



## Programmer's Guide

## **About this Manual**

This manual aims at familiarizing the SENSEnuts platform user with programming methodologies to be followed to develop the applications effectively. The entire document is divided into six sections. Though the sections can be studied in any order as per the requirement, but it is advised to refer this documentation in the same order as presented.

Section 1 aims at introducing the SENSEnuts platform to the application developer. It gives an introduction about the platform. It highlights most of the features of the platform and some technical details as well

Section 2 explains the file organization of SENSEnuts platform. It explains the directory structure and the files and code placement in the directories for successful compilation of the application written by the user.

Section 3 concentrates entirely on application development and explains the code structure of a SENSEnuts application. This section is important for those aiming at developing the applications from scratch without modifying the existing examples.

Section 4 explains the algorithms which have been implemented and provided to the user as a part of SENSEnuts platform by Eigen Technologies. User can implement their own algorithms by studying the included ones and modifying the source codes which have been provided along with the setup.

Section 5 is intended to discuss the IEEE 802.15.4 implementation in brief. This section is useful to those who are beginners in this field and want to explore the same

Section 6 provides the coding guidelines for developing applications. It explains the Dos and DONTs that should be followed wherever possible for optimized use of memory and battery.

## Revision History

Version	Date	Comments
6.1.2	09/19/2015	First Release

## **Contents**

About this manual	2
Section 1: Introduction	5
1.1 Supported Platforms	5
1.2 Third Party Tools used	5
Section 2: File Organization	6
2.1 AppCode	6
2.2 Documentation	6
2.3 Include	7
2.4 lib	7
2.5 libraries	7
2.6 main	7
2.7 tools	7
Section 3: Code Development and Architecture	8
3.1 Flow Diagram	8
3.2 Tasks	8
3.3 Application Code Structure	8
3.4 Routing	11
Section 4: Algorithms	12
4.1 MAC Bases Routing	12
4.2 Level Based Routing	12
4.3 AODV	13
Section 5: IEEE 802.15.4 MAC Implementation	14
Section 6: Coding Guidelines	15

## Section 1: Introduction

**S**ENSEnuts takes you into the real world scenario where algorithms can be tested on real devices. User can write his/her own algorithms and program the real motes according to the algorithm. The assumptions been made in the simulation environment are automatically overruled as the transfer of data between the devices takes place through the actual medium, i.e. air with all available interferences.

It allows a user to work on IEEE 802.15.4 and harness the features specified in the document for the same. With these many features, a user can work on various research topics and contribute to the research industry.

Besides research, SENSEnuts is very well suitable for applications as well. Wide range of sensors can be connected with the motes depending upon the requirement of the application. Environment monitoring, home automation, agriculture monitoring, industrial control, location tracing are a few to name.

### HIGHLIGHTS

- Developed by Eigen Technologies for WSN research
- Flexible MAC and PHY layer
- Programming in C language
- APIs to perform different tasks
- Easy to code routing algorithms and write applications
- Add custom sensors

### 1.1 Supported Platforms

SENSEnuts development platform is supported on most of the windows platform but it has been tested on the below mentioned and must be used only on these platforms.

- i. Windows XP
- ii. Windows 7
- iii. Windows 8

### 1.2 Third Party Tools used

Though most of the APIs provided to the user are a part of SENSEnuts platform, but certain third party tools have been integrated with the same for ease of programming and accelerating development process. The third party tools used are JRE, Cygwin, Eclipse IDE, Jennic compiler toolchain, and FTDI hardware drivers. It must be noted that for JRE, the PATH environment variable has to be set for the system to function properly.

## **Section 2: File Organization**


SENSEnuds platform has a well-organized directory structure. Each directory is dedicated to meet some specific functionality. The purpose of each directory is explained in this section.

### **2.1 AppCode**

This directory contains the demo examples which comes along with the SENSEnuds toolchain setup. All the application related code, whether demo or the ones created by the user must reside in this directory.

