SIM800 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:	SIM800 Series_Embedded AT_Application Note
Version:	1.04
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	2012.10.20	Bin.Mao	Origin
V1.01	2013.11.10	Bin.Mao	1.Add the application note for GPIO's reading/writing and controlling interface 2.Modifiy the enumeration definition of GPIO's reading/writing and controlling interface
V1.02	2014.05.30	Ping.Zhang Dingfen.Zhu	1.Add the Socket chapter 2.Add the SMS chapter
V1.03	2015.07.30	Bin.Mao	Add SIM800C project
V1.04	2020.06.15	Fan.Fang	

Scope

This document is applicable to SIM800 series Embedded AT module, include SIM800W, SIM840W, SIM800V, SIM800H, SIM800, SIM800M64, SIM808 and SIM800C.

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

Contents

1 Introduction

1.1 Purpose of the document

Based on module AT command manual, this document will introduce Embedded AT application process.

Developers could understand and develop application quickly and efficiently based on this document.

1.2 Related documents

[1] SIM800 Series_AT Command Manual

1.3 Conventions and abbreviations

In this document, the GSM engines are referred to as following term:
ME (Mobile Equipment);
MS (Mobile Station);
TA (Terminal Adapter);
DCE (Data Communication Equipment) or facsimile DCE (FAX modem, FAX board);

In application, controlling device controls the GSM engine by sending AT Command via its serial interface.
The controlling device at the other end of the serial line is referred to as following term:
TE (Terminal Equipment);
DTE (Data Terminal Equipment) or plainly "the application" which is running on an embedded system;

2 Embedded AT Introduction

2.1 Overview

Embedded AT is mainly used for customer to do the secondary development on SIM800 series modules. SIMCom provides the relative API functions, resource and operating environment. Customer's APP code runs inside SIM800 series modules, so external host will be saved.

2.2 Code Style

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

2.3 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.
Socket API	Socket function.
SMS API	Receive, read, write and send SMS.

3 Embedded AT Basic Conception

The software architecture of Embedded AT is as following:

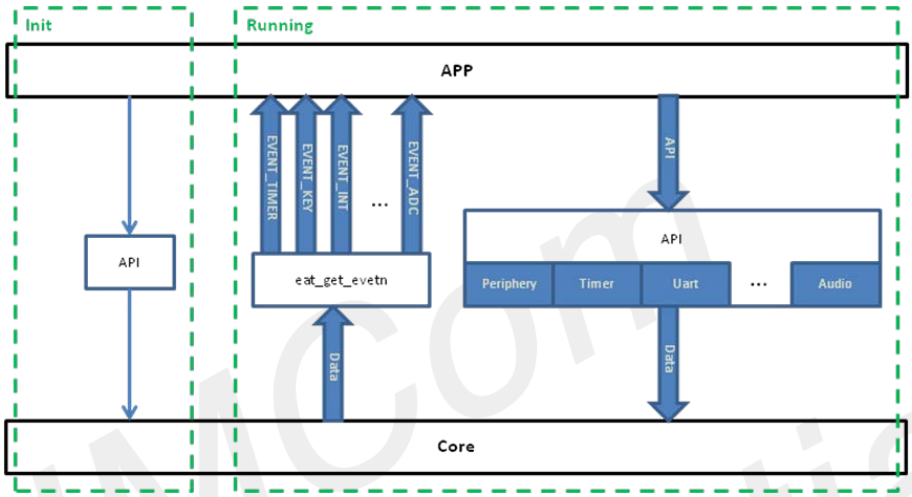


Figure 1General Software Architecture

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.

4 Multi Threads

4.1 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.

4.2 Tread Introduction

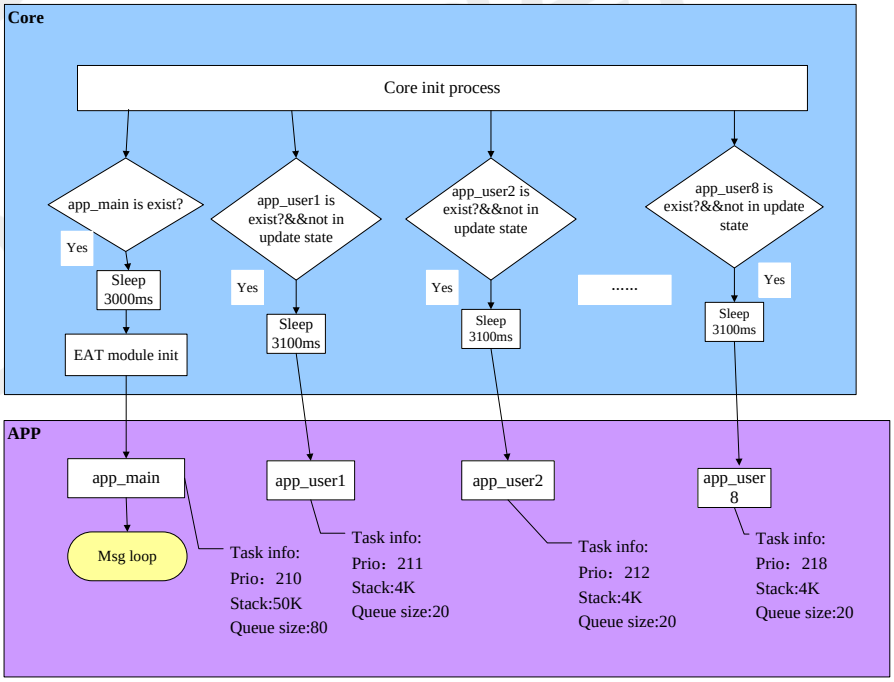


Figure 2 Tread Initialization Information

Illustration:

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

```
/* Add APP_CFG in SIM800H and SIM800 platform EAT app begin*/
#pragma arm section rodata = "APP_CFG"
APP_ENTRY_FLAG
#pragma arm section rodata
/* Add APP_CFG in SIM800H and SIM800 platform EAT app end*/

#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,
    ……
};
```

In SIM800W and SIM800V, EAT_USER_0 thread supports stack size 50k bytes, queue size is 80. Other threads supports stack space 4K Byte, queue size 20.

