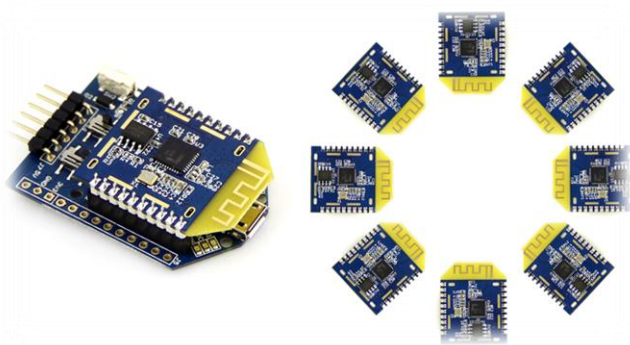




# MeshBee®

## Open Source ZigBee RF Module

### User's Manual



©2014 Seeed Technology Inc. All rights reserved  
[www.seeedstudio.com](http://www.seeedstudio.com)

---

| Doc Version | Date              | Author             | Remark         |
|-------------|-------------------|--------------------|----------------|
| <i>v0.1</i> | <i>2014/04/28</i> | <i>oliver wang</i> | <i>Created</i> |

## Table Of Contents

|                                      |    |
|--------------------------------------|----|
| Overview .....                       | 4  |
| About this document .....            | 4  |
| Introduction .....                   | 4  |
| Acronyms and Abbreviations .....     | 5  |
| 1. Key Features.....                 | 6  |
| 1.1 Physical features: .....         | 6  |
| 1.2 Operation features: .....        | 6  |
| 2. Pin definition .....              | 7  |
| 2.1 Pin assignment .....             | 8  |
| 3. Operation Mode .....              | 9  |
| 3.1 AT mode .....                    | 9  |
| 3.2 API Mode .....                   | 10 |
| 3.3 MCU Mode .....                   | 10 |
| 3.4 Data Mode .....                  | 11 |
| 4. AT commands .....                 | 12 |
| 4.1 Node information commands.....   | 12 |
| 4.2 Data transmit commands.....      | 13 |
| 4.3 Network formation commands ..... | 14 |
| 4.4 OTA commands.....                | 16 |
| 5. API frame .....                   | 18 |
| 5.1 Structure of API Frame .....     | 18 |
| 6. AUPS Function list.....           | 23 |
| 6.1 Set run-time parameters .....    | 23 |
| 6.2 Send RF data .....               | 24 |
| 6.3 Suli API .....                   | 24 |
| Appendix a: AT command index.....    | 32 |

# Overview

## *About this document*

This manual gives a single point of reference for information relating to the MeshBee. Including:

- Chapt1: Key features;
- Chapt2: Pin definition;
- Chapt3: Operation mode;
- Chapt4: AT commands;
- Chapt5: API frames;
- Chapt6: Functions that AUPS can call

Information shown in this document is all based on the firmware v1003. The manual should be used as a reference resource throughout MeshBee application development. It does not provide in-depth introduction of the MeshBee programming. Please refer to the MeshBee CookBook(*MB\_2014\_D02*) for further references on the firmware architecture and programming issue.

## *Introduction*

MeshBee<sup>®</sup> is a 2.4 GHz wireless zigbee RF module together with high level open source software driven by community. It uses microchip JN516x from NXP that enables several different standards-based zigbee mesh networking. User can easily and cost-effectively integrate ZigBee functionality into target project. Our factory firmware supports latest fully **Zigbee Pro** stack.

MeshBee<sup>®</sup> is the best choice to make your connected thing.

