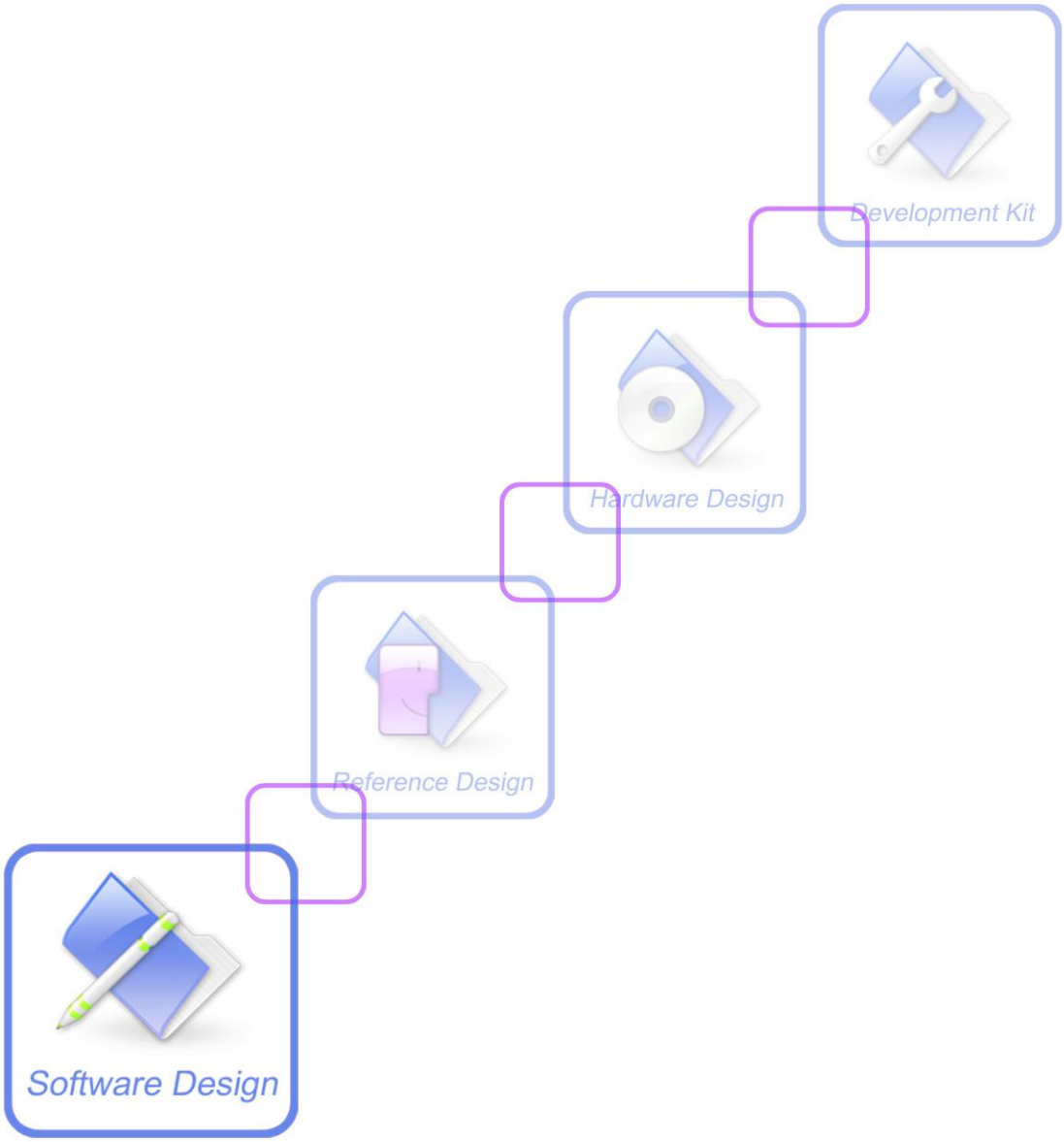
SIMCom
A company of SIM Tech

SIM5360 EBDAT Application Note



Document Title:	SIM5360 EBDAT Application Note
Version:	0.01
Date:	2014-04-29
Status:	Developing
Document ID:	SIM5360_EBDAT_Application_Note_V0.01

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

Version History

Version	Chapter	Comments
V0.01	New Version	

Contents

1. Introduction

SIMCom EBDAT extension is a feature that allows customer applications control and drive the module internally and easily.

The SIMCom EBDAT extension is aimed at light applications where the application was usually done by a small microcontroller that managed some I/O pins and the module through the AT command interface.

By using the EBDAT extension APIs, customer can write applications using C language very quickly.

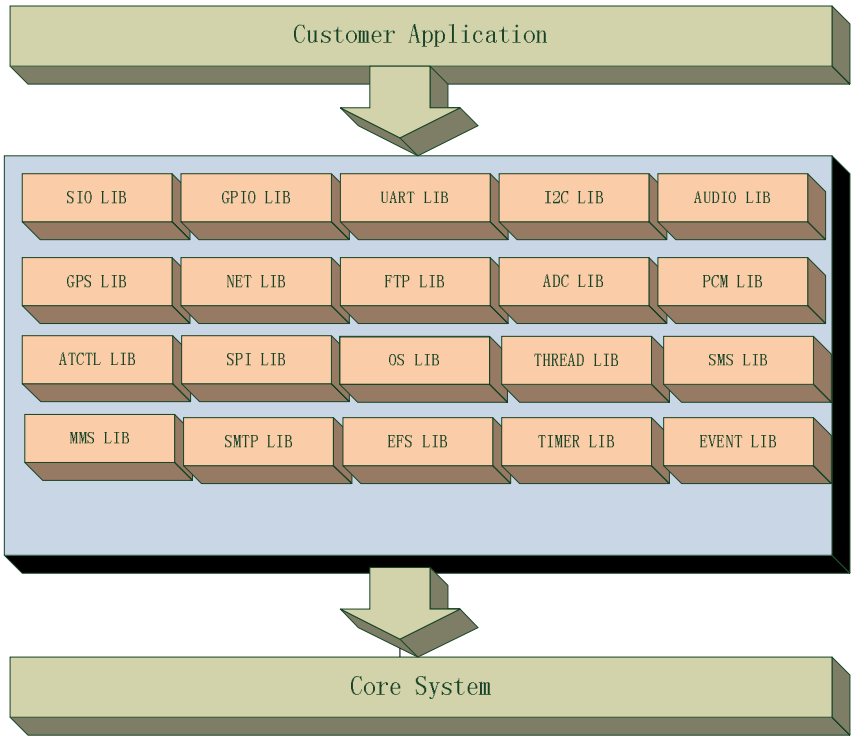
1.1. Features:

