SIM800 Series_Embedded AT®_Application Note_V1.02

Development Kit

Hardware Design

Reference Design

Software Design

Document Titile	SIM800 Series_Embedded AT®_Application Note
Version	1.02
Date	2014-05-30
Status	Released
Document Control ID	SIM800 Series_Embedded AT®_Application Note_V1.02

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

Contents

Version History

Date	Version	Description of Change	Author
2012-10-20	1.00	Origin	Maobin
2013-11-10	1.01	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	Maobin
2014-05-30	1.02	1.Add the Socket chapter 2.Add the SMS chapter	zhangping zhudingfen

Application Scope

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

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

1. Embedded AT Introduction

1.1 Brief Introduction

Embedded AT is mainly used for customer to do the secondary development with SIM800 series modules . SIMCOM provides the relative API functions , resource and operating environment. Customer's APP code runs inner SIM800 series module , so we don't need external MCU and save the cost.

1.2 Code Style

Embedded AT can basically implement customer's code structure of MCU , and it supports multi threads that allows customer to run different subtask with different threads.

1.3 Sources Supplied by SIM800 Series Modules' Embedded AT

The main API functions are 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. |

2   Embedded AT Basic Conception

The software architecture of Embeded AT is as following :
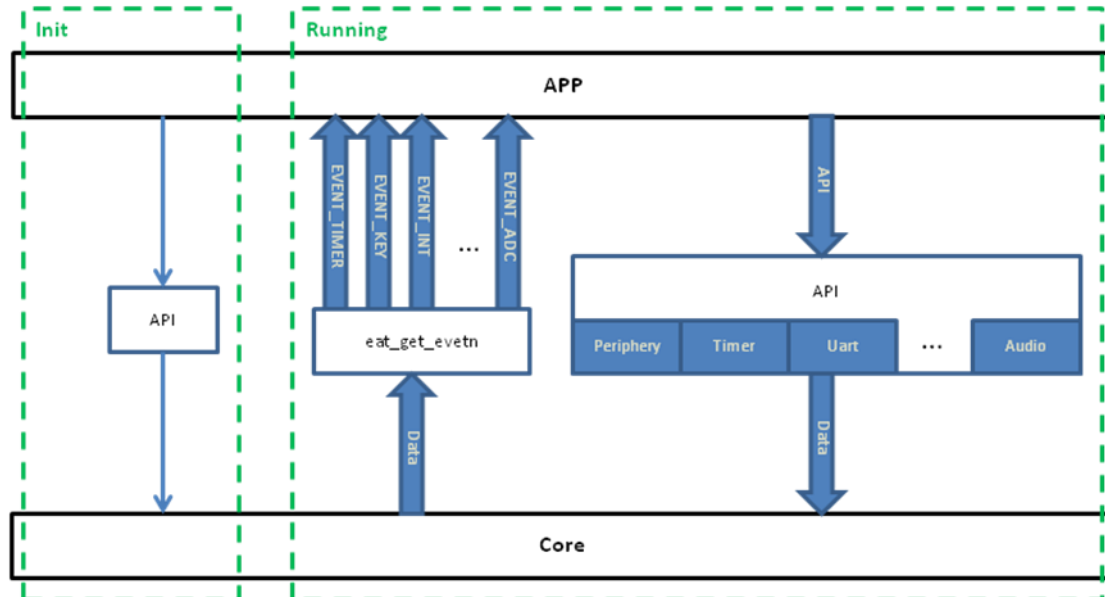
Figure 1   General Software Architecture

Illustration :

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

Note : EAT is the abbreviation of Embedded AT , will replace Embeded AT with EAT in the following chapters .

