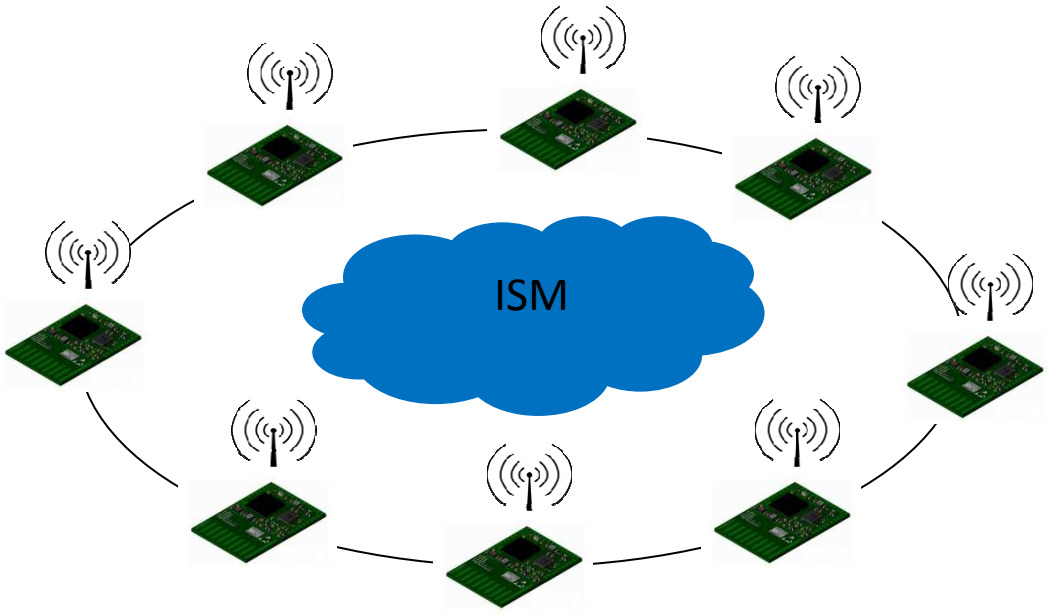
www.iotmesh.eu

IOTMesh_Library

User's Manual



Content table :

Introduction

IOTMesh_Library is a software library compiled to work with RFM3x boards. The RFM3x electronic boards are composed of ARM Cortex-M3 32bit microcontroller from ST Microelectronics (STM32) and a Texas Instruments sub GHz low power radio transceiver (CC1101). The library was compiled with GNU ARM toolchain (release 4.8 2013q4) and use ST Microelectronic Standard Peripheral Driver library (release STM32L1xx_StdPeriph_Lib_V1.2.0) to manage CPU peripherals (like GPIO, SPI, …).

IOTMesh_Library is built to drastically reduce your time to market. The library can be included in your own application to take in charge the transceiver driver part. The radio settings file can be easily generated thanks to SmartRF Studio[1] free software from Texas Instruments. To simplify your development, common settings are provided in software example.

This document will give you details about library API functions written in C.

The main API functions are:

- IOTMesh_Com_Init : for initialization
- IOTMesh_Com_Management : for background management
- IOTMesh_Com_Rx : to read received messages
- IOTMesh_Com_Tx : to send messages
- IOTMesh_Com_RX_Callback : to get notified each time a message is received
- IOTMesh_Timer_Isr : to manage library internal timings
- IOTMesh_Transceiver_Isr : to manage CC1101 interrupts
- IOTMesh_Address_Get : to get the module's address
- IOTMesh_Group_Get : to get the module's group
- IOTMesh_Group_Set : to join the module to a group
- IOTMesh_Cipher_Set : to change the cipher type
- IOTMesh_PassPhrase_Get : to get the current cipher key
- IOTMesh_PassPhrase_Set : to change the current cipher key
- IOTMesh_Key_Update : to change the cipher key of a remote node
- IOTMesh_Rssi_Get : to read current RSSI
- IOTMesh_Rx_Rssi_Lqi_Read : to read last received message RSSI and LQI
- IOTMesh_Rf_Power_Get : to read RF transmit power
- IOTMesh_Rf_Power_Set : to set RF transmit power
- IOTMesh_Rf_Channel_Get : to read RF channel
- IOTMesh_Rf_Channel_Set : to set RF channel
- IOTMesh_Temperature_Set : to set ambient temperature
- IOTMesh_Version_Get : to get the version of the IOTMesh library

