MeshBee®
Open Source ZigBee RF Module
User's Manual

©2014 Seeed Technology Inc.
www.seeedstudio.com
<table>
<thead>
<tr>
<th>Doc Version</th>
<th>Date</th>
<th>Author</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>v0.1</td>
<td>2014/04/28</td>
<td></td>
<td>Created</td>
</tr>
<tr>
<td>v0.2</td>
<td>2014/6/17</td>
<td>Oliver wang</td>
<td>Add some AT commands</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Add some api descriptions</td>
</tr>
<tr>
<td>v0.3</td>
<td>2016/1/22</td>
<td>Jack</td>
<td>Updated the API frame structure, updated the AT command index, added IO index.</td>
</tr>
</tbody>
</table>
# Table of Contents

**USER’S MANUAL** .................................................................................................................. 1

**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. **ATCOMMANDS** ................................................................................................................ 12
  4.1 Node information commands .......................................................................................... 12
  4.2 Data transmit commands ............................................................................................... 13
  4.3 Network formation commands ....................................................................................... 14
  4.4 OTA commands ............................................................................................................... 16
  4.5 Sleep commands ............................................................................................................. 18
  4.6 State commands ............................................................................................................... 18

5. **API FRAME** ....................................................................................................................... 19
  5.1 Structure of API Frame ..................................................................................................... 19

6. **AUPS FUNCTION LIST** .................................................................................................... 25
  6.1 Set run-time parameters ............................................................................................... 25
  6.2 Send RF data ................................................................................................................... 25
  6.3 Receive RF data .............................................................................................................. 26
  6.4 Suli API .......................................................................................................................... 27
  6.5 RPC function .................................................................................................................. 34
  6.6 Tools function .................................................................................................................. 35

**APPENDIX A: ATCOMMAND INDEX AND API FRAME STRUCTURE** .................................. 36

**APPENDIX B: IO INDEX** ....................................................................................................... 38
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 v1.0.4. 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(\textit{MB_2014_D02}) for further references on the firmware architecture and programming issue.

Introduction

MeshBee\textsuperscript{®} 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 \textbf{Zigbee Pro} stack.

MeshBee\textsuperscript{®} 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: Microcontroller

JenOS: Jennic operating system

RPC: Remote procedure call
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, Start 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) Supports for RPC;
8) Open source hardware and firmware.
2. Pin definition

![Figure 2.1: Pin definition of MeshBee](image)

**Note:** Please refer to datasheet of JN516x for more information about each pin.
## 2.1 Pin assignment

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

Syntax for sending AT commands:

```
AT ASCII Parameter
Prefix Command Optional
```