## ***Acronyms and Abbreviations***

AUPS: Arduino-ful user programming space

SPM: Stream processing machine

CMI: Communication interface

ADS: Airport data server

UDS: Uart data server

HAL: Hardware abstract layer

SULI: Seeed Unified Library Interface

API: Application programming interface

MCU: Microcontroler

JenOS: Jennic operating system

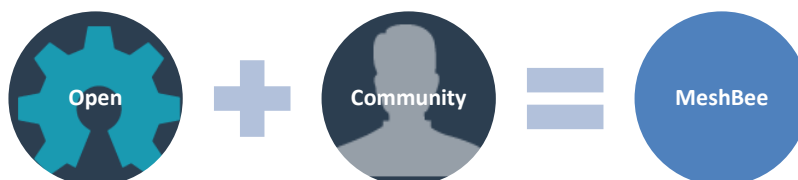
# 1. Key Features

## 1.1 Physical features:

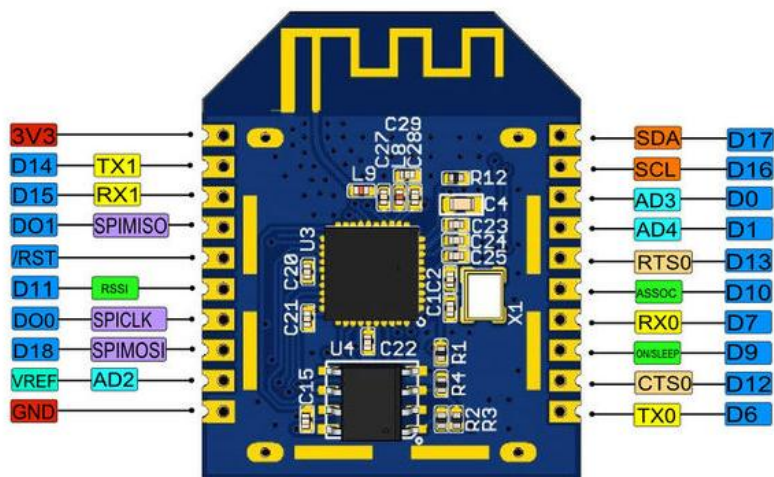
- 1) *Range: Indoor/Urban: up to 30m;  
Outdoor line-of-sight: up to 100m;*
- 2) *Receiver Sensitivity: -95dBm;*
- 3) *Working Frequency: unlicensed  
2.4GHz band;*
- 4) *Data Transmission Rate: 4800, 9600,  
19200, 38400, 57600, 115200 (bps);*
- 5) *Programmable 32-bit RISC CPU:  
32M clock, 256KB Flash, 32KB RAM,  
4KB EEPROM;*
- 6) *Socket compatible with the Xbee, so  
you can plug it into any Xbee socket  
as a quick replacement*

## 1.2 Operation features:

- 1) *Easy-to-Use Serial Interface and rich  
extendable ports;*
- 2) *Communication type: Point to Point,  
Star Network , Mesh Network;*
- 3) *Support for OTA(upgrade firmware  
over the air);*
- 4) *Easy-to-Use AT Command: Setup  
ZigBee network, set Serial Baud Rate,  
etc;*
- 5) *API configuration and control mode;*
- 6) *Arduino-ful user programming  
space;*
- 7) *Open source hardware and  
firmware;*



## 2. Pin definition



*Figure 2.1: Pin definition of MeshBee*



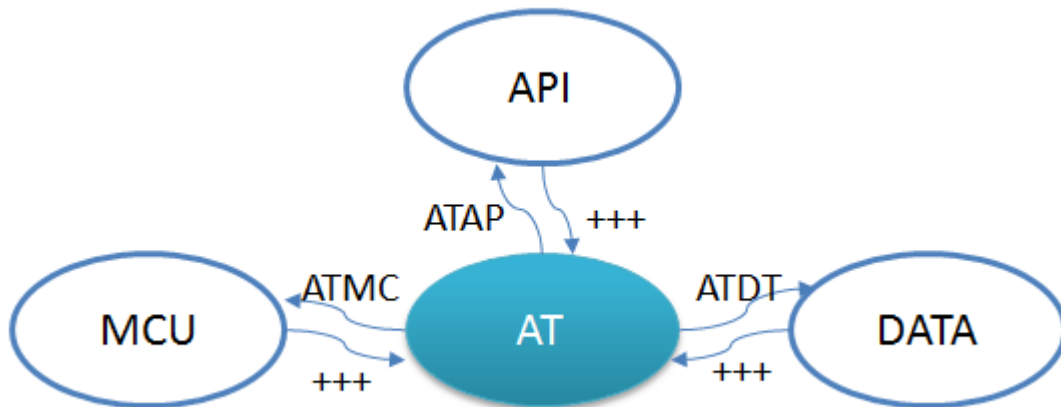
**Note:** please refer to datasheet of JN516x for more information about each pin.

## 2.1 Pin assignment

| Pin No | Name            | Direction | Description                                                  |
|--------|-----------------|-----------|--------------------------------------------------------------|
| 1      | 3V3             | —         | Power supply                                                 |
| 2      | D14/TX1         | Output    | Digital IO14 or UART1 TX                                     |
| 3      | D15/RX1         | Input     | Digital IO15 or UART1 RX                                     |
| 4      | DO1/SPIMISO     | Both      | Digital Output 1 or SPI Master In Slave Out Input            |
| 5      | RST             | —         | Reset pin                                                    |
| 6      | D11/PWM1        | Both      | Digital IO11 (default usage: RSSI Indicator) or PWM1 Output  |
| 7      | DO0/SPICLK      | Output    | Digital Output 0 or SPI Master Clock Output                  |
| 8      | D18/ SPIMOSI    | Both      | Digital IO 18 or SPI Master Out Slave In Output              |
| 9      | Vref/ ADC2      | Input     | Analogue peripheral reference voltage; ADC input 2           |
| 10     | GND             | —         | GND                                                          |
| 11     | D6/TX0          | Both      | UART0 TX or Digital IO6                                      |
| 12     | D12/CTS0        | Both      | Digital IO12 or UART0 clear to send input                    |
| 13     | D9              | Both      | Digital IO 9 (default usage: Mesh Bee ON/Sleep Indicator)    |
| 14     | D7/RX0          | Both      | UART0 RX or Digital IO 7                                     |
| 15     | D10             | Both      | Digital IO 10 (default usage: Network Association Indicator) |
| 16     | RTS0            | Both      | Digital IO 13 or UART0 request to send output                |
| 17     | D1/SPISEL2/ADC4 | Both      | Digital IO 1; SPI Master Select Output 2; ADC input 4        |
| 18     | D0/SPISEL1/ADC3 | Both      | Digital IO 0; SPI Master Select Output 1; ADC input 3        |
| 19     | D16/SCL         | Both      | Digital IO 16 or I2C clock                                   |
| 20     | D17/SDA         | Both      | Digital IO 17 or I2C data                                    |