Otherwise, include SIM800H, SIM800, SIM800M64, SIM808 and SIM800C, EAT_USER_0 (EatEntry_st.entry) stack size is 10k bytes, queue size is 80. From EAT_USER_1 to EAT_USER_4, stack size is 10K bytes, queue size is 50. 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.

4.3 Message

4.3.1 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	Forward message once Modem's receive buffer turn into non-full status from full status
EAT_EVENT_MDM_RI	Reserve for now
EAT_EVENT_UART_READY_RD	Serial port receive data
EAT_EVENT_UART_READY_WR	Forward message once serial port's receive buffer turn into non-full status from full status
EAT_EVENT_ADC	ADC message
EAT_EVENT_UART_SEND_COMPLETE	Message indicates serial port's underlying hardware data sent successfully
EAT_EVENT_USER_MSG	Forward message once a thread receive other threads' message
EAT_EVENT_IME_KEY	Info of IME (Only available on SIM800W)

4.3.2 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	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)
            {
            }
        }
}
```

4.3.3 Note

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

- SIM800H, SIM800, SIM800M64, SIM808 and SIM800C system message can be sent to sub thread, and the principle is that message will be send to the thread in which the API is called.

For example, if customer 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 forwarded to user1 once uart1 receive date .

For example, if customer 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 call 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.

5 Timer Application

5.1 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.

5.2 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 stop ms-grade timer
eat_gpt_start /eat_gpt_stop	Start and stop 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);
```

5.3 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.

6 Memory Application

6.1 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.

6.2 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
      //...
}
```

6.3 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.

7 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 “SIM800 Series Embedded AT Sleep Application”.

8 Serial Port interface

8.1 Function Description

Serial port API functions could do following works:

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

Reserve 3 ports for app, either of them could be specified as AT port or debug port. Different module has different port, for example, there have 2 UART ports and 1 USB port in SIM800H module.

8.2 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.

8.3 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:

```
typedefstruct {
        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

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);
```

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

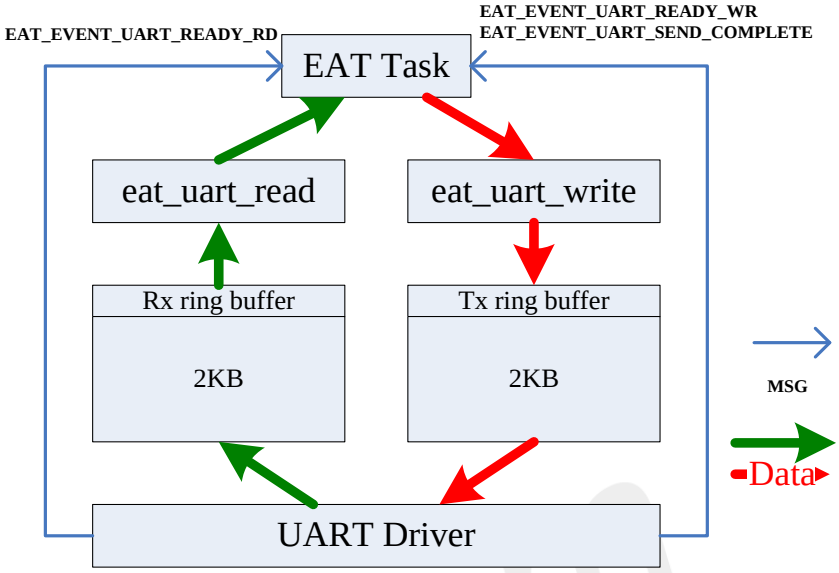
```
unsigned short size;
size = eat_uart_get_free_space (EAT_UART_1);
```

Data transmitting between EAT and MODEM

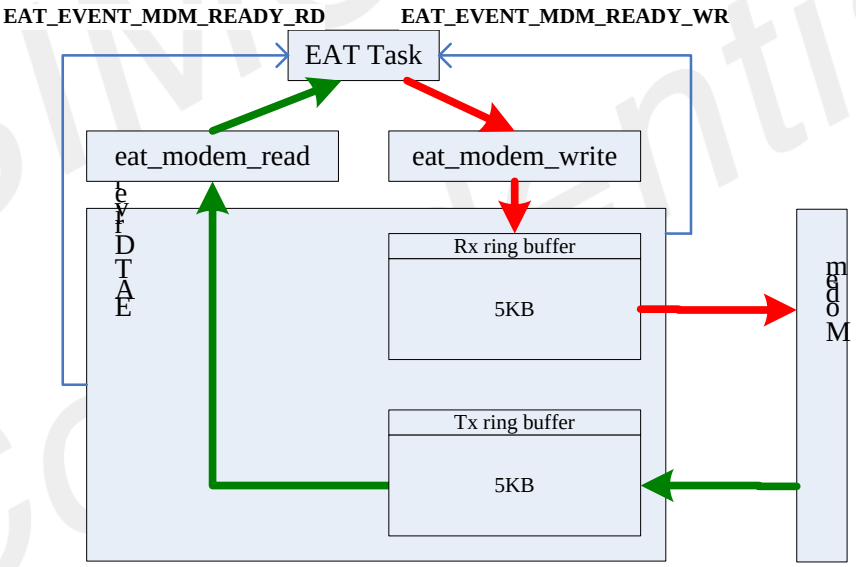
```
u16 len;
u8 rx_buf[5120];

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

8.4 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



8.5 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

➢ API eat_uart_set_at_port is not available before firmware version 1116B02V01.
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

9 Flash Allocation

9.1 Function Description

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

9.2 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

9.3 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.