Example: ATDA0000

*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:**

```c
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:   ATDAxxxx
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:

![ATIF screen shot](image)

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](image)

**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).
Operation Mode

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.
Operation Mode

**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 ATLN 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.

**ATLN**

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

**ATJN**

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) ATJN command is also an action trigger command. The node will join the network specified by the index of ATLN output. ATJN 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.

![ATOT screen shot](image)

TOR

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.
Operation Mode

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:

![ATOA screen shot](image)

**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.

![ATOS screen shot](image)

**Figure 3.7: ATOS screen shot**
Operation Mode

4.5 Sleep commands

ATSM

1) Register RW AT, for end device
2) Bits: 1, decimal, max: 5, default: 0
3) 0: disable sleep mode, 1/2/3: not defined, 4: cyclic sleep, 5: cyclic sleep with pio wake
4) Set sleep mode of end device

ATSP

1) Register RW AT, for end device
2) Bits: 4, decimal, max: 9999, default: 0
3) Sleep period
4) Set an end device’s sleep period

ATST

1) Register RW AT, for end device
2) Bits: 4, decimal, max: 9999, default: 0
3) wait time before sleep
4) Set an end device’s wait time before sleep

4.6 State commands

ATMF

1) Register RW AT, for all of the nodes
Operation Mode

2) Bits: 4, decimal, max: 3000, default: 0
3) Delay period between each Arduino_Loop
4) Set the delay period between each Arduino_Loop

ATMC

1) Action AT
2) Enter MCU mode

ATDT

1) Action AT
2) Enter DATA mode

ATAP

1) Action AT
2) Enter API mode

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):
Start Delimiter: Fixed to 0x7e.

Length: The length of payload section.

API Identifier: The ID of API frame type, see 5.1.1 for detail.

Payload: see 5.1.1 for detail.

Checksum: The summary of all bytes in payload section.

5.1.1 API Specific Structures

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.

<table>
<thead>
<tr>
<th>Frame ID</th>
<th>Option</th>
<th>AT Index Num</th>
<th>AT Cmd Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Byte</td>
<td>1 Byte</td>
<td>1 Byte</td>
<td>4 Byte</td>
</tr>
</tbody>
</table>

Figure 5.2: local AT request

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

Option:

Bit0: ACK Mode, 0 with ACK; 1 without ACK. Please don’t set this to 1 if you’re executing a read command.

AT Index Num: Index of the AT commands, see appendix A for detail.

AT Cmd Structure: The structure for this AT command, see appendix A for detail.

API_LOCAL_AT_RESP

API identifier value: 0x88
AUPS Function list

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](image)

**Frame ID**: Corresponding frame ID which is set when the API_LOCAL_AT_REQ request is constructed.

**AT index Num**: Index of the AT commands, see appendix A for detail.

**Status**: Command execution status, 0x0: OK, 0x1: ERR, 0x2: INVALID_CMD, 0x3: INVALID_PARAM.

**Length of Valid Data**: The length of valid data in response body.

**Response Body**: See appendix A for detail.

### 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 request](image)

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

**Option**:

- **Bit0**: ACK Mode, 0 with ACK; 1 without ACK.
- **Bit1**: Cast Mode, 0 unicast; 1 broadcast.

**AT index Num**: Index of the AT commands, see appendix A for detail.

**AT Cmd Structure**: The structure for this AT cmd, see appendix A for detail.

**Dest Addr Short**: Set to the 16-bit network address of the remote, if this address is set to 0xffff, the long address(MAC) will be used.
AUPS Function list

**Dest Addr Long**: Set to the 64-bit network address of the remote (MAC address).

---

**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](image)

**Frame ID**: Corresponding frame ID which is set when the API_REMOTE_AT_REQ request is constructed.

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

**Status**: Command execution status, 0x0: OK, 0x1: ERR, 0x2: INVALID_CMD, 0x3: INVALID_PARAM.

**Addr Short**: The 16-bit network address of the remote.

**Addr Long**: The 64-bit network address of the remote (MAC address).

**Length of Valid Data**: The length of valid data in response body.

**Response Body**: See appendix A for detail.

---

**API_DATA_PACKET**

API identifier value: 0x02

The API type “data packet” is a user defined data frame. This request message will send a block of data to the remote node. This request will not be responded except that the remote sends another data packet as a response.

![Figure 5.6: data packet request](image)

**Frame ID**: To Identify the data frame subsequence at the remote side.

**Option**:
AUPS Function list

**Bit0:** Reserved.

**Bit1:** Cast Mode, 0 unicast; 1 broadcast.

**Addr Short:** For the packet to send, this is the 16-bit network address of the remote, if this address is set to 0xffff, the long address (MAC) will be used. For the packet received, this is the 16-bit network address of the data sender.

**Addr Long:** For the packet to send, this is the 64-bit network address of the remote (MAC address), this address can be filled with zero if short address is specified. For the packet received, this is the 64-bit MAC address of the data sender.

**Length of Valid Data:** The length of valid data in data body.

**Body:** The data which will be transmitted to the remote node.

---

**API_TOPO_REQ**

API identifier value: 0xfb

This API type allows module to query the network’s topology, it will cause the module broadcasting a ping to the whole network, and all the nodes which listens to this network will respond a API_TOPO_RESP frame.

![Required Cmd](image)

*Figure 5.7: Structure of network Topo Require*

**Required Cmd:** Fixed to 0x01.

---

**API_TOPO_RESP**

API identifier value: 0x6b

In response to an “API_TOPO_REQ” message, the module will send a response message.

![Structure of network topology response](image)

*Figure 5.8: Structure of network topology response*
AUPS Function list

**Link Quality:** Indicate link quality of node, it’s a positive integer, the bigger the better link quality, it will be 305 when signal intensity is 0dbm.

**Dbm:** Signal intensity, \( dbm = (lqi - 305)/3 \).

**Firmware Version:** Firmware Version.

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

**Mac Address Low:** MAC address of the node, low 32bit.

**Mac Address High:** MAC address of the node, high 32bit.
6. AUPS Function list

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

6.1 Set run-time parameters

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 or BROADCAST

unicastDest: short address of unicast
buf: the pointer of data
len: length of the data
AUPS Function list

Return:

OK
ERR

API_bSendToMacDev

bool API_bSendToMacDev(uint64 unicastMacAddr, uint8 srcEpId, uint8 dstEpId, char *buf,
int len);

Description:
This function can be used to send RF packets.

Parameter:
unicastMacAddr: dest mac address
srcEpId: source end point
dstEpId: dest end point
buf: the pointer of data
len: length of the data

Return:
OK
ERR

6.3 Receive RF data

aupsAirPortReadable

PUBLIC uint32 aupsAirPortReadable(void);

Description:
This function returns the available bytes in AUPS’s airport ringbuffer.

Parameter:
none

Return:
the number of available bytes
aupsAirPortRead

PUBLIC uint8 aupsAirPortRead(void *dst, int len);

Description:
This function reads len bytes of data to dst from AUPS’s airportringbuffer.

Parameter:
dst: pointer to the destination Address
len: data len

Return:
real number of bytes you read

6.4 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:
one

suli_pin_dir

void _suli_pin_dir(IO_T *pio, DIR_T dir);

Description:
AUPS Function list

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**

