SIM868 Series_Embedded AT_Application Note

GPRS Module

SIMCom Wireless Solutions Limited
Building B, SIM Technology Building, No.633, Jinzhong Road
Changning District, Shanghai P.R. China
Tel: 86-21-31575100
support@simcom.com
www.simcom.com

Document Title:	SIM868 Series_Embedded AT_Application Note
Version:	1.01
Date:	2020.6.15
Status:	Released

GENERAL NOTES

SIMCOM OFFERS THIS INFORMATION AS A SERVICE TO ITS CUSTOMERS, TO SUPPORT APPLICATION AND ENGINEERING EFFORTS THAT USE THE PRODUCTS DESIGNED BY SIMCOM. THE INFORMATION PROVIDED IS BASED UPON REQUIREMENTS SPECIFICALLY PROVIDED TO SIMCOM BY THE CUSTOMERS. SIMCOM HAS NOT UNDERTAKEN ANY INDEPENDENT SEARCH FOR ADDITIONAL RELEVANT INFORMATION, INCLUDING ANY INFORMATION THAT MAY BE IN THE CUSTOMER'S POSSESSION. FURTHERMORE, SYSTEM VALIDATION OF THIS PRODUCT DESIGNED BY SIMCOM WITHIN A LARGER ELECTRONIC SYSTEM REMAINS THE RESPONSIBILITY OF THE CUSTOMER OR THE CUSTOMER'S SYSTEM INTEGRATOR. ALL SPECIFICATIONS SUPPLIED HEREIN ARE SUBJECT TO CHANGE.

COPYRIGHT

SIMCom Wireless Solutions Limited
Building B, SIM Technology Building, No.633 Jinzhong Road, Changning District, Shanghai P.R. China
Tel: +86 21 31575100
Email: simcom@simcom.com

For more information, please visit:
https://www.simcom.com/download/list-863-en.html

For technical support, or to report documentation errors, please visit:
https://www.simcom.com/ask/ or email to: support@simcom.com

About Document

Version History

Version	Date	Owner	What is new
V1.00	2016.10.20	Bin.Mao	Origin
V1.01	2020.06.15	Fan.Fang	All

Scope

This document describes the development guide of Embedded AT and relative notes.

The document is applicable to SIM868 Embedded AT module.

Contents

About Document **3**
Version History 3
Scope 3

Contents **4**

1 Embedded AT Introduction **6**
1.1 Overview 6
1.2 Code Style 6
1.3 Sources Supplied by Embedded AT 6

2 Embedded AT Basic Conception **8**

3 Multi Threads **9**
3.1 Multi Threads Function Description 9
3.2 Tread Introduction 9
3.3 Message 11
3.3.1 Message Definition 11
3.3.2 Interface Definition 11
3.3.3 Note 13

4 Timer Application **14**
4.1 Overview 14
4.2 Function Interface and Example 14
4.3 Note 16

5 Memory Application **17**
5.1 Function Description 17
5.2 Function Interface and Example 17
5.3 Note 18

6 Sleep **19**

7 Serial Port interface **20**
7.1 Function Description 20
7.2 Message 20
7.3 Function Interface and Example 21
7.4 Serial Port Data Flow 25
7.5 Note 25

8 Flash Allocation **28**
8.1 Function Description 28
8.2 Function Interface and Example 28
8.3 Space allocation 28
8.4 APP upgrade 29

8.5 Note 31

9 File System 32

9.1 Function introduction 32
9.2 Function Interface 32
9.3 Note 33

10 Peripheral Interface 34

10.1 Function Introduction 34
10.2 Relative Function Interface and Example 34
10.2.1 GPIO Read/write and control Interface 34
10.2.2 Module Interrupt configuration Interface 36
10.2.3 SPI Interface 37
10.2.4 PWM Output Control Interface 39
10.2.5 ADC Interface 39
10.2.6 PowerKey , LED Control Interface 40

11 Audio 42

11.1 Function Introduction 42
11.2 Function Interface and Example 42
11.3 Note 43

12 GPS Function 45

12.1 Function Introduction 45
12.2 Related function interface with examples 45
12.3 Caution items for use 45

13 Socket 46

13.1 Function Introduction 46

14 SMS 47

14.1 Function Introduction 47

1 Embedded AT Introduction

1.1 Overview

Embedded AT is mainly used for customer to do the secondary development on SIM868 series modules. SIMCom provides the relative API functions, resource and operating environment. Customer's APP code runs inside SIM868 series modules, so external host will be saved.

1.2 Code Style

Embedded AT can basically implement customer's code, and supports multiple threads that allows customer to run different subtask.

1.3 Sources Supplied by Embedded AT

The main API functions are listed as following:

API Type	Functions
Audio API	Display the audio in AMR&MIDI format, and generate the customized-frequency sound
AT Commands API	Mainly used for the AT interaction between client's code and module's code
Flash API	Some API function about Flash, such as erase , program, update application code
System API	Includes such functions: send and receive message, power off and restart, allocate dynamic memory, capture event, semaphore, inquire firmware version.
Peripherals API	Includes such functions: SPI, GPIO control, external interruption, PWM and ADC.
Timer API	Timer's control and relative time management, which includes timer manipulation, RTC configuration and time function.

File system API	The interface to manipulate the file: read/write/create/delete file, create/delete folder, acquire the file's information.
Serial Port API	Read/Write serial port
Debug API	Open/close the Debug port, output date to Debug port, print customer's debug information to Debug port.
Update API	Used to update the firmware, customer can download the new firmware to module by network, and then update the firmware by API function.
GPS	Including gprs power-off & power-down, startup mode, nmea output setting, GPRS sleep mode, GPS geo fence.

2 Embedded AT Basic Conception

The software architecture of Embedded AT is as following:

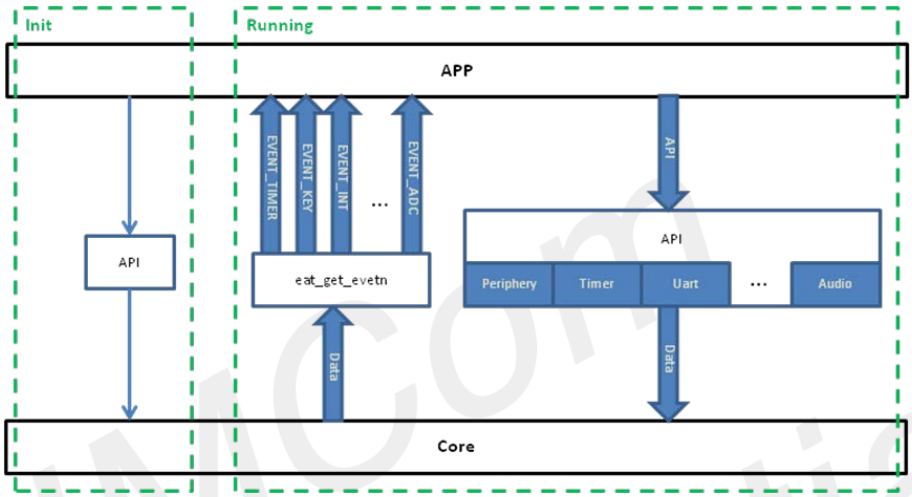


Figure 1General Software Architecture

Illustration:

Embedded AT consists of two parts, one is the main program namely core system, another is customers' program namely Embedded application. Core system provides the module's main features and API for customer program. The main features include standard AT commands, file system manipulation, timer control, peripheral API and some common system API.

