

12.3.6.52 struct ApiMac_mlmePollCnf_t

MAC_MLME_POLL_CNF type

12.3.6.52.1 Data Fields:

ApiMac_status_t	status	status of the poll request
uint8_t	framePending	Set if framePending bit in data packet is set

12.3.6.53 struct ApiMac_mlmeCommStatusInd_t

MAC_MLME_COMM_STATUS_IND type

12.3.6.53.1 Data Fields:

ApiMac_status_t	status	status of the event

ApiMac_sAddr_t	srcAddr	The source address associated with the event
ApiMac_sAddr_t	dstAddr	The destination address associated with the event
uint16_t	panId	The PAN ID associated with the event
ApiMac_commStatusReason_t	reason	The reason the event was generated
ApiMac_sec_t	sec	The security parameters for this message

12.3.6.54 struct ApiMac_mlmePollInd_t

MAC_MLME_POLL_IND type

12.3.6.54.1 Data Fields:

ApiMac_sAddr_t	srcAddr	Address of the device sending the data request
uint16_t	srcPanId	Pan ID of the device sending the data request
bool	noRsp	indication that no MAC_McpsDataReq() is required. It is set when MAC_MLME_POLL_IND is generated, to simply indicate that a received data request frame was acked with pending bit cleared.

12.3.6.55 struct ApiMac_mlmeWsAsyncCnf_t

MAC_MLME_WS_ASYNC_FRAME_CNF type

12.3.6.55.1 Data Fields:

ApiMac_status_t	status	status of the Async request

12.3.6.56 struct ApiMac_callbacks_t

Structure containing all the MAC callbacks (indications). To receive the confirmation or indication fill in the associated callback with a pointer to the function that will handle that callback. To ignore a callback set that function pointer to NULL.

12.3.6.56.1 Data Fields:

ApiMac_associateIndFp_t	pAssocIndCb	Associate Indicated callback
ApiMac_associateCnfFp_t	pAssocCnfCb	Associate Confirmation callback
ApiMac_disassociateIndFp_t	pDisassociateIndCb	Disassociate Indication callback
ApiMac_disassociateCnfFp_t	pDisassociateCnfCb	Disassociate Confirmation callback
ApiMac_beaconNotifyIndFp_t	pBeaconNotifyIndCb	Beacon Notify Indication callback
ApiMac_orphanIndFp_t	pOrphanIndCb	Orphan Indication callback
ApiMac_scanCnfFp_t	pScanCnfCb	Scan Confirmation callback
ApiMac_startCnfFp_t	pStartCnfCb	Start Confirmation callback
ApiMac_syncLossIndFp_t	pSyncLossIndCb	Sync Loss Indication callback
ApiMac_pollCnfFp_t	pPollCnfCb	Poll Confirm callback

ApiMac_commStatusIndFp_t	pCommStatusCb	Comm Status Indication callback
ApiMac_pollIndFp_t	pPollIndCb	Poll Indication Callback
ApiMac_dataCnfFp_t	pDataCnfCb	Data Confirmation callback
ApiMac_dataIndFp_t	pDataIndCb	Data Indication callback
ApiMac_purgeCnfFp_t	pPurgeCnfCb	Purge Confirm callback
ApiMac_wsAsyncIndFp_t	pWsAsyncIndCb	WiSUN Async Indication callback
ApiMac_wsAsyncCnfFp_t	pWsAsyncCnfCb	WiSUN Async Confirmation callback
ApiMac_unprocessedFp_t	pUnprocessedCb	Unprocessed message callback

12.3.6.57 union ApiMac_sAddr_t.addr

The address can be either a long address or a short address depending the addrMode field.

12.3.6.57.1 Data Fields:

uint16_t	shortAddr	16 bit address
ApiMac_sAddrExt_t	extAddr	Extended address

12.3.6.58 union ApiMac_mlmeBeaconNotifyInd_t.beaconData

Beacon data union depending on beaconType, select beaconData or or eBeaconData.

12.3.6.58.1 Data Fields:

ApiMac_beaconData_t	beacon	beacon data
ApiMac_eBeaconData_t	eBeacon	enhanced beacon data

12.3.6.59 union ApiMac_mlmeScanCnf_t.result

Depending on the scanType the results are in this union

12.3.6.59.1 Data Fields:

uint8_t *	pEnergyDetect	The list of energy measurements, one for each channel scanned
ApiMac_panDesc_t *	pPanDescriptor	The list of PAN descriptors, one for each beacon found

---

12.3.7 Macro Definition Documentation

12.3.7.1 #define APIMAC_KEY_MAX_LEN 16

Key Length

12.3.7.2 #define APIMAC_SADDR_EXT_LEN 8

IEEE Address Length

12.3.7.3 #define APIMAC_MAX_KEY_TABLE_ENTRIES 2

Maximum number of key table entries

12.3.7.4 #define APIMAC_KEYID_IMPLICIT_LEN 0

Key identifier field length - Implicit mode

12.3.7.5 #define APIMAC_KEYID_MODE1_LEN 1

Key identifier field length - mode 1

12.3.7.6 #define APIMAC_KEYID_MODE4_LEN 5

Key Identifier field length - mode 4

12.3.7.7 #define APIMAC_KEYID_MODE8_LEN 9

Key Identifier field length - mode 8

12.3.7.8 #define APIMAC_KEY_SOURCE_MAX_LEN 8

Key source maximum length in bytes

12.3.7.9 #define APIMAC_KEY_INDEX_LEN 1

Key index length in bytes

12.3.7.10 #define APIMAC_FRAME_COUNTER_LEN 4

Frame counter length in bytes

12.3.7.11 #define APIMAC_KEY_LOOKUP_SHORT_LEN 5

Key lookup data length in bytes - short length

12.3.7.12 #define APIMAC_KEY_LOOKUP_LONG_LEN 9

Key lookup data length in bytes - long length

12.3.7.13 #define APIMAC_MAX_KEY_LOOKUP_LEN APIMAC KEY LOOKUP LONG LEN

Key lookup data length in bytes - lookup length

12.3.7.14 #define APIMAC_DATA_OFFSET 24

Bytes required for MAC header in data frame

12.3.7.15 #define APIMAC_MAX_BEACON_PAYLOAD 16

Maximum length allowed for the beacon payload

12.3.7.16 #define APIMAC_MIC_32_LEN 4

Length required for MIC-32 authentication

12.3.7.17 #define APIMAC_MIC_64_LEN 8

Length required for MIC-64 authentication

12.3.7.18 #define APIMAC_MIC_128_LEN 16

Length required for MIC-128 authentication

12.3.7.19 #define APIMAC_MHR_LEN 37

MHR length for received frame

- FCF (2) + Seq (1) + Addr Fields (20) + Security HDR (14)

12.3.7.20 #define APIMAC_CHANNEL_PAGE_9 9

Channel Page - standard-defined SUN PHY operating modes

12.3.7.21 #define APIMAC_CHANNEL_PAGE_10 10

Channel Page - MR-FSK Generic-PHY-defined PHY modes

12.3.7.22 #define APIMAC_STANDARD_PHY_DESCRIPTOR_ENTRIES 3

Maximum number of Standard PHY descriptor entries

12.3.7.23 #define APIMAC_GENERIC_PHY_DESCRIPTOR_ENTRIES 3

Maximum number of Generic PHY descriptor entries

12.3.7.24 #define APIMAC_STD_US_915_PHY_1 1

PHY IDs - 915MHz US Frequency band operating mode # 1

12.3.7.25 #define APIMAC_STD_US_915_PHY_2 2

PHY IDs - 915MHz US Frequency band operating mode # 2

12.3.7.26 #define APIMAC_STD_ETSI_863_PHY_3 3

863MHz ETSI Frequency band operating mode #1

12.3.7.27 #define APIMAC_MRFSK_GENERIC_PHY_ID_BEGIN 128

PHY IDs - MRFSK Generic Phy ID start

12.3.7.28 #define APIMAC_MRFSK_GENERIC_PHY_ID_END 143

PHY IDs - MRFSK Generic Phy ID end

12.3.7.29 #define APIMAC_MRFSK_STD_PHY_ID_BEGIN APIMAC_STD_US_915_PHY_1

PHY IDs - MRFSK Standard Phy ID start

12.3.7.30 #define APIMAC_MRFSK_STD_PHY_ID_END APIMAC_STD_ETSI_863_PHY_3

PHY IDs - MRFSK Standard Phy ID end

12.3.7.31 #define APIMAC_PHY_DESCRIPTOR 0x01

PHY descriptor table entry

12.3.7.32 #define APIMAC_ADDR_USE_EXT 0xFFFE

Special address value - Short address value indicating extended address is used

12.3.7.33 #define APIMAC_SHORT_ADDR_BROADCAST 0xFFFF

Special address value - Broadcast short address

12.3.7.34 #define APIMAC_SHORT_ADDR_NONE 0xFFFF

Special address value - Short address when there is no short address

12.3.7.35 #define APIMAC_RANDOM_SEED_LEN 32

The length of the random seed is set for maximum requirement which is 32

12.3.7.36 #define APIMAC_FH_UTT_IE 0x00000002

Frequency Hopping UTT IE Selection Bit

12.3.7.37 #define APIMAC_FH_BT_IE 0x00000008

Frequency Hopping BT IE Selection Bit

12.3.7.38 #define APIMAC_FH_US_IE 0x00010000

Frequency Hopping US IE Selection Bit

12.3.7.39 #define APIMAC_FH_BS_IE 0x00020000

Frequency Hopping BS IE Selection Bit

12.3.7.40 #define APIMAC_FH_HEADER_IE_MASK 0x000000FF

Frequency hopping header IE's mask

12.3.7.41 #define APIMAC_FH_PROTO_DISPATCH_NONE 0x00

Frequency hopping Protocol dispatch values - Protocol dispatch none

12.3.7.42 #define APIMAC_FH_PROTO_DISPATCH_MHD_PDU 0x01

Frequency hopping Protocol dispatch values - Protocol dispatch MHD-PDU

12.3.7.43 #define APIMAC_FH_PROTO_DISPATCH_6LOWPAN 0x02

Frequency hopping Protocol dispatch values - Protocol dispatch 6LOWPAN

12.3.7.44 #define APIMAC_154G_MAX_NUM_CHANNEL 129

Maximum number of channels

12.3.7.45 #define APIMAC_154G_CHANNEL_BITMAP_SIZ ((APIMAC_154G_MAX_NUM_CHANNEL + 7) / 8)

Bitmap size to hold the channel list

12.3.7.46 #define APIMAC_HEADER_IE_MAX 2

Maximum number of header IEs

12.3.7.47 #define APIMAC_PAYLOAD_IE_MAX 2

Maximum number of payload-IEs

12.3.7.48 #define APIMAC_PAYLOAD_SUB_IE_MAX 4

Maximum number of sub-IEs

12.3.7.49 #define APIMAC_SFS_BEACON_ORDER(s) ((s) & 0x0F)