## 3. Operation Mode

MeshBee has four different types of mode: AT, API, DATA, MCU, illustrated in figure below:



**Figure 3.1:** No matter which mode MeshBee works in, input “+++” can come back to AT command mode.

### 3.1 AT mode

Mesh Bee communicates with outside through UART1 including data and command communicating. The default setting of UART1 is: *115200 baud rate, data bits 8, parity none, stop bit 1*. “+++<CR>” can put Mesh Bee into AT mode. The mode switch is illustrated in figure 3.1.

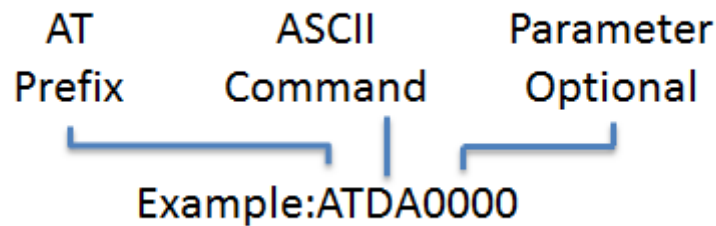
AT command can be classified into two types: register R/W AT and action AT.

The pattern of AT command is “ATXX[DDDD]<CR>” in which XX stands for the register/action name and DDDD stands for the written value of a register. All letters’ case is ignored.

Register R/W AT can operate a virtual register of Mesh Bee. Absence of DDDD means reading the register value out and meanwhile ATXXDDDD means setting the register value to DDDD.

Action AT can trigger a specific action. The execution of command may be immediate or time-consuming.

*Syntax for sending AT commands:*



*Figure 3.2: syntax for AT commands*

## 3.2 API Mode

API is simply a set of standard interfaces created to allow other MCU to interact with MeshBee. For our purposes, API supports local operation and remote operation. For example, a host application can send an “ATIO” frame to Coordinator A, A will set its GPIO when it receives this frame. The most important thing to note is that APIs are specifically engineered to enable MeshBee to talk efficiently to other MCU. The target of API-mode is to transmit highly structured data quickly and reliably.

## 3.3 MCU Mode

In order to simplify the development of application for user, we create an Arduino-ful user programming space(AUPS).

In AT mode, using “ATMC” to enter MCU mode, then the `arduino_loop` will be executed periodically.

Write your own code in “[ups\\_arduino\\_sketch.c](#)”.

### **Example :**

```
IO_T led_io;
int16 state = HAL_PIN_HIGH;
void arduino_setup(void)
{
    setLoopIntervalMs(1000);           //set loop period
```

```
suli_pin_init(&led_io, 9);           //init led
suli_pin_dir(&led_io, HAL_PIN_OUTPUT);
}

void arduino_loop(void)
{
    suli_pin_write(&led_io, state);    //set led
    if(state == HAL_PIN_HIGH)
        state = HAL_PIN_LOW;
    else
        state = HAL_PIN_HIGH;
}
```



**Note:** In MCU mode, Uart1 is under the control of the AUPS, user should not send API frame to MeshBee.

## 3.4 Data Mode

When operating in Data mode, the modules act as a serial line. All UART data received through the UART1 is transmitted to a specified remote device.

To use a transparent connection, take the following steps:

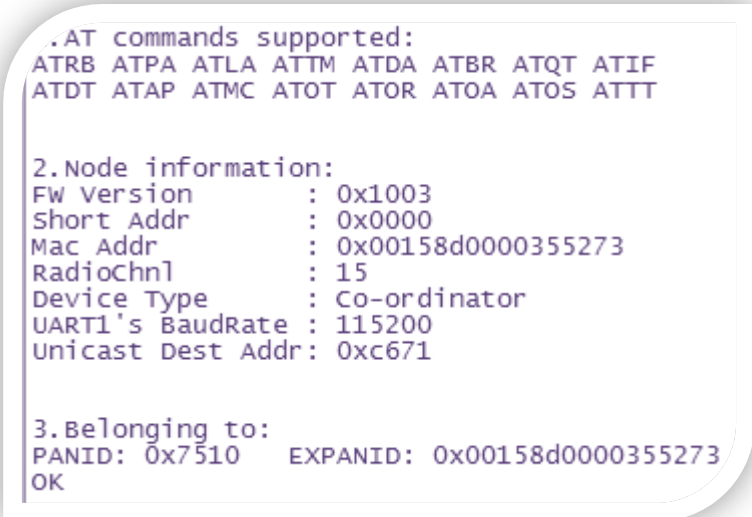
- 1) Set unicast address: *ATDAxxx*
- 2) Enter Data Mode: *ATDT*