3   Multi Threads

3.1   Multi Threads Function Description

The platform provide multi threads functions , 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
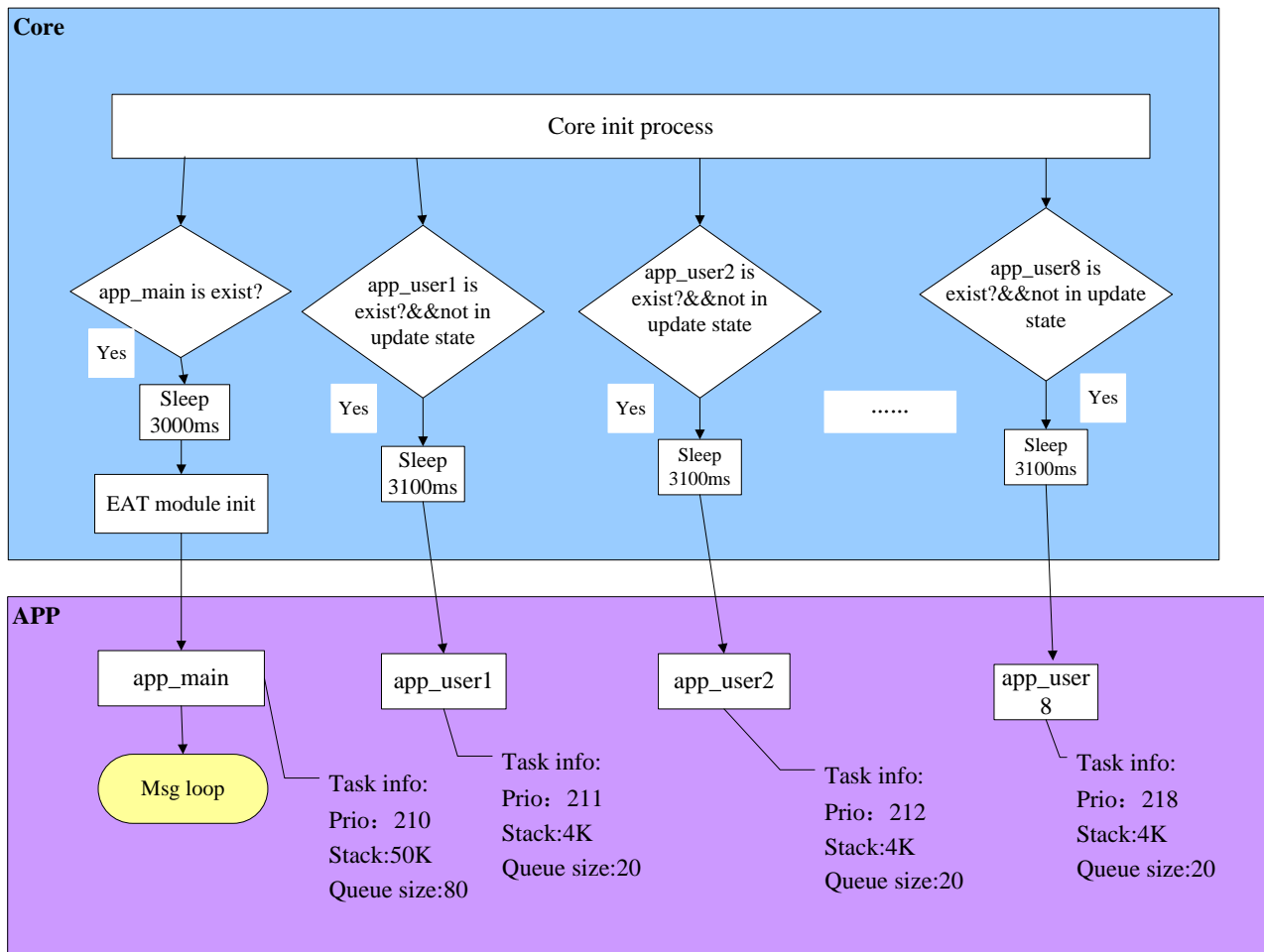
Figure 2 Tread Initialization Information

Illustration :

1)   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"
const EatEntry_st AppEntry =
{
    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

2)   Task Description

    App_main, the main app thread , is to receive the message from system (core) , has highest priority which initializes earlier than other 8 threads . The function interface would be called if the value in EatEntry_st is not 0xFFFFFFFF or 0 , and also system would allocate the task's information : stack space 10K Byte , queue size 80 , has the highest priority of app threads .

    App_user1 , app_user2 …app_user8 is the 8 threads which customer can use . The entrance would be called if the value in EatEntry_st is not null nor 0xFFFFFFFF , and also system would allocate task's information: stack space 4K Byte , queue size 20 . The priority degrades successively , i.e. app_main>app_user1>...>app_user8 .

    Note:

       For EAT application of SIM800H and SIM800 , the stack space for main app thread is 5K , and sub thread's stack space is 3K.