Note:

EAT (named in following chapters) is the abbreviation of Embedded AT.

3 Multi Threads

3.1 Multi Threads Function Description

The platform provides multi threads function, supports one main thread and 8 sub threads for now , mainly used to communicate with system such as receive system event .

The suspended thread which has a high priority will be called prior than the running thread which has a low priority once the condition is triggered.

3.2 Tread Introduction

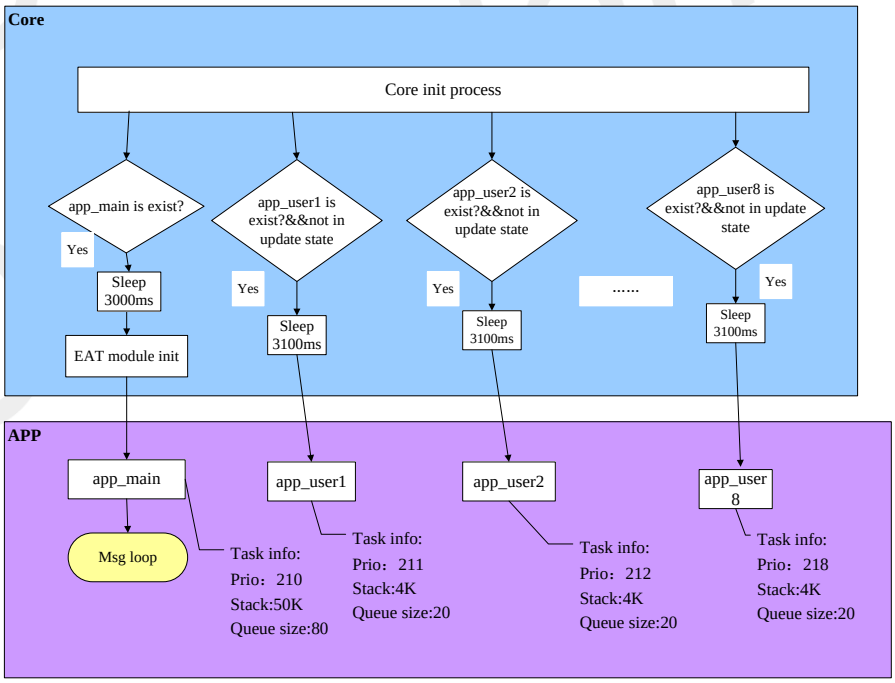


Figure 2 Tread Initialization Information

Illustration:

➢ **The Corresponding structure in main.c**

```
#pragma arm section rodata = "APP_CFG"
APP_ENTRY_FLAG
#pragma arm section rodata

#pragma arm section rodata="APPENTRY"

constEatEntry_stAppEntry =
{
    app_main,
    app_func_ext1,
    (app_user_func)EAT_NULL,//app_user1,
    (app_user_func)EAT_NULL,//app_user2,
    (app_user_func)EAT_NULL,//app_user3,
    (app_user_func)EAT_NULL,//app_user4,
    (app_user_func)EAT_NULL,//app_user5,
    (app_user_func)EAT_NULL,//app_user6,
    (app_user_func)EAT_NULL,//app_user7,
    (app_user_func)EAT_NULL,//app_user8,
};
#pragma arm section rodata
```

➢ **Task Description**

There have 9 threads for user app, they are EAT_USER_0 to EAT_USER_8.
If the member in struct EatEntry_st is configured, also system is not in upgrade process, then this entrance will be called, and task related message will be allocated.

Following example shows app_main, app_func_ext1, app_user1 and app_user3 will be called.

```
constEatEntry_stAppEntry =
{
    app_main,
    app_func_ext1,
    (app_user_func)app_user1, //app_user1,
    (app_user_func)app_user2, //app_user2,
    (app_user_func)app_user3, //app_user3,
    (app_user_func) EAT_NULL, //app_user4,
    (app_user_func) EAT_NULL, //app_user5,
    (app_user_func) EAT_NULL, //app_user6,
    (app_user_func) EAT_NULL, //app_user7,
    (app_user_func) EAT_NULL, //app_user8,
     ……
};
```

EAT_USER_0 (EatEntry_st.entry) stack size and queue size is 10k bytes and 80 respectively. From

EAT_USER_1 to EAT_USER_4, stack size and queue size is 10K bytes and 50 respectively. And stack size is 2k bytes, queue size is 20 in other threads.
The priority degrades successively, i.e. app_main>app_user1>...>app_user8.

Note:

Do not to use large array in thread to avoid stack overflow.

3.3 Message

3.3.1 Message Definition

Function Interface	Function
EAT_EVENT_TIMER	Timer message
EAT_EVENT_KEY	Key message
EAT_EVENT_INT	External GPIO interruption triggered message
EAT_EVENT_MDM_READY_RD	EAT receives data sent from Modem
EAT_EVENT_MDM_READY_WR	Reported message once Modem buffer turning into non-full status from full status
EAT_EVENT_MDM_RI	Reserving currently
EAT_EVENT_UART_READY_RD	Serial port receive data
EAT_EVENT_UART_READY_WR	Reported message once buffer of serial buffer turning into non-full status from full status
EAT_EVENT_ADC	ADC message
EAT_EVENT_UART_SEND_COMPLETE	Message that indicates underlying hardware data of serial ports has been sent successfully
EAT_EVENT_USER_MSG	Reported message once receiving message from other threads.

3.3.2 Interface Definition

The following two functions, which can only be used in app_main, is to acquire message sent from core, or acquire message the EAT_EVENT_USER_MSG sent from app_user task by eat_send_msg_to_user.

Function Interface :

Function Interface	Function

eat_get_event	Acquire the queue message
eat_get_event_num	Acquire the queue message number

Example:

```
EatEvent_st event;
u32 num;
void app_main(void *data)
{
	eat_get_event(&event);
	num=eat_get_event_num();
	if(event.event == EAT_EVENT_UART_SEND_COMPLETE)
	{
	}
}
```

For the 8 tasks which can be used by customer, i.e. app_user1, app_user2, etc, functions which correspond to above features are as following:

Function Interface:

Function Interface	Function
eat_get_event_for_user	Acquire the queue message
eat_get_event_num_for_user	Acquire the queue message number

Example:

```
void app_user1(void *data)
{
	u32 num;
	EatEvent_st event;
	while(1)
	{
		num= eat_get_event_num_for_user(EAT_USER_1);
		eat_get_event_for_user(EAT_USER_1, &event);

		if(event.event == EAT_EVENT_USER_MSG)
		{
		}
	}
}
```

3.3.3 Note

- To send message, function eat_send_msg_to_user can be used in app_main and app_user.