```c
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**

```c
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**

```c
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 macro
  - A1 : ADC1
  - A2 : ADC2
  - A3 : ADC3
  - A4 : ADC4
  - TEMP: On-chip temperature ADC
  - VOL : On-chip voltage ADC

**Return:**
- none

**suli_analog_read**

```c
int16 suli_analog_read(ANALOG_T*aio);
```

**Description:**
This function can be used to read the ADC value.

**Parameter:**
AUPS Function list

aio: pointer of the ANALOG_T entity

**Return:**
ADC value

### suli_i2c_init

```c
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

```c
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**

```c
uint8 suli_i2c_read(void *i2c_device, uint8 dev_addr, uint8 *buff, uint8 len);
```

**Description:**
This function can be used to read a buffer 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**

```c
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:**
one
**suli_uart_send**

```c
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**

```c
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**

```c
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:**
AUPS Function list

uart_device: any dummy value
uart_num: any dummy value
data: float data

Return:
none

**suli_uart_write_int**

```c
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**

```c
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

6.5 RPC function

RPC_vCaller

PUBLIC void RPC_vCaller(uint64 mac, char* cmd);
AUPS Function list

**Description:**
User can use this function to make an Rpc call.

**Parameter:**
- mac: mac address of the server node
- cmd: rpc command string

**Return:**
- none

6.6 Tools function

**random**

PUBLIC uint16 random();

**Description:**
This function is used to generate a 16bits random number.

**Parameter:**
- none

**Return:**
- random number
## Appendix A: AT Command Index and API Frame Structure

### Version v1004

<table>
<thead>
<tr>
<th>Command</th>
<th>Index</th>
<th>Description</th>
<th>Request Frame (4 Byte)</th>
<th>Response Frame (20 Byte)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATRB</td>
<td>0x30</td>
<td>Reboot the node</td>
<td>Byte[3:0]: 0x00</td>
<td>None</td>
</tr>
<tr>
<td>ATPA</td>
<td>0x32</td>
<td>Power up action</td>
<td>Byte0: 0x00 0x00 0x00 0x00</td>
<td>Byte[1:0]: uint16, register value</td>
</tr>
<tr>
<td>ATAJ</td>
<td>0x34</td>
<td>Auto join network</td>
<td>Byte[0:3]: 0x00 0x00 0x00 0x00</td>
<td>Byte[1:0]: uint16, register value</td>
</tr>
<tr>
<td>ATRS</td>
<td>0x36</td>
<td>Re-Scan network</td>
<td>Byte[3:0]: 0x00</td>
<td>Byte[1:0]: uint16, register value</td>
</tr>
<tr>
<td>ATLN</td>
<td>0x38</td>
<td>List scanned network, only for Router and End Device</td>
<td>Byte[3:0]: 0x00</td>
<td>Multiple response frame may be returned, each has a structure like this: Byte0: uint8, index of this response Byte1: Uint8, radio channel Byte2: Uint8, permitted to join Byte[6:3]: Uint32, PAN ID low 32bit Byte[10:7]: Uint32, PAN ID high 32bit</td>
</tr>
<tr>
<td>ATJN</td>
<td>0x40</td>
<td>Join specified network</td>
<td>Byte0: 0x00 0x00 0x00 0x00</td>
<td>String, error message The result of joining can be obtained from the status byte.</td>
</tr>
<tr>
<td>ATRJ</td>
<td>0x41</td>
<td>Rejoin the last network</td>
<td>Byte[3:0]: 0x00</td>
<td>Byte[1:0]: uint16, register value</td>
</tr>
<tr>
<td>ATTM</td>
<td>0x44</td>
<td>Transmit mode</td>
<td>Byte0: 0x00 0x00 0x00 0x00</td>
<td>Byte[1:0]: uint16, register value</td>
</tr>
<tr>
<td>Command</td>
<td>Address</td>
<td>Function</td>
<td>Byte 0 Action</td>
<td>Byte 1 Action</td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
<td>----------</td>
<td>---------------</td>
<td>---------------</td>
</tr>
<tr>
<td>ATDA</td>
<td>0x46</td>
<td>Unicast address</td>
<td>0 – read the register</td>
<td>1 – write the register</td>
</tr>
<tr>
<td>ATBR</td>
<td>0x48</td>
<td>Set baud rate for UART1</td>
<td>0 – read the register</td>
<td>1 – write the register</td>
</tr>
<tr>
<td>ATQT</td>
<td>0x50</td>
<td>Query on-chip temperature</td>
<td>0x00</td>
<td></td>
</tr>
<tr>
<td>ATIF</td>
<td>0x54</td>
<td>Query node information</td>
<td>0x00</td>
<td></td>
</tr>
<tr>
<td>ATIO</td>
<td>0x70</td>
<td>Read/write IO</td>
<td>0 – read/write flag 0-write 1-read</td>
<td>Uint8, pin number, see appendix B</td>
</tr>
<tr>
<td>ATAD</td>
<td>0x72</td>
<td>Sample the ADC</td>
<td>0x00</td>
<td></td>
</tr>
</tbody>
</table>