SIM800W 64 M Versions:

Section	Start Address	End Address	Size (Byte)
Modem	08000000	083FFFFF	4M(0x400000)
APP	08400000	0867FFFF	2.5M (0x280000)
FS	08680000	087BFFFF	1.25M (0x140000)
RAM	F0220000	F03FFFFF	1.875M (0x1E0000)

SIM800V 128 M Version

Section	Start Address	End Address	Size (Byte)
Modem	08000000	084FFFFF	5M (0x500000)
APP	08500000	086FFFFF	2M (0x200000)
FS	08700000	08EEFFFF	7.9M (0x7F0000)
RAM	F0220000	F03FFFFF	1.875M (0x1E0000)

SIM800H Version

Section	Start Address	End Address	Size (Byte)
Modem	10000000	101FFFFF	2M (0x200000)
APP	10200000	1037FFFF	1.5M (0x180000)
FS	10380000	103FDFFF	504K (0x7E000)
RAM	F0380000	F03FFFFF	512K (0x80000)

SIM800 32M Version

Section	Start Address	End Address	Size (Byte)
Modem	10000000	101FFFFF	2M (0x200000)
APP	10200000	1037FFFF	1.5M (0x180000)
FS	10380000	103FDFFF	504K (0x7E000)
RAM	F0380000	F03FFFFF	512K (0x80000)

SIM800 64M Version

Section	Start Address	End Address	Size (Byte)
Modem	10000000	103FFFFF	4M (0x200000)
APP	10400000	1066FFFF	2.43M (0x270000)
FS	10680000	107BFFFF	1.25M (0x140000)
RAM	F0600000	F077FFFF	1.5M (0x180000)

SIM800 BT 64M Version

Section	Start Address	End Address	Size (Byte)

Modem	10000000	105BFFFF	5.625M (0x5A0000)
APP	905A0000	9066FFFF	832K (0xD0000)
FS	10680000	107BFFFF	1.25M (0x140000)
RAM	F0600000	F077FFFF	1.5M (0x180000)

SIM800C 32M Version

Section	Start Address	End Address	Size (Byte)
Modem	10000000	10308FFF	3.1M (0x309000)
APP	90309000	9039BFFF	588K (0x93000)
FS	1039E000	103FE000-1	384K(0x60000)
RAM	F0380000	F03EFFFF	448K (0x70000)

SIM808 Version

Section	Start Address	End Address	Size (Byte)
Modem	10000000	10308FFF	3.1M (0x309000)
APP	90309000	9039BFFF	588K (0x93000)
FS	1039E000	103FE000-1	384K(0x60000)
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.

9.4 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.

9.5 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.

10 File System

10.1 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

10.2 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 :

```
00000000h: 43 00 3A 00 5C 00 5C 00 66 00 69 00 6C 00 65 00 ; C.:.\.\.f.i.l.e.
00000010h: 2E 00 74 00 78 00 74 00                         ; ..t.x.t.
```

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.

11 Peripheral Interface

11.1 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

11.2 Relative Function Interface and Example

11.2.1 GPIO Read/write and control Interface

Below is the enumeration definition for SIM800W PIN, please refer to the the definition in eat_periphery.h for other platform (module).

```
typedefenum     {
    EAT_PIN6_ADC0 = 6,            // ADC
    EAT_PIN8_GPIO1 = 8,           // GPIO
    EAT_PIN9_I2C_SDA = 9,         // GPIO, KEY_ROW, EINT, I2C_SDA
    EAT_PIN10_I2C_SCL = 10,       // GPIO, KEY_COL, I2C_SCL
    EAT_PIN11_KPLED = 11,         // KPLED
    EAT_PIN16_NETLIGHT = 16,    // PWM
    EAT_PIN28_GPIO2 = 28,         // GPIO, EINT
```

```
    EAT_PIN29_KBC5 = 29,          // GPIO, KEY_COL, EINT
    EAT_PIN30_KBC4 = 30,          // GPIO, KEY_COL
    EAT_PIN31_KBC3 = 31,          // GPIO, KEY_COL
    EAT_PIN32_KBC2 = 32,          // GPIO, KEY_COL
    EAT_PIN33_KBC1 = 33,          // GPIO, KEY_COL
    EAT_PIN34_KBC0 = 34,          //KEY_COL
    EAT_PIN35_KBR5 = 35,          // GPIO, KEY_ROW, EINT
    EAT_PIN36_KBR4 = 36,          // GPIO, KEY_ROW
    EAT_PIN37_KBR3 = 37,          // GPIO, KEY_ROW
    EAT_PIN38_KBR2 = 38,          // GPIO, KEY_ROW
    EAT_PIN39_KBR1 = 39,          // GPIO, KEY_ROW
    EAT_PIN40_KBR0 = 40,          // GPIO, KEY_ROW, SPI_LSDI
    EAT_PIN45_GPIO3 = 45,         // GPIO, EINT
    EAT_PIN46_DISP_DATA = 46, // GPIO, SPI_LSDA
    EAT_PIN47_DISP_CLK = 47,    // GPIO, SPI_SCK
    EAT_PIN48_DISP_RST = 48,    // GPIO
    EAT_PIN49_DISP_DC = 49,     // GPIO, KEY_ROW, SPI_LSA
    EAT_PIN50_DISP_CS = 50,     // GPIO, SPI_LSCE
    EAT_PIN51_VDD_EXT = 51,     // VDD_EXT
    EAT_PIN52_PCM_SYNC = 52,    // GPIO
    EAT_PIN53_PCM_IN = 53,      // GPIO
    EAT_PIN54_PCM_CLK = 54,     // GPIO
    EAT_PIN55_PCM_OUT = 55,     // GPIO
    EAT_PIN57_GPIO4 = 57,         // GPIO, KEY_COL, EINT
    EAT_PIN58_RXD3 = 58,          // GPIO, UART3
    EAT_PIN59_TXD3 = 59,          // GPIO, UART3
    EAT_PIN60_RXD = 60,           // UART1
    EAT_PIN61_TXD = 61,           // UART1
    EAT_PIN62_DBG_RXD = 62,     // GPIO, UART2
    EAT_PIN63_DBG_TXD = 63,     // GPIO, UART2
    EAT_PIN65_LCD_LIGHT = 65, // LCD_LIGTH
    EAT_PIN_NUM = 68
} 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

Example