[1] http://www.ti.com/tool/smartrftm-studio

- IOTMesh_QOS : to get QOS info for the current node
- IOTMesh_QOS_Remote : to get QOS info for a remote node
- IOTMesh_TTL_MAX_Set : to define the max TTL in the network
- IOTMesh_TTL_MAX_Get : to get the max TTL in this network
- IOTMesh_Netconf_Set : to define the network mode
- IOTMesh_Netconf_Get : to get the network mode
- IOTMesh_Sleep : to put the transceiver in sleeping mode
- IOTMesh_Wake : to wake transceiver
- IOTMesh_Status_Get : to get the status of transceiver (awake or sleeping)
- IOTMesh_Service_Register : to inform IOTMesh that the application offer a service
- IOTMesh_Service_Unregister : to inform IOTMesh that the application no longer offer a service
- IOTMesh_Service_Get_Service : to discover the IOTMesh node offering a service
- IOTMesh_Service_Get_All_Services : to discover all IOTMesh nodes offering a service

All of them are detailed in this document.

The library also includes a messages buffer in RAM memory to manage up to 15 messages from 1 byte to 64 bytes (receive and/or transmit) to be buffered (a 127 bytes message uses 2 buffer slots, up to a 255 bytes message that uses 4 buffer slots).

The package includes:

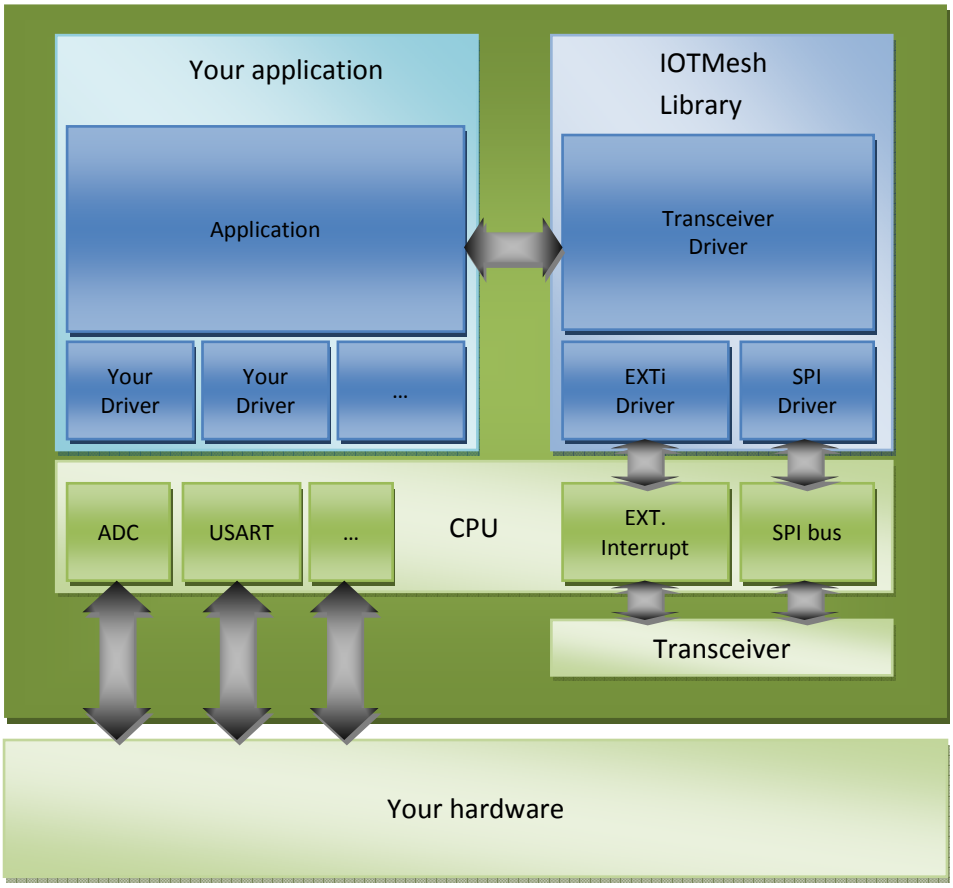
- IOTMesh binary library file
- IOTMesh.h header file that gives the API
- ST Microelectronic Standard Peripheral Driver minimum source code to be able to compile and link your application. The best way is to use the complete ST library freely available on ST web site[2].
- zipped example application