- System message can be distributed to sub-thread, adhering the principle that message will be send to the thread which called the API.

For example, if users call eat_uart_open (EAT_UART_1) to open uart 1 in user1, then the message EAT_EVENT_UART_READY_RD (event.uart.uart=EAT_UART_1) will be to user1 once uart1 receive date.

For example, if users call eat_time_start (EAT_TIMER_1) in app_main and call eat_timer_start (EAT_TIMER_2) in user1, message EAT_EVENT_TIMER (event.timer.timer_id = EAT_TIMER_1) will be forwarded to app_main once EAT_TIMER1 is triggered, and message EAT_EVENT_TIMER (event.timer.timer_id = EAT_TIMER_2) will be forwarded to user1 once EAT_TIMER_2 is triggered.

For example, if we call eat_modem_write ("AT\r\n", 4) to send AT command to Core in user1, then the AT command's execution result will be forwarded to user1 by message EAT_EVENT_MDM_READY_RD. Afterwards the URC report would also be forwarded to user1 until other thread call eat_modem_write. Such as send AT command in user2, then the AT command execution result and URC report will be forwarded to user2.

The AT URC message (EAT_EVENT_MDM_READY_RD) in the start-up process is forwarded to main thread as default, and can use eat_modem_set_poweron_urc_dir() to configure that forwarding power on URC to assigned thread .

- eat_get_event or eat_get_event_for _user is synchronous interface. Once customer calls this interface, response will return immediately if there is event in the thread, and if there is not event, the thread will be suspended.

If customer doesn't need to suspend the thread, then can use eat_get_event_num() or eat_get_event_num_for_user(EAT_USER_x) to acquire the event number in this thread's event queue. If the event number is 0, then don't call eat_get_event_for _user interface. Customer would call this interface if the number is above 0.

4 Timer Application

4.1 Overview

EAT provides timer interfaces for following usage:

- Timers for customer. There are two kinds of timers, 16 ms-grade timers and 1 μs-grade timer;
- Interface to set thread sleep;
- Interface to set and get system time & date;
- Interface to get the time difference between two points.

4.2 Function Interface and Example

EVENT :

EAT_EVENT_TIMER

Struct :

```
typedefstruct {
unsigned char sec; /* [0, 59] */
unsigned char min; /* [0,59]    */
unsigned char hour; /* [0,23]    */
unsigned char day; /* [1,31]    */
unsigned char mon; /* [1,12] */
unsigned char wday; /* [1,7] */
unsigned char year; /* [0,127] */
} EatRtc_st;
```

Callback Function:

```
typedef void (*eat_gpt_callback_func)(void);
```

Function Interface:

Function Interface	Function
eat_timer_start/eat_timer_stop	Start and terminate ms-grade timer
eat_gpt_start /eat_gpt_stop	Start and terminate μs-grade timer
eat_sleep	Thread sleep
eat_get_current_time	Acquire the current time
eat_get_duration_us eat_get_duration_ms	Acquire the time interval
eat_get_rtc/ eat_set_rtc	Set and acquire the system time

Example:

Application for ms-grade timer

```
//Start the timer
eat_timer_start(EAT_TIMER_1, 100);
//Get timer EVENT
eat_get_event(&event);
if( EAT_EVENT_TIMER == event.event)
{
    //do something
}
```

Application for μs-grade timer

```
voidgpt_time_handle(void)
{
    //do something...
}
eat_gpt_start(, EAT_FALSE, gpt_time_handle);
```

Acquire the time interval

```
unsignedint time =  eat_get_current_time();
    //do something
unsignedinttime_ms = eat_get_duration_ms(time);
```

Set and acquire system time

```
EatRtc_strtc;
eat_set_rtc(&rtc);
rtc.year = 12;
eat_get_rtc(&rtc);
```

4.3 Note

- eat_gpt_start is a hardware timer , will be executed timer callback function in interrupt . So timer callback function should not occupy too much time, should not use blocking function , such as sleep , memory allocation , signal , etc .

- Timer may affect sleep function. System in sleep mode would be woken up once the timer is out.

5 Memory Application

5.1 Function Description

EAT platform provides an interface by managing an array to realize memory initialization, allocation and release. Memory space which needs to be applied and released dynamically could be defined by array flexibly.

The maximum memory space size can be applied at a time is N (The value of N is 168 currently, may be different based on different module.).

The memory and global variables both occupy app's RAM space. For the size of RAM space, please refer to chapter Flash allocation.

5.2 Function Interface and Example

Function Interface:

Function Interface	Function
eat_mem_init	Initialize memory block
eat_mem_alloc	Apply memory
eat_mem_free	Release memory
APP_InitRegions	Initialize ZI section and RW section for APP

Example:

```
#define DYNAMIC_MEM_SIZE 1024*400
static unsigned char app_dynic_mem_test[DYNAMIC_MEM_SIZE]; /* space , used to initialize memory
void* mem_prt=EAT_NULL;

/ * initialize memory */
eat_mem_init(app_dynic_mem_test, sizeof(app_dynic_mem_test));

/* apply memory */
```

```
mem_prt = eat_mem_alloc(size);
...

/*release memory */
eat_mem_free(mem_prt);

/*Initialize ZI section and RW section for APP*/
voidapp_main(void *data)
{
      // Local variables defined
      //.....
App_initRedions;// Init app RAM, first step
App_initclib();    //C library initialize,second step
      //...
}
```

5.3 Note

- APP_InitRegions is used to initialize ZI section and RW section for app in EAT TASK0. The global data can be read and write only after the function is executed.
- APP_init_clib is implemented in EAT TASK0 for C library environment of APP. If the function is not executed for the C library initialization, the C library function is not correct.
- APP_init_clib and APP_InitRegions can only and must be executed once in TASK0 EAT.

6 Sleep

API eat_sleep_enable is to enable or disable the system sleep mode. And it is disabled as default.

Enable the system to into sleep mode

```
eat_sleep_enable( EAT_TRUE );
```

Disable the system to enter into sleep mode

```
eat_sleep_enable(EAT_FALSE );
```

During sleep mode, module will be woken up regularly and automatically by network paging.

There have only key event, GPIO interrupt, timer, incoming SMS and call could wake up module in sleep mode. For detail, please refer to document “SIM868 Series Embedded AT Sleep Application_V1.0”.