1. C language, easy to develop.
2. Support AT command operation using API
3. Support GPIO/IIC/SPI/ADC/PCM/Audio/GPS/UART operation in script.
4. Support FTP operation instead of using AT commands.
5. Application can run automatically when module powers up if AT+CEBDATAUTORUN=1.<max_autorun_times>, if <max_autorun_times> is 0, the module will run the EBDAT application each time when powering up.
6. The priority of the EBDAT task can be adjusted to HIGH, NORMAL, LOW(default).
7. High priority is not recommended to be used when external MCU is connected.
8. Support event operation.
9. Support multi-thread operation with separate thread library.
10. Support timer operation
11. Support module entering sleep mode when EBDAT is idle(calling ebdat_thread_sleep, ebdat_wait_evt)

2. SOFTWARE ARCHITECTURE

2.1. Software Organization

The Embedded AT facility is a software mechanism, it follows the software architecture as shown below:



2.2. Embedded AT Library Information

The Embedded AT facility is a software mechanism, it follows the software architecture as shown below:

Library	Functions
sio	The sio library is used by EBDAT to send AT commands, receive responses and URC.
Gpio	The gpio library allow EBDAT to handle general purpose input output directly instead of using AT commands.
Uart	The uart library allows EBDAT to handle UART operations directly instead of using AT commands.
I2c	The i2c library allows EBDAT to handle IIC read/write operations directly instead of using AT commands.

Adc	The adc library allows EBDAT to read analogue value on the specified ADC channel instead of using AT commands.
Pcm	The pcm library allows EBDAT to switch PCM and GPIO pins directly instead of using AT commands.
Audio	The audio library allows EBDAT to manage AUDIO functions directly instead of using AT commands.
Gps	The GPS library allows EBDAT to set and query GPS setting and geographic information instead of using AT commands.
Net	The net library allows EBDAT to query the wireless network information directly instead of using AT commands.
ftp	The ftp library allows EBDAT to put/get files to/from FTP server directly instead of using AT commands.
Atctl	The atctl library allows EBDAT to handle the data received from external serial port instead of processing it using the internal AT handling engine.
Spi	The SPI library allows EBDAT to write or read SPI device.
Thread	The thread library allows EBDAT to handle multi-thread operations.
Sms	The sms library allows EBDAT to manage SMS operations directly instead of using AT commands.
mms	The mms library allows EBDAT to manage MMS operations directly instead of using AT commands.
Smtp	The smtp library allows EBDAT to manage SMTP operations directly instead of using AT commands.
Socket	The socket library allows EBDAT to do socket operation.

Table 2.1 SIMCom extended libraries

2.3. AT Commands

Below is the EBDATl associated with AT commands, detailed information please refer to document [1]. Through these AT commands can achieve the following functions.

1) Write elf file to flash partition.
2) Run the EBDAT application manually.
3) Enable or disable autorun function.
4) Enable or disable power on sanity check function.

Command	Description
AT+CEBDAT	Write EBDAT elf file to specific flash partition which can be loaded on next power cycle.
AT+CEBDATSTART	Run the EBDAT application manually.
AT+CEBDATAUTORUN	Enable or disable EBDAT aurotun function.

AT+EBDATPRINTDIR	Enable or disable ebdat_os_print() function like ebdat_os_printdir()
AT+CPWRONCHK	Enable or disable power on sanity check function.

3. Embedded AT Custom Libraries

3.1. os library

3.1.1. ebdat_os_printdir

Description

The function is used to set the direction of the ebdat_os_print function

Prototype	void ebdat_os_printdir (boolean enable_print)
Parameters	enable_print: the direction of the print 0: not print (default) 1: print to external AT interface
Return value	None

Example

```
--print to AT interface
printdir(1)
print("this is printed to AT interface\r\n")
```

3.1.2. ebdat_os_print

Description

The function is used to print trace information to DIAG or external AT interface,

Prototype	void ebdat_os_print (const byte* msg, uint32 msg_len)
Parameters	msg: the message to print

	msg_len: the length of the message to print
Return value	None

Example

```
char* prompt = “This is my prompt\r\n”;
ebdat_os_print(prompt, strlen(prompt));
```

3.1.3. ebdat_os_gettick

Description

The function is used to get the seconds passed from the starting of the operating system.

Prototype	uint64 ebdat_os_gettick (void)
Parameters	None
Return value	The tick count value of the operating system

Example

```
uint64 seconds_passed = ebdat_os_gettick();
```

3.1.4. ebdat_os_mktime

Description

The function is used to work like mktime() function in other platform.

Prototype	time_t ebdat_os_mktime (struct tm* pt)
Parameters	pt: Pointer to time structure
Return value	The return value is the specified calendar time encoded as a value of type time_t

Example

```
time_t value = ebdat_os_mktime(NULL);
```

3.1.5. ebdat_os_localtime

Description

The function is used to work like localtime() function in other platform.

Prototype	struct tm* ebdat_os_localtime (time_t *timer)
Parameters	timer : Pointer to stored time.
Return value	The return value is a pointer to the structure result.

Example

```
struct tm* value_p = ebdat_os_localtime(&time_t_value);
```

3.1.6. ebdat_os_gmtime

Description

The function is used to work like gmtime() function in other platform.

Prototype	struct tm* ebdat_os_gmtime (time_t *timer)
Parameters	timer : Pointer to stored time.
Return value	The return value is a pointer to the structure result.

Example

```
struct tm* value_p = ebdat_os_gmtime(&time_t_value);
```

3.1.7. ebdat_os_time

Description

The function is used to work like time() function in other platform.

Prototype	time_t ebdat_os_time (time_t* timer)
Parameters	timer: Pointer to the storage location for time.
Return value	Return the time as seconds elapsed since midnight, January 1, 1970. There is no error return.

Example

```
time_t value = ebdat_os_time(&ltime);
```

3.1.8. ebdat_os_strftime

Description

The function is used to work like strftime() function in other platform.

Prototype	void ebdat_os_strftime (char *strDest, size_t maxsize, const char *format, const struct tm *timeptr)
Parameters	strDest: Output string. maxsize: Size of strDest buffer. format: Format-control string. timeptr: tm data structure.
Return value	returns the number of characters placed in strDest

Example

```
ebdat_os_strftime( tmpbuf, 128,  "Today is %A, day %d of %B in the year %Y.\n", &today );
```

3.1.9. ebdat_os_mem_alloc

Description

The function is used to work like malloc() function in other platform.

Prototype	void* ebdat_os_mem_alloc (uint32 size)
Parameters	size: the length of the memory to allocate
Return value	The pointer to the allocated memory

Example

```
char* tmp = (char*)ebdat_os_mem_alloc(1024);
```

3.1.10. ebdat_os_mem_realloc

Description

The function is used to work like realloc() function in other platform.

Prototype	void* ebdat_os_mem_realloc (void* buffer_p, uint32 size)
Parameters	buffer_p: Pointer to previously allocated memory block. size: the new length of the memory to allocate
Return value	The pointer to the new allocated memory

Example

```
char* new_string = (cahr*)ebdat_os_mem_realloc(old_ptr, 10000);
```

3.1.11. ebdat_os_mem_free

Description

The function is used to free the memory allocated previously.