```
/*set PIN52 as GPIO output mode , initialized to low electrical level */
eat_gpio_setup(EAT_PIN52_PCM_SYNC,
  EAT_GPIO_DIR_OUTPUT, EAT_GPIO_LEVEL_LOW);

/* set PIN52 to high electrical level
eat_gpio_write(EAT_PIN52_PCM_SYNC, EAT_GPIO_LEVEL_HIGH);

/*read PIN52*/
eat_gpio_read (EAT_PIN52_PCM_SYNC);

/* set PIN53 to high level */
eat_gpio_write_ext(EAT_PIN53_PCM_IN, EAT_GPIO_LEVEL_HIGH);

/* set PIN40 as key value mode */
eat_pin_set_mode(EAT_PIN40_KBR0, EAT_PIN_MODE_KEY);
```

Note:

- eat_gpio_write_ext interface is available in SIM800W only. Compared to eat_gpio_write ,eat_gpio_write_ext has a faster speed to write . eat_gpio_write_ext may take about 1μs (microsecond) to write which is only half of the time eat_gpio_write takes. eat_gpio_write_ext doesn't do the fault-tolerant check and it is only effective to some specific pins of SIM800W. The pins supported for now are as following :

EAT_PIN8_GPIO1,
EAT_PIN9_I2C_SDA,
EAT_PIN10_I2C_SCL,
EAT_PIN52_PCM_SYNC,
EAT_PIN53_PCM_IN,
EAT_PIN54_PCM_CLK,
EAT_PIN55_PCM_OUT ,
EAT_PIN57_GPIO4,
EAT_PIN58_RXD3
EAT_PIN59_TXD3,
EAT_PIN62_DBG_RXD
EAT_PIN63_DBG_TXD,

- Customer should pay attention to some pins' status of pull-up and pull-down in hardware design. For example, the following 4 pins of SIM800W series modules should be noticed carefully, whose status may affect the start-up of module.

  - PIN16, PIN34, PIN52 should not be forced to pull-up or pull-down before module start up.

  - If PIN64 is used to detect the SIM card status (AT+CSDT=1), this pin should be pulled up to VDD_EXT by 10k~100k resistor. If PIN64 is not used to detect the SIM card (AT+CSDT=0), this pin can be kept open. We could use AT&W to save the configuration of AT+CSDT, then it would follow the configuration when the module start up next time.

Pin	Name	Status before start up	The possible abnormity caused by changing electrical level compulsively in the start-up process	Explanation
16	NETLIGHT	Low	can't start up	NETLIGHT has been pulled down inner module already
34	KBC0	High	can't start up	KBCO can't be pulled down before start up .Otherwise , Module would enter into USB download mode and can't start up
52	PCM_SYNC	High	can't start up	pulled up inner module already
64	GPIO5	High	restart after start up , read the SIM card abnormally , fail to register to network	If GPIO5 is used as SIM card detection , it should be pulled up to VDD_EXT by 10-100K resistor externally , can't keep it open

11.2.2 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 weather or not auto set the trigger polarity
eat_int_set_trigger	set the trigger mode of interrupt

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.

11.2.3 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.

11.2.4 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)

11.2.5 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

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 SIM800W , 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 .

11.2.6 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);// there is one more parameter for SIM800H and SIM800 which is used to set the current magnitude */
/* 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.

11.2.7 KEYPAD

The message of keypad is forwarded by EAT_EVENT_KEY.

EVENT：

EAT_EVENT_KEY

Key Value Enumeration Definition:

```
typedefenum {
    EAT_KEY_C0R0,
    EAT_KEY_C0R1,
    EAT_KEY_C0R2,
    EAT_KEY_C0R3,
    EAT_KEY_C0R4,
    EAT_KEY_C0R5,
    EAT_KEY_C1R0,
    EAT_KEY_C1R1,
    EAT_KEY_C1R2,
    EAT_KEY_C1R3,
    EAT_KEY_C1R4,
    EAT_KEY_C1R5,
    EAT_KEY_C2R0,
    EAT_KEY_C2R1,
    EAT_KEY_C2R2,
    EAT_KEY_C2R3,
    EAT_KEY_C2R4,
    EAT_KEY_C2R5,
```

```
    EAT_KEY_C3R0,
    EAT_KEY_C3R1,
    EAT_KEY_C3R2,
    EAT_KEY_C3R3,
    EAT_KEY_C3R4,
    EAT_KEY_C3R5,
    EAT_KEY_C4R0,
    EAT_KEY_C4R1,
    EAT_KEY_C4R2,
    EAT_KEY_C4R3,
    EAT_KEY_C4R4,
    EAT_KEY_C4R5,
    EAT_KEY_C5R0,
    EAT_KEY_C5R1,
    EAT_KEY_C5R2,
    EAT_KEY_C5R3,
    EAT_KEY_C5R4,
    EAT_KEY_C5R5,
    EAT_KEY_POWER,
    EAT_KEY_NUM
} EatKey_enum;

/* EAT KEY configuration structure. */
typedefstruct {
    EatKey_enumkey_value; /* key value */
    eat_boolis_pressed; /* 1-key press down; 0-key release up */
} EatKey_st;
```

Example:

```
voidapp_main(void *data)
{
    EatEvent_st event;
    eat_get_event(&event);
    switch(event.event)
    {
        ....
        case EAT_EVENT_KEY:
            eat_trace("Get key value:%d pressed:%d", event.data.key.key_value, event.data.key.
is_pressed);
        break;
        ....
    }
```

12 Audio

12.1 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).

12.2 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

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();
```

12.3 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 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 } }

13 Socket

13.1 Function Introduction

Platform supports SOCKET interface, customer can use it to create TCP or UDP connection and transfer data.