7 Serial Port interface

7.1 Function Description

Serial port API functions could do following works:

- Configure Serial port parameter ;
- Transfer data via UART;
- Configure UART mode;

Reserve 2 ports for app, either of them could be specified as AT port or debug port. Different module has different ports.

7.2 Message

EAT_EVENT_MDM_READY_RD

During the process of modem sending data to EAT, the TX buffer status changing from empty status to non-empty status will trigger a message to EAT. This TX buffer size is 5K bytes.

EAT_EVENT_MDM_READY_WR

During the process of Modem receiving data from EAT, RX buffer changing from full status to non-full status will trigger a message to EAT. This RX buffer size is 5K bytes.

EAT_EVENT_UART_READY_RD

When UART receives data, RX buffer changing from empty status to non-empty status will trigger this message to EAT. This RX buffer size is 2K bytes.

EAT_EVENT_UART_READY_WR

When UART sends data, TX buffer changing from full status to non-full status will trigger this message to EAT. This TX buffer is 2K bytes.

EAT_EVENT_UART_SEND_COMPLETE

When UART sends data, TX buffer changing from non-empty status to empty status and the empty status of DMA FIFO will trigger this message.

7.3 Function Interface and Example

Enumeration Varialble:

```
typedefenum {
    EAT_UART_1,
    EAT_UART_2,
    EAT_UART_3,
    EAT_UART_NUM,
    EAT_UART_NULL = 99
} EatUart_enum;

typedefenum {
    EAT_UART_BAUD_1200          =1200,
    EAT_UART_BAUD_2400          =2400,
    EAT_UART_BAUD_4800          =4800,
    EAT_UART_BAUD_9600          =9600,
    EAT_UART_BAUD_19200         =19200,
    EAT_UART_BAUD_38400         =38400,
    EAT_UART_BAUD_57600         =57600,
    EAT_UART_BAUD_115200        =115200,
    EAT_UART_BAUD_230400        =230400,
    EAT_UART_BAUD_460800        =460800
} EatUartBaudrate;

typedefenum {
      EAT_UART_DATA_BITS_5=5,
      EAT_UART_DATA_BITS_6,
      EAT_UART_DATA_BITS_7,
      EAT_UART_DATA_BITS_8
} EatUartDataBits_enum;

typedefenum {
      EAT_UART_STOP_BITS_1=1,
      EAT_UART_STOP_BITS_2,
      EAT_UART_STOP_BITS_1_5
```

```
} EatUartStopBits_enum;

typedefenum {
        EAT_UART_PARITY_NONE=0,
        EAT_UART_PARITY_ODD,
        EAT_UART_PARITY_EVEN,
        EAT_UART_PARITY_SPACE
} EatUartParity_enum;
```

Struct:

```
typedef struct {
        EatUart_enumuart;
} EatUart_st;

typedefstruct {
        EatUartBaudrate    baud;
        EatUartDataBits_enumdataBits;
        EatUartStopBits_enumstopBits;
        EatUartParity_enum    parity;
} EatUartConfig_st;
```

Function Interface:

Function Interface	Function
eat_uart_open eat_uart_close	Open and close serial port
eat_uart_set_config eat_uart_get_config	Set and acquire the parameter of serial port
eat_uart_set_baudrate eat_uart_get_baudrate	Set and acquire the transmitting baud rate
eat_uart_write eat_uart_read	Send and receive data
eat_uart_set_debug	Set debug port
eat_uart_set_at_port	Set Core system and AT port
eat_uart_set_send_complete_event	Set whether enable or disable the report of sending data completely
eat_uart_get_send_complete_status	Acquire the status of sending data completely
eat_uart_get_free_space	Acquire the remaining space size for send buffer
eat_modem_write eat_modem_read	Send date to MODEM or receive data from MODEM
eat_uart_control_by_app	Set app control of DTR, RI and DCD pin function.
eat_uart_power_off	Power off the serial port

Example:

Open and close serial port

```
// open serial port
eat_uart_open(EAT_UART_1);

// close serial port
eat_uart_close (EAT_UART_1);
```

Set the parameter for serial port

```
EatUartConfig_st uart_config;

    uart_config.baud = 115200;
    uart_config.dataBits = EAT_UART_DATA_BITS_8;
    uart_config.parity = EAT_UART_PARITY_NONE;
    uart_config.stopBits = EAT_UART_STOP_BITS_1;

    eat_uart_set_config(EAT_UART_1, &uart_config);
```

Set app control of DTR,RI and DCD pin function.

```
eat_uart_control_by_app_core(EAT_TRUE);
```

Set serial port power-off

```
Eat_uart_power_off (EAT_UART_1);
```

Acquire the parameter for serial port

```
EatUartConfig_st uart_config;

eat_uart_get_config(EAT_UART_1, &uart_config);
```

Set the baud rate

```
eat_uart_set_baudrate (EAT_UART_1, EAT_UART_BAUD_115200);
```

Acquire the baud rate

```
EatUartBaudrate baudrate;

baudrate = eat_uart_get_baudrate (EAT_UART_1);
```

Send and receive data

```
u16len;
u8 rx_buf[EAT_UART_RX_BUF_LEN_MAX];

//receive data
len = eat_uart_read(EAT_UART_1, rx_buf, EAT_UART_RX_BUF_LEN_MAX);
if(len != 0)
{
      // send data
      eat_uart_write(EAT_UART_1, rx_buf, len);
}
```

Set function port

```
eat_uart_set_at_port(EAT_UART_1);
eat_uart_set_debug(EAT_UART_2);
```

Set whether enable the report of sending data completely

```
//Enable the report of sending data completely
eat_uart_set_send_complete_event (EAT_UART_1, EAT_TRUE);

//Disable the report of sending data completely
eat_uart_set_send_complete_event (EAT_UART_1, EAT_FALSE);
```

Acquire the status of sending data completely

```
eat_bool status;
status = eat_uart_get_send_complete_status (EAT_UART_1);
```

Acquire the remaining space size of sending buffer

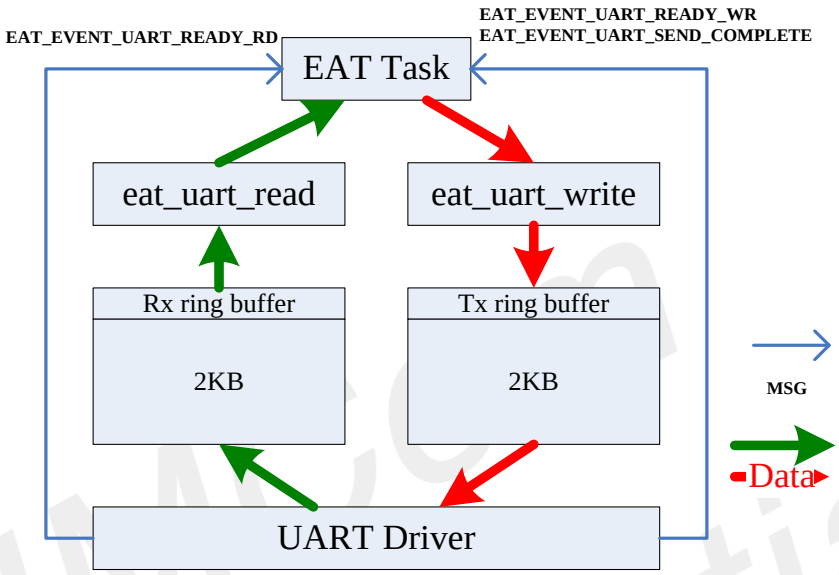
```
unsigned short size;
size = eat_uart_get_free_space (EAT_UART_1);
```