[2] http://www.st.com

Library description

The library is offered to reduce time to market of application based on IOTMesh electronic board.

The library handles the transceiver driver and makes the use of the RFM3x boards easier. The next diagram show how the library and your application are organized:



Library memory footprint

Library FLASH Footprint: **~48.2 kBytes (18.7kBytes for ST library and 29.5kBytes for stack)**

Library RAM Footprint: **~3.7 kBytes**

Library timing

Background timing (IOTMesh_Management function in standby)): between 150 and 180µs

Timer interrupt timing (IOTMesh_Timer_Isr): 7µs

External interrupt timing (IOTMesh_Transceiver_Isr handler): between 50 and 150µS

(All timings are considering CPU core frequency is 32MHz)

Transmission security

The library provides AES 128 and AES 256 ciphering. All transmitted messages are ciphered using one of these protocols. An application can set the cipher type and cipher key by calling the API *IOTMesh_Cipher_Set* and *IOTMesh_PassPhrase_Set.* If no cipher type is precised, transmitted data are ciphered using AES 128.

Low Power Functionality

The library provides low power functionalities. Consumption can be reduced up to 2µA in sleep mode. The library provides API to turn on and off the transceiver: *IOTMesh_Sleep* and *IOTMesh_Wake*. A low power example is available on the IOTMesh website[3].

API functions

IOTMesh_Com_Init

Description:

This function is used to initialize the library. It should be called only once in the main after CPU clock initialization.

Prototype:

void IOTMesh_Com_Init(void);

Input parameters :

void: no parameters.

Return parameter:

void: no output parameters.

IOTMesh_Com_Management

Description:

This function is the background process of the library. It should be called in the main loop to ensure a periodic processing.

Prototype:

void IOTMesh_Com_Management(void);

Input parameters:

void: no parameters.

[3] http://www.iotmesh.eu/index.php?route=information/download

Return parameter:

void : no parameters.

IOTMesh_Com_Rx

Description :

This function is used to read one received message if there is. It should be called each time the application has to process a message. The maximum user message size is 225 data bytes.

Prototype :

unsigned short IOTMesh_Com_Rx(t_IOTMesh_RX_MSG *msg, unsigned short dataFieldMax);

Input parameters :

pointer to t_IOTMesh_RX_MSG structured type variable : this structured variable includes four members:

1. datafield: This member is an unsigned char pointer that must be initialized to a memory segment reserved to receive the message.
2. ad_source: This member is an unsigned long pointer that must be initialized to a memory segment reserved to hold the sender address.
3. rssi: This member is an unsigned char that gives the rssi[4] of the received message
4. lqi: This member is an unsigned char that gives the lqi[5] of the received message

unsigned short : maximum number of bytes that could be copied at datafield pointed position.

Return parameter:

unsigned short : the length of received message. 0 if no message was received.