MACRO that returns the beacon order from the superframe specification

12.3.7.50 #define APIMAC_SFS_SUPERFRAME_ORDER(s) (((s) >> 4) & 0x0F)

MACRO that returns the superframe order from the superframe specification

12.3.7.51 #define APIMAC_SFS_FINAL_CAP_SLOT(s) (((s) >> 8) & 0x0F)

MACRO that returns the final CAP slot from the superframe specification

12.3.7.52 #define APIMAC_SFS_BLE(s) (((s) >> 12) & 0x01)

MACRO that returns the battery life extension bit from the superframe specification

12.3.7.53 #define APIMAC_SFS_PAN_COORDINATOR(s) (((s) >> 14) & 0x01)

MACRO that returns the PAN coordinator bit from the superframe specification

12.3.7.54 #define APIMAC_SFS_ASSOCIATION_PERMIT(s) (((s) >> 15) & 0x01)

MACRO that returns the Associate Permit bit from the superframe specification

12.3.7.55 #define APIMAC_FH_MAX_BIT_MAP_SIZE 32

Max size of the Frequency Hopping Channel Map Size

12.3.7.56 #define APIMAC_FH_NET_NAME_SIZE_MAX 32

Max size of the Frequency Hopping Network Name

12.3.7.57 #define APIMAC_FH_GTK_HASH_SIZE 8

Size of the Frequency Hopping GTK Hash Size

12.3.8 Typedef Documentation

12.3.8.1 typedef uint8_t ApiMac_sAddrExt_t[APIMAC_SADDR_EXT_LEN]

Extended address

12.3.8.2 typedef ApiMac_mcpsDataInd_t ApiMac_mlmeWsAsyncInd_t

MAC_MLME_WS_ASYNC_FRAME_IND type

12.3.8.3 typedef void(ApiMac_associateIndFp_t) (ApiMac_mlmeAssociateInd_t *pAssocInd)*

Associate Indication Callback function pointer prototype for the callback table

12.3.8.4 typedef void(ApiMac_associateCnfFp_t) (ApiMac_mlmeAssociateCnf_t *pAssocCnf)*

Assocate Confirmation Callback function pointer prototype for the callback table

12.3.8.5 typedef void(ApiMac_disassociateIndFp_t) (ApiMac_mlmeDisassociateInd_t *pDisassociateInd)*

Disassociate Indication Callback function pointer prototype for the callback table

12.3.8.6 typedef void(ApiMac_disassociateCnfFp_t) (ApiMac_mlmeDisassociateCnf_t *pDisassociateCnf)*

Disassociate Confirm Callback function pointer prototype for the callback table

12.3.8.7 typedef void(* ApiMac_beaconNotifyIndFp_t) (ApiMac_mlmeBeaconNotifyInd_t *pBeaconNotifyInd)

Beacon Notify Indication Callback function pointer prototype for the callback table

12.3.8.8 typedef void(* ApiMac_orphanIndFp_t) (ApiMac_mlmeOrphanInd_t *pOrphanInd)

Orphan Indication Callback function pointer prototype for the callback table

12.3.8.9 typedef void(* ApiMac_scanCnfFp_t) (ApiMac_mlmeScanCnf_t *pScanCnf)

Scan Confirmation Callback function pointer prototype for the callback table

12.3.8.10 typedef void(* ApiMac_startCnfFp_t) (ApiMac_mlmeStartCnf_t *pStartCnf)

Start Confirmation Callback function pointer prototype for the callback table

12.3.8.11 typedef void(* ApiMac_syncLossIndFp_t) (ApiMac_mlmeSyncLossInd_t *pSyncLossInd)

Sync Loss Indication Callback function pointer prototype for the callback table

12.3.8.12 typedef void(* ApiMac_pollCnfFp_t) (ApiMac_mlmePollCnf_t *pPollCnf)

Poll Confirm Callback function pointer prototype for the callback table

12.3.8.13 typedef void(* ApiMac_commStatusIndFp_t) (ApiMac_mlmeCommStatusInd_t *pCommStatus)

Comm Status Indication Callback function pointer prototype for the callback table

12.3.8.14 typedef void(* ApiMac_pollIndFp_t) (ApiMac_mlmePollInd_t *pPollInd)

Poll Indication Callback function pointer prototype for the callback table

12.3.8.15 typedef void(* ApiMac_dataCnfFp_t) (ApiMac_mcpsDataCnf_t *pDataCnf)

Data Confirmation Callback function pointer prototype for the callback table

12.3.8.16 typedef void(* ApiMac_dataIndFp_t) (ApiMac_mcpsDataInd_t *pDataInd)

Data Indication Callback function pointer prototype for the callback table

12.3.8.17 typedef void(* ApiMac_purgeCnfFp_t) (ApiMac_mcpsPurgeCnf_t *pPurgeCnf)

Purge Confirmation Callback function pointer prototype for the callback table

12.3.8.18 typedef void(* ApiMac_wsAsyncIndFp_t) (ApiMac_mlmeWsAsyncInd_t *pWsAsyncInd)

WiSUN Async Indication Callback function pointer prototype for the callback table

12.3.8.19 typedef void(* ApiMac_wsAsyncCnfFp_t) (ApiMac_mlmeWsAsyncCnf_t *pWsAsyncCnf)

WiSUN Async Confirmation Callback function pointer prototype for the callback table

12.3.8.20 typedef void(ApiMac_unprocessedFp_t) (uint16_t param1, uint16_t param2, void *pMsg)*

Unprocessed Message Callback function pointer prototype for the callback table. This function will be called when an unrecognized message is received.

12.3.9 Enumeration Type Documentation

12.3.9.1 enum ApiMac_assocStatus_t

Associate Response status types

Enumerator

ApiMac_assocStatus_success Success, join allowed

ApiMac_assocStatus_panAtCapacity PAN at capacity

ApiMac_assocStatus_panAccessDenied PAN access denied

12.3.9.2 enum ApiMac_addrType_t

Address types - used to set addrMode field of the ApiMac_sAddr_t structure.

Enumerator

ApiMac_addrType_none Address not present

ApiMac_addrType_short Short Address (16 bits)

ApiMac_addrType_extended Extended Address (64 bits)

12.3.9.3 enum ApiMac_beaconType_t

Beacon types in the ApiMac_mlmeBeaconNotifyInd_t structure.

Enumerator

ApiMac_beaconType_normal normal beacon type

ApiMac_beaconType_enhanced enhanced beacon type

12.3.9.4 enum ApiMac_disassocateReason_t

Disassociate Reasons

Enumerator

ApiMac_disassocateReason_coord The coordinator wishes the device to disassociate

ApiMac_disassocateReason_device The device itself wishes to disassociate

12.3.9.5 enum ApiMac_commStatusReason_t

Comm Status Indication Reasons

Enumerator

ApiMac_commStatusReason_assocRsp Reason for comm status indication was in response to an Associate Response

ApiMac_commStatusReason_orphanRsp Reason for comm status indication was in response to an Orphan Response

ApiMac_commStatusReason_rxSecure Reason for comm status indication was result of recieving a secure frame

12.3.9.6 enum ApiMac_status_t

General MAC Status values

Enumerator

ApiMac_status_success Operation successful

ApiMac_status_subSystemError MAC Co-Processor only - Subsystem Error

ApiMac_status_commandIDError MAC Co-Processor only - Command ID error

ApiMac_status_lengthError MAC Co-Processor only - Length error

ApiMac_status_unsupportedType MAC Co-Processor only - Unsupported Extended Type

ApiMac_status_autoAckPendingAllOn The AUTOPEND pending all is turned on

ApiMac_status_autoAckPendingAllOff The AUTOPEND pending all is turned off

ApiMac_status_beaconLoss The beacon was lost following a synchronization request

ApiMac_status_channelAccessFailure The operation or data request failed because of activity on the channel

ApiMac_status_counterError The frame counter puportedly applied by the originator of the received frame is invalid

ApiMac_status_denied The MAC was not able to enter low power mode

ApiMac_status_disabledTrxFailure Unused

ApiMac_status_frameTooLong The received frame or frame resulting from an operation or data request is too long to be processed by the MAC

ApiMac_status_improperKeyType The key purportedly applied by the originator of the received frame is not allowed

ApiMac_status_improperSecurityLevel The security level purportedly applied by the originator of the received frame does not meet the minimum security level

ApiMac_status_invalidAddress The data request failed because neither the source address nor destination address parameters were present

ApiMac_status_invalidGts Unused

ApiMac_status_invalidHandle The purge request contained an invalid handle

ApiMac_status_invalidIndex Unused

ApiMac_status_invalidParameter The API function parameter is out of range

ApiMac_status_limitReached The scan terminated because the PAN descriptor storage limit was reached

ApiMac_status_noAck The operation or data request failed because no acknowledgement was received

ApiMac_status_noBeacon The scan request failed because no beacons were received or the orphan scan failed because no coordinator realignment was received

ApiMac_status_noData The associate request failed because no associate response was received or the poll request did not return any data

ApiMac_status_noShortAddress The short address parameter of the start request was invalid

ApiMac_status_onTimeTooLong Unused

ApiMac_status_outOfCap Unused

ApiMac_status_panIdConflict A PAN identifier conflict has been detected and communicated to the PAN coordinator

ApiMac_status_pastTime Unused

ApiMac_status_readOnly A set request was issued with a read-only identifier

ApiMac_status_realignment A coordinator realignment command has been received

ApiMac_status_scanInProgress The scan request failed because a scan is already in progress

ApiMac_status_securityError Cryptographic processing of the received secure frame failed

ApiMac_status_superframeOverlap The beacon start time overlapped the coordinator transmission time

ApiMac_status_trackingOff The start request failed because the device is not tracking the beacon of its coordinator

ApiMac_status_transactionExpired The associate response, disassociate request, or indirect data transmission failed because the peer device did not respond before the transaction expired or was purged

ApiMac_status_transactionOverflow The request failed because MAC data buffers are full

ApiMac_status_txActive Unused

ApiMac_status_unavailableKey The operation or data request failed because the security key is not available

ApiMac_status_unsupportedAttribute The set or get request failed because the attribute is not supported

ApiMac_status_unsupportedLegacy The received frame was secured with legacy security which is not supported

ApiMac_status_unsupportedSecurity The security of the received frame is not supported

ApiMac_status_unsupported The operation is not supported in the current configuration

ApiMac_status_badState The operation could not be performed in the current state

ApiMac_status_noResources The operation could not be completed because no memory resources were available

ApiMac_status_ackPending For internal use only

ApiMac_status_noTime For internal use only

ApiMac_status_txAborted For internal use only

ApiMac_status_duplicateEntry For internal use only - A duplicated entry is added to the source matching table

ApiMac_status_fhError Frequency Hopping - General error

ApiMac_status_fhIeNotSupported Frequency Hopping - IE is not supported

ApiMac_status_fhNotInAsync Frequency Hopping - There is no ASYNC message in the MAC TX queue

ApiMac_status_fhNotInNeighborTable Frequency Hopping - Destination address is not in neighbor table