The setup comes with five examples for user's reference. SENSEnuds team motivates the users to use these examples to understand the coding structure and modify them to create their own applications. The functionality of the examples are explained as under:

- i. Demo 1:- blinking the LED on the Radio Module
- ii. Demo 2:- read the light and temperature data from the TL module connected to the computer directly with the help of USB gateway
- iii. Demo 3:- broadcast the data in the network from a coordinator with PAN coordinator as the destination address
- iv. Demo 4:- MBR, MAC Based Routing with multi-hop scenario where data is sent to the PAN coordinator. The algorithm of MBR is explained in upcoming sections
- v. Demo 5:- example to demonstrate the use of LBR, Level Based Routing
- vi. Demo 6:- example implementation of AODV routing protocol
- vii. Demo 7:- example to demonstrate the implementation of a critical task



For information on how to create a project from scratch, please refer section 6 of quick start guide

### **2.2 Documentation**

All the documentation and help for SENSEnuds development platform is stored in this directory. Moreover, all the new documents in future releases of toolchain will also be present in the same directory. The documents available in this directory are as follow:

- i. API reference guide:- this guide is a resource for an application developer. All the APIs available in SENSEnuds platform are explained in this guide.
- ii. Installation guide:- this guide explains the installation steps of SENSEnuds toolchain.
- iii. Programmer's guide:- this guide explains the entire structure of SENSEnuds platform and includes the information on programming methodologies to be followed to write your own applications
- iv. Quick Start guide:- this is the first guide that any user must read. It provides the entire information needed to start working on SENSEnuds platform, from installation of toolchain to creation of a new project in Eclipse IDE.

- v. SENSEnuts GUI user guide:- this guide provides information on the usage of SENSEnuts GUI to program the motes and receive data from the network.

## 2.3 Include

All the header files for the source codes are available in this directory. The header files also specify the usage of APIs with ample amount of commenting. It is advised that along with studying the API reference guide, a user must have a look at the corresponding header file mentioned in the API reference guide itself.

## 2.4 lib

Kernel specific files are present in this directory. A user should not modify these files unless he/she knows what exactly he/she is doing. Modifying any file present in this directory may result in a corrupt development environment.

## 2.5 libraries

All the SENSEnuts code resides in this directory. The difference between libraries and “AppCode” is that “AppCode” contains application related code, but libraries contains the definition of APIs used to develop an application code. To clear the difference between the two, let us take an example of light sensor. The API used to read the light intensity is `readLux()`. The definition of `readLux()` is given in “lightSensor.c” which is present in libraries directory, whereas the call `readLux()` is used in source code which resides in the “source” directory inside “AppCode”.

### Guideline:

Application specific code must reside in “AppCode” directory and system specific source code in “libraries”

A new source file can be added in either of the two directories, namely, AppCode and “libraries”. But it is advised to add application specific source code files in the “source” folder inside “AppCode” directory.

## 2.6 main

The main source file “main.c” resides in this directory. The function of main.c is to start the node by initializing the clock, dio, scheduler and calling the `startNode()` function, where a user writes a first task to be executed in the source file of the application.