       So try 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	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

3.3.2 Interface Definition

The following two functions , which can only be used in app_main , is to to acquire message sent from core , or acquire message the EAT_EVENT_USER_MSG sent from app_user1 app_user2 …app_user8 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 task which can be used by customer , i.e. app_user1 app_user2 …app_user8 , the functions which correspond to above features is 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

1) For sending message , function eat_send_msg_to_user can be used in app_main and app_user1 app_user2 …app_user8 .

2) **SIM800H and SIM800** 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 we 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 we 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 .

3) eat_get_event or eat_get_event_for _user is synchronous interface . Once we call the interface , will get return immediately if there is event in the thread , and if there is not event , the thread will be suspended .

If we don'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 . We would call this interface if the number is above 0 .

4 Timer Application

4.1 Function Introduction

There are two kinds of timers supplied , 16 ms (millisecond) timer and 1 us (microsecond) timer .
Set the thread sleep .
Set and acquire the current date and time of system .
Acquire the interval between two time .

4.2 Function Interface and Example

EVENT：
   EAT_EVENT_TIMER
Struct :
   typedef struct {
       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 Name	Description
eat_timer_start/eat_timer_stop	Start and stop the ms timer
eat_gpt_start /eat_gpt_stop	Start and stop the us 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 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 us timer

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

Acquire the time interval

```
unsigned int time =   eat_get_current_time();
//do something
unsigned int time_ms = eat_get_duration_ms(time);
```

Set and acquire system time

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

4.3 Note

4) eat_gpt_start is hardware timer , is to execute the timer callback function in interrupt . So the timer callback function should not take too much time , should not use blocking function , such as sleep , memory allocation , signal , etc .

5) The application of timer may affect the sleep feature . System in sleep would be waken up once the timer is triggered . So when we are trying to control the system to enter into sleep mode , we should consider the influence of timer .

5 Memory Application

The EAT platform allows us to manage an array by which we can implement the memory initialization , allocation and release of interface . It is flexible that the space which need to be applied and released dynamically is defined by array .

The maximum memory space size can be applied at a time is : Initialized array size-N (The value of N is 168 for now , it may be different based on different platform , but it is easy to be measured .)

The memory and global read-write variables both occupy app's RAM space . And for the size of RAM space , we can refer to the allocation of RAM space size in the chapter Flash allocation .

Function Interface:

Function Interface	Function
eat_mem_init	Initialize memory block
eat_mem_alloc	Apply memory
eat_mem_free	Release memory

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);
```

6 Sleep

eat_sleep_enable is to enable or disable the system to enter into 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 );
```

After module going into sleep mode , it would be waken up in cycles automatically to communicate with network .

For the period not waken up in sleep mode , system can only be waken up by key , external interrupt (GPIO) , timer , incoming message , incoming call .

For the detailed description of sleep feature , please refer to the document 《SIM800 Series_Embedded AT Sleep Illustration _V1.01》 .

7   Serial Port

7.1   Function Description

Serial port parameter configuration

Serial port data send and receive

Serial port function mode configuration

There are three serial ports which can be used to send and receive data in app . And each port of them can be used as AT command port or system's DEBUG port . (SIM800H and SIM800 have two serial ports and one USB port ,SIM808 only has one serial port and one USB port.)

For different platform , the ports quantity may be different , please confirm it based on platform function description .

7.2   Message

EAT_EVENT_MDM_READY_RD

Once the send buffer's status turns to non-empty from empty status in the process of Modem sending data to EAT , this message would be triggered and send to EAT . The send buffer's size is 5KB .

EAT_EVENT_MDM_READY_WR

Once the receive buffer's status turns to non-full from full status in the process of Modem receiving data from EAT , this message would be triggered and send to EAT . The receive buffer's size is 5KB .

EAT_EVENT_UART_READY_RD

Once the receive buffer's status turns to non-empty from empty status in the process of UART port receiving data , this message would be triggered . The receive buffer's size is 2KB .

EAT_EVENT_UART_READY_WR

Once the send buffer's status turns to non-full from full status in the process of UART port sending data , this message would be triggered . The send buffer's size is 2KB .

EAT_EVENT_UART_SEND_COMPLETE

Once the send buffer's status turns to empty from non-empty status in the process of UART port sending data , and also if the underlying DMA FIFO is in empty status , then this message would be triggered .