ApiMac_status_fhOutSlot Frequency Hopping - Not in UC or BC dwell time slot

ApiMac_status_fhInvalidAddress Frequency Hopping - Invalid address

ApiMac_status_fhIeFormatInvalid Frequency Hopping - IE format is wrong

ApiMac_status_fhPibNotSupported Frequency Hopping - PIB is not supported

ApiMac_status_fhPibReadOnly Frequency Hopping - PIB is read only

ApiMac_status_fhPibInvalidParameter Frequency Hopping - PIB API invalid parameter

ApiMac_status_fhInvalidFrameType Frequency Hopping - Invalid frame type

ApiMac_status_fhExpiredNode Frequency Hopping - Expired node

12.3.9.7 enum ApiMac secLevel t

MAC Security Levels

Enumerator

ApiMac_secLevel_none No security is used

ApiMac_secLevel_mic32 MIC-32 authentication is used

ApiMac_secLevel_mic64 MIC-64 authentication is used

ApiMac_secLevel_mic128 MIC-128 authentication is used

ApiMac_secLevel_enc AES encryption is used

ApiMac_secLevel_encMic32 AES encryption and MIC-32 authentication are used

ApiMac_secLevel_encMic64 AES encryption and MIC-64 authentication are used

ApiMac_secLevel_encMic128 AES encryption and MIC-128 authentication are used

12.3.9.8 enum ApiMac keyIdMode t

Key Identifier Mode

Enumerator

ApiMac_keyIdMode_implicit Key is determined implicitly

ApiMac_keyIdMode_1 Key is determined from the 1-byte key index

ApiMac_keyIdMode_4 Key is determined from the 4-byte key index

ApiMac_keyIdMode_8 Key is determined from the 8-byte key index

12.3.9.9 enum ApiMac attribute bool t

Standard PIB Get and Set Attributes - size bool

Enumerator

ApiMac_attribute_associatePermit TRUE if a coordinator is currently allowing association

ApiMac_attribute_autoRequest TRUE if a device automatically sends a data request if its address is listed in the beacon frame

ApiMac_attribute_battLifeExt TRUE if battery life extension is enabled

ApiMac_attribute_gtsPermit TRUE if the PAN coordinator accepts GTS requests

ApiMac_attribute_promiscuousMode TRUE if the MAC is in promiscuous mode

ApiMac_attribute_RxOnWhenIdle TRUE if the MAC enables its receiver during idle periods

ApiMac_attribute_associatedPanCoord TRUE if the device is associated to the PAN coordinator

ApiMac_attribute_timestampSupported TRUE if the MAC supports RX and TX timestamps

ApiMac_attribute_securityEnabled TRUE if security is enabled

ApiMac_attribute_includeMPMIE TRUE if MPM IE needs to be included

ApiMac_attribute_fcsType FCS type

12.3.9.10 enum ApiMac_attribute_uint8_t

Standard PIB Get and Set Attributes - size uint8_t

Enumerator

ApiMac_attribute_ackWaitDuration The maximum number of symbols to wait for an acknowledgment frame

ApiMac_attribute_battLifeExtPeriods The number of backoff periods during which the receiver is enabled following a beacon in battery life extension mode

ApiMac_attribute_beaconPayloadLength The length in bytes of the beacon payload, the maximum value for this parameters is APIMAC_MAX_BEACON_PAYLOAD.

ApiMac_attribute_beaconOrder How often the coordinator transmits a beacon

ApiMac_attribute_bsn The beacon sequence number

ApiMac_attribute_dsn The data or MAC command frame sequence number

ApiMac_attribute_maxCsmaBackoffs The maximum number of backoffs the CSMA-CA algorithm will attempt before declaring a channel failure

ApiMac_attribute_backoffExponent The minimum value of the backoff exponent in the CSMA-CA algorithm. If this value is set to 0, collision avoidance is disabled during the first iteration of the algorithm. Also for the slotted version of the CSMA-CA algorithm with the battery life extension enabled, the minimum value of the backoff exponent will be at least 2

ApiMac_attribute_superframeOrder This specifies the length of the active portion of the superframe

ApiMac_attribute_maxBackoffExponent The maximum value of the backoff exponent in the CSMA-CA algorithm

ApiMac_attribute_maxFrameRetries The maximum number of retries allowed after a transmission failure

ApiMac_attribute_responseWaitTime The maximum number of symbols a device shall wait for a response command to be available following a request command in multiples of aBaseSuperframeDuration

ApiMac_attribute_syncSymbolOffset The timestamp offset from SFD in symbols

ApiMac_attribute_eBeaconSequenceNumber Enhanced beacon sequence number

ApiMac_attribute_eBeaconOrder Enhanced beacon order in a beacon enabled network

ApiMac_attribute_offsetTimeslot Offset time slot from the beacon

ApiMac_attribute_phyTransmitPowerSigned Duplicate transmit power attribute in signed (2's complement) dBm unit

ApiMac_attribute_logicalChannel The logical channel

ApiMac_attribute_altBackoffExponent alternate minimum backoff exponent

ApiMac_attribute_deviceBeaconOrder Device beacon order

ApiMac_attribute_rf4cePowerSavings valid values are true and false

ApiMac_attribute_frameVersionSupport Currently supports 0 and 1. If 0, frame Version is always 0 and set to 1 only for secure frames. If 1, frame version will be set to 1 only if packet len > 102 or for secure frames

ApiMac_attribute_channelPage Channel Page

ApiMac_attribute_phyCurrentDescriptorId PHY Descriptor ID, used to support channel page number and index into descriptor table

12.3.9.11 enum ApiMac attribute uint16 t

Standard PIB Get and Set Attributes - size uint16_t

Enumerator

ApiMac_attribute_coordShortAddress The short address assigned to the coordinator with which the device is associated. A value of MAC_ADDR_USE_EXT indicates that the coordinator is using its extended address

ApiMac_attribute_panId The PAN identifier. If this value is 0xffff, the device is not associated

ApiMac_attribute_shortAddress The short address that the device uses to communicate in the PAN. If the device is a PAN coordinator, this value shall be set before calling MAC_StartReq(). Otherwise the value is allocated during association. Value MAC_ADDR_USE_EXT indicates that the device is associated but not using a short address

ApiMac_attribute_transactionPersistenceTime The maximum time in beacon intervals that a transaction is stored by a coordinator and indicated in the beacon

ApiMac_attribute_maxFrameTotalWaitTime The maximum number of CAP symbols in a beacon-enabled PAN, or symbols in a non beacon-enabled PAN, to wait for a frame intended as a response to a data request frame

ApiMac_attribute_eBeaconOrderNBPAN Enhanced beacon order in a non-beacon enabled network

12.3.9.12 enum ApiMac attribute uint32 t

Standard PIB Get and Set Attributes - size uint32_t

Enumerator

ApiMac_attribute_beaconTxTime The time the device transmitted its last beacon frame, in backoff period units

ApiMac_attribute_diagRxCrcPass Diagnostics PIB - Received CRC pass counter

ApiMac_attribute_diagRxCrcFail Diagnostics PIB - Received CRC fail counter

ApiMac_attribute_diagRxBroadcast Diagnostics PIB - Received broadcast counter

ApiMac_attribute_diagTxBroadcast Diagnostics PIB - Transmitted broadcast counter

ApiMac_attribute_diagRxUnicast Diagnostics PIB - Received unicast counter

ApiMac_attribute_diagTxUnicast Diagnostics PIB - Transmitted unicast counter

ApiMac_attribute_diagTxUnicastRetry Diagnostics PIB - Transmitted unicast retry counter

ApiMac_attribute_diagTxUnicastFail Diagnostics PIB - Transmitted unicast fail counter

ApiMac_attribute_diagRxSecureFail Diagnostics PIB - Received Security fail counter

ApiMac_attribute_diagTxSecureFail Diagnostics PIB - Transmit Security fail counter

12.3.9.13 enum ApiMac attribute array t

Standard PIB Get and Set Attributes - these attributes are array of bytes

Enumerator

ApiMac_attribute_beaconPayload The contents of the beacon payload

ApiMac_attribute_coordExtendedAddress The extended address of the coordinator with which the device is associated

ApiMac_attribute_extendedAddress The extended address of the device

12.3.9.14 enum ApiMac securityAttribute uint8 t

Security PIB Get and Set Attributes - size uint8_t

Enumerator

ApiMac_securityAttribute_keyTableEntries The number of entries in macKeyTable

ApiMac_securityAttribute_deviceTableEntries The number of entries in macDeviceTable

ApiMac_securityAttribute_securityLevelTableEntries The number of entries in macSecurityLevelTable

ApiMac_securityAttribute_autoRequestSecurityLevel The security level used for automatic data requests

ApiMac_securityAttribute_autoRequestKeyIdMode The key identifier mode used for automatic data requests

ApiMac_securityAttribute_autoRequestKeyIndex The index of the key used for automatic data requests

12.3.9.15 enum ApiMac securityAttribute uint16 t

Security PIB Get and Set Attributes - size uint16_t

Enumerator

ApiMac_securityAttribute_panCoordShortAddress The 16-bit short address assigned to the PAN coordinator

12.3.9.16 enum ApiMac securityAttribute array t

Security PIB Get and Set Attributes - array of bytes

Enumerator

ApiMac_securityAttribute_autoRequestKeySource The originator of the key used for automatic data requests

ApiMac_securityAttribute_defaultKeySource The originator of the default key used for key ID mode 0x01

ApiMac_securityAttribute_panCoordExtendedAddress The 64-bit address of the PAN coordinator

12.3.9.17 enum ApiMac_securityAttribute_struct_t

Security PIB Get and Set Attributes - these attributes are structures

Enumerator

ApiMac_securityAttribute_keyTable A table of KeyDescriptor, entries, each containing keys and related information required for secured communications. This is a SET only attribute. Call ApiMac_mlmeSetSecurityReqStruct() with pValue set to NULL, the MAC will build the table.

ApiMac_securityAttribute_keyIdLookupEntry The key lookup table entry, part of an entry of the key table. To GET or SET to this attribute, setup the keyIndex and keyIdLookupIndex fields of ApiMac_securityPibKeyIdLookupEntry_t, call ApiMac_mlmeGetSecurityReqStruct() or ApiMac_mlmeSetSecurityReqStruct() with a pointer to the ApiMac_securityPibKeyIdLookupEntry_t structure. For the GET, the lookupEntry field will contain the required data.

ApiMac_securityAttribute_keyDeviceEntry The key device entry, part of an entry of the key table. To GET or SET to this attribute, setup the keyIndex and keyDeviceIndex fields of ApiMac_securityPibKeyDeviceEntry_t, call ApiMac_mlmeGetSecurityReqStruct() or ApiMac_mlmeSetSecurityReqStruct() with a pointer to the ApiMac_securityPibKeyDeviceEntry_t structure. For the GET, the deviceEntry field will contain the required data.