13.2 Function Interface and Example

SOCKET supports 3 type interfaces, they are GPRS bear, SOCKET about, and DNS query. The GPRS bear is basic, SOCKET and DNS is based on it。

Function Interface：

Function Interface	Function
eat_gprs_bearer_open	Open GPRS bear
eat_gprs_bearer_hold	Hold GPRS bear
eat_gprs_bearer_release	Release GPRS
eat_soc_create	Create socket
eat_soc_notify_register	Register call back function while socket notify
eat_soc_connect	Connects to the server
eat_soc_setsockopt	Sets the socket options.
eat_soc_getsockopt	Gets the socket options.
eat_soc_bind	Bind local port
eat_soc_listen	makes a socket to a server socket to wait client connections
eat_soc_accept	waits for the incoming connections and return a socket id of new connection.
eat_soc_send	Send the data to the destination
eat_soc_recv	Receive data and return the source address
eat_soc_sendto	Send the data to the destination, more use TCP connection
eat_soc_recvfrom	Receive data and return the source address, more use UDP connection
eat_soc_getsockaddr	Get local IP address

eat_soc_close	Close Socket
eat_soc_gethostbyname	gets the IP of the given domain name
eat_soc_gethost_notify_register	set call back function about eat_soc_gethostbyname

Call back function Type：

```
typedef void(*eat_soc_notify)(s8 s,soc_event_enumevent,eat_bool result, u16 ack_size);
//socket event notify
typedef void(*eat_bear_notify)(cbm_bearer_state_enum state,u8 ip_addr[4]);
//gprs bear state notify
typedef void(*eat_hostname_notify)(u32 request_id,eat_bool result,u8 ip_addr[4]);
//DNS query notify
```

Struct：

```
/* Bearer state */
typedefenum
{
    CBM_DEACTIVATED              = 0x01, /* deactivated */
    CBM_ACTIVATING               = 0x02, /* activating */
    CBM_ACTIVATED                = 0x04, /* activated */
    CBM_DEACTIVATING             = 0x08, /* deactivating */
    CBM_CSD_AUTO_DISC_TIMEOUT    = 0x10, /* csd auto disconnection timeout */
    CBM_GPRS_AUTO_DISC_TIMEOUT   = 0x20, /* gprs auto disconnection timeout */
    CBM_NWK_NEG_QOS_MODIFY       = 0x040, /* negotiated network qos modify notification */
    CBM_WIFI_STA_INFO_MODIFY     = 0x080, /* wifi hot spot sta number is changed */
    CBM_BEARER_STATE_TOTAL
} cbm_bearer_state_enum;
/* Socket return codes, negative values stand for errors */
typedefenum
{
    SOC_SUCCESS          = 0,    /* success */
    SOC_ERROR            = -1,   /* error */
    SOC_WOULDBLOCK       = -2,   /* not done yet */
    SOC_LIMIT_RESOURCE   = -3,   /* limited resource */
    SOC_INVALID_SOCKET   = -4,   /* invalid socket */
    SOC_INVALID_ACCOUNT  = -5,   /* invalid account id */
    SOC_NAMETOOLONG      = -6,   /* address too long */
    SOC_ALREADY          = -7,   /* operation already in progress */
    SOC_OPNOTSUPP        = -8,   /* operation not support */
    SOC_CONNABORTED      = -9,   /* Software caused connection abort */
    SOC_INVAL            = -10,  /* invalid argument */
    SOC_PIPE             = -11,  /* broken pipe */
    SOC_NOTCONN          = -12,  /* socket is not connected */
    SOC_MSGSIZE          = -13,  /* msg is too long */
```

```
    SOC_BEARER_FAIL          = -14,     /* bearer is broken */
    SOC_CONNRESET            = -15,     /* TCP half-write close, i.e., FINED */
    SOC_DHCP_ERROR           = -16,     /* DHCP error */
    SOC_IP_CHANGED           = -17,     /* IP has changed */
    SOC_ADDRINUSE            = -18,     /* address already in use */
    SOC_CANCEL_ACT_BEARER = -19       /* cancel the activation of bearer */
} soc_error_enum;
/* error cause */
typedefenum
{
    CBM_OK                       = 0,    /* success */
    CBM_ERROR                    = -1, /* error */
    CBM_WOULDBLOCK               = -2, /* would block */
    CBM_LIMIT_RESOURCE           = -3, /* limited resource */
    CBM_INVALID_ACCT_ID          = -4, /* invalid account id*/
    CBM_INVALID_AP_ID            = -5, /* invalid application id*/
    CBM_INVALID_SIM_ID           = -6, /* invalid SIM id */
    CBM_BEARER_FAIL              = -7, /* bearer fail */
    CBM_DHCP_ERROR               = -8, /* DHCP get IP error */
    CBM_CANCEL_ACT_BEARER    = -9, /* cancel the account query screen, such as always ask or
bearer fallback screen */
    CBM_DISC_BY_CM               = -10 /* bearer is deactivated by the connection management */
} cbm_result_error_enum;

/* Socket Type */
typedefenum
{
    SOC_SOCK_STREAM = 0,    /* stream socket, TCP */
    SOC_SOCK_DGRAM,          /* datagram socket, UDP */
    SOC_SOCK_SMS,           /* SMS bearer */
    SOC_SOCK_RAW             /* raw socket */
} socket_type_enum;

/* Socket Options */
typedefenum
{
    SOC_OOBINLINE        = 0x01 << 0,    /* not support yet */
    SOC_LINGER           = 0x01 << 1,    /* linger on close */
    SOC_NBIO             = 0x01 << 2,    /* Nonblocking */
    SOC_ASYNC            = 0x01 << 3,    /* Asynchronous notification */

    SOC_NODELAY          = 0x01 << 4,    /* disable Nagle algorithm or not */
    SOC_KEEPALIVE        = 0x01 << 5,    /* enable/disable the keepalive */
    SOC_RCVBUF           = 0x01 << 6,    /* set the socket receive buffer size */
    SOC_SENDBUF          = 0x01 << 7,    /* set the socket send buffer size */
```

```
    SOC_NREAD          = 0x01 << 8,    /* no. of bytes for read, only for soc_getsockopt */
    SOC_PKT_SIZE       = 0x01 << 9,    /* datagram max packet size */
    SOC_SILENT_LISTEN = 0x01 << 10, /* SOC_SOCK_SMS property */
    SOC_QOS            = 0x01 << 11, /* set the socket qos */

    SOC_TCP_MAXSEG     = 0x01 << 12, /* set the max segmemnt size */
    SOC_IP_TTL         = 0x01 << 13, /* set the IP TTL value */
    SOC_LISTEN_BEARER = 0x01 << 14, /* enable listen bearer */
    SOC_UDP_ANY_FPORT = 0x01 << 15, /* enable UDP any foreign port */

    SOC_WIFI_NOWAKEUP = 0x01 << 16, /* send packet in power saving mode */
    SOC_UDP_NEED_ICMP = 0x01 << 17, /* deliver NOTIFY(close) for ICMP error */
    SOC_IP_HDRINCL     = 0x01 << 18,    /* IP header included for raw sockets */
    SOC_IPSEC_POLICY       = 0x01 << 19, /* ip security policy */
    SOC_TCP_ACKED_DATA   = 0x01 << 20,    /* TCPIP acked data */
    SOC_TCP_DELAYED_ACK = 0x01 << 21, /* TCP delayed ack */
    SOC_TCP_SACK           = 0x01 << 22, /* TCP selective ack */
    SOC_TCP_TIME_STAMP   = 0x01 << 23,    /* TCP time stamp */
    SOC_TCP_ACK_MSEG   = 0x01 << 24     /* TCP ACK multiple segment */
} soc_option_enum;

/* event */
typedefenum
{
    SOC_READ     = 0x01,    /* Notify for read */
    SOC_WRITE    = 0x02,    /* Notify for write */
    SOC_ACCEPT   = 0x04,    /* Notify for accept */
    SOC_CONNECT = 0x08,    /* Notify for connect */
    SOC_CLOSE    = 0x10,    /* Notify for close */
    SOC_ACKED    = 0x20    /* Notify for acked */

} soc_event_enum;

/* socket address structure */
typedefstruct
{
socket_type_enum     sock_type; /* socket type */
    s16 addr_len; /* address length */
    u16 port; /* port number */
    u8   addr[MAX_SOCK_ADDR_LEN];
    /* IP address. For keep the 4-type boundary,
    * please do not declare other variables above "addr"*/
} sockaddr_struct;
```