IOTMesh_Com_Tx

Description :

This function is used to transmit a message. It could be called anytime in the application. The transmitted message size depend on the route taken by the message. It can be max 225 bytes for direct communication (node A sends to node B and node B is in the range of node A, that mean the message makes 1 hop only). It decreases to 5 bytes max when the destination is 56 hops away. IOTMesh_Lib_EC search for optimal routes to rout messages. However, if a node A wants to send a 200 bytes message to node B, IOTMesh_Lib_EC will search for a route with a number of hops allowing to send the 200 bytes, if there is no route respecting this condition, IOTMesh_Lib_EC notify the application with the callback function in the passed t_IOTMesh_TX_MSG structure with the necessity to decompose this message.

[4] Received signal strength indication
[5] Link quality indication

Prototype :

unsigned short IOTMesh_Com_Tx(t_IOTMesh_TX_MSG *msg);

Input parameters :

pointer to t_IOTMesh_TX_MSG structured type variable : this structured variable includes six members:

1. ad_dest: This member is an unsigned long pointer that must be initialized to a memory segment where the destination's node address is stored.
2. ad_group: This member is an unsigned short that must be initialized to the destination's group address. It can take values from 0x0000 to 0x0fff. If *ad_group* is initialized to 0x0000,the message is addressed to *ad_dest* only. If *ad_group* is > 0x0000 then the message is addressed to all nodes within the group *ad_group*.
3. data_field: This member is an unsigned char pointer that must be initialized to a memory segment where the message to be transmitted is stored.
4. data_field_len : This member is an unsigned short that must be initialized to the number of bytes to be transmitted.
5. ack_required: this member is a short that tells the library whether to ask or not an acknowledge from the destination. The presence or absence of acknowledge is reported by the callback function. It can take:
    - 1: an acknowledge is required
    - 0: no acknowledge

6. callback: this member is a pointer to a callback function, it is used by the stack to notify the application that:
    - the message was acknowledged,
    - the message was not acknowledged,
    - the destination could not be reached or
    - that you should decompose the message because the route is too long to send these amount of data.

   This function takes one parameter and returns void. The parameter is a pointer to **t_IOTMesh_Callback_Params structured type**. This structured variable includes three members:

    - msg_id: is an unsigned short representing the ID of the message. (an ID is assigned to each sent message and is returned by the IOTMesh_Com_Tx function. see the Return parameter below.)
    - param_code: is an unsigned short representing the issue, it can take these values:
        - IOTMESH_CALLBACK_ACK : Acknowledge received
        - IOTMESH_CALLBACK_NACK : Acknowledge not received
        - IOTMESH_CALLBACK_DEST_UNREACHABLE : Destination unreachable
        - IOTMESH_CALLBACK_DECOMPOSE : Message too long to be routed to destination, it needs decomposition

        - IOTMESH_CALLBACK_NOMEMORY : not enough buffer slots available to hold the message
    - max_data: max number of bytes that can be routed to the destination. This member is applicable only when the param_code is evaluated to IOTMESH_CALLBACK_DECOMPOSE. So that application has to decompose his message to max max_data bytes

Return parameter:

unsigned short : gives the result of transmit request:

- 0 : sending failed, it is not possible to store the message.
- x : a unique ID assigned to the message by the library. The ID is assigned randomly. This same ID is reported by the callback function to report the reception or not of the acknowledge.

IOTMesh_Com_RX_Callback

Description :

This function should be implemented in the application. It is called by the library to notify the reception of a message.