ApiMac_securityAttribute_keyUsageEntry The key usage entry, part of an entry of the key table. To GET or SET to this attribute, setup the keyIndex and keyUsageIndex fields of ApiMac_securityPibKeyUsageEntry_t, call ApiMac_mlmeGetSecurityReqStruct() or ApiMac_mlmeSetSecurityReqStruct() with a pointer to the ApiMac_securityPibKeyUsageEntry_t structure. For the GET, the usageEntry field will contain the required data.

ApiMac_securityAttribute_keyEntry The MAC key entry, an entry of the key table. To GET or SET to this attribute, setup the keyIndex field of ApiMac_securityPibKeyEntry_t, call ApiMac_mlmeGetSecurityReqStruct() or ApiMac_mlmeSetSecurityReqStruct() with a pointer to the ApiMac_securityPibKeyEntry_t structure. For the GET, the rest of the fields will contain the required data.

ApiMac_securityAttribute_deviceEntry The MAC device entry, an entry of the device table. To GET or SET to this attribute, setup the deviceIndex field of ApiMac_securityPibDeviceEntry_t, call ApiMac_mlmeGetSecurityReqStruct() or ApiMac_mlmeSetSecurityReqStruct() with a pointer to the ApiMac_securityPibDeviceEntry_t structure. For the GET, the deviceEntry field will contain the required data.

ApiMac_securityAttribute_securityLevelEntry The MAC security level entry, an entry of the security level table. To GET or SET to this attribute, setup the levelIndex field of ApiMac_securityPibSecurityLevelEntry_t, call ApiMac_mlmeGetSecurityReqStruct() or ApiMac_mlmeSetSecurityReqStruct() with a pointer to the

ApiMac_securityPibSecurityLevelEntry_t structure. For the GET, the levelEntry field will contain the required data.

12.3.9.18 enum ApiMac_FHAttribute_uint8_t

Frequency Hopping PIB Get and Set Attributes - size uint8_t

Enumerator

ApiMac_FHAttribute_unicastDwellInterval Duration of node's unicast slot (in milliseconds) - uint8_t

ApiMac_FHAttribute_broadcastDwellInterval Duration of node's broadcast slot (in milliseconds) - uint8_t

ApiMac_FHAttribute_clockDrift Clock drift in PPM - uint8_t

ApiMac_FHAttribute_timingAccuracy Timing accuracy in 10 microsecond resolution - uint8_t

ApiMac_FHAttribute_unicastChannelFunction Unicast channel hopping function - uint8_t

ApiMac_FHAttribute_broadcastChannelFunction Broadcast channel hopping function - uint8_t

ApiMac_FHAttribute_useParentBSIE Node is propagating parent's BS-IE - uint8_t

ApiMac_FHAttribute_routingCost Estimate of routing path ETX to the PAN coordinator - uint8_t

ApiMac_FHAttribute_routingMethod RPL(1), MHDS(0) - uint8_t

ApiMac_FHAttribute_eapolReady Node can accept EAPOL message - uint8_t

ApiMac_FHAttribute_fanTPSVersion Wi-SUN FAN version - uint8_t

ApiMac_FHAttribute_gtk0Hash Low order 64 bits of SHA256 hash of GTK

- APIMAC_FH_NET_NAME_SIZE_MAX uint8_t

ApiMac_FHAttribute_gtk1Hash Next low order 64 bits of SHA256 hash of GTK

- APIMAC_FH_NET_NAME_SIZE_MAX uint8_t

ApiMac_FHAttribute_gtk2Hash Next low order 64 bits of SHA256 hash of GTK

- APIMAC_FH_NET_NAME_SIZE_MAX uint8_t

ApiMac_FHAttribute_gtk3Hash Next low order 64 bits of SHA256 hash of GTK

- APIMAC_FH_NET_NAME_SIZE_MAX uint8_t

12.3.9.19 enum ApiMac_FHAttribute_uint16_t

Frequency Hopping PIB Get and Set Attributes - size uint16_t

Enumerator

ApiMac_FHAttribute_broadcastSchedId Broadcast schedule ID for broadcast channel hopping sequence - uint16_t

ApiMac_FHAttribute_unicastFixedChannel Unicast channel number when no hopping - uint16_t

ApiMac_FHAttribute_broadcastFixedChannel Broadcast channel number when no hopping - uint16_t

ApiMac_FHAttribute_panSize Number of nodes in the PAN - uint16_t

ApiMac_FHAttribute_panVersion PAN version to notify PAN configuration changes - uint16_t

ApiMac_FHAttribute_neighborValidTime Time in min during which the node info considered as valid - uint16_t

12.3.9.20 enum ApiMac_FHAttribute_uint32_t

Frequency Hopping PIB Get and Set Attributes - size uint32_t

Enumerator

ApiMac_FHAttribute_BCInterval Time between start of two broadcast slots (in milliseconds) - uint32_t

12.3.9.21 enum ApiMac_FHAttribute_array_t

Frequency Hopping PIB Get and Set Attributes - array of bytes

Enumerator

ApiMac_FHAttribute_trackParentEUI The parent EUI address - ApiMac_sAddrExt_t

ApiMac_FHAttribute_unicastExcludedChannels Unicast excluded channels - APIMAC_FH_MAX_BIT_MAP_SIZE

ApiMac_FHAttribute_broadcastExcludedChannels Broadcast excluded channels - APIMAC_FH_MAX_BIT_MAP_SIZE

ApiMac_FHAttribute_netName Network Name - APIMAC_FH_NET_NAME_SIZE_MAX uint8_t

12.3.9.22 enum ApiMac_fhFrameType_t

FH Frame Types

Enumerator

ApiMac_fhFrameType_panAdvert WiSUN PAN advertisement

ApiMac_fhFrameType_panAdvertSolicit WiSUN PAN advertisement solicit

ApiMac_fhFrameType_config WiSUN PAN config

ApiMac_fhFrameType_configSolicit WiSUN PAN config solicit

ApiMac_fhFrameType_data WiSUN Data frame

ApiMac_fhFrameType_ack WiSUN Ack frame

ApiMac_fhFrameType_eapol WiSUN Ack frame

ApiMac_fhFrameType_invalid Internal: WiSUN Invalid frame

12.3.9.23 enum ApiMac_payloadIEGroup_t

Payload IE Group IDs

Enumerator

ApiMac_payloadIEGroup_ESDU Payload ESDU IE Group ID

ApiMac_payloadIEGroup_MLME Payload MLME IE Group ID

ApiMac_payloadIEGroup_WiSUN Payload WiSUN IE Group ID

ApiMac_payloadIEGroup_term Payload Termination IE Group ID

12.3.9.24 enum ApiMac_MLMESubIE_t

MLME Sub IEs

Enumerator

ApiMac_MLMESubIE_coexist MLME Sub IEs - short format - Coexistence IE

ApiMac_MLMESubIE_sunDevCap MLME Sub IEs - short format - SUN Device capabilities IE

ApiMac_MLMESubIE_sunFSKGenPhy MLME Sub IEs - short format - SUN FSK Generic PHY IE

12.3.9.25 enum ApiMac_wisunSubIE_t

WiSUN Sub IEs

Enumerator

ApiMac_wisunSubIE_USIE WiSUN Sub IE - Long format - Unicast Schedule IE

ApiMac_wisunSubIE_BSIE WiSUN Sub IE - Long format - Broadcast Schedule IE

ApiMac_wisunSubIE_PANIE WiSUN Sub IE - Short format - PAN IE

ApiMac_wisunSubIE_netNameIE WiSUN Sub IE - Short format - Network Name IE

ApiMac_wisunSubIE_PANVersionIE WiSUN Sub IE - Short format - PAN Version IE

ApiMac_wisunSubIE_GTKHashIE WiSUN Sub IE - Short format - GTK Hash IE

12.3.9.26 enum ApiMac_scantype_t

Scan Types

Enumerator

ApiMac_scantype_energyDetect Energy detect scan. The device will tune to each channel and perform and energy measurement. The list of channels and their associated measurements will be returned at the end of the scan

ApiMac_scantype_active Active scan. The device tunes to each channel, sends a beacon request and listens for beacons. The PAN descriptors are returned at the end of the scan

ApiMac_scantype_passive Passive scan. The device tunes to each channel and listens for beacons. The PAN descriptors are returned at the end of the scan

ApiMac_scantype_orphan Orphan scan. The device tunes to each channel and sends an orphan notification to try and find its coordinator. The status is returned at the end of the scan

ApiMac_scantype_activeEnhanced Enhanced Active scan. In addition to Active scan, this command is also used by a device to locate a subset of all coordinators within its POS during an active scan

12.3.9.27 enum ApiMac_wisunAsycnOperation_t

WiSUN Async Operations

Enumerator

ApiMac_wisunAsycnOperation_start Start Async

ApiMac_wisunAsycnOperation_stop Stop Async

12.3.9.28 enum ApiMac_wisunAsyncFrame_t

WiSUN Async Frame Types

Enumerator

ApiMac_wisunAsyncFrame_advertisement WiSUN Async PAN Advertisement Frame type

ApiMac_wisunAsyncFrame_advertisementSolicit WiSUN Async PAN Advertisement Solicitation Frame type

ApiMac_wisunAsyncFrame_config WiSUN Async PAN Configuration Frame type

ApiMac_wisunAsyncFrame_configSolicit WiSUN Async PAN Configuration Solicitation Frame type

12.3.9.29 enum ApiMac_fhDispatchType_t

Freqency Hopping Dispatch Values

Enumerator

ApiMac_fhDispatchType_none No protocol dispatch

ApiMac_fhDispatchType_MHD_PDU MHD-PDU protocol dispatch

ApiMac_fhDispatchType_6LowPAN 6LowPAN protocol dispatch

12.3.10 Function Documentation

12.3.10.1 void ApiMac_init (bool* **enableFH***)*

Initialize this module.

12.3.10.1.1 Parameters:

enableFH	- true to enable frequency hopping, false to not.

12.3.10.1.2 Returns:

pointer to a wakeup variable (semaphore in some systems)

*12.3.10.2 void ApiMac_registerCallbacks (ApiMac_callbacks_t * **pCallbacks**)*

Register for MAC callbacks.

12.3.10.2.1 Parameters:

pCallbacks	- pointer to callback structure

12.3.10.3 void ApiMac_processIncoming (void)

Process incoming messages from the MAC stack.

*12.3.10.4 ApiMac_status_t ApiMac_mcpsDataReq (ApiMac_mcpsDataReq_t * pData)*

This function sends application data to the MAC for transmission in a MAC data frame.

The MAC can only buffer a certain number of data request frames. When the MAC is congested and cannot accept the data request it will initiate a callback (ApiMac_dataCnfFp_t) with an overflow status (ApiMac_status_transactionOverflow) . Eventually the MAC will become uncongested and initiate the callback (ApiMac_dataCnfFp_t) for a buffered request. At this point the application can attempt another data request. Using this scheme, the application can send data whenever it wants but it must queue data to be resent if it receives an overflow status.