Data transmitting between EAT and MODEM

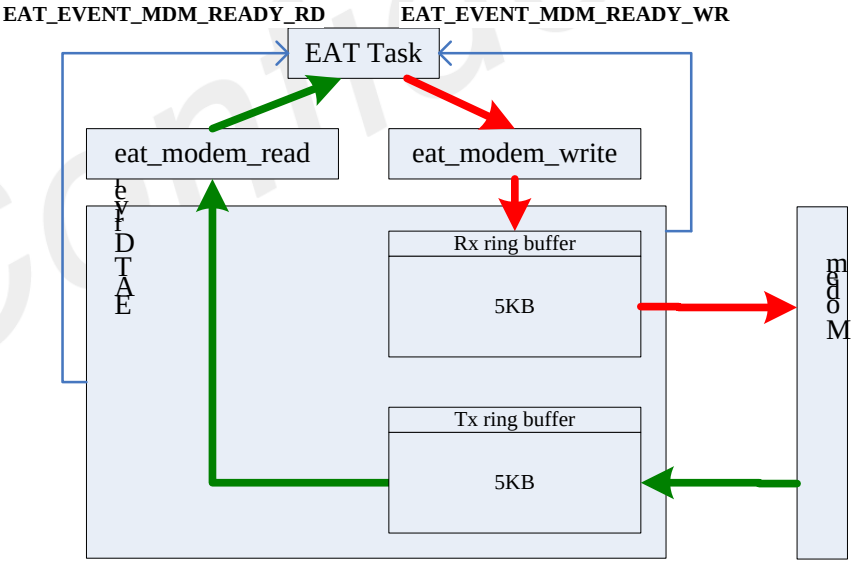
```
u16 len;
u8 rx_buf[5120];

//Receive data
len = eat_modem_read (EAT_UART_1, rx_buf, 5120);
//Send ata
eat_modem_write (EAT_UART_1, rx_buf, len);
```

7.4 Serial Port Data Flow

Flow Chart of Serial Port data and message in EAT Application



Flow Chart of Serial Port Data and Message between EAT and Modem



7.5 Note

- Once APP receives the message from EAT_EVENT_MDM_READY_RD or EAT_EVENT_UART_READY_RD, it would read the data from RX buffer with eat_modem_read or eat_uart_read interface. If there has data unread in buffer, then READY_RD message will not report when receiving new data.

- The following functions should be used after eat_uart_open and before eat_uart_clode :
  - eat_uart_set_config
  - eat_uart_get_config
  - eat_uart_set_baudrate
  - eat_uart_get_baudrate
  - eat_uart_write
  - eat_uart_read
  - eat_uart_set_send_complete_event
  - eat_uart_get_send_complete_status
  - eat_uart_get_free_space

- Following functions should be used in initialization phase (in the member function func_ext1 of structEatEntry_st)
  - eat_uart_set_debug
  - eat_uart_set_at_port
  - eat_uart_set_debug_config
  - eat_uart_set_at_port_baud

- eat_uart_control by appeat_uart_control_by_app must be used after closing port by shutting down the serial port. with TX RX being low-level. The serial port can be power-off only in idle status.

- Set eat_uart_control_by_app to TRUE,when DTR,RI and DCD is used by APP as other features,to make module process no relevant initializations of the pins. If the pins are not used as other functions, this setting can be ignored.

- EAT API eat_get_version() or eat_get_buildno() could get FW version, also could use AT command to read as following.

at+cgmr
Revision:1116B02SIM840W64_WINBOND_EMBEDDEDAT
OK

at+csub
+CSUB: V01
OK

8 Flash Allocation

8.1 Function Description

There have interfaces for flash writing, erasing and app upgrade.

8.2 Function Interface and Example

Function Interface

Function Interface	Function
eat_flash_erase	Erase FLASH block
eat_flash_write	Write data to FLASH
eat_update_app	Upgrade app
eat_get_app_base_addr	Get app base address
eat_get_app_space	Get the space size of app
eat_get_flash_block_size	Get the size of FLASH block

8.3 Space allocation

FLASH allocation of standard EAT version is shown in the following table. The address configuration may be different based on customer's requirement.

The base address and space size of app can be acquired by interfaces **eat_get_app_base_addr()** and **eat_get_app_space()**.

Following tables show different flash allocation based on different modules.

Version SIM868

Section	Start Address	End Address	Size (Byte)
Modem	10000000	102FFFFF	3M (0x300000)
APP	90300000	9037FFFF	512K (0x80000)
FS	10380000	103FE000-1	504K (0x7E000)
RAM	F0380000	F03EFFFF	448K (0x70000)

APP: customer's app code and ROM space

FS: File system space includes system parameter, calibration parameter, etc. File system space supplied to customer is also contained in this space. The system occupies 200K space, so size of space customer can use is equal to file system size minus 200K.

Note:

- Customer has different requirements for different platform, so the address configuration of Flash and RAM space size may also be different. Please refer to the released version.

8.4 APP upgrade

App could be updated during module in working status.

There has a flag for app upgrade status. Only after app upgraded completely finished, this flag will be clear. Supposed the process was interrupted in middle, after next reboot, app upgrade process will continue from beginning. So, this protection will make module recovery from abnormal status.