Prototype :

unsigned short IOTMesh_Com_RX_Callback(t_IOTMesh_RX_MSG *msg, unsigned short data_size);

Input parameters :

pointer to t_IOTMesh_RX_MSG structured type variable : this structured variable includes five members:

1. datafield: This member is an unsigned char pointer initialized to a reserved memory segment containing the message.
2. ad_source: This member is an unsigned long pointer initialized to a reserved memory segment containing the sender's address.
3. rssi: This member is an unsigned char that gives the rssi of the received message
4. lqi: This member is an unsigned char that gives the lqi of the received message
5. destination: is an unsigned short representing the destination type of the message.It can take these values:
    - IOTMESH_DESTINATION_NODE : Message sent directly to the destination address
    - IOTMESH_DESTINATION_GROUP : Message sent to all nodes within a group
    - IOTMESH_DESTINATION_BROADCAST : Message sent in broadcast

unsigned short : number of bytes of the received message.

Return parameter:

unsigned short : this function should return 1 or 0:

- 1: means that the received message has been treated. A treated message will be discarded and will not be available through the function IOTMesh_Com_Rx
- 0: means that the received message has not been treated. It will remain retrievable by calling the function IOTMesh_Com_Rx

note: if this function is not implemented by the application, the application can retrieve received messages by calling the IOTMesh_Com_Rx function

IOTMesh_Timer_Isr

Description :

This function is used for timing management of IOTMesh library. It should be called every 1ms in timer interrupt handler for example.

Prototype :

```
void IOTMesh_Timer_Isr(void);
```

Input parameters :

void : no parameters.

Return parameter:

void : no parameters.

IOTMesh_Transceiver_Isr

Description :

This function process transceiver I/O interrupts. It must be called in External Interrupt 15 to 10 handler commonly called EXTI15_10_IRQHandler() (refer to ST Standard Peripheral Library for details). This function manages only EXTI10 and EXTI11 interrupts flags (flags are reset by the function). Other interrupt sources of the EXTI15_10_IRQHandler are not managed.

Prototype :

```
void IOTMesh_Transceiver_Isr(void);
```

Input parameters :

void : no parameters.

Return parameter:

void : no parameters.

IOTMesh_Address_Get

Description :
This function returns the local address.

Prototype :
unsigned long IOTMesh_Address_Get(void);

Input parameters :
void : no parameters.

Return parameter:
unsigned long : the module's address.

IOTMesh_Group_Get

Description :
This function returns the module's group address. A module receive and process messages addressed to its group.

Prototype :
unsigned short IOTMesh_Group_Get(void);

Input parameters :
void : no parameters.

Return parameter:
unsigned short : the module's group.

IOTMesh_Group_Set

Description :
This function joins the module to a group. This function should be called during initialization phase. It can also be called to switch from one group to another. Once joined to a group, a module can receive and process messages addressed to this group.

Prototype :
void IOTMesh_Group_Set(unsigned short);

Input parameters :
unsigned short : the module's group.

Return parameter:
void : no parameters.

IOTMesh_Cipher_Set

Description :
This function changes the cipher type of the transmitted data. This version supports two cipher types: AES 128 and AES 256. If no value is set, all transmitted data are ciphered using AES 128.

Prototype :
void IOTMesh_Cipher_Set(unsigned short);

Input parameters :
unsigned short : the cipher type. Accepted values are:

- 0: for AES 128
- 1: for AES 256

.

Return parameter:
void : no parameters.

IOTMesh_PassPhrase_Get

Description :
This function returns the cipher/decipher key. this key identifies the network of the node. A node can only process messages within his network.

Prototype :
void IOTMesh_PassPhrase_Get(unsigned char*);

Input parameters :
1. **unsigned char*** : the cipher key. It is an unsigned char pointer that must be initialized to a reserved memory segment of 32 bytes where the key will be stored. If the cipher type used is AES 128, only the first 16 bytes are used.

Return parameter:
void : no parameters.

IOTMesh_PassPhrase_Set

Description :

This function changes the cipher/decipher key. this key identifies the network of the node. A node can only process messages within his network. In general, this function should be called during initialization phase. But it can also be called later to switch from one network to another.