Example 1：

```
/*TCP Client*/
/*define GPRS bear call back*/
eat_bear_notifybear_notify_cb(cbm_bearer_state_enum state,u8 ip_addr[4])
{
    switch (state) {
    case CBM_DEACTIVATED:    /* GPRS deactivated */
    break;
    case CBM_ACTIVATED :   /* GPRS activated */
            //here get local IP address from parameter “ip_addr”
    break;
    default:
    break;
     }                    /* -----   end switch   ----- */

}

/*define socket event notify call back*/
eat_soc_notifysoc_notify_cb(s8 s,soc_event_enumevent,eat_bool result, u16 ack_size)
{
    switch ( event ) {
    case SOC_READ:     /* socket received data */
    break;
    case SOC_WRITE:    /* socket can send data */
    break;
    case SOC_ACCEPT:  /* client accepting    */
    break;
    case SOC_CONNECT:     /* TCP connect notify */
    if (result == TRUE){ /*connect success*/
    }
    else{                 /*connect failed*/
    }
    break;
    case SOC_CLOSE:    /* socket disconect */
    break;
    case SOC_ACKED:    /* The remote has received data*/
            // ack_size is acked data’s length
    break;
    default:
    break;
    }                    /* -----   end switch   ----- */
}

//.....................................................
/*open GPRS bear*/
ret = eat_gprs_bearer_open("CMNET",NULL,NULL,bear_notify_cb);    //open GPRS bear
ret = eat_gprs_bearer_hold() ; //hold GPRS bear
```

```
//.....................................................

//in bear_notify_cb function, if (state    == CBM_ACTIVATED), socket can be created to connect.
/*create SOCKET connection*/
eat_soc_notify_register(soc_notify_cb);    //register call back
socket_id = eat_soc_create(SOC_SOCK_STREAM,0); //create TCP SOCKET
val = (SOC_READ | SOC_WRITE | SOC_CLOSE | SOC_CONNECT|SOC_ACCEPT);
ret = eat_soc_setsockopt(socket_id,SOC_ASYNC,&val,sizeof(val));//set async event
val = TRUE;
ret = eat_soc_setsockopt(socket_id, SOC_NBIO, &val, sizeof(val));//set no block mode
address.sock_type = SOC_SOCK_STREAM;
address.addr_len = 4;
address.port = 5107;                          /* TCP server port */
address.addr[0]=116;                          /* TCP server ip address */
address.addr[1]=247;
address.addr[2]=119;
address.addr[3]=165;
ret = eat_soc_connect(socket_id,&address);//connect TCP server 116.247.119.165, port is 5107

//.....................................................
/*Close SOCKET */
ret = eat_soc_close(socket_id);

//.....................................................
/*release GPRS*/
ret = eat_gprs_bearer_release();
```