## 4. AT commands

### 4.1 Node information commands

#### ATIF

- 1) Action AT, immediate execution, for any zigbee role.
- 2) Get node InFormation
- 3) ATIF command will print information of node including: supported AT commands, node's firmware version, node's zigbee short address, node's MAC address, node's radio channel, node's zigbee role, etc.
- 4) Example:



```
.AT commands supported:
ATRB ATPA ATLA ATTM ATDA ATBR ATQT ATIF
ATDT ATAP ATMC ATOT ATOR ATOA ATOS ATTT

2.Node information:
FW Version      : 0x1003
Short Addr      : 0x0000
Mac Addr        : 0x00158d0000355273
RadioChnl       : 15
Device Type     : Co-ordinator
UART1's BaudRate : 115200
Unicast Dest Addr: 0xc671

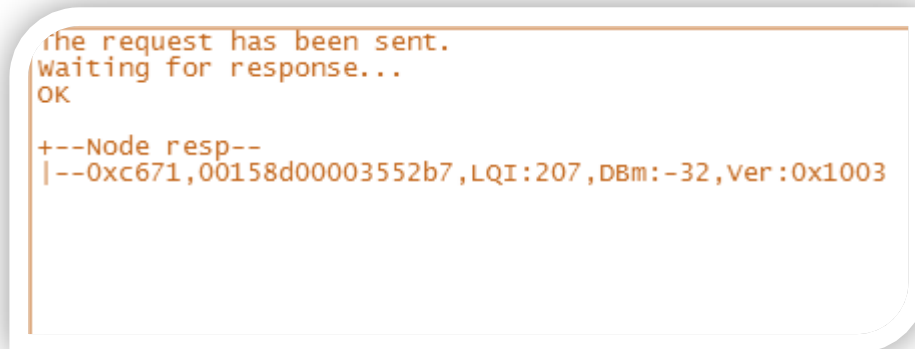
3.Belonging to:
PANID: 0x7510   EXPANID: 0x00158d0000355273
OK
```

*Figure 3.3: ATIF screen shot*

#### ATLA

- 1) Action AT, time-consuming execution, for any zigbee role
- 2) List All nodes within the network

- 3) ATLA will broadcast a topology query packet into the whole network. The node that's still alive may response to that. The querying node will print responding nodes' short address, MAC address, Link-Quality-Indication (LQI), etc. LQI is a positive integer, the bigger LQI the better link quality.



*Figure 3.4: ATLA screen shot*

## ATQT

- 1) Action AT, immediate execution, for any zigbee role
- 2) Get on-chip temperature.

## 4.2 Data transmit commands

### ATTM

- 1) Register R/W AT, for any zigbee role
- 2) Bits:1, decimal, max:1, default:0
- 3) Set node's TX Mode
- 4) 0 - broadcast, 1- unicast (need setting destination address by ATDA command first).

### ATDA

- 1) Register RW AT, for any zigbee role
- 2) Bits:4, hex, max: ffff, default:0000
- 3) Set node's unicast destination address
- 4) This address will also be used as the OTA target address, means that this destination address will be used for ATOT and ATOS command. It has a pattern of HHHH that is 4 bits of HEX number ignoring case.
- 5) Example: ATDA14ad<CR>

### ATBR

- 1) Register R/W AT, for any zigbee role
- 2) Bits:1, decimal, max:5, default:5
- 3) Set UART1's Baud Rate
- 4) 0- 4800, 1-9600, 2-19200, 3-38400, 4-57600, 5-115200.
- 5) Example: ATBR5<CR>

## ***4.3 Network formation commands***

### ATPA

- 1) Register RW AT, for any zigbee role but with different effect.
- 2) Bits:1, decimal, max:1, default:0
- 3) Set node's Power up Action
- 4) The node's default power-up behavior is restoring the last network state before power down. But when setting PA register to 1 and then reboot, the node will not restore the last network. In this case, coordinator node will re-create a network and router/End device will re-scan the network. The PA register will be cleared to 0 after reboot.

### ATRS

- 1) Action AT, time-consuming execution, for router/End device
- 2) Re-Scan network
- 3) The scanning process will take a while and you can use ATL N command to monitor the scan result. If node finds nothing after a long time scanning, retry ATRS command or reset Mesh Bee. The node will automatically join the first found network when AJ register has a value of 1.

### ATL N

- 1) Action AT, immediate execution, for router/End device
- 2) List Network scanned
- 3) The index value will be used by ATJ N command.

### ATJ N

- 1) Register R/W AT, for router/End device
- 2) Bits:1, decimal, max:8, default:0
- 3) Join a Network with specific index
- 4) ATJ N command is also an action trigger command. The node will join the network specified by the index of ATL N output. ATJ N will return error when the node's already in that network.