Prototype	void ebdat_os_mem_free (void* buffer_p)
Parameters	buffer_p: the memory allocated previously
Return value	None

Example

```
ebdat_os_mem_free(buffer_p);
```

3.1.12. ebdat_os_autodog

Description

The function is used to set whether kick the module dogs automatically.

Prototype	voi ebdat_os_autodog (boolean autodog)
Parameters	autodog: Whether kick the module dogs automatically. true: yes

	false: no
Return value	None

Example

```
ebdat_os_autodog(TRUE);
```

3.1.13. ebdat_os_assert_okts

Description

The function is used to enbale the module enter sleep module when EBDAT task is IDLE.

Prototype	void ebdat_os_assert_okts (void)
Parameters	None
Return value	None

Example

```
ebdat_os_assert_okts();
```

3.1.14. ebdat_os_negate_okts

Description

The function is used to forbidden the module enter sleep module when EBDAT task is IDLE.

Prototype	void ebdat_os_negate_okts ()
Parameters	None
Return value	None

Example

```
ebdat_os_negate_okts();
```

3.1.15. ebdat_os_get_usb_mode

Description

The function is used to get the current USB state.

Prototype	ebdat_usb_bus_mode_enum ebdat_os_get_usb_mode (void)
Parameters	None
Return value	the current state of the USB.

Example

```
ebdat_usb_bus_mode_enum state = ebdat_os_get_usb_mode()
```

3.1.16. ebdat_os_set_portmode

Description

The function is used to set the port mode of the module.

Prototype	boolean ebdat_os_set_portmode (uint32 mode)
Parameters	mode: the mode to set
Return value	The result of the setting TRUE: successfully FALSE: failed

Example

```
ebdat_os_set_portmode(63);
```

3.1.17. ebdat_os_get_portmode

Description

The function is used to get the port mode value.

Prototype	uint32 ebdat_os_get_portmode (void)
Parameters	None
Return value	The port mode value of current setting.

Example

```
uint32 portmode = ebdat_os_get_portmode()
```

3.1.18. ebdat_os_get_cscs

Description

The function is used to get the AT+CSCS setting.

Prototype	int ebdat_os_get_cscs()
Parameters	None
Return value	the cscs value

Example

```
val = ebdat_os_get_cscs();
```

3.1.19. ebdat_os_set_cscs

Description

The function is used to set the CSCS setting.

Prototype	int ebdat_os_set_cscs(int cscs)
Parameters	int: the cscs value IRA: 0 GSM: 1 UCS2: 2
Return value	the result of setting TRUE: successful FALSE: failed

Example

```
result= ebdat_os_set_cscs(1);
```

3.2. thread library

3.2.1. ebdat_create_thread

Description

The function is used to create a thread.

Prototype	int ebdat_create_thread (unsigned int stack_size, unsigned int priority, ebdat_thread_func_type thread_entry, unsigned int param, char* thread_name, boolean suspend)
Parameters	stack_size: The bytes of the stack for the new created thread. priority: the priority of the new created thread. If it is -1, the new thread will use the same priority of the parent thread. thread_entry: the entry of the new created thread. param: the parameter to the entry of the new created thread thread_name: the name of the new created thread Suspend: whether suspend the thread after creating the thread.
Return value	the thread identity.

Example

```
int thread_id = ebdat_thread_create(
4096,
-1,
my_thread_func,
NULL,
“my thread”,
FALSE
```

);

3.2.2. ebdat_suspend_thread

Description

The function is used to suspend a thread.

Prototype	void ebdat_suspend_thread (unsigned char thread_index)
Parameters	thread_index: the identity of the thread
Return value	None

Example

```
ebdat_suspend_thread(thread_id);
```

3.2.3. ebdat_resume_thread

Description

The function is used to resume a suspended thread.

Prototype	void ebdat_resume_thread (unsigned char thread_index)
Parameters	thread_index: the identity of the thread
Return value	None

Example

```
ebdat_resume_thread(thread_id);
```

3.2.4. ebdat_kill_thread

Description

The function is used to kill a thread.

Prototype	void ebdat_kill_thread (unsigned char thread_index)
Parameters	thread_index: the identity of the thread
Return value	None

Example

```
ebdat_kill_thread(thread_id);
```

3.2.5. ebdat_thread_running

Description

The function is check whether a thread is in running state.

Prototype	boolean ebdat_thread_running (unsigned char thread_index)
Parameters	thread_index: the identity of the thread
Return value	Whether the thread is in running state TRUE: running FALSE: not running

Example

```
boolean is_running = ebdatl_thread_running(thread_id);
```

3.2.6. ebdat_thread_suspended

Description

The function is check whether a thread is in suspend state.

Prototype	boolean ebdat_thread_suspended (unsigned char thread_index)

Parameters	thread_index: the identity of the thread
Return value	Whether the thread is in suspend state TRUE: suspended FALSE: not suspended

Example

```
boolean is_running = ebdatl_thread_running(thread_id);
```

3.2.7. ebdat_thread_get_current_index

Description

The function is used to get the identity of the current thread.

Prototype	int ebdat_thread_get_current_index (void)
Parameters	None
Return value	the identity of the thread

Example

```
int thread_identity = ebdatl_thread_get_current_index();
```

3.2.8. ebdat_set_thread_priority

Description

The function is used to set the priority of the thread.

Prototype	boolean ebdat_set_thread_priority (int thread_index, int priority)
Parameters	thread_index: the identity of the thread priority: the priority to set
Return value	The result of setting TRUE: succeeded FALSE: failed

Example

boolean result = ebdatl_set_thread_priority(1, 1);

3.2.9. ebdat_set_current_thread_priority

Description

The function is used to set the priority of the current thread.

Prototype	boolean ebdat_set_current_thread_priority (int priority)
Parameters	priority: the priority to set
Return value	The result of setting TRUE: succeeded FALSE: failed

Example

boolean result = ebdatl_set_current_thread_priority(1);

3.2.10. ebdat_get_thread_priority

Description

The function is used to set the priority of the current thread.

Prototype	int ebdat_set__thread_priority (int thread_index)
Parameters	thread_index: the identity of the thread
Return value	The priority of the thread

Example

Int priority = ebdatl_get_thread_priority(1);

3.2.11. ebdat_get_current_thread_priority

Description

The function is used to set the priority of the current thread.

Prototype	int ebdat_set__thread_priority (void)

Parameters	None
Return value	The priority of the current thread

Example

```
int priority = ebdatl_get_current_thread_priority();
```

3.2.12. ebdat_thread_sleep

Description

The function is used to let the current thread sleep.

Prototype	void ebdat_thread_sleep (uint32 ms)
Parameters	ms: the milliseconds to sleep
Return value	None

Example

```
ebdat_thread_sleep(1000);
```

3.2.13. ebdat_enter_crit_sect

Description

The function is used to let the current thread enter critical section.