12.3.10.4.1 Parameters:

pData	- pointer to parameter structure

12.3.10.4.2 Returns:

The status of the request, as follows:
ApiMac_status_success

- Operation successful
- ApiMac_status_noResources - Resources not available

12.3.10.5 ApiMac_status_t ApiMac_mcpsPurgeReq (uint8_t msduHandle)

This function purges and discards a data request from the MAC data queue. When the operation is complete the MAC sends a MCPS Purge Confirm which will initiate a callback (ApiMac_purgeCnfFp_t).

12.3.10.5.1 Parameters:

msduHandle	- The application-defined handle value

12.3.10.5.2 Returns:

The status of the request, as follows:
ApiMac_status_success

- Operation successful
- ApiMac_status_noResources - Resources not available

*12.3.10.6 ApiMac_status_t ApiMac_mlmeAssociateReq (ApiMac_mlmeAssociateReq_t * pData)*

This function sends an associate request to a coordinator device. The application shall attempt to associate only with a PAN that is currently allowing association, as indicated in the results of the

scanning procedure. In a beacon-enabled PAN the beacon order must be set by using ApiMac_mlmeSetReq() before making the call to ApiMac_mlmeAssociateReq().

When the associate request is complete the appliction will receive the ApiMac_associateCnfFp_t callback.

12.3.10.6.1 Parameters:

pData	- Pointer to parameters structure.

12.3.10.6.2 Returns:

The status of the request, as follows:
ApiMac_status_success

- Operation successful
- ApiMac_status_noResources - Resources not available

*12.3.10.7 ApiMac_status_t ApiMac_mlmeAssociateRsp (ApiMac_mlmeAssociateRsp_t * pData)*

This function sends an associate response to a device requesting to associate. This function must be called after the ApiMac_associateIndFp_t callback. When the associate response is complete the callback ApiMac_commStatusIndFp_t is called to indicate the success or failure of the operation.

12.3.10.7.1 Parameters:

pData	- Pointer to parameters structure.

12.3.10.7.2 Returns:

The status of the request, as follows:
ApiMac_status_success

- Operation successful
- ApiMac_status_noResources - Resources not available

*12.3.10.8 ApiMac_status_t ApiMac_mlmeDisassociateReq (ApiMac_mlmeDisassociateReq_t * pData)*

This function is used by an associated device to notify the coordinator of its intent to leave the PAN. It is also used by the coordinator to instruct an associated device to leave the PAN. When the disassociate procedure is complete the applications callback ApiMac_disassociateCnfFp_t is called.

12.3.10.8.1 Parameters:

pData	- Pointer to parameters structure.

12.3.10.8.2 Returns:

The status of the request, as follows:
ApiMac_status_success

- Operation successful
- ApiMac_status_noResources - Resources not available

12.3.10.9 ApiMac_status_t ApiMac_mlmeGetReqBool (ApiMac_attribute_bool_t **pibAttribute***, bool* * **pValue***)*

This direct execute function retrieves an attribute value from the MAC PIB.

12.3.10.9.1 Parameters:

pibAttribute	- The attribute identifier
pValue	- pointer to the attribute value

12.3.10.9.2 Returns:

The status of the request

12.3.10.10 ApiMac_status_t ApiMac_mlmeGetReqUint8 (ApiMac_attribute_uint8_t **pibAttribute***, uint8_t* * **pValue***)*

This direct execute function retrieves an attribute value from the MAC PIB.

12.3.10.10.1 Parameters:

pibAttribute	- The attribute identifier
pValue	- pointer to the attribute value

12.3.10.10.2 Returns:

The status of the request

12.3.10.11 ApiMac_status_t ApiMac_mlmeGetReqUint16 (ApiMac_attribute_uint16_t **pibAttribute***, uint16_t* * **pValue***)*

This direct execute function retrieves an attribute value from the MAC PIB.

12.3.10.11.1 Parameters:

pibAttribute	- The attribute identifier
pValue	- pointer to the attribute value

12.3.10.11.2 Returns:

The status of the request

12.3.10.12 ApiMac_status_t ApiMac_mlmeGetReqUint32 (ApiMac_attribute_uint32_t **pibAttribute***, uint32_t** * **pValue***)*

This direct execute function retrieves an attribute value from the MAC PIB.

12.3.10.12.1 Parameters:

pibAttribute	- The attribute identifier
pValue	- pointer to the attribute value

12.3.10.12.2 Returns:

The status of the request

12.3.10.13 ApiMac_status_t ApiMac_mlmeGetReqArray (ApiMac_attribute_array_t **pibAttribute***, uint8_t** * **pValue***)*

This direct execute function retrieves an attribute value from the MAC PIB.

12.3.10.13.1 Parameters:

pibAttribute	- The attribute identifier
pValue	- pointer to the attribute value

12.3.10.13.2 Returns:

The status of the request

12.3.10.14 ApiMac_status_t ApiMac_mlmeGetFhReqUint8 (ApiMac_FHAttribute_uint8_t **pibAttribute***, uint8_t** * **pValue***)*

This direct execute function retrieves an attribute value from the MAC Frequency Hopping PIB.

12.3.10.14.1 Parameters:

pibAttribute	- The attribute identifier
pValue	- pointer to the attribute value

12.3.10.14.2 Returns:

The status of the request, as follows:
ApiMac_status_success
- Operation successful
- ApiMac_status_unsupportedAttribute - Attribute not found

12.3.10.15 ApiMac_status_t ApiMac_mlmeGetFhReqUint16 (ApiMac_FHAttribute_uint16_t **pibAttribute***, uint16_t** * **pValue***)*

This direct execute function retrieves an attribute value from the MAC Frequency Hopping PIB.

12.3.10.15.1 Parameters:

pibAttribute	- The attribute identifier
pValue	- pointer to the attribute value

12.3.10.15.2 Returns:

The status of the request, as follows:
ApiMac_status_success
- Operation successful
- ApiMac_status_unsupportedAttribute - Attribute not found

12.3.10.16 ApiMac_status_t ApiMac_mlmeGetFhReqUint32 (ApiMac_FHAttribute_uint32_t **pibAttribute***, uint32_t** * **pValue***)*

This direct execute function retrieves an attribute value from the MAC Frequency Hopping PIB.

12.3.10.16.1 Parameters:

pibAttribute	- The attribute identifier
pValue	- pointer to the attribute value

12.3.10.16.2 Returns:

The status of the request, as follows:
ApiMac_status_success

- Operation successful
- ApiMac_status_unsupportedAttribute - Attribute not found

12.3.10.17 ApiMac_status_t ApiMac_mlmeGetFhReqArray (ApiMac_FHAttribute_array_t **pibAttribute***, uint8_t** **pValue***)*

This direct execute function retrieves an attribute value from the MAC Frequency Hopping PIB.

12.3.10.17.1 Parameters:

pibAttribute	- The attribute identifier
pValue	- pointer to the attribute value

12.3.10.17.2 Returns:

The status of the request, as follows:
ApiMac_status_success
- Operation successful
- ApiMac_status_unsupportedAttribute - Attribute not found

12.3.10.18 ApiMac_status_t ApiMac_mlmeGetSecurityReqUint8 (ApiMac_securityAttribute_uint8_t **pibAttribute***, uint8_t** **pValue***)*

This direct execute function retrieves an attribute value from the MAC Secutity PIB.

12.3.10.18.1 Parameters:

pibAttribute	- The attribute identifier
pValue	- pointer to the attribute value

12.3.10.18.2 Returns:

The status of the request, as follows:
ApiMac_status_success
- Operation successful
- ApiMac_status_unsupportedAttribute - Attribute not found

12.3.10.19 ApiMac_status_t ApiMac_mlmeGetSecurityReqUint16 (ApiMac_securityAttribute_uint16_t **pibAttribute***, uint16_t** **pValue***)*

This direct execute function retrieves an attribute value from the MAC Secutity PIB.

12.3.10.19.1 Parameters:

pibAttribute	- The attribute identifier
pValue	- pointer to the attribute value

12.3.10.19.2 Returns:

The status of the request, as follows:
ApiMac_status_success
- Operation successful
- ApiMac_status_unsupportedAttribute - Attribute not found

12.3.10.20 ApiMac_status_t ApiMac_mlmeGetSecurityReqArray (ApiMac_securityAttribute_array_t **pibAttribute***, uint8_t** * **pValue***)*

This direct execute function retrieves an attribute value from the MAC Secutity PIB.

12.3.10.20.1 Parameters:

pibAttribute	- The attribute identifier
pValue	- pointer to the attribute value

12.3.10.20.2 Returns:

The status of the request, as follows:
ApiMac_status_success
- Operation successful
- ApiMac_status_unsupportedAttribute - Attribute not found

12.3.10.21 ApiMac_status_t ApiMac_mlmeGetSecurityReqStruct (ApiMac_securityAttribute_struct_t **pibAttribute***, void** * **pValue***)*

This direct execute function retrieves an attribute value from the MAC Secutity PIB.

12.3.10.21.1 Parameters:

pibAttribute	- The attribute identifier
pValue	- pointer to the attribute value

12.3.10.21.2 Returns:

The status of the request, as follows:
ApiMac_status_success

- Operation successful
- ApiMac_status_unsupportedAttribute - Attribute not found

*12.3.10.22 ApiMac_status_t ApiMac_mlmeOrphanRsp (ApiMac_mlmeOrphanRsp_t * pData)*

This function is called in response to an orphan notification from a peer device. This function must be called after receiving an Orphan_Indication_Callback. When the orphan response is complete the Comm_Status_Indication_Callback is called to indicate the success or failure of the operation.

12.3.10.22.1 Parameters:

pData	- Pointer to parameters structure.

12.3.10.22.2 Returns:

The status of the request, as follows:
ApiMac_status_success
- Operation successful
- ApiMac_status_noResources - Resources not available

*12.3.10.23 ApiMac_status_t ApiMac_mlmePollReq (ApiMac_mlmePollReq_t * pData)*

This function is used to request pending data from the coordinator. When the poll request is complete the Poll_Confirm_Callback is called. If a data frame of nonzero length is received from the coordinator the Poll_Confirm_Callback has a status ApiMac_status_success and then calls the Data_Indication_Callback for the received data.

12.3.10.23.1 Parameters:

pData	- Pointer to parameters structure.

12.3.10.23.2 Returns:

The status of the request, as follows:
ApiMac_status_success
- Operation successful
- ApiMac_status_noResources - Resources not available

12.3.10.24 ApiMac_status_t ApiMac_mlmeResetReq (bool setDefaultPib)

This direct execute function resets the MAC. This function must be called once at system startup before any other function in the management API is called.

12.3.10.24.1 Parameters:

setDefaultPib	- Set to TRUE to reset the MAC PIB to its default values.

12.3.10.24.2 Returns:

always ApiMac_status_success

*12.3.10.25 ApiMac_status_t ApiMac_mlmeScanReq (ApiMac_mlmeScanReq_t * pData)*

This function initiates an energy detect, active, passive, or orphan scan on one or more channels. An energy detect scan measures the peak energy on each requested channel. An active scan sends a beacon request on each channel and then listening for beacons. A passive scan is a receive-only operation that listens for beacons on each channel. An orphan scan is used to locate the coordinator with which the scanning device had previously associated. When a scan operation is complete the Scan Confirm callback is called.