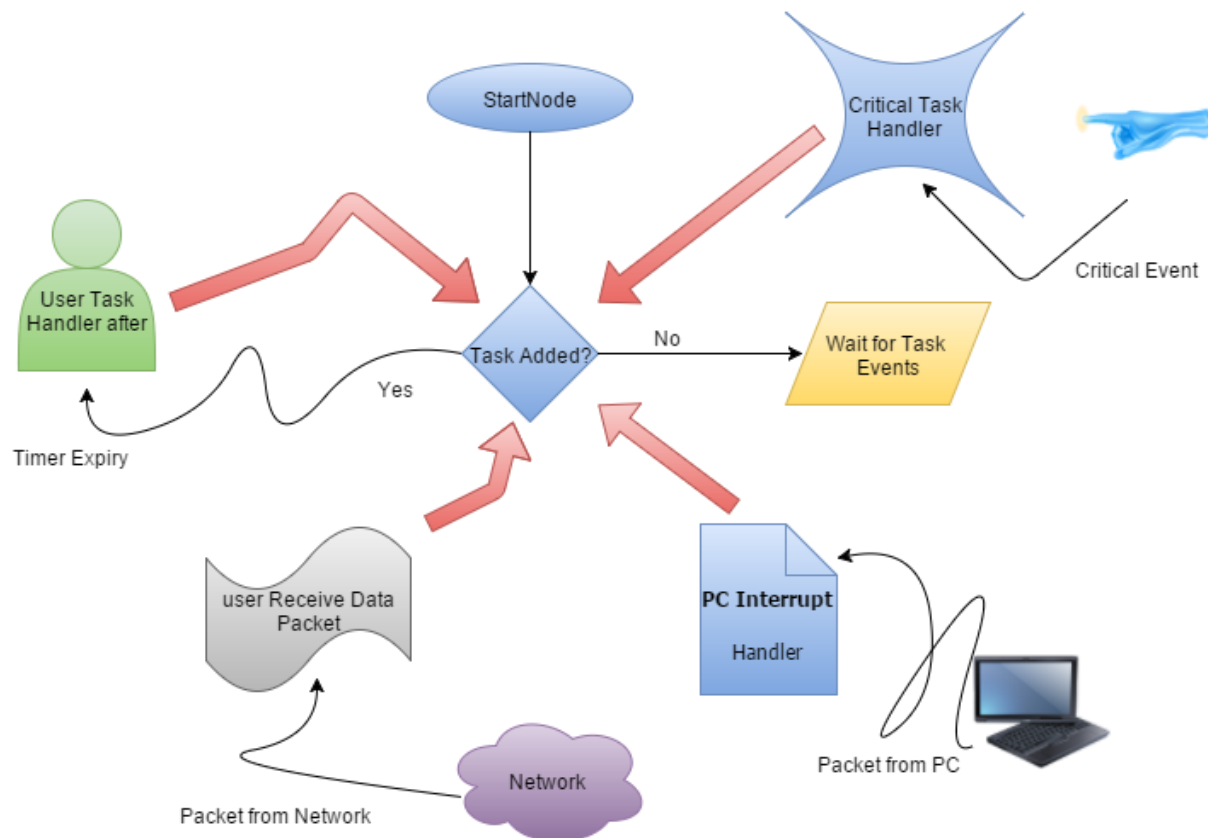
## 2.7 tools

All third party tools and SENSEnuts GUI is present in this directory. It is strongly recommended to read the license agreement of these tools before you try to make any change in them. The license agreement is displayed at the time of installation of the SENSEnuts toolchain.

## Section 3: Code Development and Architecture

This section presents the information to write an application for SENSEnuts platform. It is advised to install the toolchain before reading this section. It is also advised to open an example in Eclipse IDE and study all the examples sided by side to have a better understanding of this section.

### 3.1 Flow Diagram



### 3.2 Tasks

Any operation to be performed by the processor is considered to be a task. It can be turning on a LED, sending a packet to the network or take a reading from the sensors. Tasks are added to the queues and are executed once the timer for the same expires. Adding tasks into queues is simple and is added using `addTask()` API. Refer to API reference guide for information on this API.

Tasks are handled by handlers. The handler which serves the task depends upon the first argument of `addTask()`. If the argument is `USER`, user task handler is the one which handles the task. If the argument is `ROUTING`, then routing task handler is being called after the expiry of the timer and it executes the code specified in the handler.

### 3.3 Application Code Structure

The entire application code is divided among five functions which **must** be present in any source file of the application (even if they have to be blank). The functions are listed in this section.



i. **startNode**

This is the first function which is called when the device is turned on (boot time entry). All the hardware peripherals and software components are to be initialized here. The components that can be initialized here are sensors, MAC interface, routing (works only if MAC interface is initialized), gateway communication, etc. User can also specify tasks to be performed at a specific time, for example, sending the packet to the network every second using the API `addTask()`.

```
void startNode()
{
    //initialize sensors
    tmpInit();
    lightSensorInit();

    //initialize mac layer
    macInit();

    /*set task to read sensor info and send the data to mac. If
    the coordinator has already associated, it will broadcast
    this packet or else it will drop the packet.*/

    addTask(USER, SEND_PACKET_TO_MAC, 1*SECOND);
}
```

The above code snippet initializes the temperature sensor, light sensor and the wireless interface (using `macInit`). Post these steps, a task is added which is “USER” defined and task type is “SEND\_PACKET\_TO\_MAC” which should be defined previously (for example using `#define` preprocessor).

The above code snippet is taken from the source code for a coordinator. To check the above code in detail, refer to the source code of `bcast_Coord` in demo 3 included in the `AppCode` directory.

ii. **userTaskHandler**

This function gets called whenever the timer for a user specific task expires. Based on the type of task, user can execute a specific code in `userTaskHandler`. Refer to the code snippet below:

```
void userTaskHandler(uint8 taskType)
{
    switch (taskType)
    {
        case LED_ON:
            ledOn();
            addTask(USER, LED_OFF, 1*SECOND);
            break;
        case LED_OFF:
            ledOff();
            addTask(USER, LED_ON, 1*SECOND);
            break;
    }
}
```

```
}
```

`TaskType` is added using the second argument of `addTask()`. A user can add multiple tasks using `addTask()` API and specify the time using the third argument of `addTask()`. Operating system sets a timer as specified using `addTask()` API and it calls `userTaskHandler()`