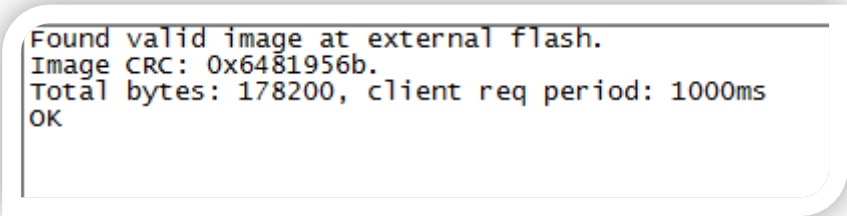
### ATAJ

- 1) Register RW AT, for router/End device
- 2) Bits:1, decimal, max:1, default:1
- 3) Whether Auto Join network scanned
- 4) If AJ register has a value of 1, the node will automatically join the first network scanned after ATRS command or power up with PA register equals to 1.

## 4.4 OTA commands

### ATOT

- 1) Action AT, immediate execution, for coordinator
- 2) OTA Trigger
- 3) Non-coordinator nodes can upgrade firmware over-the-air. This is called OTA. ATOT command will trigger the OTA upgrade download of a destination node. OTA architecture consists of OTA server and client. Coordinator will be the server side and router/End device is the client side. To OTA a client node, you should firstly enter the AT mode on server side and set the unicast destination address (DA register) to the short address of the client node, and then execute the ATOT command. And now trace serial port (usually UART0) will print some information about OTA process if trace is enabled. After downloading all image blocks which are saved in the external Flash, the client node will trigger the upgrade process automatically. The process is: mark the internal firmware invalid, then reboot, and then the bootloader will copy the new image from the external Flash into the internal Flash, and then run the new firmware.



```
Found valid image at external flash.  
Image CRC: 0x6481956b.  
Total bytes: 178200, client req period: 1000ms  
OK
```

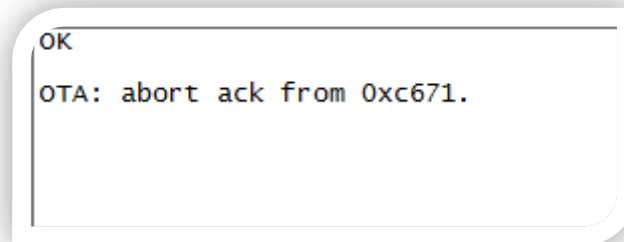
*Figure 3.5: ATOT screen shot*

### ATOR

- 1) Register RW AT, for coordinator
- 2) Bits:5, decimal, max:60000, default:1000
- 3) OTA block request Rate
- 4) Set the interval of two image block requests. The value's unite is milliseconds. The smaller, the faster.

## ATOA

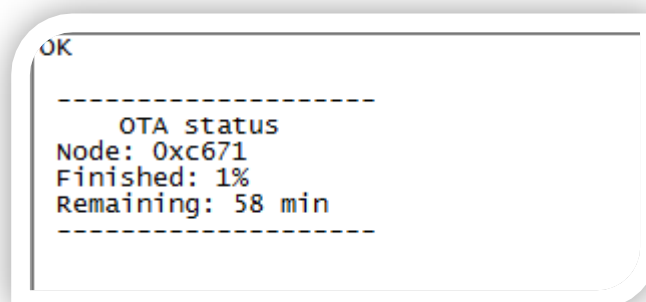
- 1) Action AT, immediate execution, for coordinator
- 2) OTA Abort
- 3) Abort the OTA downloading process of a specific node specified by the DA register.
- 4) Example:



*Figure 3.6: ATOA screen shot*

## ATOS

- 1) Action AT, time-consuming execution, for coordinator
- 2) Query OTA Status
- 3) Query the status of the OTA downloading process of a specific node specified by the DA register.



*Figure 3.7: ATOS screen shot*

## 5. API frame

### 5.1 Structure of API Frame

Every transfer of information requires a protocol. We defined the API frame like this(structure was defined in firmware\_at\_api.h):

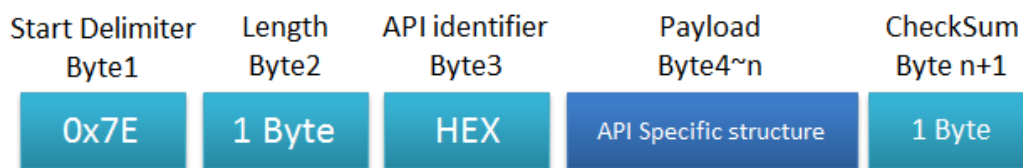


Figure 5.1: API Frame structure

#### 5.1.1 API structure types

##### *API\_LOCAL\_AT\_REQ*

API identifier value: 0x08

These packet types are useful only if the host wants to send commands to its local MeshBee. You don't need to specify the unicast address.