Prototype :

void IOTMesh_PassPhrase_Set(unsigned char*);

Input parameters :

2. **unsigned char*** : the cipher key. It is an unsigned char pointer that must be initialized to a memory segment of 32 bytes where the key is stored. If the cipher type used is AES 128, only the first 16 bytes are used.

Return parameter:

void : no parameters.

IOTMesh_Key_Update

Description :

This function changes the cipher/decipher key of a remote node. this key identifies the network of the node. A node can only process messages within his network.

Prototype :

void IOTMesh_PassPhrase_Set(unsigned char*, unsigned char*, unsigned long);

Input parameters :

1. **unsigned char*** : the old cipher key. It is an unsigned char pointer that must be initialized to a memory segment of 32 bytes where the key is stored. If the cipher type used is AES 128, only the first 16 bytes are used.
2. **unsigned char*** : the new cipher key. It is an unsigned char pointer that must be initialized to a memory segment of 32 bytes where the key is stored. If the cipher type used is AES 128, only the first 16 bytes are used.
3. **unsigned long** : the address of the destination node.

Return parameter:

void : no parameters.

IOTMesh_Rssi_Get

Description :

This function return the current RF link RSSI. As example, it could be used to get the RF background RSSI level. The return value is given in dBm.

Prototype :

signed char IOTMesh_Rssi_Get(void);

Input parameters :

void : no parameters.

Return parameter:

signed char : The current RSSI level.

IOTMesh_Rx_Rssi_Lqi_Read

Description :

This function could be used to get the last received messages RSSI and LQI. The library keep in memory the five last received messages values.

Prototype :

unsigned char IOTMesh_Rx_Rssi_Lqi_Read(unsigned char, unsigned long *, signed char *, unsigned char *);

Input parameters :

1. **unsigned char** : gives the number of messages to backward in history table (from 0 for most recent message to 4 for the oldest message)
2. **unsigned long *** : pointer to variable where address of transmitter will be written
3. **signed char *** : pointer to variable where RSSI of message will be written
4. **unsigned char *** : pointer to variable where LQI will be written

Return parameter:

unsigned char : TRUE if number of messages to backward is valid, otherwise FALSE.

IOTMesh_Rf_Power_Get

Description :

This function is used to read the RF transmitter power. The returned value will be in dBm between

-30dBm and +12dBm.

Prototype :
signed char IOTMesh_Rf_Power_Get(void);

Input parameters :
void : no parameters.

Return parameter:
signed char : RF transmit power

IOTMesh_Rf_Power_Set

Description :
This function is used to set the RF transmit power. This value must be between -30dBm (lowest power) and +12dBm (highest transmit power).

Prototype :
void IOTMesh_Rf_Power_Set(signed char);

Input parameters :
1. **signed char** : RF transmit power

Return parameter:
void : no parameters.

IOTMesh_Rf_Channel_Get

Description :
This function is used to read RF channel. The channel spacing depends on RF settings.

Prototype :
unsigned char IOTMesh_Rf_Channel_Get(void);

Input parameters :
void : no parameters.

Return parameter:
unsigned char : RF channel

IOTMesh_Rf_Channel_Set

Description :
This function is used to set RF channel.

Prototype :
void IOTMesh_Rf_Channel_Set(unsigned char);

Input parameters :
1. **unsigned char** : RF channel

Return parameter:
void : no parameters.

IOTMesh_Temperature_Set

Description :
This function is used to give the ambient temperature of the RFM3L board. The temperature value is in °C between -128°C and +127°C (to limit to electronic board working temperature).

Prototype :
void IOTMesh_Temperature_Set(signed short);

Input parameters :
1. **signed short** : current temperature

Return parameter:
void : no parameters.

IOTMesh_Version_Get

Description :
This function returns the operating version of the IOTMesh library.

Prototype :
void IOTMesh_Version_Get(const unsigned char**, unsigned char*);

Input parameters :
1. **const unsigned char**** : pointer to pointer to a memory segment containing the series of bytes representing the constant version number.