7.3   Function Interface and Example

Enumeration Varialble

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

typedef enum {
```

```
        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;


    typedef enum {
        EAT_UART_DATA_BITS_5=5,
        EAT_UART_DATA_BITS_6,
        EAT_UART_DATA_BITS_7,
        EAT_UART_DATA_BITS_8
    } EatUartDataBits_enum;


    typedef enum {
        EAT_UART_STOP_BITS_1=1,
        EAT_UART_STOP_BITS_2,
        EAT_UART_STOP_BITS_1_5
    } EatUartStopBits_enum;


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

Struct :

```
    typedef struct {
        EatUart_enum uart;
    } EatUart_st;


    typedef struct {
        EatUartBaudrate    baud;
        EatUartDataBits_enum    dataBits;
        EatUartStopBits_enum    stopBits;
        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

```
u16 len;
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
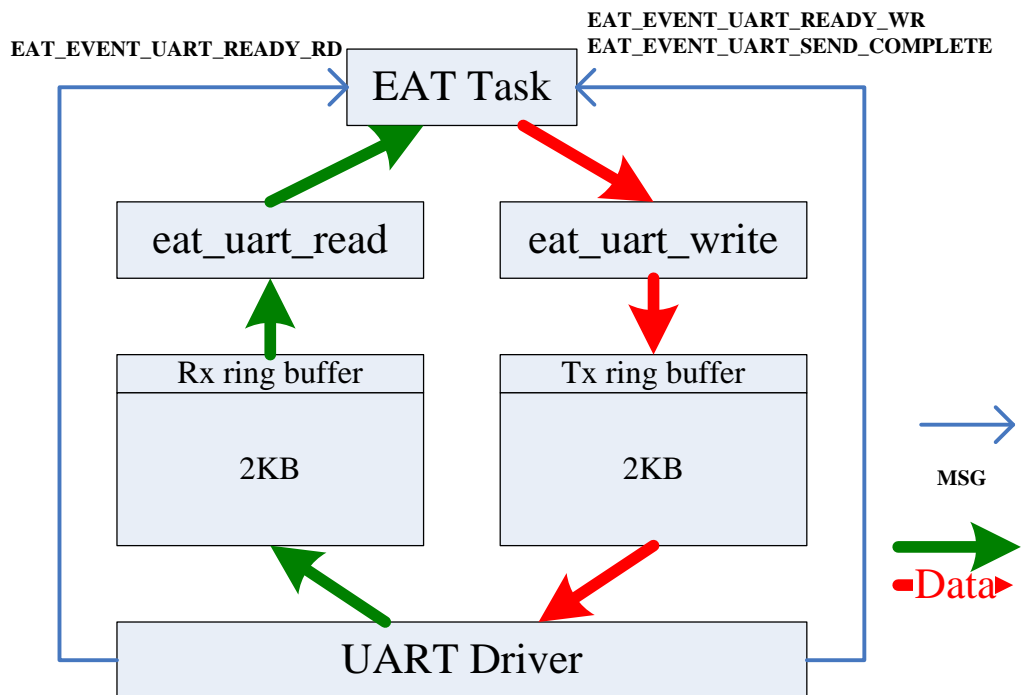
Flow Chart of Serial Port Data and Message Transmitted Between EAT and Modem