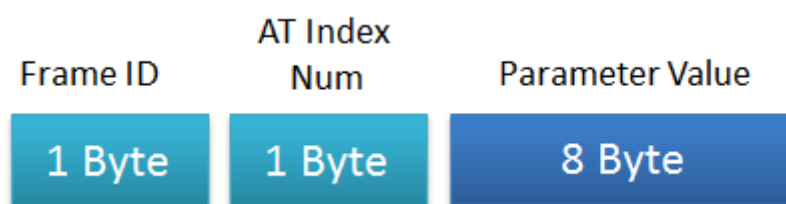


Figure 5.2: local AT require

**Frame ID:** To Identifies the UART data frame for the host to correlate with a subsequent ACK (acknowledgement).

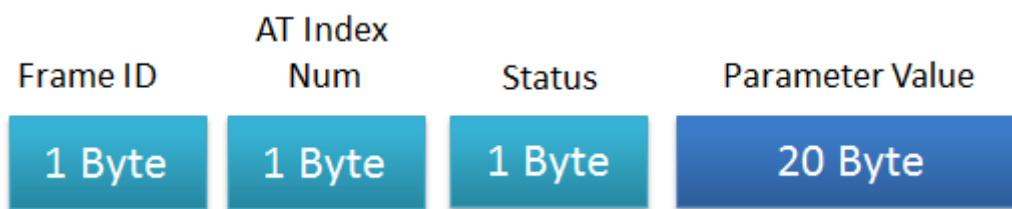
**AT index Num:** Index of the AT commands.

**Parameter value:** Parameter value to be set.

### *API\_LOCAL\_AT\_RESP*

API identifier value: 0x88

The API type “local AT response” is an ACK frame which is returned to the host from MeshBee after handling a local AT request frame. To set a register, these frame types indicate whether the request execution is successful or not. To read a register, it contains the value of the register you query.



*Figure 5.3: local AT response*

**Frame ID:** To Identifies the UART data frame for the host to correlate with a subsequent ACK (acknowledgement).

**AT index Num:** Index of the AT commands.

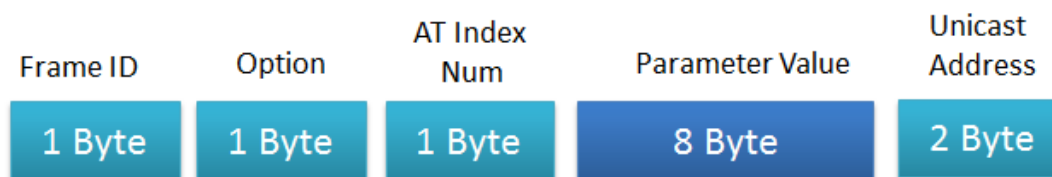
**Status:** Command execution status.

**Parameter value:** Return register value.

### *API\_REMOTE\_AT\_REQ*

API identifier value: 0x17

These types allows for module parameter registers on a remote device to be queried or set, or perform an action (example: reboot) on a remote device.



*Figure 5.4: remote AT require*

**Frame ID:** To Identifies the UART data frame for the host to correlate with a subsequent ACK (acknowledgement).

**Option:**

[8:0]:ACK Mode, 0 with ACK; 1 without ACK.

[8:1]:Cast Mode, 0 unicast; 1 broadcast.

**AT index Num:** Index of the AT commands.

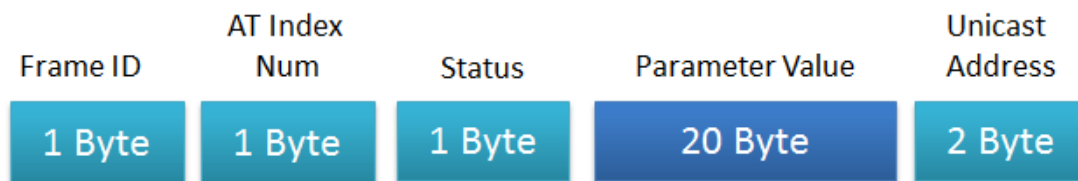
**Parameter value:** Parameter value to be set.

**Unicast Address:** Set to the 16- bit network address of the remote.

### [API\\_REMOTE\\_AT\\_RESP](#)

API identifier value: 0x97

The API type “remote AT response” is an ACK frame which is returned to the host from the remote node after handling a remote AT request frame.



**Figure 5.5: remote AT response**

**Frame ID:** To Identifies the UART data frame for the host to correlate with a subsequent ACK (acknowledgement).

**AT index Num:** Index of the AT commands.

**Status:** Command execution status.

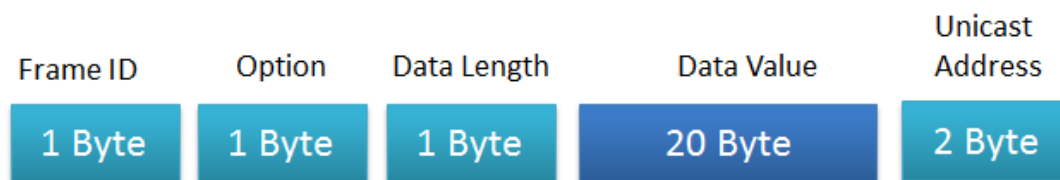
**Parameter value:** Return register value.

**Unicast Address:** 16- bit network address of the remote.

### [API\\_DATA\\_PACKET](#)

API identifier value: 0x02

The API type “data packet” is a user data frame. This request message will cause the module to send RF Data as an RF Packet.



**Figure 5.6: Data packets TX require**

**Frame ID:** To Identifies the UART data frame for the host to correlate with a subsequent ACK (acknowledgement).

**Option:**

[8:0]:ACK Mode, 0 with ACK; 1 without ACK.