#### iii. `userReceiveDataPacket`

If MAC layer is initialized and if the node receives a data packet through wireless interface, this function is called. First byte of any packet received by the mote helps the MAC layer to decide if the received packet is a data packet or a routing packet. If the value of first byte is greater than 0x23, then the packet will be considered as data packet. So if the user wants to send any data packet wirelessly, the first byte of the packet must be greater than 0x23. Refer demo 3 for an understanding of the use of the function.

```
void userReceiveDataPacket(uint8* payload,uint8 length,uint16
                           prevAddr,uint8 linkQuality)
{
    uint8 tmp;
    uint16 light;

    if (payload[0]==USER_PACKET_TYPE)
    {
        tmp=payload[1];
        light=payload[2];
        light=light<<8;
        light=light | payload[3];

        //update info in database present in PC
        updateAmbientdb(prevAddr,light,tmp);
    }
}
```

#### iv. `critTaskHandler`

This handler gets called whenever a DIO causes an interrupt to the processor. In the current release, the DIO interrupt for temperature and light sensor is present. Whenever the temperature sensor crosses a threshold value set by the user, it causes an interrupt on DIO and this function then gets called and executed. Same is true for light sensor. Highest priority is given to the DIOs on which interrupts from light and temperature sensors takes place. Whenever there is a critical interrupt, the node will handle it once it finishes the current task in process, giving it a priority over the other tasks already in line to be executed. Refer to the code snippet below which highlights the critical task from the light sensor to be handled.

```

void userCriticalTaskHandler(uint8 critTaskType)
{
    if (critTaskType==LIGHT_CRIT_TASK)
    {
        uint8 tmp;
        uint16 light;

        //read sensor values
        tmp=readTmp();
        light=readLux();

        //update info in database present in PC
        updateAmbientdb(getNodeId(),light,tmp);

        clearLightInterrupt();
    }
}

```

v. **pcInterruptHandler**

This function is called whenever the Radio Module connected with a system (for example a Laptop) receives data from it. The data packet arriving from the PC can be read using the API `readByteFromPc()`. This function is executed in interrupt context, and hence, it should have as few instructions as possible. Using of loops (such as `while`) should be strictly avoided for the proper functioning of operating system. A wise programming strategy would be to add a task to be executed after a particular interval using `addTask()` API.

### 3.4 Routing

Users can develop their own routing protocols in SENSEnuts platform. Data packet can be sent directly to the MAC layer if there is only one hop or a routing protocol is not needed (for example, a static path). Routing protocol will be needed when the path is not static and the scenario is a multi-hop scenario. The routing protocols supported as of now by SENSEnuts platform are MBR, LBR and AODV.

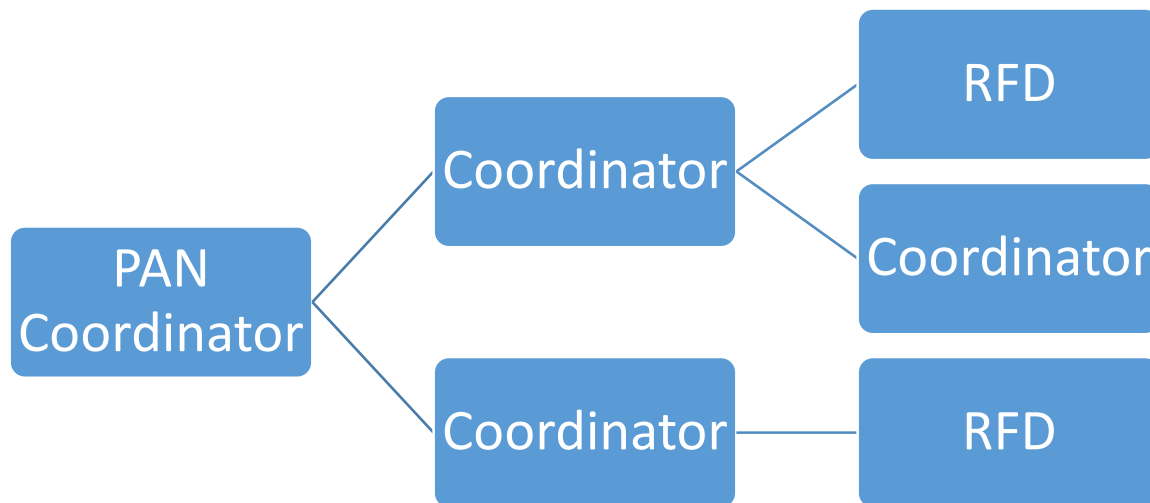
There are five APIs for routing. If a user wants to add a new routing protocol, the first four functions must be modified in such a way that when any of the functions are called by application layer, MAC layer or a task, equivalent functions of newly added protocols are called.