In the end of process, module will reboot app, and pass parameter to app_main.

app upgrade flow is as following:

```
u32 APP_DATA_RUN_BASE;        //running address of app
u32  APP_DATA_STORAGE_BASE;  //storage  address  of                                    app
updating code
const unsigned char app_new_data[] = {
#include app_new
};//data of updating code

void update_app_test_start（）
{
        ……..
    //Acquire address
    APP_DATA_RUN_BASE = eat_get_app_base_addr(); // acquire app address
```

```
    app_space_value = eat_get_app_space(); //acquire app space size
    APP_DATA_STORAGE_BASE = APP_DATA_RUN_BASE + (app_space_value>>1); // save the address of app updating file

    // erase the flash space area of updating storage address
    eat_flash_erase(APP_DATA_STORAGE_BASE,update_app_data_len);
    ……..
    // download the updating program into flash space area , the starting address is APP_DATA_STORAGE_BASE

    eat_flash_write(APP_DATA_STORAGE_BASE, app_new_data, app_datalen);
      ……
     // update
    eat_update_app((void*)(APP_DATA_RUN_BASE),          (void*)(APP_DATA_STORAGE_BASE), app_datalen, EAT_PIN_NUM, EAT_PIN_NUM, EAT_FALSE);
}
```

The updating process after calling eat_update_app is as following:

- Write the related parameters into the APP update flag area;
- Reboot module, check the value of APP update flag, move the updating program in the address of APP_DATA_STORAGE_BASE to the app running address APP_DATA_RUN_BASE , and set the value of flag.
- Reboot module again and run new app code. Module will check the parameter from app_main(void param) and judge the result of app upgrade process, then clear the flag in APP updata_flag area.

```
voidapp_main(void *data)
{
    EatEvent_st event;
    EatEntryPara_st *para;

    APP_InitRegions();//Init app RAM

    para = (EatEntryPara_st*)data;

    memcpy(&app_para, data, sizeof(EatEntryPara_st));
    if(app_para.is_update_app&&app_para.update_app_result)
    {
        //APP update succeed
        eat_update_app_ok(); //clear update APP flag
    }
    .....
}
```

Customer should clear update flag by calling eat_updata_app_ok() in new app_main code.
If not, module will write data in APP_DATA_STORAGE_BASE to APP_DATA_DATA_RUN_BASE.

8.5 Note

- Block is the basic unit for flash operation. The address for eat_flash_erase(const void*address, unsigned int size) should be in integral multiple of block. If not, EAT system will handle it, and set the operating address from the starting address of the block..

If size is not in integral multiple of block, module will erase (size/block size)+1 pcs of block.

The interface to get flash block size is eat_get_flash_block_size.

- if the block was written before, customer should erase first before writing again.

9 File System

9.1 Function introduction

Support related interfaces for file system for corresponding operation.

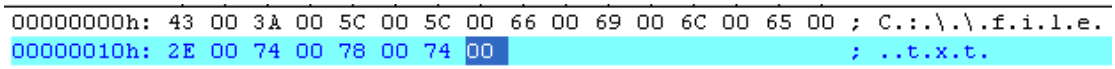
9.2 Function Interface

Following are interface API functions.

Function Interface	Function
eat_fs_Open	Open or create file
eat_fs_Close	Close the opened file
eat_fs_Read	Read file
eat_fs_Write	Write file
eat_fs_Seek	Seek the file pointer
eat_fs_Commit	Push file to disk
eat_fs_GetFileSize	Acquire the file size
eat_fs_GetFilePosition	Acquire the current file's pointer
eat_fs_GetAttributes	Acquire the file's attribute
eat_fs_SetAttributes	Configure the file's attribute
eat_fs_Delete	Delete file
eat_fs_CreateDir	Create file directory
eat_fs_RemoveDir	Delete file directory
eat_fs_Truncate	Truncate file
eat_fs_GetDiskFreeSize	Acquire the size of remained file system space
eat_fs_GetFolderSize	Acquire file size
eat_fs_FindFirst	Research the first instance corresponding to the specified file.
eat_fs_FindNext	Research the follow instance corresponding to the specified file.
eat_fs_FindClose	Close the handle created by the function eat_fs_FindFirst
eat_fs_XDelete	Delete the files and directory in batch.
eat_fs_Abort	Terminate the related operations.

9.3 Note

- There have only four kinds of file operations which include FS_READ_WRITE , FS_READ_ONLY , FS_CREATE and FS_CREATE_ALWAYS intrerfaces. It supports to open or create maximum 24 files at the same time. The created file name need two-byte alignment and in UCS2 code. For example, customer can't use "C:\\file.txt" to open a file directly but have to use L"C:\\file.txt". The actual value is :



The example of converting char to unicode

```
for(i=0;i<filename_len;i++)
{
    filename_l[i*2] = filename[i];
    filename_l[i*2+1] = 0x00;
}
```

- Eat_fs_GetDiskFreeSize supports to acquire the remained space size of inner file system and T-Flash. It may get corresponding error value if there is no external T-Flash .

- eat_fs_Write has no size limitation for writing data at one time, but it would make error once write data more than the remained space size of file system or SD card. The ctual length of data written in file system is referring to last parameter return of this interface.

- The drive of inner file system is "C", root directory is "C:\\". The drive of T_Flash is "D:", root directory is "D:\\" . If customer operate files without drive name, module will use inner flash memory.

10 Peripheral Interface

10.1 Function Introduction

EAT provides some API functions to operate peripheral interfaces, like LCD, Keypad, ADC and so on.

- GPIO read/write control interface
- Interrupt configuration interface
- Analog SPI interface
- PWM output control interface
- ADC read interface
- Powerkey , LED control interface

10.2 Relative Function Interface and Example

10.2.1 GPIO Read/write and control Interface

Below is the enumeration definition for SIM868W PIN, please refer to the the definition in eat_periphery.h for other platform (module).

```
typedef enum {
    EAT_PIN1_UART1_TXD    = 1,    /* GPIO, UART1_TXD, EINT */
    EAT_PIN2_UART1_RXD    = 2,    /* GPIO, UART1_RXD, EINT */
    EAT_PIN3_UART1_RTS    = 3,    /* GPIO, KCOL1, UART1_RTS */
    EAT_PIN4_UART1_CTS    = 4,    /* GPIO, KCOL2, EINT, UART1_CTS */
    EAT_PIN5_UART1_DCD    = 5,    /* GPIO, SPI_CS */
    EAT_PIN6_UART1_DTR    = 6,    /* GPIO, KCOL3, EINT, PWM0,SPI_SCK */
    EAT_PIN7_UART1_RI     = 7,    /* GPIO, SPI_MOSI*/
    EAT_PIN14_SIM1_DET    = 14, /* GPIO, KROW1, EINT ,SPI_MISO*/
```

```
    EAT_PIN22_UART2_TXD = 22, /* GPIO, KROW4, UART2_TXD */
    EAT_PIN23_UART2_RXD = 23, /* GPIO, KCOL4, UART2_RXD */
    EAT_PIN38_ADC       = 38, /* ADC */
    EAT_PIN41_NETLIGHT  = 41, /* GPIO */
    EAT_PIN42_STATUS    = 42, /* GPIO, SPI_DC */
    EAT_PIN52_SIM2_DET  = 52, /* GPIO, KROW3, EINT  */
    EAT_PIN57_GPIO1     = 57, /* GPIO, KROW2, PWM1*/
    EAT_PIN58_GPIO2     = 58, /* GPIO, KROW0 */
    EAT_PIN64_SDA       = 64, /* GPIO, SDA */
    EAT_PIN65_SCL       = 65, /* GPIO, SCL */
    EAT_PIN_NUM         = 66
} EatPinName_enum;

typedefenum {
    EAT_PIN_MODE_GPIO,
    EAT_PIN_MODE_KEY,
    EAT_PIN_MODE_EINT,
    EAT_PIN_MODE_UART,
    EAT_PIN_MODE_SPI,
    EAT_PIN_MODE_PWM,
    EAT_PIN_MODE_I2C,
    EAT_PIN_MODE_CLK,
    EAT_PIN_MODE_NUM
} EatPinMode_enum;

typedefenum {
    EAT_GPIO_LEVEL_LOW,
    EAT_GPIO_LEVEL_HIGH
} EatGpioLevel_enum;

typedefenum {
    EAT_GPIO_DIR_INPUT,
    EAT_GPIO_DIR_OUTPUT,
} EatGpioDir_enum;
```