Prototype	void ebdat_enter_crit_sect (uint8 cs_no)
Parameters	cs_no: the index of the critical section. It's range is from 0 to 11.
Return value	None

Example

```
ebdat_enter_crit_sect(1);
```

3.2.14. ebdat_leave_crit_sect

Description

The function is used to let the current thread leave critical section.

Prototype	void ebdat_leaver_crit_sect (uint8 cs_no)
Parameters	cs_no: the index of the critical section. It's range is from 0 to 11.
Return value	None

Example

```
ebdat_leave_crit_sect(1);
```

3.2.15. ebdat_signal_clean

Description

The function is used to clean signal for current thread.

Prototype	void ebdat_singal_clean (uint8 sig_mask)
Parameters	uint8 sig_mask: the signal to clean
Return value	None

Example

```
ebdat_signal_clean(EBDAT_SIG_1 | EBDAT_SIG_2|EBDAT_SIG_8);
```

3.2.16. ebdat_signal_notify

Description

The function is used to set signal for target thread.

Prototype	void ebdat_singal_notify(int thread_index, uint8 sig_mask)
Parameters	int thread_index: the index of the thread to set signal uint8 sig_mask: the signal to set
Return value	None

Example

```
ebdat_signal_notify(thread_index, EBDAT_SIG_1 | EBDAT_SIG_2|EBDAT_SIG_8);
```

3.2.17. ebdat_signal_wait

Description

The function is used to set signal for target thread.

Prototype	int ebdat_singal_wait(uint8 sig_mask, int timeout)
Parameters	uint8 sig_mask: the signal to wait int timeout: the millseconds to wait.
Return value	waited signal mask. If no signal waited, return 0.

Example

```
waited_mask = ebdat_signal_wait(EBDAT_SIG_1 | EBDAT_SIG_2|EBDAT_SIG_8, 5000);
if (waited_mask & EBDAT_SIG_1)
{
}
```

3.3. event library

3.3.1. ebdat_set_evt

Description

The function is used to set a event to a EBDAT thread.

Prototype	void ebdat_set_evt (uint32 evt_id, uint32 evt_p1, uint32 evt_p2, uint32 evt_p3, double evt_clock, int thread_index)
Parameters	evt_id: the identity of the event

	evt_p1: the first parameter of the event evt_p2: the second parameter of the event evt_p3: the third parameter of the event evt_clock: the tick count of the event generated. If it is 0, this value will be replaced with current tick count. thread_index: the destination thread identity.
Return value	None

Example

```
ebdat_set_evt(1, 0, 0, 0, 0, 1);
```

3.3.2. ebdat_wait_evt

Description

The function is used to get the first event in the current thread event queue.

Prototype	boolean ebdat_wait_evt (ebdat_evt_info_s_type* evt_p, uint32 timeout)
Parameters	evt_p: pointer to the event structure for output. timeout: the milliseconds to wait. 0 means not wait.
Return value	the result of waiting: true: successful false: failed

Event and parameters description

event	event_id	event_param1	event_param2	event_param3	event_clock
GPIO_EVENT	0	<gpio_param1>	<gpio_param2>	0	os.clock()
UART_EVENT	1	0	0	0	os.clock()
KEYPAD_EVENT	2	<keypad_param1>	<keypad_param2>	0	os.clock()
USB_EVENT	3	0	0	0	os.clock()
AUDIO_EVENT	4	0	0	0	os.clock()
ZIGBEE_EVENT	7	0	0	0	os.clock()
VOICECALL_EVENT	21	<vcall_param1>	<vcall_param2>	0	os.clock()
SOCKET_EVENT	22	<socket_param1>	<socket_param2>	<socket_param3>	os.clock()

SLEEP_EVENT	23	<sleep_param1>	0	0	os.clock()
COMMON_CH_EVENT	24	<comch_param1>	<comch_param2>	<comch_param2>	os.clock()
FTPS_EVENT	25	<ftps_param1>	<ftps_param2>	0	os.clock()
SMS_EVENT	26	<sms_param1>	<sms_param2>	<sms_param3>	os.clock()
TIMER_EVENT	28	<timer_param1>	<timer_param2>	<timer_param3>	os.clock()
SIO_RCVD_EVENT	29	0	0	0	os.clock()
ATCTL_EVENT	30	0	0	0	os.clock()
OUT_CMD_EVENT	31	0	0	0	os.clock()
LED_EVENT	32	<led_param1>	<led_param2>	0	os.clock()
PDP_EVENT	33	<reserved>	<reserved>		os.clock()
NMEA_EVENT	35	0	0	0	os.clock()
MULTIMEDIA_EVENT	36	<multimedia_param1>	<multimedia_param2>	0	os.clock()

Parameter defined values

event	event_id	event_param1	event_param2	event_param3	event_clock
GPIO_EVENT	0	<gpio_param1>	<gpio_param2>	0	os.clock()
UART_EVENT	1	0	0	0	os.clock()
KEYPAD_EVENT	2	<keypad_param1>	<keypad_param2>	0	os.clock()
USB_EVENT	3	0	0	0	os.clock()
AUDIO_EVENT	4	0	0	0	os.clock()
ZIGBEE_EVENT	7	0	0	0	os.clock()
VOICECALL_EVENT	21	<vcall_param1>	<vcall_param2>	0	os.clock()
SOCKET_EVENT	22	<socket_param1>	<socket_param2>	<socket_param3>	os.clock()
SLEEP_EVENT	23	<sleep_param1>	0	0	os.clock()
COMMON_CH_EVENT	24	<comch_param1>	<comch_param2>	<comch_param2>	os.clock()
FTPS_EVENT	25	<ftps_param1>	<ftps_param2>	0	os.clock()
SMS_EVENT	26	<sms_param1>	<sms_param2>	<sms_param3>	os.clock()
TIMER_EVENT	28	<timer_param1>	<timer_param2>	<timer_param3>	os.clock()
SIO_RCVD_EVENT	29	0	0	0	os.clock()
ATCTL_EVENT	30	0	0	0	os.clock()
OUT_CMD_EVENT	31	0	0	0	os.clock()
LED_EVENT	32	<led_param1>	<led_param2>	0	os.clock()
PDP_EVENT	33	<reserved>	<reserved>		os.clock()
NMEA_EVENT	35	0	0	0	os.clock()
MULTIMEDIA_EVENT	36	<multimedia_param1>	<multimedia_param2>	0	os.clock()

Example

```
boolean result = ebdat_wait_evt(&new_event, 5000);
```

3.3.3. ebdat_peek_evt

Description

The function is used to peek a event in the current thread event queue.

Prototype	boolean ebdat_peek_evt (int evt_id)
Parameters	evt_id: the identity of the event.
Return value	the result of peekting: true: successful false: failed

Example

```
boolean has_evt = ebdat_peek_evt(1);
```

3.3.4. ebdat_clear_evts

Description

The function is used to clear the current thread event queue.

Prototype	void ebdat_clear_evts (void)
Parameters	None
Return value	None

Example

```
ebdat_clear_evts();
```

3.3.5. ebdat_evt_set_evt_priority

Description

The function is used to set the priority of a event.