## Section 4: Algorithms

This section explains the working of the algorithms implemented in SENSEnuts WSN platform. This must be noticed that a user can implement his/her own custom routing protocols. The source code for the existing protocols can be modified to implement custom algorithms. The protocols included in SENSEnuts platform are explained in the upcoming sections.

### 4.1 MAC Based Routing

MBR or MAC Based Routing can be used when the destination is always the PAN Coordinator. As per IEEE 802.15.4, during network setup, each node gets associated with a coordinator. In MBR, routing is done in such a way that every node sends the packet to the coordinator to which it is associated. This process keeps on repeating till the time the packet is received by the destination node, i.e. PAN Coordinator. If the packet gets dropped, it generates an exception and the node resets itself.



Flow of MBR from right to left

### 4.2 Level Based Routing

LBR or Level Based Routing is used in multihop scenario and can be used when destination is PAN Coordinator. This algorithm can be modified to work with any other node acting as a sink.

The protocol initiates with an assumption that the MAC layer will take some time to setup throughout the network. The routing process initiates when the timer expires for which the task is added to be executed after the time assumed for the setup. When the timer expires, PAN Coordinator sends a packet in the network with its level (hop) set to '0'. All the nodes which are

in range of PAN coordinator receives the packet and set their level to '1' after which, they rebroadcast the packet. The next hop now may receive the packets from multiple nodes in level '1'. Each node saves the information of four nodes (at max) and set their level as '2'. They do a rebroadcast only for the first packet received from level '1'. This is how each node gets a level and saves the information about the nodes in previous level.

Now to send a data packet to the previous level, the node follows the Round Robin principle between the nodes in the previous level whose information is saved in its memory (which can be four at max). This may help in saving the battery and hence increase the network lifetime.

#### 4.3 AODV

For AODV implementation, refer RFC 3561. The limitation of the present implementation is mentioned in the source code of AODV.

## **Section 5: IEEE 802.15.4 MAC implementation**

According to IEEE 802.15.4, there are three configurations of a device possible. They are PAN coordinator, Coordinator and a Reduced Function Device (RFD). It must be noticed that in SENSEnuts, any mote can be configured in any of the three mentioned devices.

PAN Coordinator initiates the network. It chooses a channel on which the networks operates. Coordinators are the devices which can act as a router and other nodes can get associated with. RFD (Reduced Function Device) as the name suggests, are the devices with reduced capability. As already mentioned, in SENSEnuts platform RFD and FFD are equally capable in terms of hardware. The difference is in the software features available to them. RFD cannot act as a router and they can send the data only to the Coordinator they are associated with.

When MAC layer is initialized by calling `macInit()`, depending upon the information present in `config.h`, the node is programmed as a PAN Coordinator, Coordinator or a RFD. If the node is programmed as a PAN Coordinator, it performs an energy scan and selects the quietest of the available channels and starts a PAN over the same. If the node is set as a coordinator, the node performs active scan and selects most appropriate coordinator to get associated with based on the link quality.

MAC layer generates MLME (MAC Layer Management Entity) messages for network setup. The data packets generated from the higher layers are treated as MCPS (MAC Common Port Layer).

## **Section 6: Coding Guidelines**

SENSEnuts platform enables the user to program in C programming language. A user can make use of all the features that C language provides. Some coding methodologies suggested for developing applications for SENSEnuts devices are enlisted in this section.

- i. Do not use dynamic memory allocation
- ii. Try to keep the functions and tasks as short as possible with minimum possible statements.
- iii. Use LED on the radio module and debug messages to be printed on GUI in “Print Window” section for debugging purposes.
- iv. Implement an algorithm in a progressive manner. Divide it in modules and implement it stepwise. Test a smaller part and then implement the next module of the algorithm. This will reduce the development time.
- v. Do not forget to include the relevant header files in the source code of the application. Failing to ensure this will create errors while compilation.
- vi. Avoid use floating point variables. They increase the code size. If possible, perform the floating point operations on a PC after receiving data from the device on the system.

## Important Notice

**Limited warranty and liability** - Information in this document is believed to be accurate and reliable. However, Eigen Technologies does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. Eigen Technologies takes no responsibility for the content in this document if provided by an information source outside of Eigen Technologies.

In no event shall Eigen Technologies be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or work charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory. Notwithstanding any damages that customer might incur for any reason whatsoever, Eigen Technologies' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the *Terms and conditions of commercial sale* of Eigen Technologies.

**Right to make changes** – Eigen Technologies reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.