2. **unsigned char *****: pointer to unsigned char containing the number of bytes

Return parameter:
void : no parameters.

IOTMesh_QOS

Description :
This function returns the QOS info of the current node.

Prototype :
void IOTMesh_QOS(t_IOTMesh_QOS*);

Input parameters :
1. t_IOTMesh_QOS**** : pointer to a memory segment containing the QOS information.

The structured type t_IOTMesh_QOS: includes 15 members:

1. RSSI_Ambient: ambient rssi seen by the node
2. CCA_Rate: Channel occupation indicator (0%: channel free, 100%: channel fully used)
3. Neighbors_Nb: number of known neighbors
4. Routes_Nb: number of available routes
5. MER: Message error rate
6. Message_APP_Sent: number of data messages sent
7. Message_APP_Received: number of data messages received
8. Message_APP_NACK: number of messages lost (sent and never aknowledged for messages where ACK is required)
9. Message_MAC_Sent: number of all messages sent including network protocol messages (ACK, ..)
10. Message_MAC_Received: number of all messages received
11. Message_NoRoute: number of messages that were not routed
12. Message_RREQ: number of route discovery
13. Message_RRESP: number of routes received
14. AVG_ACK_Time: average time between sending a message and receiving the acknowledge in milli seconds
15. rssi[5]: rssi of the last received 5 messages

Return parameter:
void : no parameters.

IOTMesh_QOS_Remote

Description :

This function returns the QOS info of a given node.

Prototype :

void IOTMesh_QOS_Remote(unsigned long , void (*callback)(t_IOTMesh_QOS*));

Input parameters :

1. **unsigned long** : the remote node address
2. **void (*callback)(t_IOTMesh_QOS*)**: a callback function that should be called once we have the answer of the remote node

Return parameter:

void : no parameters.

IOTMesh_TTL_MAX_Set

Description :

This function initializes the TTL max in the current network. The default value is 54. By defining a max TTL, we inform the IOTMesh Library to adapt delays / routes to this max.

Prototype :

void IOTMesh_TTL_MAX_Set(unsigned short);

Input parameters :

1. **unsigned short**: the new value between 1 and 54

Return parameter:

void : no parameters.

IOTMesh_TTL_Max_Get

Description :

This function returns the max TTL defined for this network

Prototype :

unsigned short IOTMesh_TTL_Max_Get(void);

Input parameters :

void : no parameters.

Return parameter:
unsigned short : the current max TTL

IOTMesh_Netconf_Set

Description :
This function defines the network mode of the current node. Three modes are available:

- IOTMESH_NETCONF_MESH : fully participates in the network
- IOTMESH_NETCONF_ENDNODE : does not participate to network, it only sends his own messages and receives messages addressed to him, his group or broadcasted messages
- IOTMESH_NETCONF_ENDNODE_BROADCAST : same as previous but it transfers broadcasted messages.

Prototype :
void IOTMesh_Netconf_Set(unsigned short);

Input parameters :
unsigned short : the network mode

Return parameter:
void : no parameters.

IOTMesh_Netconf_Get

Description :
This function returns current network mode.

Prototype :
unsigned short IOTMesh_Netconf_Get(void);

Input parameters :
void : no parameters.

Return parameter:
unsigned short : the current mode

IOTMesh_Sleep

Description :
This function allows to turn off the transceiver to save energy.

Prototype :
void IOTMesh_Sleep(unsigned short);

Input parameters :
unsigned short : the sleep mode, there is currently only one supported mode *IOTMESH_SLEEP_RF*

Return parameter:
void : no parameters.

IOTMesh_Wake

Description :
This function allows to turn on the transceiver.

Prototype :
void IOTMesh_Wake(unsigned short);

Input parameters :
unsigned short : the wake mode, there is currently only one supported mode *IOTMESH_WAKE_RF*

Return parameter:
void : no parameters.