Prototype	void ebdat_evt_set_evt_priority(int evt, int priority)
Parameters	evt: the event identity priority: the priority to set

Return value	NOne

Example

```
ebdat_evt_set_evt_priority(1, 100);
```

3.3.6. ebdat_evt_get_evt_priority

Description

The function is used to get the priority of a event.

Prototype	int ebdat_evt_get_evt_priority(int evt)
Parameters	evt: the event identity
Return value	the priority of the event

Example

```
int evt_priority = ebdat_evt_get_evt_priority(1);
```

3.3.7. ebdat_evt_set_evt_owner_thread_idx

Description

The function is used to set the owner thread identity of a event

Prototype	void ebdat_evt_set_evt_owner_thread_idx(int evt, int thread_index)
Parameters	evt: the event identity thread_index: the index of the thread
Return value	None

Example

```
ebdat_evt_set_evt_owner_thread_idx(1, 1);
```

3.3.8. ebdat_evt_get_evt_owner_thread_idx

Description

The function is used to get the owner thread index.

Prototype	int ebdat_evt_get_evt_owner_thread_idx(int evt)
Parameters	evt: the event identity
Return value	The thread identity that own the event.

Example

```
int event_thread_id = ebdat_evt_get_evt_owner_thread_idx(1);
```

3.3.9. ebdat_evt_set_evt_as_ignored

Description

The function is used to set whether the event should be discarded

Prototype	void ebdat_evt_set_evt_as_ignored(int evt, boolean discarded)
Parameters	evt: the event identity discarded: whether the event should be discarded
Return value	None

Example

```
ebdat_evt_set_evt_as_ignored(1, TRUE);
```

3.3.10. ebdat_evt_is_evt_ignored

Description

The function is used to check whether an event is discarded

Prototype	boolean ebdat_evt_is_evt_ignored(int evt)
Parameters	evt: the event identity
Return value	Whether the event is discarded: TRUE: yes FALSE: no

Example

```
boolean is_discarded = ebdat_evt_is_evt_ignored(1);
```

3.3.11. ebdat_get_evt_count_in_queue

Description

The function is used to statistics the count of events for a thread with specific event id and parameters.

Prototype	int ebdat_get_evt_count_in_queue(int* evt_id_p, int* evt_p1_p, int* evt_p2_p, int evt_p3_p)
Parameters	int* evt_id_p: the event id. If it is NULL, this function will return the total count of events cached for this thread. int* evt_p1_p: the first parameter. If it is NULL, it will be ignored. int* evt_p2_p: the second parameter. If it is NULL, it will be ignored. int* evt_p3_p: the third parameter. If it is NULL, it will be ignored.
Return value	the count of events with specific event_id and parameters.

Example

```
int evt_count = ebdat_get_evt_count_in_queue(NULL, NULL, NULL, NULL);
```

3.3.12. ebdat_delete_spec_evt_with_params_from_queue

Description

The function is used to delete event with specified parameters from queue.

Prototype	void ebdat_delete_spec_evt_with_params_from_queue(int delete_count, int evt_id, int* evt_p1_p, int* evt_p2_p, int evt_p3_p)
Parameters	int delete_count: the count of events to delete. int evt_id: the event id. int* evt_p1_p: the first parameter. If it is NULL, it will be ignored.

	int* evt_p2_p: the second parameter. If it is NULL, it will be ignored. int* evt_p3_p: the third parameter. If it is NULL, it will be ignored.
Return value	None

Example

ebdat_delete_spec_evt_with_params_from_queue(1000, 5, NULL, NULL, NULL);

3.3.13. ebdat_evt_filter_add

Description

The function is used to add event filter for queue.

Prototype	int ebdat_evt_filter_add(int evt_id, int* evt_p1_p, int* evt_p2_p, int evt_p3_p, int max_count_in_queue, boolean replace_oldest)
Parameters	int evt_id: the event id. int* evt_p1_p: the first parameter. If it is NULL, it will be ignored. int* evt_p2_p: the second parameter. If it is NULL, it will be ignored. int* evt_p3_p: the third parameter. If it is NULL, it will be ignored. int max_count_in_queue: the maximum count of specified event can be pushed ino queue. boolean replace_oldest: If the count reaches max_count_in_queue, whether replace the oldest event with specified id.
Return value	The filter index. 0 to 9. If failed, return -1.

Example

int filter_index = ebdat_evt_filter_add(5, NULL, NULL, NULL, 2, TRUE);

3.3.14. ebdat_evt_filter_delete

Description

The function is used to delete event filter for queue.

Prototype	boolean ebdat_evt_filter_delete(int filter_index)
Parameters	int filter_index: the index of the filter to delete.
Return value	The result of operating: TRUE: successful FALSE: failed

Example

```
boolean rst = ebdat_evt_filter_delete(filter_index);
```

3.4. timer library

3.4.1. ebdat_starttimer

Description

The function is used to start a timer.

Prototype	boolean ebdat_starttimer(uint8 timer_id, uint32 time_out, boolean periodic)
Parameters	timer_id: the identity of the timer. It's range is from 0 to 19. time_out: the timeout value for the timer. periodic: the type of the timer.
Return value	Whether the timer is started successfully: TRUE: yes FALSE: no

Example

```
ebdat_starttimer(1, 1000, FALSE);
```

3.4.2. ebdat_stoptimer

Description

The function is used to stop the timer.

Prototype	void ebdat_stoptimer(uint8 timer_id)
Parameters	timer_id: the timer identity.
Return value	None

Example

```
ebdat_stoptimer(1);
```

3.5. sio library

3.5.1. ebdat_sio_send

Description

The function is used to send AT command to the module.

Prototype	boolean ebdat_sio_send (const char* cmd, uint32 len)
Parameters	cmd: the AT command string to be send len: the length of the AT command string.
Return value	the result of sending: true: successful false: failed

Example

```
ebdat_sio_send("ATI\r\n");
```

3.5.2. ebdat_sio_recv

Description

The function is used to receive AT response or URC from the module.

Prototype	int ebdat_sio_recv (byte* recv_data_p, const uint32 size_of_recv_buf, uint32 timeout)
Parameters	recv_data_p: the buffer to receive response size_of_recv_buf: the size of the buffer. timeout: the timeout value of receiving.
Return value	The received data length.

Example

```
int rcvd_len = ebdat_sio_recv(tmp_buf, sizeof(tmp_buf), 5000);
```

3.5.3. ebdat_sio_enable_recv

Description

The function is used to set whether or not to cache the AT response or URC in EBDAT core layer.

Prototype	void ebdat_sio_enable_recv (boolean enable)
Parameters	enable: whether or not to cache the AT response or URC TRUE: enable FALSE: disable
Return value	None

Example

```
ebdat_sio_enable_recv(FALSE);
```

3.5.4. ebdat_sio_clear

Description

The function is used to clear the AT response and URC cached in EBDAT core layer.