EAT_EVENT_MDM_READY_RD   **EAT_EVENT_MDM_READY_WR**
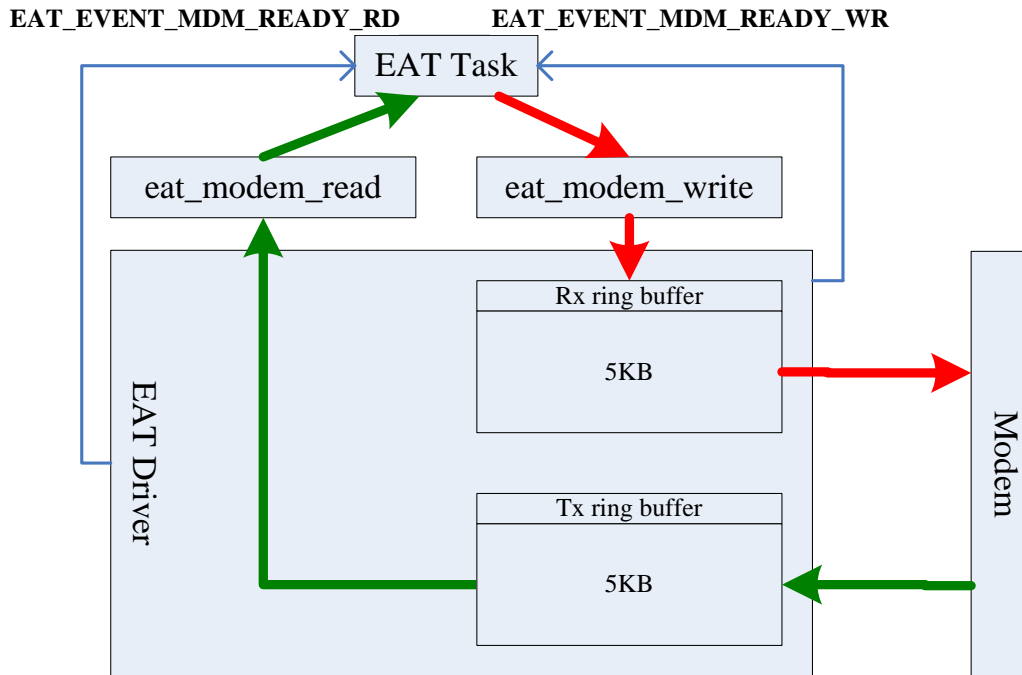
7.5   Note

1) **Once APP receive the message EAT_EVENT_MDM_READY_RD or EAT_EVENT_UART_READY_RD , would read the data in receive buffer by eat_modem_read or eat_uart_read . If still there is data remained in receive buffer not be read , the message READY_RD will not be forwarded when the receive buffer receive data next time .**

2) **The following functions is to 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

3) **The following functions is to be called in system initialization process (i.e. in the member function func_ext1 of struct EatEntry_st)**
- eat_uart_set_debug
- eat_uart_set_at_port
- eat_uart_set_debug_config

4) **eat_uart_set_at_port , the firmware 1116B02V01 and also the previous firmware versions don't support this feature . firmware version check : use EAT API**

eat_get_version() and eat_get_buildno() .

Or check it by AT command :

at+cgmr

Revision:1116B02SIM840W64_WINBOND_EMBEDDEDAT

OK

at+csub

+CSUB: V01

OK

8   Flash Allocation

8.1   Function Description

Provides Write, Erase and app upgrade function.

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

The FLASH allocation of standard EAT version as shown in the following table . The address configuration may change based on customer's requirement , please refer to customer's actual requirement .

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

SIM800 series modules have various FLASH versions , the address configuration as default of different version is following below table :

SIM800W 64M Version

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 128M 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	1067FFFF	2.5M (0x280000)
FS	10682000	107C1FFF	1.25M (0x140000)
RAM	F0600000	F077FFFF	1.5M (0x180000)

APP: customer's app code and ROM space

FS : File system space includes system parameter , calibration parameter , etc . The 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: Customers have 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

Provides the function of updating app in running process .

There is updating mark inner module which can only be erased once update app successfully . If the module is power off in the updating process , then it would continue to update app based on the updating mark when the module is power on next time , and it would not happen that module can't start up if power off in the updating process .

Once update successfully , module would start app and transmit parameter to app_main to indicate the updating result .

app update flow is as following example :

```
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);
    }
    void app_main(void* data)
    {
      ......
        // initialize
        APP_DATA_RUN_BASE = eat_get_app_base_addr(); // acquire app address
        app_space_value = eat_get_app_space(); // acquire the size of app space
        APP_DATA_STORAGE_BASE = APP_DATA_RUN_BASE + (app_space_value>>1);
// save the address of app updating file
      ......
    }
```

The updating process after call eat_update_app is as following :

1) Write the relative mark of updating program into the APP updata flag area of flash .

2) Module restart , check the value of APP updata flag , remove 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 .

3) Module restart again , run the new program . Customer would judge if update successfully by the transmitted parameter of app_main(void param) in new program , and erase the area APP updata flag to avoid that any relative indication affects the program .

```
void app_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
    }
    ............
}
```

We need to erase the updating mark by "eat_update_app_ok()" in "app_main" of new program after update successfully . If the updating mark is not erased , the data in APP_DATA_STORAGE_BASE would be downloaded to area APP_DATA_RUN_BASE when the module start up again .