Function Interface

Function Interface	Function
eat_gpio_setup	set PIN's GPIO attribute
eat_gpio_write	Write GPIO's electrical level
eat_gpio_read	Read GPIO's electrical level
eat_gpio_write_ext	write GPIO's electrical level (only available for some specific pin of SIM800W)
eat_pin_set_mode	set PIN's mode
eat_gpio_set_pull	Set GPIO status of pull-up and pull-down

Note:

- PIN status of pull-up and pull-down in hardware design should be taken notice.

10.2.2 Module Interrupt configuration Interface

EVENT：

EAT_EVENT_INT

Enumeration Definition：

```
typedef enum {
    EAT_INT_TRIGGER_HIGH_LEVEL, /* high level*/
    EAT_INT_TRIGGER_LOW_LEVEL, /* low level*/
    EAT_INT_TRIGGER_RISING_EDGE, /* rising edge*/
    EAT_INT_TRIGGER_FALLING_EDGE, /* falling edge*/
    EAT_INT_TRIGGER_NUM
} EatIntTrigger_enum; /* The GPIO EINT trigger mode */
```

Struct：

```
typedefstruct {
    EatPinName_enum pin; /* the pin */
    EatGpioLevel_enum level; /* 1-high level; 0-low level*/
} EatInt_st; /* EAT_EVENT_INT data struct*/
```

Callback Function：

```
typedef void (*eat_gpio_int_callback_func)(EatInt_st *interrupt);
```

Function Interface：

Function Interface	Function
eat_int_setup	set interrupt
eat_int_setup_ext	set interrupt, there is a parameter that wether or not auto set the trigger polarity
eat_int_set_trigger	set the trigger mode of interrupt
eat_sim_detect_en	Enable/Disable the interrupt funtion

Example:

```
/* define the interrupt callback function */
voidint_test_handler_func (EatInt_st *interrupt)
{
      //do something
}
/* set interrupt */
eat_int_setup(EAT_PIN45_GPIO3, EAT_INT_TRIGGER_LOW_LEVEL,
100, int_test_handler_func);
/* if the interrupt callback function is not defined (NULL) */
      //Get INT EVENT
      eat_get_event(&event);
      if( EAT_EVENT_INT == event.event)
      {
            //do something
      }
/* reconfigure the trigger condition of interrupt */
eat_int_set_trigger(EAT_PIN45_GPIO3, EAT_INT_TRIGGER_RISING_EDGE);
```

Note:

- The unit of parameter debounce_ms in eat_int_setup is 10ms. For example, it means 100ms if debounce_ms is equal to 10. This parameter debounce_ms is only effective to level-triggered interrupt.
- If customer wants to invert interrupt mode after pin triggered, then customer needs to configure the interrupt inversion in callback function.
- The response time of edge-triggered interrupt is about 1ms. The response time of level-triggered interrupt equals debounce_ms plus 1ms, it depends on the configuration of debounce_ms.
- If level-triggered interrupt mode and EVENT interrupt mode used, then customer should check current status of pin before configuration, and configure the different status. For example, pin's current status is LOW_LEVEL, then trigger level should be configured to HIGH_LEVEL. Otherwise the core will get continuous interrupt report which may cause module reboot. Or customer could try eat_int_setup_ext interface, and configure last parameter as opposite level.
- When using interrupt callback mode, do not use functions which will occupy long time or resource of system in callback function.

10.2.3 SPI Interface

Enumeration Definition

```
typedefenum {
    EAT_SPI_3WIRE, /* 3 wire SPI */
    EAT_SPI_4WIRE /* 4 wire SPI */
} EatSpiWire_enum;

typedef enum {
    EAT_SPI_CLK_52M = 1, /* 52M clock */
    EAT_SPI_CLK_26M = 2, /* 26M clock    */
    EAT_SPI_CLK_13M = 4    /* 13M clock    */
} EatSpiClk_enum; /* Can turn down the freg if you need ,the scope is 1~1024 */

typedefenum {
    EAT_SPI_BIT8, /* 8 bit */
    EAT_SPI_BIT9, /* 9 bit */
    EAT_SPI_BIT16, /* 16 bit */
    EAT_SPI_BIT24, /* 24 bit */
    EAT_SPI_BIT32    /* 32 bit */
} EatSpiBit_enum;
```

Function Interface:

Function Interface	Function
eat_spi_init	Initialize SPI configuration
eat_spi_write	Write SPI
eat_spi_read	Read SPI

Example:

```
/* initialize SPI configuration */
eat_spi_init(EAT_SPI_CLK_13M, EAT_SPI_4WIRE,
EAT_SPI_BIT8, EAT_FALSE, EAT_TRUE);
/* write data or commands */
static void lcd_write_cmd(unsigned char cmd)
{
    eat_spi_write(&cmd, 1, EAT_TRUE);
}
static void lcd_write_data(unsigned char data)
{
    eat_spi_write(&data, 1, EAT_FALSE);
}
/* read data of len bytes */
static void lcd_read_data(unsigned char *data,unsigned char len)
{
    eat_spi_read(data,len);
}
```

Note:

- If SPI operation is not controlled by external DISP_CS signal, customer should control this in app code,enable_cs should be false in eat_spi_init.

10.2.4 PWM Output Control Interface

Function Interface

Function Interface	Function
eat_pwm_start	Output PWM
eat_pwm_stop	Stop output PWM

Example：

```
/* output PWM */
eat_pwm_start(200,50);

/* stop output PWM */
eat_pwm_stop();
```

Note:

- Output cycle : 0 (all low) – 100 (all high) , Only PWM0 currently.