Prototype	void ebdat_sio_clear (void)
Parameters	None

Return value	None

Example

```
ebdat_sio_clear();
```

3.5.5. ebdat_sio_exclrpt

Description

The function is used to set whether the AT response and URC should only be reported to EBDAT core layer.

Prototype	void ebdat_sio_exclrpt (boolean exclusive_report)
Parameters	exclusive_report: Whether the AT response and URC should only be reported to EBDAT core layer. TRUE: only report to EBDAT core layer FALSE: report to all.(default)
Return value	None

Example

```
Ebdat_sio_exclrpt(FALSE);
```

3.6. efs library

3.6.1. ebdat_efs_creat

Description

The function is used to create a file.

Prototype	int ebdat_efs_creat (const char* path, uint16 mode)
Parameters	path: the full path of the file to create mode: the mode to create the file
Return value	the file handle

Example

rst = ebdat_efs_creat("c:\\mydir\\test.txt", S_IREAD|S_IWRITE);

3.6.2. ebdat_efs_open

Description

The function is used to create or open a file.

Prototype	int ebdat_efs_open (const char* path, int flag)
Parameters	path: the full path of the file to create or open flag: the flag value of opening
Return value	the file handle

Example

rst = ebdat_efs_open("c:\\mydir\\test.txt", O_CREAT | O_TRUNC | O_WRONLY);

3.6.3. ebdat_efs_close

Description

The function is used to close an opened file.

Prototype	int ebdat_efs_close (int filedes)
Parameters	filedes: the file handle to close
Return value	the result of operating: 1: successful 0: failed

Example

rst = ebdat_efs_close(filedes);

3.6.4. ebdat_efs_read

Description

The function is used to read data from an opened file.

Prototype	int ebdat_efs_read (int filedes, void* buf, fs_size_t

	nbyte)
Parameters	filedes: the file handle buf: the buffer to contain the read data nbyte: the bytes to read
Return value	the length of data read into buffer

Example

```
int bytes_read = ebdat_efs_read(filedes, buf, sizeof(buf));
```

3.6.5. ebdat_efs_write

Description

The function is used to write data to an opened file.

Prototype	int ebdat_efs_write (int filedes, const void* buf, fs_size_t nbytes)
Parameters	filedes: the file handle buf: the buffer to write nbytes: the length of data to write
Return value	the length of data written.

Example

```
length_written = ebdat_efs_write(filedes, buf, sizeof(buf));
```

3.6.6. ebdat_efs_ftruncate

Description

The function is used to truncate an opened file.

Prototype	int ebdat_efs_ftruncate (int fd, fs_off_t length)
Parameters	fd: the file handle length: the length reserved after truncating
Return value	the result of operating:

	0: successful 1: failed

Example

```
rst= ebdat_efs_ftruncate(filedes, 0);
```

3.6.7. ebdat_efs_lseek

Description

The function is used to seek in an opened file.

Prototype	int ebdat_efs_lseek (int filedes, fs_off_t offset, int whence)
Parameters	filedes: the file handle offset: the offset to seek whence: the initial position to seek
Return value	the result of operating: 0: successful 1: failed

Example

```
rst = ebdat_efs_lseek(filedes, 0, SEEK_END);
```

3.6.8. ebdat_efs_stat

Description

The function is used to statistic a file.

Prototype	int ebdat_efs_stat (const char* path, struct fs_stat* buf)
Parameters	path: the full path of the file buf: the buffer to contain the result
Return value	the result of operating: 0: successful

	1: failed

Example

rst = ebdat_efs_stat("c:\\mydir\\1.txt", buf);

3.6.9. ebdat_efs_fstat

Description

The function is used to statistic an opened file.

Prototype	int ebdat_efs_fstat (int filedes, struct fs_stat* buf)
Parameters	filedes: the file identity buf: the buffer to contain the result
Return value	the result of operating: 0: successful 1: failed

Example

rst = ebdat_efs_fstat(filedes, buf);

3.6.10. ebdat_efs_lstat

Description

The function is used to statistic a file. If the file is a symbol, it will return the information of the symbol.

Prototype	int ebdat_efs_lstat (const char* path, struct fs_stat* buf)
Parameters	path: the full path of the file or symbol buf: the buffer to contain the result
Return value	the result of operating: 0: successful 1: failed

Example

rst = ebdat_efs_lstat("c:\\mydir\\1.txt", buf);

3.6.11. ebdat_efs_tell

Description

The function is used to get the offset of the file pointer.

Prototype	int ebdat_efs_tell (int filedes)
Parameters	filedes: the file identity
Return value	the offset of the file pointer

Example

offset = ebdat_efs_tell(filedes);

3.6.12. ebdat_efs_get_opened_filesize

Description

The function is used to get the opened file size.

Prototype	int ebdat_efs_get_opened_filesize (int filedes)
Parameters	filedes: the file identity
Return value	The size of the opened file

Example

fsize = ebdat_efs_get_opened_filesize(filedes);

3.6.13. ebdat_efs_file_exist

Description

The function is used to check whether a file exist.

Prototype	boolean ebdat_efs_file_exist (const char* path)
Parameters	path: the full path of the file
Return value	Whether the file exist: TRUE: yes

	FALSE: no

Example

```
exist = ebdat_efs_file_exist("c:\\mydir\\1.txt");
```

3.6.14. ebdat_efs_dir_exist

Description

The function is used to check whether a directory exist.

Prototype	boolean ebdat_efs_dir_exist (const char* path)
Parameters	path: the full path of the directory
Return value	Whether the directory exist: TRUE: yes FALSE: no

Example

```
exist = ebdat_efs_dir_exist("c:\\mydir\\mysubdir");
```

3.6.15. ebdat_efs_delete_file

Description

The function is used to delete a file by name.

Prototype	boolean ebdat_efs_delete_file (const char* path)
Parameters	path: the full path of the file
Return value	The result of operating: TRUE: successful FALSE: failed

Example

```
rst = ebdat_efs_delete_file("c:\\mydir\\1.txt");
```

3.6.16. ebdat_efs_rmdir

Description

The function is used to remove a directory by name.

Prototype	boolean ebdat_efs_rmdir (const char* path)
Parameters	path: the full path of the directory
Return value	The result of operating: TRUE: successful FALSE: failed

Example

```
rst = ebdat_efs_rmdir("c:\\mydir\\mysubdir");
```

3.6.17. ebdat_efs_mkdir

Description

The function is used to create a directory by name.

Prototype	boolean ebdat_efs_mkdir (const char* path)
Parameters	path: the full path of the directory
Return value	The result of operating: TRUE: successful FALSE: failed

Example

```
rst = ebdat_efs_mkdir("c:\\mydir\\mysubdir");
```

3.6.18. ebdat_efs_get_filesize

Description

The function is used to get the length of a file.

Prototype	int ebdat_efs_get_filesize (const char* path)

Parameters	path: the full path of the file
Return value	The length of the file.

Example

length = ebdat_efs_get_filesize("c:\\mydir\\1.txt");

3.6.19. ebdat_efs_truncate_file

Description

The function is used to truncate a file by name.