**Byte0:**
- Uint8, node role, 0 – COO, 1 – ROU, 2 – END

**Byte1:**
- Uint8, Radio channel
- Uint16, firmware version
- Uint16, short address
- Uint16, pan id belonging
- Uint32, MAC low 32bit
- Uint32, MAC high 32bit

**Byte2:**
- The state of this IO currently

**Byte3:**
- 0x00
# Appendix B: IO Index

## Digital IO Index

<table>
<thead>
<tr>
<th>IO Name</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>D0</td>
<td>0x00</td>
</tr>
<tr>
<td>D1</td>
<td>0x01</td>
</tr>
<tr>
<td>D2</td>
<td>0x02</td>
</tr>
<tr>
<td>D3</td>
<td>0x03</td>
</tr>
<tr>
<td>D4</td>
<td>0x04</td>
</tr>
<tr>
<td>D5</td>
<td>0x05</td>
</tr>
<tr>
<td>D6</td>
<td>0x06</td>
</tr>
<tr>
<td>D7</td>
<td>0x07</td>
</tr>
<tr>
<td>D8</td>
<td>0x08</td>
</tr>
<tr>
<td>D9</td>
<td>0x09</td>
</tr>
<tr>
<td>D10</td>
<td>0x0a</td>
</tr>
<tr>
<td>D11</td>
<td>0x0b</td>
</tr>
<tr>
<td>D12</td>
<td>0x0c</td>
</tr>
<tr>
<td>D13</td>
<td>0x0d</td>
</tr>
<tr>
<td>D14</td>
<td>0x0e</td>
</tr>
<tr>
<td>D15</td>
<td>0x0f</td>
</tr>
<tr>
<td>D16</td>
<td>0x10</td>
</tr>
<tr>
<td>D17</td>
<td>0x11</td>
</tr>
<tr>
<td>D18</td>
<td>0x12</td>
</tr>
<tr>
<td>D19</td>
<td>0x13</td>
</tr>
<tr>
<td>D20</td>
<td>0x14</td>
</tr>
<tr>
<td>D00</td>
<td>0x21</td>
</tr>
<tr>
<td>D01</td>
<td>0x22</td>
</tr>
</tbody>
</table>

## ADC Source Index

<table>
<thead>
<tr>
<th>ADC Source</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>A3</td>
<td>0x00</td>
</tr>
<tr>
<td>A4</td>
<td>0x01</td>
</tr>
<tr>
<td>A2</td>
<td>0x32</td>
</tr>
<tr>
<td>A1</td>
<td>0x33</td>
</tr>
<tr>
<td>TEMP</td>
<td>0x34</td>
</tr>
</tbody>
</table>