For active or passive scans the application sets the maxResults parameter the maximum number of PAN descriptors to return. If maxResults is greater than zero then the application must also set result.panDescriptor to point to a buffer of size maxResults * sizeof(ApiMac_panDesc_t) to store the results of the scan. The application must not access or deallocate this buffer until the Scan Confirm Callback is called. The MAC will store up to maxResults PAN descriptors and ignore duplicate beacons.

An alternative way to get results for an active or passive scan is to set maxResults to zero or set PIB attribute ApiMac_attribute_autoRequest to FALSE. Then the MAC will not store results but rather call the Beacon Notify Indication Callback" for each beacon received. The application will not need to supply any memory to store the scan results but the MAC will not filter out duplicate beacons.

For energy detect scans the application must set result.energyDetect to point to a buffer of size 18 bytes to store the results of the scan. The application must not access or deallocate this buffer until the Scan Confirm Callback is called.

An energy detect, active or passive scan may be performed at any time if a scan is not already in progress. However a device cannot perform any other MAC management operation or send or receive MAC data until the scan is complete.

12.3.10.25.1 Parameters:

pData	- Pointer to parameters structure.

12.3.10.25.2 Returns:

The status of the request, as follows:
ApiMac_status_success

- Operation successful
- ApiMac_status_scanInProgress - already scanning
- ApiMac_status_noResources - memory allocation error

12.3.10.26 ApiMac_status_t ApiMac_mlmeSetReqBool (ApiMac_attribute_bool_t pibAttribute, bool value)

This direct execute function sets an attribute value in the MAC PIB.

12.3.10.26.1 Parameters:

pibAttribute	- The attribute identifier
value	- the attribute value

12.3.10.26.2 Returns:

The status of the request

12.3.10.27 ApiMac_status_t ApiMac_mlmeSetReqUint8 (ApiMac_attribute_uint8_t **pibAttribute***, uint8_t* **value***)*

This direct execute function sets an attribute value in the MAC PIB.

12.3.10.27.1 Parameters:

pibAttribute	- The attribute identifier
value	- the attribute value

12.3.10.27.2 Returns:

The status of the request

12.3.10.28 ApiMac_status_t ApiMac_mlmeSetReqUint16 (ApiMac_attribute_uint16_t **pibAttribute***, uint16_t* **value***)*

This direct execute function sets an attribute value in the MAC PIB.

12.3.10.28.1 Parameters:

pibAttribute	- The attribute identifier
value	- the attribute value

12.3.10.28.2 Returns:

The status of the request

12.3.10.29 ApiMac_status_t ApiMac_mlmeSetReqUint32 (ApiMac_attribute_uint32_t **pibAttribute***, uint32_t* **value***)*

This direct execute function sets an attribute value in the MAC PIB.

12.3.10.29.1 Parameters:

pibAttribute	- The attribute identifier
value	- the attribute value

12.3.10.29.2 Returns:

The status of the request

12.3.10.30 ApiMac_status_t ApiMac_mlmeSetReqArray (ApiMac_attribute_array_t **pibAttribute***, uint8_t** * **pValue***)*

This direct execute function sets an attribute value in the MAC PIB.

12.3.10.30.1 Parameters:

pibAttribute	- The attribute identifier
pValue	- the attribute value

12.3.10.30.2 Returns:

The status of the request

12.3.10.31 ApiMac_status_t ApiMac_mlmeSetFhReqUint8 (ApiMac_FHAttribute_uint8_t **pibAttribute***, uint8_t* **value***)*

This direct execute function sets an attribute value in the MAC Frequency Hopping PIB.

12.3.10.31.1 Parameters:

pibAttribute	- The attribute identifier
value	- the attribute value

12.3.10.31.2 Returns:

The status of the request, as follows:
ApiMac_status_success

- Operation successful
- ApiMac_status_unsupportedAttribute - Attribute not found

12.3.10.32 ApiMac_status_t ApiMac_mlmeSetFhReqUint16 (ApiMac_FHAttribute_uint16_t **pibAttribute***, uint16_t* **value***)*

This direct execute function sets an attribute value in the MAC Frequency Hopping PIB.

12.3.10.32.1 Parameters:

pibAttribute	- The attribute identifier
value	- the attribute value

12.3.10.32.2 Returns:

The status of the request, as follows:
ApiMac_status_success

- Operation successful
- ApiMac_status_unsupportedAttribute - Attribute not found

12.3.10.33 ApiMac_status_t ApiMac_mlmeSetFhReqUint32 (ApiMac_FHAttribute_uint32_t **pibAttribute***, uint32_t* **value***)*

This direct execute function sets an attribute value in the MAC Frequency Hopping PIB.

12.3.10.33.1 Parameters:

pibAttribute	- The attribute identifier
value	- the attribute value

12.3.10.33.2 Returns:

The status of the request, as follows:
ApiMac_status_success

- Operation successful
- ApiMac_status_unsupportedAttribute - Attribute not found

12.3.10.34 ApiMac_status_t ApiMac_mlmeSetFhReqArray (ApiMac_FHAttribute_array_t **pibAttribute***, uint8_t ** **pValue***)*

This direct execute function sets an attribute value in the MAC Frequency Hopping PIB.

12.3.10.34.1 Parameters:

pibAttribute	- The attribute identifier
pValue	- pointer to the attribute value

12.3.10.34.2 Returns:

The status of the request, as follows:
ApiMac_status_success
- Operation successful
- ApiMac_status_unsupportedAttribute - Attribute not found

12.3.10.35 ApiMac_status_t ApiMac_mlmeSetSecurityReqUint8 (ApiMac_securityAttribute_uint8_t pibAttribute, uint8_t value)

This direct execute function sets an attribute value in the MAC Security PIB.

12.3.10.35.1 Parameters:

pibAttribute	- The attribute identifier
value	- the attribute value

12.3.10.35.2 Returns:

The status of the request, as follows:
ApiMac_status_success
- Operation successful
- ApiMac_status_unsupportedAttribute - Attribute not found

12.3.10.36 ApiMac_status_t ApiMac_mlmeSetSecurityReqUint16 (ApiMac_securityAttribute_uint16_t pibAttribute, uint16_t value)

This direct execute function sets an attribute value in the MAC Security PIB.

12.3.10.36.1 Parameters:

pibAttribute	- The attribute identifier
value	- the attribute value

12.3.10.36.2 Returns:

The status of the request, as follows:
ApiMac_status_success
- Operation successful

- ApiMac_status_unsupportedAttribute - Attribute not found

12.3.10.37 ApiMac_status_t ApiMac_mlmeSetSecurityReqArray (ApiMac_securityAttribute_array_t **pibAttribute***, uint8_t** **pValue***)*

This direct execute function sets an attribute value in the MAC Security PIB.

12.3.10.37.1 Parameters:

pibAttribute	- The attribute identifier
pValue	- pointer to the attribute value

12.3.10.37.2 Returns:

The status of the request, as follows:
ApiMac_status_success
- Operation successful
- ApiMac_status_unsupportedAttribute - Attribute not found

12.3.10.38 ApiMac_status_t ApiMac_mlmeSetSecurityReqStruct (ApiMac_securityAttribute_struct_t **pibAttribute***, void** **pValue***)*

This direct execute function sets an attribute value in the MAC Security PIB.

12.3.10.38.1 Parameters:

pibAttribute	- The attribute identifier
pValue	- pointer to the attribute value

12.3.10.38.2 Returns:

The status of the request, as follows:
ApiMac_status_success
- Operation successful
- ApiMac_status_unsupportedAttribute - Attribute not found

*12.3.10.39 ApiMac_status_t ApiMac_mlmeStartReq (ApiMac_mlmeStartReq_t** **pData***)*

This function is called by a coordinator or PAN coordinator to start or reconfigure a network. Before starting a network the device must have set its short address. A PAN coordinator sets the short address by setting the attribute ApiMac_attribute_shortAddress. A coordinator sets the short address through association.

When parameter panCoordinator is TRUE, the MAC automatically sets attributes ApiMac_attribute_panID and ApiMac_attribute_logicalChannel to the panId and logicalChannel parameters. If panCoordinator is FALSE, these parameters are ignored (they would already be set through association).

The parameter beaconOrder controls whether the network is beacon-enabled or non beacon-enabled. For a beacon-enabled network this parameter also controls the beacon transmission interval.

When the operation is complete the Start Confirm Callback is called.

12.3.10.39.1 Parameters:

pData	- Pointer to parameters structure.

12.3.10.39.2 Returns:

The status of the request, as follows:
ApiMac_status_success

- Operation successful
- ApiMac_status_noResources - Resources not available

*12.3.10.40 ApiMac_status_t ApiMac_mlmeSyncReq (ApiMac_mlmeSyncReq_t * pData)*

This function requests the MAC to synchronize with the coordinator by acquiring and optionally tracking its beacons. Synchronizing with the coordinator is recommended before associating in a beacon-enabled network. If the beacon could not be located on its initial search or during tracking, the MAC calls the Sync Loss Indication Callback with ApiMac_status_beaconLoss as the reason.

Before calling this function the application must set PIB attributes ApiMac_attribute_beaconOrder, ApiMac_attribute_panId and either ApiMac_attribute_coordShortAddress or ApiMac_attribute_coordExtendedAddress to the address of the coordinator with which to synchronize.

The application may wish to set PIB attribute ApiMac_attribute_autoRequest to FALSE before calling this function. Then when the MAC successfully synchronizes with the coordinator it will call the Beacon_Notify_Indication Callback". After receiving the callback the application may set ApiMac_attribute_autoRequest to TRUE to stop receiving beacon notifications.

This function is only applicable to beacon-enabled networks.

12.3.10.40.1 Parameters:

pData	- Pointer to parameters structure.

12.3.10.40.2 Returns:

The status of the request, as follows:
ApiMac_status_success

- Operation successful

- ApiMac_status_noResources - Resources not available

12.3.10.41 uint8_t ApiMac_randomByte (void)

This function returns a random byte from the MAC random number generator.

12.3.10.41.1 Returns:

A random byte.

12.3.10.42 ApiMac_status_t ApiMac_updatePanId (uint16_t **panId***)*

Update Device Table entry and PIB with new Pan Id.

12.3.10.42.1 Parameters:

panId	- the new Pan ID

12.3.10.42.2 Returns:

The status of the request, as follows:
ApiMac_status_success

- Operation successful
- ApiMac_status_noResources - Resources not available

*12.3.10.43 ApiMac_status_t ApiMac_mlmeWSAsyncReq (ApiMac_mlmeWSAsyncReq_t **

pData*)*

This functions handles a WiSUN async request. The possible operation is Async Start or Async Stop. For the async start operation, the caller of this function can indicate which WiSUN async frame type to be sent on the specified channels.