Prototype	int ebdat_efs_truncate_file (const char* path, int length)
Parameters	path: the full path of the file length: the length to reserve after truncating.
Return value	The result of operating: TRUE: successful FALSE: failed

Example

rst = ebdat_efs_truncate_file("c:\\mydir\\1.txt", 100);

3.6.20. ebdat_efs_lsdir

Description

The function is used to list all directories under a specified folder.

Prototype	int ebdat_efs_lsdir (const char* dirname, ebdat_efs_list_cb list_cb, void* user_data)
Parameters	dirname: the specified folder list_cb: the callback function for the list operation. user_data: the user data that will bypass to list_cb
Return value	The count of the directories found

Example

ebdat_efs_lsdir("c:\\mydir", my_list_cb, NULL);

3.6.21. ebdat_efs_lsfile

Description

The function is used to list all files under a specified folder.

Prototype	int ebdat_efs_lsfile (const char* dirname, ebdat_efs_list_cb list_cb, void* user_data)
Parameters	dirname: the specified folder list_cb: the callback function for the list operation. user_data: the user data that will bypass to list_cb
Return value	The count of the files found

Example

ebdat_efs_lsfile("c:\\mydir", my_list_cb, NULL);

3.7. atctl library

3.7.1. ebdat_atctl_setport

Description

The function is used to set the port used as AT CONTROL mode.

Prototype	void ebdat_atctl_setport (int port)
Parameters	port: the port to set as ATCTL mode 1: UART 2: USB MODEM 3: USB AT -1: NONE
Return value	None

Example

rst = ebdat_atctl_setport(1);

3.7.2. ebdat_atctl_send

Description

The function is used to send data to the ATCTL port.

Prototype	boolean ebdat_atctl_send (const byte* cmd, uint32 len)
Parameters	cmd: the string to send len: the length of the command to send
Return value	the result of operating: TRUE: successful FALSE: failed

Example

rst= ebdat_atctl_send(buf, sizeof(buf));

3.7.3. ebdat_atctl_recv

Description

The function is used to receive data from the ATCTL port.

Prototype	int ebdat_atctl_recv (byte* recv_data_p, const uint32 size_of_recv_buf, uint32 timeout)
Parameters	recv_data_p: the buffer to receive data size_of_recv_buf: the size of the buffer. timeout: the timeout value of receiving.
Return value	The received data length.

Example

int rcvd_len = ebdat_atctl_recv(tmp_buf, sizeof(tmp_buf), 5000);

3.7.4. ebdat_atctl_clear

Description

The function is used to clear the received data cached in EBDAT ATCTL queue.

Prototype	void ebdat_atctl_clear (void)
Parameters	None
Return value	None

Example

```
ebdat_atctl_clear();
```

3.8. ftp library

3.8.1. ebdat_ftp_simpput

Description

The function is used to put a file from local EFS to the remote FTP server. For the corresponding AT command, please refer to AT+CFTPPUTFILE.

Prototype	int ebdat_ftp_simpput (const char* address, unsigned short port, const char* name, const char* password, const char* remote_filepath, const char* local_filepath, boolean passive, uint32 rest_size)

Parameters	address: FTP server address port: FTP server port name: FTP user name password: FTP user password remote_filepath: the path of the remote file . local_filepath: the path of the local EFS file passive: passive mode (1=passive mode, 0=not passive mode) rest_size : The value for FTP "REST" command which is used for broken transfer when transferring failed last time. It's range is 0 to 2147483647.
Return value	the result of the FTP putting: 0: successful other: failed

Example

```
server = "e-device.net";
port = 21;
name = "myaccount";
pass = "password";
remote_file = "/up_normal.jpg";
uplocal_file = "c:\\Picture\\normal.jpg";
passive = 0;
rst = ebdat_ftp_simpput(server, port, name, pass, remote_file, uplocal_file, passive);
```

3.8.2. ebdat_ftp_simpget

Description

The function is used to get a file from the remote FTP server to local EFS. For the corresponding AT command, please refer to AT+CFTPGETFILE.

Prototype	int ebdat_ftp_simpget (

	const char* address, unsigned short port, const char* name, const char* password, const char* remote_filepath, const char* local_filepath, boolean passive, uint32 rest_size)
Parameters	address: FTP server address port: FTP server port name: FTP user name password: FTP user password remote_filepath: the path of the remote file. local_filepath: the path of the local EFS file passive: passive mode (1=passive mode, 0=not passive mode) rest_size : The value for FTP “REST” command which is used for broken transfer when transferring failed last time. It’s range is 0 to 2147483647.
Return value	the result of the FTP getting: 0: successful other: failed

Example

```
server = "e-device.net";
port = 21;
name = "myaccount";
pass = "password";
remote_file = "/up_normal.jpg";
```

downlocal_file = "c:\\Video\\down_normal.jpg";

passive = 0;

rst = ebdat_ftp_simpget(server, port, name, pass, remote_file, downlocal_file, passive);

3.8.3. ebdat_ftp_simplist

Description

The function is used to get the file and directory list from the remote FTP server to local EFS. For the corresponding AT command, please refer to AT+CFTPLIST.

Prototype	Int string ftp.simplist (const char* address, unsigned short port, const char* name, const char* password, const char* remote_path, boolean passive)
Parameters	address: FTP server address port: FTP server port name: FTP user name password: FTP user password remote_path: the path of the remote file. passive: passive mode (1=passive mode, 0=not passive mode)
Return value	the result of the FTP listing: int result: 0: successful other: failed string list_data: the data of the list command returns.

Example

```
server = "e-device.net";
port = 21;
name = "myaccount";
pass = "password";
remote_path = "/MyDir/";
passive = 0;
rst, list_data= ebdat_ftp_simplist(server, port, name, pass, remote_path, passive);
```

3.9. smtp library

3.9.1. ebdat_smtp_config

Description

The function is used to configure SMTP server and user password.

Prototype	boolean ebdat_smtp_config(const char* server, int port, const char* username, const char* password)
Parameters	server: the server IP or domain name port: the port the server IP username: the user name of the SMTP server password: the password of the SMTP server
Return value	the result of the configuring: TRUE: successful FALSE: failed

Example

```
rst = ebdat_smtp_config("smtp.163.com",25,"songjin_hello","123456");
```

3.9.2. ebdat_smtp_set_from

Description

The function is used to configure sender address and name information of the email.

Prototype	boolean ebdat_smtp_set_from(const char* sender_address, const char* sender_name)
Parameters	sender_address: the address of the sender sender_name: the name of the sender
Return value	the result of the setting: TRUE: successful FALSE: failed

Example

```
rst = ebdat_smtp_set_from("<songjin_hello@163.com>", "");
```

3.9.3. ebdat_smtp_set_rcpt

Description

The function is used to configure recipient address and name information of the email.