Example 2：

```
/*TCP Server*/
/*define GPRS bear call back*/
s8newsocket=-1;
eat_bear_notifybear_notify_cb(cbm_bearer_state_enum state,u8 ip_addr[4])
{
    switch (state) {
        case CBM_DEACTIVATED: /* GPRS deactivated */
        break;
        case CBM_ACTIVATED :    /* GPRS activated */
                //here get local IP address from parameter “ip_addr”
        break;
        default:
        break;
     }                      /* -----    end switch    ----- */
```

```
}

/*define socket event notify call back*/
eat_soc_notifysoc_notify_cb(s8 s,soc_event_enumevent,eat_bool result, u16 ack_size)
{
      switch ( event ) {
            case SOC_READ:      /* socket received data */
            break;
            case SOC_WRITE:     /* socket can send data */
            break;
            case SOC_ACCEPT:  /* client accepting    */
                  newsocket = eat_soc_accept(s, NULL);
            break;
            case SOC_CONNECT:      /* TCP connect notify */
            break;
            case SOC_CLOSE:     /* socket disconect */
            if(s!=newsocket){
                  /*error */
}
ret = eat_soc_close(s);
newsocket=-1;
break;
case SOC_ACKED:    /* The remote has received data*/
                  // ack_size is acked data's length
break;
default:
break;
      }                       /* -----    end switch    ----- */
}

//.....................................................
/*open GPRS bear*/
ret = eat_gprs_bearer_open("CMNET",NULL,NULL,bear_notify_cb);    //open GPRS bear
ret = eat_gprs_bearer_hold() ; //hold GPRS bear

//.....................................................

//in bear_notify_cb function, if (state    == CBM_ACTIVATED), socket can be created to connect.
/*create SOCKET connection*/
eat_soc_notify_register(soc_notify_cb);    //register call back
socket_id = eat_soc_create(SOC_SOCK_STREAM,0); //create TCP SOCKET
address.sock_type = SOC_SOCK_STREAM;
address.addr_len = 0;/* any local ip address */
address.port = 5107;          /* local port*/
address.addr[0]=0;
```

```
address.addr[1]=0;
address.addr[2]=0;
address.addr[3]=0;
ret = eat_soc_bind(socket_id, &address); /*bind local ip*/
val = (SOC_READ | SOC_WRITE | SOC_CLOSE | SOC_CONNECT|SOC_ACCEPT);
ret = eat_soc_setsockopt(socket_id,SOC_ASYNC,&val,sizeof(val));//set async event
val = TRUE;
ret = eat_soc_setsockopt(socket_id, SOC_NBIO, &val, sizeof(val));//set no block mode
ret = eat_soc_listen(socket_id, 1);    /*creat 1 queue for listen client connect */
//.....................................................
/*Close SOCKET */
if(newsocket!=-1){    /* close socket which isconnecting with client */
ret = eat_soc_close(newsocket);
newsocket=-1;
}
ret = eat_soc_close(socket_id);

//.....................................................
/*release GPRS*/
ret = eat_gprs_bearer_release();
```

13.3 Note

- SOCKET just supports TCP and UDP.
- Three callbacks only need to register once.
- Multiple DNS queries were distinguished by setting the last parameter.
- SOCKET currently supports up to six channels.
- To activate GPRS bearer after “+ CGREG: 1”, if GPRS was deactivated, customer will re-do the activation action. Maximum activation time out is about 80 seconds.
- TCP maximum transmission length is 1460 bytes, UDP maximum transfer length is 1472 bytes, be careful not to send too much data.

14 SMS

14.1 Function Introduction

The SMS API supports sending some messages, read, or delete operation. The required header file is “eat_sms.h”. The example is in the “app_demo_sms.c”.

14.2 Function Interface and Example

Enumeration Definition :

```
typedef enum _eat_sms_storage_en_
{
    EAT_SM = 0, //SM
    EAT_ME = 1,//ME
    EAT_SM_P = 2, //First SM
    EAT_ME_P = 3, // First ME
    EAT_MT = 4 // SM and ME
}EatSmsStorage_en;//storage type

typedefenum
{
  EAT_SMSAL_GSM7_BIT = 0,    //7 bit code
  EAT_SMSAL_EIGHT_BIT,      // 8 bit code
  EAT_SMSAL_UCS2,            //UCS2 code
  EAT_SMSAL_ALPHABET_UNSPECIFIED

} EatSmsalAlphabet_en;    // SMS content encoding
```

Struct :

```
typedefstruct _eat_sms_read_cnf_st_
{
    u8 name[EAT_SMS_NAME_LEN+1];/Name
    u8datetime[EAT_SMS_DATA_TIME_LEN+1];//Time
```

```
    u8 data[EAT_SMS_DATA_LEN+1];// SMS content
    u8 number[EAT_SMS_NUMBER_LEN+1];//phone number
    u8 status;//status
    u16 len;/length
}EatSmsReadCnf_st;

typedefstruct _eat_sms_new_message_st_
{
    EatSmsStorage_en storage; //storage type
    u16 index;// storage index
}EatSmsNewMessageInd_st;// Receive SMS structure
```

Callback Function:

typedef void (* Sms_Send_Completed)(eat_bool result);// Send SMS callback
typedef void (* Sms_Read_Completed)(EatSmsReadCnf_stsmsReadContent); // Read SMS callback
typedef void (* Sms_Delete_Completed)(eat_bool result); // Delete SMS callback
typedef void (* Sms_New_Message_Ind)(EatSmsNewMessageInd_stsmsNewMessage); // Receive SMS callback typedef void (* Sms_Flash_Message_Ind)(EatSmsReadCnf_stsmsFlashMessage); // Receive Flash SMS callback
typedef void(* Sms_Ready_Ind)(eat_bool result); // SMS callback initialization is complete

Function Interface:

Function Interface	Description
eat_get_SMS_sc	Get the SMS center number
eat_set_sms_sc	Set the SMS center number
eat_get_sms_ready_state	Check SMS module whether ready
eat_acsii_to_ucs2	Convert ASCII code to UCS2 format
eat_acsii_to_gsm7bit	Convert ASCII code to 7bit code
eat_send_pdu_sms	Send PDU mode message
eat_send_text_sms	Send TEXT mode message
eat_get_sms_format	Get SMS format
eat_set_sms_format	Set SMS format
eat_set_sms_cnmi	Set SMS CNMI parameter
eat_set_sms_storage	Set the SMS storage type
eat_get_sms_operation_mode	Get SMS operation by API or AT mode
eat_set_sms_operation_mode	Set SMS operation by API or AT mode
eat_read_sms	Read a message
eat_delete_sms	Delete a message
eat_sms_decode_tpdu	Decode PDU message
eat_sms_orig_address_data_convert	Convert Original address

eat_sms_register_send_completed_callback	Send message completed to callback
eat_sms_register_new_message_callback	New message reported to callback
eat_sms_register_flash_message_callback	Report flash message to callback
eat_sms_register_sms_ready_callback	Report SMS ready to callback