10.2.5 ADC Interface

EVENT：

EAT_EVENT_ADC

Struct :

```
typedefstruct {
    EatPinName_enum pin; /* the pin */
    unsignedint v; /* ADC value,unit is mv*/
} EatAdc_st;
```

Callback Function:

typedef void (*eat_adc_callback_func)(EatAdc_st *adc);

Function Interface:

Function Interface	Function
eat_adc_get	Read ADC(the unit is 2.3ms)
eat_adc_get_ms	Read ADC(the unit is ms)
eat_get_adc_sync	Read ADC(in synchronization)

Example:

```
/* define interrupt callback function */
voidadc_test_handler_func (EatAdc _st * adc)
{
      //do something
}
/* read ADC */
eat_adc_get(EAT_PIN6_ADC0, 3000, adc_test_handler_func);
/* if the interrupt callback function is not defined(NULLL) */
      //Get INT EVENT
eat_get_event(&event);
if( EAT_EVENT_ADC == event.event)
{
      //do something
}
```

Note:

- Only this pin EAT_PIN6_ADC0 is available to read ADC for SIM868W , the external voltage range is 0-2.8V .
- For the ADC PIN definition of other platform , please refer to the macro definition in eat_periphery.h of the corresponding platform .

10.2.6 PowerKey , LED Control Interface

Function Interface

Function Interface	Function
eat_lcd_light_sw	LCD backlight control interface
eat_kpled_sw	Keypad backlight control interface

eat_poweroff_key_sw	PoweKey control interface. If we don't enable this function, powering off the module by key is disabled as default.

Example :

```
/* light up LCD backlight */
eat_lcd_light_sw(EAT_TRUE,EAT_BL_STEP_04_MA);

/* light up keypad backlight */
eat_kpled_sw (EAT_TRUE);

/* enable powering off the module by long-pressing PowerKey */
eat_poweroff_key_sw(EAT_TRUE);
```

Note :

- System can't enter sleep mode if LCD backlight or keypad backlight is on .
- one second for long-pressing PowerKey would power off the module if eat_poweroff_key_sw(EAT_TRUE).

The module can't be powered off if USB connected or pressing PowerKey all the time, it would restart after 30 seconds. It will restart after 6s, if the app is installed.Or it will restart after 30s.

11 Audio

11.1 Function Introduction

It provides interfaces to play or stop tone (keypad tone, dial tone, busy tone, etc) and audio data flow (in format of MIDI and WAV).

11.2 Function Interface and Example

Function Interface:

Function Interface	Function
eat_audio_play_tone_id	play tone
eat_audio_stop_tone_id	stop play tone
eat_audio_play_data	play audio data flow in MIDI or WAV format
eat_audio_stop_data	stop play audio data flow
eat_audio_set_custom_tone	generate customized tone
eat_audio_play_file	Play audio file flow
eat_audio_stop_file	Stop audio file flow
eat_audio_set_play_mode_in_call	Set play mode in call (local, remote,local&remote)
eat_audio_get_play_mode_in_call	Get play mode in call (local, remote,local&remote)

Struct :

```
typedefstruct {
    unsigned short freq1;            /* First frequency */
    unsigned short freq2;            /* Second frequency */
    unsigned short on_duration;      /* Tone on duation, in ms unit, 0 for continuous tone */
    unsigned short off_duration;     /* Tone off duation, in ms unit, 0 for end of playing */
    unsigned char next_operation;    /*Index of the next tone */
} EatAudioToneData_st;
```

Example：

eat_audio_play_tone_id application

```
eat_audio_play_tone_id(EAT_TONE_DIAL_CALL_GSM, EAT_AUDIO_PLAY_INFINITE, 15, EAT_AUDIO_PATH_SPK1);
```

eat_audio_stop_id application

```
eat_audio_stop_tone_id(EAT_TONE_DIAL_CALL_GSM);
```

eat_audio_play_data application

```
const unsigned char audio_test_wav_data[] = { /* ring2.wav */
   0x52,0x49,0x46,0x46,0x62,0x9B,0x04,0x00,0x57,0x41,0x56,0x45,0x66,0x6D,0x74,0x20,
   0x10,0x00,0x00,0x00,0x01,0x00,0x01,0x00,0x40,0x1F,0x00,0x00,0x80,0x3E,0x00,0x00,
   ......
}
eat_audio_play_data(audio_test_wav_data, sizeof(audio_test_wav_data), EAT_AUDIO_FORMAT_WAV, EAT_AUDIO_PLAY_INFINITE , 12, EAT_AUDIO_PATH_SPK2);
```

eat_audio_stop_data application

```
eat_audio_stop_data();
```

eat_audio_play_file application

```
eat_audio_play_file(AUDIO_TEST_FILE,EAT_FALSE,eat_play_stop_cb,50,EAT_AUDIO_PATH_SPK1);
```

eat_audio_stop_file application

```
eat_audio_stop_file();
```

eat_audio_set_play_mode_in_call application

```
eat_audio_set_play_mode_in_call(EAT_AUDIO_PLAY_REMOTE);
```

eat_audio_get_play_mode_in_call application

```
eat_audio_get_play_mode_in_call();
```

11.3 Note

- The tone includes key tone and audio tone, and audio tone is prior to key tone. It would stop key tone if play audio tone.
- Call is prior to audio data flow, and audio data flow is prior to tone. So call would stop audio data flow playback, audio data flow would stop tone playback, also call would stop tone playback.
- Tone can be played but audio data flow can't in call process.
- The id of eat_audio_play_tone_id() and eat_audio_stop_tone_id() should match to avoid stopping incorrectly .
- Make sure the audio data flow is in correct format for playback, only MIDI and WAV format is supported for now.
- The audio file must be in file system during audio display, only MIDI and WAV format supported.
- The common tone frequency (refer to structEatAudioToneData_st) :

---Busy Tone : EAT_TONE_BUSY_CALL_GSM：{ { 425, 0, 500, 500, 0 } }
---Dial Tone : EAT_TONE_DIAL_CALL_GSM, { { 425, 0, 0, 0, 0 } }

12 GPS Function

12.1 Function Introduction

SIM868 EAT support a series of operation API, including gprs power-off & power-down, startup mode(cold-startup, warm-startup, and hot-startup), nmea output setting(supported mode: GPS+GLONASS, GPS ONLY, GPS+BEIDOU) ,GPRS sleep mode, GPS geo fence.

12.2 Related function interface with examples

Function interface:

Function Interface	Function
eat_gps_power_req	Turn off &on GPS
eat_gps_power_status	Get GPS power status
eat_gps_start_mode_get	Get GPS start mode
eat_gps_start_mode_set	Set GPS start mode
eat_gps_nmea_info_output	Set nmea output information type
eat_gps_sleep_set	Set GPS sleep mode
eat_gps_sleep_read	Get GPS status of sleep_mode
eat_gps_status_get	Get GPS fix mode
eat_gps_geo_fence_set	Set GPS geo fence
eat_gps_geo_fence_read	Get parameters of GPS geo fence.

12.3 Caution items for use

For implementing eat_gps_power_req interface, PIN gps_en and PIN57 should be joined together. Or GPS cannot be triggered.

13 Socket

13.1 Function Introduction

Users can take GPRS data transmission and implement TCP and UDP capabilities through AT commands.

14 SMS

14.1 Function Introduction

Users can send, read and delete messages through AT commands, referring related examples in app_demo_sms.c.