8.5   Note

1） The erasure of Flash is in accordance with the "block" erase. The use of eat_flash_erase (const void *address, unsigned int size) interface to erase flash, address should be integer times of block size, minimum erasing one flash block. If the address is not a block size integer, EAT will do the processing, the address is set to the start address of the block erase. If the size value is not equal to integer times of a block, it will erase (size/Block) +1 Flash blocks.

2) If this address was written once before,you need to erase before write again.

9   File System

Provides the interface to implement the relative manipulation of file system . The function interface is as following :

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

9.1   Note

1)   eat_fs_Open open or create file , only provide four kinds of manipulation which include FS_READ_WRITE , FS_READ_ONLY , FS_CREATE and FS_CREATE_ALWAYS . It supports to open or create maximum 24 files at the same time . The created file name need …. For example , we 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;
}
```

2) eat_fs_GetDiskFreeSize It supports to acquire the remained space's size of inner file system and T-Flash . It may get corresponding error value if there is no external T-Flash .

3) eat_fs_Write There is no size limitation for writing data one time , but it would make error once write data more than the remained space's size of file system . The actual length of data written in file system is based on the returned parameter value of this interface last time .

4) The drive of inner file system is "C" , root directory is "C:\\" . The drive of T_Flash is "D:" , root directory is "D:\\" . If we omit the drive in file manipulation , then it would be taken as inner file system as default.

10 APP Update

Provides the function of updating app in running process .

There is updating mark inner module which can only be erased once update app successfully . If the module is power off in the updating process , then it would continue to update app based on the updating mark when the module is power on next time , and it would not happen that module can't start up if power off in the updating process .

Once update successfully , module would start app and transmit parameter to app_main to indicate the updating result .

app update flow is as following example :

```
    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);
    }
    void app_main(void* data)
    {
      ......
        // initialize
```

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

The updating process after call eat_update_app is as following :

4)   Write the relative mark of updating program into the APP updata flag area of flash .

5)   Module restart , check the value of APP updata flag , remove 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 .

6)   Module restart again , run the new program . Customer would judge if update successfully by the transmitted parameter of app_main(void param) in new program , and erase the area APP updata flag to avoid that any relative indication affects the program .

```
void app_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
     }
     ............
}
```

7)   We need to erase the updating mark by "eat_update_app_ok()" in "app_main" of new program after update successfully . If the updating mark is not erased , the data in APP_DATA_STORAGE_BASE would be downloaded to area APP_DATA_RUN_BASE when the module start up again .

11 Peripheral Interface

11.1 Function Introduction

The peripheral interface mainly provides some API that can easily control and read/write the peripherals such as LCD , KEYPAD , ADC , etc . It mainly provides the following functions:

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 Read/write and control Interface for GPIO

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

```
typedef enum    {
    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;

typedef enum {
    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;

typedef enum {
    EAT_GPIO_LEVEL_LOW,
    EAT_GPIO_LEVEL_HIGH
} EatGpioLevel_enum;

typedef enum {
    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:

1) Only SIM800W has the interface eat_gpio_write_ext . Compared with eat_gpio_write , eat_gpio_write_ext has a faster speed to write . eat_gpio_write_ext may need about 1us (microsecond) to write which is only half of the time eat_gpio_write needs . 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,

2) We should pay attention to the 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 we use PIN64 to detect the SIM card (AT+CSDT=1) , this pin should be pulled up to VDD_EXT by 10~100k 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	KBC0 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 :

```
typedef struct {
    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_set_trigger	set the trigger mode of interrupt

Example :

```
/* define the interrupt callback function */
void int_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 :

1) The unit of parameter debounce_ms in eat_int_setup is 10ms . For example , it means 100ms if debounce_ms equals 10 . This parameter debounce_ms is only effective to level-triggered interrupt .

2) If we want to invert interrupt after it be triggered , need to configure the interrupt inversion in callback function .

3) 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 .

4) If we choose level-triggered interrupt and use EVENT to handle the interrupt , then need to check the current electrical level and configure the opposite electrical level as trigger condition . For example , if the current electrical is low level , then we need to configure high level as trigger condition . Because for level-triggered interrupt , if we configure the trigger

condition (level) as same as the current electrical level , then the interrupt would be triggered before eaet_int_setup exit . As we use EVEN to handle the interrupt , the interrupt may send message to task , but can't enter into get_event to configure the opposite electrical level since the current task is interrupted by GPIO interrupt , so the level-triggered interrupt would be triggered internally all the time . Another method , we use eat_int_setup_ext and configure the last parameter to enable or disable setting opposite electrical level automatically after level-triggered interrupt be triggered .