IOTMesh_Status_Get

Description :
This function returns the current status of the transceiver.

Prototype :
unsigned short IOTMesh_Status_Get(void);

Input parameters :
void : no parameters.

Return parameter:
unsigned short : the current mode, it can take these values:

- IOTMESH_STATUS_UP : transceiver is on
- IOTMESH_STATUS_RF_SLEEP : transceiver is off

IOTMesh_Service_Register

Description:

This function informs IOTMesh that this application offers a service.

Prototype:

unsigned short IOTMesh_Service_Register(unsigned long code, void (*callback)(char *params));

Input parameters:

1. **Unsigned long : service ID**
2. **void (*callback)(char *params)** : a callback function that should be called once the service is invoked. (reserved for future use)

Return parameter:

unsigned short : it can take these values:

- IOTMESH_SERVICE_REGISTERED: the service was successfully registered
- IOTMESH_SERVICE_UNREGISTERED_SLOT_UNAVAILABLE: service was not registered, maximum number of services is reached
- IOTMESH_SERVICE_UNREGISTERED_CODE_NOT_ALLOWED: service was not registered. The service ID given is not allowed. Service code can take any value between 0x00000000 and 0xFFFEFFFF
- IOTMESH_SERVICE_UNREGISTERED_NULL: service was not registered

IOTMesh_Service_Unregister

Description:

This function informs IOTMesh that this application no longer offers a service.

Prototype:

unsigned short IOTMesh_Service_Unregister(unsigned long code);

Input parameters:

1. **Unsigned long : service ID**

Return parameter:

unsigned short : it can take these values:

- IOTMESH_SERVICE_UNREGISTERED: service successfully unregistered
- IOTMESH_SERVICE_NOT_FOUND: service not found

IOTMesh_Service_Get

Description:

This function discovers an IOTMesh node offering a service.

Prototype:

```
unsigned short IOTMesh_Service_Get_Service(unsigned long code,
                    void (*callback)( t_IOTMesh_Service *param ) );
```

Input parameters:

1. **Unsigned long : service ID**
2. **void (*callback**)(t_IOTMesh_Service ***params)** : a callback function that should be called once a node offering the service is discovered

Return parameter:

unsigned short : it can take these values:

- IOTMESH_SERVICE_REQUESTED: Discovering has started
- IOTMESH_SERVICE_REQUEST_SLOT_UNAVAILABLE : Discovering could be made, maximum number of requests is reached
- IOTMESH_SERVICE_REQUEST_UPDATED: A previous request to the same service is already under process, the callback function is updated

IOTMesh_Service_Get_All_Services

Description:

This function discovers all accessible IOTMesh nodes offering a service.

Prototype:

```
unsigned short IOTMesh_Service_Get_All_Services(unsigned long code,
                    void (*callback)( t_IOTMesh_Service *param ) );
```

Input parameters:

3. **Unsigned long : service ID**
4. **void (*callback**)(t_IOTMesh_Service ***params)** : a callback function that should be called once a node offering the service is discovered. It can be called several times. One call for each node found

Return parameter:

unsigned short : it can take these values:

- IOTMESH_SERVICE_REQUESTED: Discovering has started

- IOTMESH_SERVICE_REQUEST_SLOT_UNAVAILABLE : Discovering could be made, maximum number of requests is reached
- IOTMESH_SERVICE_REQUEST_UPDATED: A previous request to the same service is already under process, the callback function is updated

EEPROM Special care

Some data used by the IOTMesh library are stored in EEPROM in address range from 0x08080F00 to 0x08080FFF. Do not erase this EEPROM area. If so, please contact us to request special settings data.

Example software

Example software is available on www.iotmesh.eu website.

This example sends a 221 useful bytes message every 1000ms addressed to all nodes using the broadcast address and it manages received messages. Exchanged data are ciphered using AES 128. The example uses a 38.4kbds RF data rate at 868MHz. Other radio configuration files are available in the project.