12.3.10.43.1 Parameters:

pData	pointer to the asynchronous parameters structure

12.3.10.43.2 Returns:

The status of the request, as follows:
ApiMac_status_success

- Operation successful
- ApiMac_status_noResources - Resources not available

12.3.10.44 ApiMac_status_t ApiMac_startFH (void)

This function starts the frequency hopping. Frequency hopping operation should have been enabled using ApiMac_enableFH() before calling this API. No need to call this API if you have called ApiMac_mlmeStartReq() with the startFH field set to true.

12.3.10.44.1 Returns:

The status of the request, as follows:
ApiMac_status_success
- Operation successful
- ApiMac_status_noResources - Resources not available

*12.3.10.45 ApiMac_status_t ApiMac_parsePayloadGroupIEs (uint8_t * pPayload, uint16_t payloadLen, ApiMac_payloadIeRec_t ** pList)*

Parses the Group payload information element. This function creates a linked list (plist) from the Payload IE (pPayload). Each item in the linked list is a seperate Group IE with its own content.

If no IEs are found pList will be set to NULL.

The caller is responsible to release the memory for the linked list by calling ApiMac_freeIEList().

Call this function to create the list of Group IEs, then call ApiMac_parsePayloadSubIEs() to parse each of the group IE's content into sub IEs.

12.3.10.45.1 Parameters:

pPayload	- pointer to the buffer with the payload IEs.
payloadLen	- length of the buffer with the payload IEs.
pList	- pointer to link list pointer.

12.3.10.45.2 Returns:

The status of the request, as follows:
ApiMac_status_success
- Operation successful
- ApiMac_status_noData

- pPayload or payloadLen is NULL,
- ApiMac_status_unsupported
- invalid field found,
- ApiMac_status_noResources
- if memory allocation fails.

*12.3.10.46 ApiMac_status_t ApiMac_parsePayloadSubIEs (uint8_t * pContent, uint16_t contentLen, ApiMac_payloadIeRec_t ** pList)*

Parses the payload sub information element. This function creates a linked list (pList) of sub IEs from the Group IE content (pContent). Each item in the linked list is a seperate sub IE with its own content.

If no IEs are found pList will be set to NULL.

The caller is responsible to release the memory for the linked list by calling ApiMac_freeIEList().

Call this function after calling ApiMac_parsePayloadGroupIEs().

12.3.10.46.1 Parameters:

pContent	- pointer to the buffer with the sub IEs.
contentLen	- length of the buffer with the payload IEs.
pList	- pointer to link list pointer.

12.3.10.46.2 Returns:

The status of the request, as follows:

ApiMac_status_success

- Operation successful
- ApiMac_status_noData

- pPayload or payloadLen is NULL,
- ApiMac_status_unsupported
- invalid field found,
- ApiMac_status_noResources
- if memory allocation fails.

*12.3.10.47 void ApiMac_freeIEList (ApiMac_payloadIeRec_t * pList)*

Free the linked list allocated by ApiMac_parsePayloadGroupIEs() or ApiMac_parsePayloadSubIEs().

12.3.10.47.1 Parameters:

pList	- pointer to linked list

12.3.10.48 ApiMac_status_t ApiMac_enableFH (void)

Enables the Frequency hopping operation. Make sure you call this function before setting any FH parameters or before calling ApiMac_mlmeStartReq() or ApiMac_startFH(), if you're using FH.

12.3.10.48.1 Returns:

The status of the request, as follows:

ApiMac_status_success
- Operation successful
- ApiMac_status_unsupported

- feature not available.

*12.3.10.49 uint8_t ApiMac_convertCapabilityInfo (ApiMac_capabilityInfo_t * pMsgcapInfo)*

Convert ApiMac_capabilityInfo_t data type to uint8 capInfo.

12.3.10.49.1 Parameters:

pMsgcapInfo	- CapabilityInfo pointer

12.3.10.49.2 Returns:

capInfo bit mask byte

*12.3.10.50 void ApiMac_buildMsgCapInfo (uint8_t cInfo, ApiMac_capabilityInfo_t * pPBcapInfo)*

Convert from bitmask byte to API MAC capInfo.

12.3.10.50.1 Parameters:

cInfo	- source
pPBcapInfo	- destination

*12.3.10.51 ApiMac_status_t ApiMac_secAddDevice (ApiMac_secAddDevice_t * pAddDevice)*

Adds a new MAC device table entry.

12.3.10.51.1 Parameters:

pAddDevice	- Add device information

12.3.10.51.2 Returns:

ApiMac_status_success if successful, other status value if not.

*12.3.10.52 ApiMac_status_t ApiMac_secDeleteDevice (ApiMac_sAddrExt_t * pExtAddr)*

Removes MAC device table entries.

12.3.10.52.1 Parameters:

pExtAddr	- extended address of the device table entries that shall be removed

12.3.10.52.2 Returns:

ApiMac_status_success if successful, other status value if not.

12.3.10.53 ApiMac_status_t ApiMac_secDeleteKeyAndAssocDevices (uint8_t **keyIndex***)*

Removes the key at the specified key Index and removes all MAC device table enteries associated with this key. Also removes(initializes) the key lookup list associated with this key.

12.3.10.53.1 Parameters:

keyIndex	- mac secuirty key table index of the key to be removed.

12.3.10.53.2 Returns:

ApiMac_status_success if successful, other status value if not.

12.3.10.54 ApiMac_status_t ApiMac_secDeleteAllDevices (void)

Removes all MAC device table entries.

12.3.10.54.1 Returns:

ApiMac_status_success if successful, other status value if not.

12.3.10.55 ApiMac_status_t ApiMac_secGetDefaultSourceKey (uint8_t **keyId***, uint32_t **

pFrameCounter*)*

Reads the frame counter value associated with a MAC security key indexed by the designated key identifier and the default key source.

12.3.10.55.1 Parameters:

keyId	- Key ID.
pFrameCounter	- pointer to a buffer to store the outgoing frame counter of the key.

12.3.10.55.2 Returns:

ApiMac_status_success if successful, other status value if not.

*12.3.10.56 ApiMac_status_t ApiMac_secAddKeyInitFrameCounter (ApiMac_secAddKeyInitFrameCounter_t * pInfo)*

Adds the MAC security key, adds the associated lookup list for the key, initializes the frame counter to the value provided. It also duplicates the device table enteries (associated with the previous key if any) if available based on the flag dupDevFlag value and associates the device descriptor with this key.

12.3.10.56.1 Parameters:

pInfo	- structure need to perform this function.

12.3.10.56.2 Returns:

ApiMac_status_success if successful, other status value if not.

13 Gateway And Collector Application Interface API

The purpose of this section is to provide a description of the application programming interface between the TI-15.4 Stack Linux collector example application and the gateway application. The Collector example application implements a appsrv module which opens up a server socket to which a client application can connect. The interface allows management and data interface to the client application connecting to the socket server for the TI-15.4 Stack based network. Management functionalities include ability to open/close network for new device joins, etc whereas the data interface allows sending and receiving data to and from the network devices. It should be very easy to add new API's or modify the current implementation. TI-15.4 Stack out of box example application gateway and collector demonstrate the use of these API's.

This API is defined at a specific interface level, which is a TCP socket pipe.

API commands and parameters are serialized over the socket interface using Google protocol buffers (protobuf; for details see https://developers.google.com/protocol-buffers/). Please note that the name of some parameters in this API section does not reflect the true name of the parameter which will be used by the application using that API, as those parameters are automatically generated by the protobuf engine).

For transport using a TCP socket, the protobuf-packed packets are preceded by a 4 bytes header, containing the following fields (in this order):
- **len** – 16bit number that specifies the actual length (in bytes) of the protobuf-packed packet.
- **Subsystem** – 1 byte - specifies the subsystem to/from which the packet is sent/received. It can be either:
    - **10** – TI-15.4 Stack App server interface
- **cmd_id** – 1 byte - The command ID of the actual command being sent. This value is also available inside the packed packet. The actual command ID numbers are provided in the protobuf definition files that are part of the TI-15.4 Stack Linux SDK (Collector example application and the gateway example application). When using command IDs in your code, always use the defined names (never hardcode the command ID numbers), as the numbers may change between releases.

Management Interface

13.1 APPSRV_SET_JOIN_PERMIT_REQ

13.1.1 Description:

Allows client application to enable or disable network for join for new devices.

13.1.2 Parameter List

Parameter	Type	Description
Duration	INT32	duration - duration for join permit to be turned on in milliseconds. • 0 sets it Off, 0xFFFFFFFF sets it ON indefinitely • Any other non-zero value sets it on for that duration

13.2 APPSRV_SET_JOIN_PERMIT_CNF

13.2.1 Description

Application server notifies the client of the result of processing of permit join request message.

13.2.2 Parameter

Parameter	Type	Description
Status	INT32	0 if Success

13.3 APPSRV_NWK_INFO_IND

13.3.1 Description:

Application server notifies the client of the network information when a network is formed using this API.

13.3.2 Parameter

Parameter	Type	Description
Fh	UINT32	true if network is frequency hopping
channel	UINT32	Channel Number used, if non-frequency hopping network configuration
panID	UINT32	The 16-bit PAN identifier of the network
shortAddress	UINT32	The 16-bit short address of the pan-coordinator
extAddress	INT64	The 64-bit IEEE extended address of the pan-coordinator device
securityEnabled	INT32	'true' if security enabled, 'false' otherwise
nwkMode	ENUM	Network operation mode BEACON_ENABLED = 1 NON_BEACON = 2 FREQUENCY_HOPPING = 3

state	ENUM	Pan-coordinator state values. **State** / **Value** **Initialized waiting for user to start** – 1 **Starting Coordinator** – 2 **Restoring Coordinator (from NV)** – 3 **Started** – 4 **Restored** – 5 **Joining Allowed for new devices** – 6 **Joining not allowed for new devices** – 7

13.4 APPSRV_GET_NWK_INFO_REQ

13.4.1 Description:

Application server's Client can use this API to get the current network information

13.4.2 Parameter:

There is no parameter in the command message.

13.5 APPSRV_GET_NWK_INFO_CNF

13.5.1 Description

Application server sends the current network information as a response to the get network information request from the client using this API.

13.5.2 Parameter

Parameter	Type	Description
Status	INT32	0 if Success
Fh	UINT32	(optional) true if network is frequency hopping
channel	UINT32	(optional) Channel Number used, if non-frequency hopping network configuration
panID	UINT32	(optional) The 16-bit PAN identifier of the network
shortAddress	UINT32	(optional) The 16-bit short address of the pan-coordinator
extAddress	INT64	(optional) The 64-bit IEEE extended address of the

<table>
<tr><td></td><td></td><td>pan-coordinator device</td></tr>
<tr><td>securityEnabled</td><td>INT32</td><td>(optional) ‘true’ if security enabled, ‘false’ otherwise</td></tr>
<tr><td>nwkMode</td><td>ENUM</td><td>(optional) Network operation mode
BEACON_ENABLED = 1
NON_BEACON = 2
FREQUENCY_HOPPING = 3</td></tr>
<tr><td>state</td><td>ENUM</td><td>(optional) Pan-coordinator state values.
<table>
<tr><th>State</th><th>Value</th></tr>
<tr><td>Initialized waiting for user to start</td><td>1</td></tr>
<tr><td>Starting Coordinator</td><td>2</td></tr>
<tr><td>Restoring Coordinator (from NV)</td><td>3</td></tr>
<tr><td>Started</td><td>4</td></tr>
<tr><td>Restored</td><td>5</td></tr>
<tr><td>Joining Allowed for new devices</td><td>6</td></tr>
<tr><td>Joining not allowed for new devices</td><td>7</td></tr>
</table></td></tr>
</table>

13.6 APPSRV_GET_DEVICE_ARRAY_REQ

13.6.1 Description

Application client requests the current list of connected device using this API

13.6.2 Parameter:

There is no parameter in the command message.