11.2.3 SPI Interface

Enumeration Definition

```
typedef enum {
    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 */

typedef enum {
    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	Funciton
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 there is not DISP_CS pin to control SPI to read/write externally , please control SPI in app . The parameter enable_cs of eat_spi_init is false .

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 :

1) Output cycle : 0 (all low) – 100 (all high)

11.2.5 ADC Interface

EVENT：

EAT_EVENT_ADC

Struct :

```
typedef struct {
    EatPinName_enum pin; /* the pin */
    unsigned int v; /* ADC value,unit is mv*/
} EatAdc_st;
```

Callback Function :

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

Function Interface :

Function Interface	Funciton
eat_adc_get	Read ADC

Example :

```
/* define interrupt callback function */
void adc_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 disable 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 :
1) System can't enter into sleep mode if LCD backlight or keypad backlight is lighten up .
2) Holding about 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 we keep USB connection or press 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 :
typedef enum {
    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. */
typedef struct {
        EatKey_enum key_value; /* key value */
        eat_bool is_pressed; /* 1-key press down; 0-key release up */
} EatKey_st;
```

Example :

```
void app_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 :
```
typedef struct {
    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

1) 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 .

2) 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 .

3) Tone can be played but audio data flow can't in call process .

4) The id of eat_audio_play_tone_id() and eat_audio_stop_tone_id() should match to avoid stopping incorrectly .

5) Make sure the audio data flow is in correct format for playback , only MIDI and WAV format is supported for now .

6) The common tone frequency (refer to struct EatAudioToneData_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 support SOCKET interface, customer can use it to create TCP or UDP connection and transfer data.

13.2 Function Interface and Example

SOCKET support 3 type interfaces, There 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_enum event,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 */

typedef enum

{

    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 */

typedef enum

{

    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 */

```
typedef enum
{
    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 */
typedef enum
{
    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 */
typedef enum
{
    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 */
typedef enum
{
    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 */
typedef struct
{
    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：

```
/*define GPRS bear call back*/
eat_bear_notify bear_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_notify soc_notify_cb(s8 s,soc_event_enum event,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();
```

13.3 Note

- SOCKET just support 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 interface provides API to send some messages, read, or delete operation.The required header file is "eat_sms.h". The example 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

typedef enum

{

    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 :

typedef struct _eat_sms_read_cnf_st_

{

    u8 name[EAT_SMS_NAME_LEN+1];/Name

    u8 datetime[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;

typedef struct _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_st   smsReadContent); // Read SMS callback

typedef void (* Sms_Delete_Completed)(eat_bool result); // Delete SMS callback

typedef void (* Sms_New_Message_Ind)(EatSmsNewMessageInd_st smsNewMessage); // Receive SMS callback

typedef void (* Sms_Flash_Message_Ind)(EatSmsReadCnf_st smsFlashMessage); // 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 */
static eat_sms_flash_message_cb(EatSmsReadCnf_st smsFlashMessage)
{
    u8 format =0;

    eat_get_sms_format(&format);
    eat_trace("eat_sms_flash_message_cb, format=%d",format);
```

```c
    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 */
static eat_sms_new_message_cb(EatSmsNewMessageInd_st smsNewMessage)
{
    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_st    smsReadCnfContent)
{
    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*/
void app_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 */
u8 scNumber[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[]= "0891683108200105F0040D91683186613767F200004130125165852306 31D98C56B301";
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

1) When a thread is initialized, it need to register all callbacks SMS module. If not registered callback function accordingly, the related function will fail.

2) When eat_sms_register_sms_ready_callback registered callback function returns EAT_TRUE, SMS initialization is complete, otherwise the API operation have failed.

3) 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, you can only use AT command operation SMS.

4) 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。

Contact us:
Shanghai SIMCom Wireless Solutions Co.,Ltd.
Address: Building A, SIM Technology Building, No. 633, Jinzhong Road, Shanghai, P. R. China 200335
Tel: +86 21 3252 3300
Fax: +86 21 3252 2030
URL: www.sim.com/wm