[8:1]:Cast Mode,0 unicast; 1 broadcast.

**Data length:** the length of data payload.

**Parameter value:** Data value.

**Unicast Address:** Set to the 16- bit network address of the remote.

### [API\\_TOPO\\_REQ](#)

API identifier value: 0xfb

This API type allows module to query the network topology.

Required Cmd

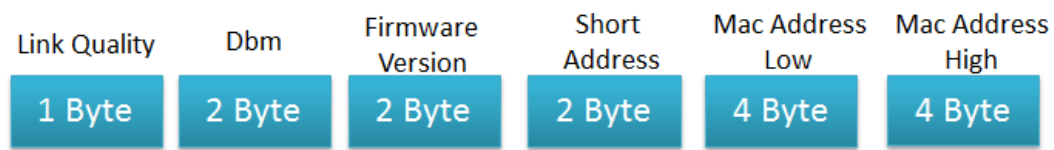
0x01

**Figure 5.7: Structure of network Topo Require**

### [API\\_TOPO\\_RESP](#)

API identifier value: 0x6b

In response to an “API\_TOPO\_REQ” message, the module will send a response message.



*Figure 5.8: Structure of network topology response*

**Link Quality:** Indicate link quality of node.

**Dbm:** Power intensity.

**Firmware Version:** Firmware Version.

**Short Address:** source address of this response.

**Mac Address Low:** MAC address of the node, low byte.

**Mac Address High:** MAC address of the node, high byte.

---

## 6. AUPS Function list

This chapter contains the information of functions that AUPS can call.

### 6.1 Set run-time parameters

#### setLoopIntervalMs

void setLoopIntervalMs(uint32 ms);

**Description:**

This function can be used to set the period of “*arduino\_Loop()*”.

**Parameter:**

ms : millisecond

**Return:**

None

#### setNodeState

void setNodeState(uint32 state);

**Description:**

This function can be used to set the working state of MeshBee.

**Parameter:**

state : working state

E\_MODE\_AT

E\_MODE\_API

E\_MODE\_DATA

E\_MODE\_MCU

**Return:**

None

---

## 6.2 Send RF data

### bSendToAirPort

bool API bSendToAirPort(uint16 txMode, uint16 unicastDest, uint8 \*buf, int len);

**Description:**

This function can be used to send RF packets.

**Parameter:**

txMode: mode of transmit

UNICAST

BROADCAST

unicastDest: short address of unicast

buf: the pointer of data

len: length of the data

**Return:**

OK

ERR

## 6.3 Suli API

### suli\_pin\_init

void suli\_pin\_init(IO\_T \*pio, PIN\_T pin);

**Description:**

This function can be used to initialize a digital IO of MeshBee.

**Parameter:**

pio: pointer of the IO\_T entity

pin: pin No

D0, D1, D2, D3, D4, D5, D6, D7, D8, D9, D10, D11, D12, D13, D14,  
D15, D16, D17, D18, D19, D20, DO0, DO1.

**Return:**

none

---

## suli\_pin\_dir

void suli\_pin\_dir(IO\_T \*pio, DIR\_T dir);

### Description:

This function can be used to set direction of digital IO.

### Parameter:

pio: pointer of the IO\_T entity

dir: direction

HAL\_PIN\_INPUT

HAL\_PIN\_OUTPUT

### Return:

none

## suli\_pin\_write

void suli\_pin\_write(IO\_T \*pio, int16 state);

### Description:

This function can be used to write a digital IO.

### Parameter:

pio: pointer of the IO\_T entity

state: state of IO

HAL\_PIN\_LOW

HAL\_PIN\_HIGH

### Return:

none

## suli\_pin\_read

int16 suli\_pin\_read(IO\_T \*pio);

### Description:

---

This function can be used to read a digital IO.

**Parameter:**

pio: pointer of the IO\_T entity

**Return:**

state: state of IO

HAL\_PIN\_LOW

HAL\_PIN\_HIGH

## suli\_analog\_init

void suli\_analog\_init(ANALOG\_T \* aio, PIN\_T pin);

**Description:**

This function can be used to initialize an analog pin.

**Parameter:**

aio: pointer of the ANALOG\_T entity

pin: pin No

A1 : ADC1

A2 : ADC2

A3 : ADC3

A4 : ADC4

TEMP: On-chip temperature ADC

VOL : On-chip voltage ADC

**Return:**

none

## suli\_analog\_read

int16 suli\_analog\_read(ANALOG\_T \* aio);

**Description:**

This function can be used to read the ADC value.

**Parameter:**

aio: pointer of the ANALOG\_T entity

---

**Return:**

ADC value

## **suli\_i2c\_init**

void suli\_i2c\_init(void \* i2c\_device);

**Description:**

This function can be used to initialize I2C of MeshBee (D16, D17).

**Parameter:**

i2c\_device: any dummy value

**Return:**

none

## **suli\_i2c\_write**

uint8 suli\_i2c\_write(void \* i2c\_device, uint8 dev\_addr, uint8 \*data, uint8 len);

**Description:**

This function can be used to write a buff to I2C.