Example:

```
/*Defined callback function receives flash message */
staticeat_sms_flash_message_cb(EatSmsReadCnf_stsmsFlashMessage)
{
    u8 format =0;

    eat_get_sms_format(&format);
    eat_trace("eat_sms_flash_message_cb, format=%d",format);
    if(1 == format)//TEXTmode
    {
        eat_trace("eat_sms_read_cb, msg=%s",smsFlashMessage.data);
        eat_trace("eat_sms_read_cb, datetime=%s",smsFlashMessage.datetime);
        eat_trace("eat_sms_read_cb, name=%s",smsFlashMessage.name);
        eat_trace("eat_sms_read_cb, status=%d",smsFlashMessage.status);
        eat_trace("eat_sms_read_cb, len=%d",smsFlashMessage.len);
        eat_trace("eat_sms_read_cb, number=%s",smsFlashMessage.number);
    }
    else//PDU mode
    {
        eat_trace("eat_sms_read_cb, msg=%s",smsFlashMessage.data);
        eat_trace("eat_sms_read_cb, len=%d",smsFlashMessage.len);
    }
}
/* Defined callback function receives a message */
staticeat_sms_new_message_cb(EatSmsNewMessageInd_stsmsNewMessage)
{
    eat_trace("eat_sms_new_message_cb,
    storage=%d,index=%d",smsNewMessage.storage,smsNewMessage.index);
}
/* Defined callback function read a message */
static void eat_sms_read_cb(EatSmsReadCnf_stsmsReadCnfContent)
{
    u8 format =0;

    eat_get_sms_format(&format);
```

```
    eat_trace("eat_sms_read_cb, format=%d",format);
    if(1 == format)//TEXTmode
    {
        eat_trace("eat_sms_read_cb, msg=%s",smsReadCnfContent.data);
        eat_trace("eat_sms_read_cb, datetime=%s",smsReadCnfContent.datetime);
        eat_trace("eat_sms_read_cb, name=%s",smsReadCnfContent.name);
        eat_trace("eat_sms_read_cb, status=%d",smsReadCnfContent.status);
        eat_trace("eat_sms_read_cb, len=%d",smsReadCnfContent.len);
        eat_trace("eat_sms_read_cb, number=%s",smsReadCnfContent.number);
    }
    else//PDU mode
    {
        eat_trace("eat_sms_read_cb, msg=%s",smsReadCnfContent.data);
        eat_trace("eat_sms_read_cb, name=%s",smsReadCnfContent.name);
        eat_trace("eat_sms_read_cb, status=%d",smsReadCnfContent.status);
        eat_trace("eat_sms_read_cb, len=%d",smsReadCnfContent.len);
    }
}
/* Defined callback function delete a message */
static void eat_sms_delete_cb(eat_bool result)
{
    eat_trace("eat_sms_delete_cb, result=%d",result);
}
/* Defined callback function send a message */
static void eat_sms_send_cb(eat_bool result)
{
    eat_trace("eat_sms_send_cb, result=%d",result);
}
/* Defined callback function complete SMS module initialization */
static void eat_sms_ready_cb(eat_bool result)
{
    eat_trace("eat_sms_ready_cb, result=%d",result);
}
/*APP MAIN start，Registrate above the callback function*/
voidapp_main(void *data)
{
    //do something

    // Registrate related the callback function
    eat_set_sms_operation_mode(EAT_TRUE);//Set SMS module API mode operation
    eat_sms_register_new_message_callback(eat_sms_new_message_cb);
    eat_sms_register_flash_message_callback(eat_sms_flash_message_cb);
    eat_sms_register_send_completed_callback(eat_sms_send_cb);
    eat_sms_register_sms_ready_callback(eat_sms_ready_cb);

while(EAT_TRUE)
```

```
{
    //do something
}
}
/*Set SMS PDU mode */
eat_set_sms_format(0);
/*Set SMS service center number */
u8scNumber[40] = {"+8613800210500"};
eat_set_sms_sc(scNumber);
/*Set SMS storage type */
u8 mem1, mem2, mem3;
eat_bool ret_val = EAT_FALSE;

mem1 = EAT_ME;
mem2 = EAT_ME;
mem3 = EAT_ME;
ret_val = eat_set_sms_storage(mem1, mem2, mem3);
/*Set CNMI Parameter */
mode = 2;
mt = 1;
bm = 0;
ds = 0;
bfr = 0;
ret_val= eat_set_sms_cnmi(mode,mt,bm,ds,bfr);
/*Reads the message content */
u16 index = 1;
ret_val = eat_read_sms(index,eat_sms_read_cb);
/* Send SMS content */
u8 format = 0;
eat_bool ret_val = EAT_FALSE;

eat_get_sms_format(&format);
if(1 == format)
{
    ret_val = eat_send_text_sms("13681673762","123456789");
}
else
{
    ret_val = eat_send_pdu_sms("0011000D91683186613767F20018010400410042",19);
}
/* Deletes the message content */
u16 index = 1;
ret_val = eat_delete_sms(index,eat_sms_delete_cb);
/* Parse the content of messages received */
u8
ptr[]= "0891683108200105F0040D91683186613767F2000041301251658523063 1D98C56B301";
```

```
u8 len = strlen(ptr);
EatSmsalPduDecode_st sms_pdu = {0};
u8 useData[320] = {0};
u8 useLen = 0;
u8 phoneNum[43] = {0};

ret = eat_sms_decode_tpdu(ptr, len, &sms_pdu);
```

NOTE：
More specifically, a more comprehensive example, you can reference the file “app_demo_sms.c”

14.3 Note

- When a thread is initialized, customer needs to register all callbacks SMS module. If not registered callback function accordingly, the related function will fail.
- When eat_sms_register_sms_ready_callback registered callback function returns EAT_TRUE, SMS initialization is complete, otherwise the API operation will be failed.
- When eat_set_sms_operation_mode (eat_bool mode) interface parameter settings EAT_TRUE, only use the API provided by the SMS operation. If set EAT_FALSE, customer can only use AT command operation SMS.
- eat_send_text_sms and eat_send_pdu_sms length based interface to send SMS messages Encoding: 7bit code is 160; 8bit code is 140; UCS2 code 70.
- When operating SMS API, it is recommended to use the interface to determine whether eat_get_sms_ready_state SMS module initialization finished, perform SMS operations after initialization is complete.