13.7 APPSRV_GET_DEVICE_ARRAY_CNF

13.7.1 Description

Application server sends the current list of connected device as a response to the get device array request message using this API.

13.7.2 Parameter:

Parameter	Type	Description
Status	UINT32	0 if Success
devInfo	Csf_deviceInformation	Multiple entries of this structure element. Number of entries is equal to the number of connected devices in the network.

devInfo (Csf_deviceInformation):

Parameter	Type	Description
panID	UINT32	The 16-bit PAN identifier of the network
shortAddress	UINT32	The 16-bit short address of the network device
extAddress	INT64	The 64-bit IEEE extended address of the network device
panCoord	UINT32	True if the device is pan-coordinator
ffd	UINT32	True if the device is a full function device
mainsPower	UINT32	True if the device is mains powered
rxOnWhenIdle	UINT32	True if the device's RX is on when the device is idle
security	UINT32	True if the device is capable of sending and receiving secured frames
allocAddr	UINT32	True if allocation of a short address in the associate procedure is needed.

13.8 APPSRV_DEVICE_JOINED_IND

13.8.1 Description

Application server informs the client of a new device join in the network using this API.

13.8.2 Parameter:

Parameter	Type	Description
panID	UINT32	The 16-bit PAN identifier of the network
shortAddress	UINT32	The 16-bit short address of the network device
extAddress	INT64	The 64-bit IEEE extended address of the network device
panCoord	UINT32	True if the device is pan-coordinator
ffd	UINT32	True if the device is a full function device
mainsPower	UINT32	True if the device is mains powered
rxOnWhenIdle	UINT32	True if the device's RX is on when the device is idle
security	UINT32	True if the device is capable of sending and receiving secured frames
allocAddr	UINT32	True if allocation of a short address in the associate procedure is needed.

13.9 APPSRV_DEVICE_NOTACTIVE_UPDATE_IND

13.9.1 Description

Application server informs the client of an inactive device using this API.

13.9.2 Parameters

Parameter	Type	Description
panID	UINT32	The 16-bit PAN identifier of the network
shortAddress	UINT32	The 16-bit short address of the network device
extAddress	INT64	The 64-bit IEEE extended address of the network device
timeout	UINT32	true if not active because of tracking timeout. meaning that the device didn't respond to the tracking request within the timeout period.

13.10 APPSRV_COLLECTOR_STATE_CNG_IND

13.10.1 Description

Application server informs the client of change in the state of the collector application using this API.

Parameter	Type	Description
state	UINT32	**State** / **Value** **Initialized waiting for user to start** / 1 **Starting Coordinator** / 2 **Restoring Coordinator (from NV)** / 3 **Started** / 4 **Restored** / 5 **Joining Allowed for new devices** / 6 **Joining not allowed for new devices** / 7

Data Interface:

13.11 APPSRV_DEVICE_DATA_RX_IND

13.11.1 Description

Application server informs the client of receipt of sensor data from a network device using this API.

13.11.2 Parameters

Parameter	Type	Description
srcAddr	UINT32	The 16-bit PAN identifier of the network
Rssi	SINT32	RSSI of the message received
sDataMsg	Smsgs_sensorMsg	(optional) Received sensor message
sConfigMsg	Smsgs_configRspMsg	(optional) Received config response message

Smsgs_sensorMsg

<table>
<tr><th>Parameter</th><th>Type</th><th>Description</th></tr>
<tr><td>cmdId</td><td>ENUM</td><td>Sensor message command id
<table>
<tr><th>Command ID</th><th>Description</th><th>Value</th></tr>
<tr><td>Smsgs_cmdIds_configReq</td><td>Configuration message, sent from the collector to the sensor</td><td>1</td></tr>
<tr><td>Smsgs_cmdIds_configRsp</td><td>Configuration Response message, sent from the sensor to the collector</td><td>2</td></tr>
<tr><td>Smsgs_cmdIds_trackingReq</td><td>Tracking request message, sent from the the collector to the sensor</td><td>3</td></tr>
<tr><td>Smsgs_cmdIds_trackingRsp</td><td>Tracking response message, sent from the sensor to the collector</td><td>4</td></tr>
<tr><td>Smsgs_cmdIds_sensorData</td><td>Sensor data message, sent from the sensor to the collector</td><td>5</td></tr>
<tr><td>Smsgs_cmdIds_toggleLedReq</td><td>Toggle LED message, sent from the collector to the sensor</td><td>6</td></tr>
<tr><td>Smsgs_cmdIds_toggleLedRsp</td><td>Smsgs_cmdIds_toggleLedRsp</td><td>7</td></tr>
</table></td></tr>
<tr><td>Framecontrol</td><td>UINT32</td><td>Frame Control field states what data fields are included in reported sensor data, each value is a bit mask value so that they can be combined (OR'd together) in a control field.</td></tr>
</table>

<table>
<tr><td></td><td></td><td>When sent over-the-air in a message this field is 2 bytes.

<table>
<tr><th>Parameter</th><th>Description</th><th>Value</th></tr>
<tr><td>Smsgs_dataFields_tempSensor</td><td>Bit mask for temperature sensor</td><td>0x00 01</td></tr>
<tr><td>Smsgs_dataFields_lightSensor</td><td>Bit mask for light sensor</td><td>0x00 02</td></tr>
<tr><td>Smsgs_dataFields_humiditySensor</td><td>Bit mask for humidity sensor</td><td>0x00 04</td></tr>
<tr><td>Smsgs_dataFields_msgStats</td><td>Bit mask for stats message</td><td>0x00 08</td></tr>
<tr><td>Smsgs_dataFields_configSettings</td><td>Bit mask for configuration settings</td><td>0x00 10</td></tr>
<tr><td>Smsgs_dataFields_toggleSettings</td><td>Bit mask for toggle settings</td><td>0x00 20</td></tr>
</table></td></tr>
<tr><td>tempSensor</td><td>Smsgs_tempSensorField</td><td>(optional) Lists the reported temperature sensor data.

Smsgs_tempSensorField:
<table>
<tr><th>Parameter</th><th>Type</th><th>Description</th></tr>
<tr><td>ambienceTemp</td><td>UINT32</td><td>Ambience Chip Temperature - each value represents a 0.01 C degree, so a value of 2475 represents 24.75 C.</td></tr>
<tr><td>objectTemp</td><td>UINT32</td><td>Object Temperature - each value represents a 0.01 C degree, so a value of 2475 represents 24.75 C.</td></tr>
</table></td></tr>
<tr><td>lightSensor</td><td>Smsgs_lightSensorField</td><td>(optional) Lists the reported light sensor data

Smsgs_lightSensorField

<table>
<tr><th>Parameter</th><th>Type</th><th>Description</th></tr>
</table></td></tr>
</table>

<table>
<tr><td></td><td></td><td>
<table>
<tr><td>rawData</td><td>UINT32</td><td>Raw Sensor Data read out of the OPT2001 light sensor</td></tr>
</table>
</td></tr>
<tr><td>humiditySensor</td><td>Smsgs_humiditySensorField</td><td>(optional) Lists the reported humidity sensor data

Smsgs_humiditySensorField
<table>
<tr><th>Parameter</th><th>Type</th><th>Description</th></tr>
<tr><td>temp</td><td>UINT32</td><td>Raw Temp Sensor Data</td></tr>
<tr><td>humidity</td><td>UINT32</td><td>Raw Humidity Sensor Data</td></tr>
</table>
</td></tr>
<tr><td>configSettings</td><td>Smsgs_configSettingsField</td><td>(optional) Lists the reported configuration settings

Smsgs_configSettingsField:
<table>
<tr><th>Parameter</th><th>Type</th><th>Description</th></tr>
<tr><td>reportingInterval</td><td>UINT32</td><td>Reporting Interval - in millseconds, how often to report sensor data to the pan-coordinator, 0 means reporting is off</td></tr>
<tr><td>pollingInterval</td><td>UINT32</td><td>Polling Interval - in millseconds (32 bits) - If the sensor device is a sleep device, this states how often the device polls its parent for data. This field is 0 if the device doesn't sleep.</td></tr>
</table>
</td></tr>
</table>

Smsgs_configRspMsg

Parameter	Type	Description
cmdId	Smsgs_cmdIds	Sensor message command id
Status	Smsgs_statusValues	Status of the processing of the request message
Framecontrol	Smsgs_dataFields	bit mask of Smsgs_dataFields
treportingInterval	UINT32	Sensor data reporting interval
pollingInterval	UINT32	Polling interval if the device is a sleepy device

13.12 APPSRV_TX_DATA_REQ

13.12.1 Description

Application client uses this to send data to a network device.

13.12.2 Parameters

<table>
<tr><th>Parameter</th><th>Type</th><th>Description</th></tr>
<tr><td>msgId</td><td>Smsgs_cmdIds</td><td>Sensor message command id</td></tr>
<tr><td>panID</td><td>UINT32</td><td>The 16-bit PAN identifier of the network</td></tr>
<tr><td>shortAddress</td><td>UINT32</td><td>The 16-bit short address of the network device</td></tr>
<tr><td>extAddress</td><td>INT64</td><td>The 64-bit IEEE extended address of the network device</td></tr>
<tr><td>configReqMsg</td><td>Smsgs_configReqMsg</td><td>(optional) configuration request message parameters
<table>
<tr><th>Parameter</th><th>Type</th><th>Description</th></tr>
<tr><td>cmdId</td><td>Smsgs_cmdIds</td><td>The value will be ‘Smsgs_cmdIds_configReq’ (= 1).</td></tr>
<tr><td>frameControl</td><td>UINT32</td><td>Frame Control field states what data fields are included in reported sensor data, each value is a bit mask value so that they can be combined (OR'd together) in a control field. When sent over-the-air in a message this field is 2 bytes.</td></tr>
<tr><td>reportingInterval</td><td>UINT32</td><td>Sensor data reporting interval</td></tr>
<tr><td>pollingInterval</td><td>UINT32</td><td>Polling interval if the device is a sleepy device</td></tr>
</table></td></tr>
<tr><td>toggleLedReq</td><td>Smsgs_toggleLedReqMsg</td><td>(optional) toggle led request message parameters</td></tr>
</table>

13.13 APPSRV_TX_DATA_CNF

13.13.1 Description

Application server informs the client of result of the transmit data request

13.13.2 Parameter

Parameter	Type	Description
Status	INT32	0 if Success