**Parameter:**

i2c\_device: any dummy value

dev\_addr: device address

data: data array

len: length of the data

**Return:**

The number of bytes already been written

## **suli\_i2c\_read**

uint8 suli\_i2c\_read(void \*i2c\_device, uint8 dev\_addr, uint8 \*buff, uint8 len);

---

**Description:**

This function can be used to read a buff from I2C.

**Parameter:**

i2c\_device: any dummy value

dev\_addr: device address

data: pointer of data array

len: length of the data

**Return:**

The number of bytes already been read

## suli\_uart\_init

void suli\_uart\_init(void \* uart\_device, int16 uart\_num, uint32 baud);

**Description:**

This function can be only used to initialize uart1 of MeshBee. Because uart0 is under the control of the system.

**Parameter:**

uart\_device: any dummy value

uart\_num: any dummy value

baud: baud rate

4800

9600

19200

38400

57600

115200

**Return:**

none

## suli\_uart\_send

void suli\_uart\_send(void \* uart\_device, int16 uart\_num, uint8 \*data, uint16 len);

**Description:**

---

This function can be only used to send data through uart1.

**Parameter:**

uart\_device: any dummy value

uart\_num: any dummy value

data: pointer of the data array

len: length of the data

**Return:**

none

## suli\_uart\_send\_byte

void suli\_uart\_send\_byte(void \*uart\_device, int16 uart\_num, uint8 data);

**Description:**

This function can be only used to send one byte through uart1.

**Parameter:**

uart\_device: any dummy value

uart\_num: any dummy value

data: data byte

**Return:**

none

## suli\_uart\_write\_float

void suli\_uart\_write\_float(void \*uart\_device, int16 uart\_num, float data, uint8 prec);

**Description:**

This function can be only used to send float data through uart1.

**Parameter:**

uart\_device: any dummy value

uart\_num: any dummy value

data: float data

**Return:**

none

---

## suli\_uart\_write\_int

void suli\_uart\_write\_int(void \*uart\_device, int16 uart\_num, int32 num);

### Description:

This function can be only used to send int data through uart1.

### Parameter:

uart\_device: any dummy value

uart\_num: any dummy value

num: int value

### Return:

none

## suli\_uart\_printf

void suli\_uart\_printf(void \*uart\_device, int16 uart\_num, const char \*fmt, ...);

### Description:

This function can be only used to send formatted string to uart1.

### Parameter:

uart\_device: any dummy value

uart\_num: any dummy value

fmt: format of string

### Return:

none

## suli\_uart\_read\_byte

uint8 suli\_uart\_read\_byte(void \*uart\_device, int16 uart\_num);

### Description:

---

This function can be only used to read a byte from uart1.

**Parameter:**

uart\_device: any dummy value

uart\_num: any dummy value

**Return:**

Returned byte

## **suli\_uart\_readable**

uint16 suli\_uart\_readable(void \* uart\_device, int16 uart\_num);

**Description:**

This function can be only used to judge if uart1 is readable.

**Parameter:**

uart\_device: any dummy value

uart\_num: any dummy value

**Return:**

The number of bytes which can be read

---

## Appendix a: AT command index

| Command Name | AT Index    | usage                         |
|--------------|-------------|-------------------------------|
| <i>ATRB</i>  | <i>0x30</i> | <i>reboot</i>                 |
| <i>ATPA</i>  | <i>0x32</i> | <i>power up action</i>        |
| <i>ATAJ</i>  | <i>0x34</i> | <i>Auto Join Network</i>      |
| <i>ATRS</i>  | <i>0x36</i> | <i>Re-Scan Network</i>        |
| <i>ATLN</i>  | <i>0x38</i> | <i>List scanned network</i>   |
| <i>ATJN</i>  | <i>0x40</i> | <i>Join specified Network</i> |
| <i>ATLA</i>  | <i>0x42</i> | <i>List All nodes</i>         |
| <i>ATTM</i>  | <i>0x44</i> | <i>Transmit mode</i>          |
| <i>ATDA</i>  | <i>0x46</i> | <i>Unicast Address</i>        |
| <i>ATQT</i>  | <i>0x48</i> | <i>Query Chip temperature</i> |
| <i>ATIO</i>  | <i>0x50</i> | <i>Handle IOs</i>             |
| <i>ATIF</i>  | <i>0x52</i> | <i>Node information</i>       |

---

**Note:** *v1003 firmware compatible*

---

---

**S T U**

Copyright (c) 2014 Seeed Technology Inc.

F5, Bldg 8, Shiling Industrial Park,  
Xinwei, #32 Tongsha Road,  
Xili Town, Nanshan Dist.  
Shenzhen 518055 China

+86 755 33552591

*[www.seeedstudio.com](http://www.seeedstudio.com)*

# Mouser Electronics

Authorized Distributor

Click to View Pricing, Inventory, Delivery & Lifecycle Information:

[Seeed Studio:](#)

[114990002](#)