Prototype	boolean ebdat_smtp_set_rcpt(int kind, int index, const char* rcpt_address, const char* rcpt_name)

Parameters	kind: the type of the address 0: To normal recipient 1: CC, Carbon Copy recipient 2: BCC Bind Carbon Copy recipient index: the index of the recipient rcpt_address: the address of the recipient rcpt_name: the name of the recipient
Return value	the result of the setting: TRUE: successful FALSE: failed

Example

```
rst = smtp.set_rcpt(0, 0, "<songjin_hello@163.com>", "");
```

3.9.4. ebdat_smtp_set_subject

Description

The function is used to configure the subject information of the email.

Prototype	boolean ebdat_smtp_set_subject(const char* subject)
Parameters	subject: the subject of the email
Return value	the result of the setting: TRUE: successful FALSE: failed

Example

```
rst = ebdat_smtp_set_subject("test subject");
```

3.9.5. ebdat_smtp_set_body

Description

The function is used to configure the subject information of the email.

Prototype	boolean ebdat_smtp_set_body(const char* body, uint32 len)
Parameters	body: the body of the email len: the length of body
Return value	the result of the setting: TRUE: successful FALSE: failed

Example

rst = ebdat_smtp_set_body("test body", strlen(“test body”));

3.9.6. ebdat_smtp_set_body_charset

Description

The function is used to configure character set of the subject information of the email.

Prototype	boolean ebdat_smtp_set_body_charset(const char* charset)
Parameters	charset: the character set of the email body
Return value	the result of the setting: TRUE: successful FALSE: failed

Example

rst = ebdat_smtp_set_body_character("utf-8");

3.9.7. ebdat_smtp_set_file

Description

The function is used to configure the attachment information of the email.

Prototype	boolean ebdat_smtp_set_file(

	int attach_no, const char* filepath)
Parameters	attach_no: the index of the attachment. filepath: the full path of the file
Return value	the result of the setting: TRUE: successful FALSE: failed

Example

```
rst = ebdat_smtp_set_file(0, "C:\\Picture\\test.jpg");
```

3.9.8. ebdat_smtp_send

Description

The function is used to send the email.

Prototype	int ebdat_smtp_send()
Parameters	None
Return value	the result of the sending: 1: successful Other: failed

Example

```
rst = ebdat_smtp_send();
```

3.10. mms library

3.10.1. ebdat_mms_accquire_module

Description

The function is used to acquire MMS module for EBDAT.

Prototype	boolean ebdat_mms_accquire_module()
Parameters	None
Return value	the result of the accquiring: TRUE: successful FALSE: failed

Example

```
rst = ebdat_mms_accquire_module();
```

3.10.2. ebdat_mms_release_module

Description

The function is used to release MMS module for EBDAT.

Prototype	void ebdat_mms_release_module()
Parameters	None
Return value	None

Example

```
ebdat_mms_release_module();
```

3.10.3. ebdat_mms_set_mmsc

Description

The function is used to set MMSC.

Prototype	boolean ebdat_mms_set_mmsc(const char* url)
Parameters	url: the MMSC URL
Return value	the result of the setting: TRUE: successful FALSE: failed

Example

```
rst = ebdat_mms_set_mmsc("http://my.mmsc.com/service");
```

3.10.4. ebdat_mms_get_mmsc

Description

The function is used to set MMSC.

Prototype	boolean ebdat_mms_get_mmsc(char* mmsc_buf, uint32 buf_size)
Parameters	mmsc_buf: the buffer to contain the MMSC URL buf_size: the size of the buffer
Return value	the result of the getting: TRUE: successful FALSE: failed

Example

```
rst = ebdat_mms_get_mmsc(url, sizeof(url));
```

3.10.5. ebdat_mms_set_protocol

Description

The function is used to set MMS protocol and gateway address.

Prototype	boolean ebdat_mms_set_protocol(int protocol, const char* proxy, int port)
Parameters	protocol: the protocol used for MMS 0=WAP, 1=HTTP proxy: the IP address of MMSC gateway port: the port of MMSC gateway
Return value	the result of the setting: TRUE: successful FALSE: failed

Example

rst = ebdat_mms_set_protocol(1, “10.0.0.172”, 80);

3.10.6. ebdat_mms_get_protocol

Description

The function is used to get MMS protocol and gateway address.

Prototype	boolean ebdat_mms_get_protocol(int* protocol, char* proxy, uint32 proxy_buf_size, int* port)
Parameters	protocol: the protocol used for MMS 0=WAP, 1=HTTP proxy: the IP address of MMSC gateway proxy_buf_size: the size of buffer to contain proxy port: the port of MMSC gateway
Return value	the result of the getting: TRUE: successful FALSE: failed

Example

rst = ebdat_mms_get_protocol(&protocol, proxy, sizeof(proxy), &port);

3.10.7. ebdat_mms_set_edit

Description

The function is used to set MMS edit state.

Prototype	boolean ebdat_mms_set_edit(int edit_state)
Parameters	edit_state: the state of edit

	0=not editable, 1=editable
Return value	the result of the setting: TRUE: successful FALSE: failed

Example

```
rst = ebdat_mms_set_edit(1);
```

3.10.8. ebdat_mms_set_title

Description

The function is used to set MMS title.

Prototype	boolean ebdat_mms_set_title(const char* title, int len)
Parameters	title: the title of the MMS len: the length of the title
Return value	the result of the setting: TRUE: successful FALSE: failed

Example

```
rst = ebdat_mms_set_title("test title", strlen("test title"));
```

3.10.9. ebdat_mms_get_title

Description

The function is used to get MMS title.

Prototype	int ebdat_mms_get_title(char* title_buf,

	uint32 title_buf_size, boolean convert_utf8_to_unicode)
Parameters	title_buf: the buffer to contain the title title_buf_size: the size of buffer to contain the title convert_utf8_to_unicode: whether to convert the title to Unicode format if it is UTF-8 format TRUE: convert FALSE: not convert
Return value	the length of title got

Example

```
title_len = ebdat_mms_get_title(title, sizeof(title), FALSE);
```

3.10.10. ebdat_mms_attach_file

Description

The function is used to add an attachment to the MMS from EFS.

Prototype	int ebdat_mms_attach_file(const char* file_path)
Parameters	Char* file_path: the path of the file to attach
Return value	int: the result of the attaching: 0: successful other value: failed

Example

```
err_code= ebdat_mms_attach_file("c:\\Picture\\1.jpg");
```

3.10.11. ebdat_mms_attach_file_from_memory

Description

The function is used to add an attachment to the MMS from memory.

Prototype	int ebdat_mms_attach_file_from_memory (int source_type, const char* source_name, const char* content, uint32 content_len)
Parameters	source_type: the type of the attachment 1=image, 2=text, 3=audio, 4=video source_name: the attachment name content: the attachment content content_len: the length of the content.
Return value	int: the result of the attaching: 0: successful other value: failed

Example

err_code= ebdat_mms_attach_file_from_memory(2, “1.log”, “this is the log content”, strlen(“this is the log content”));

3.10.12. ebdat_mms_add_receipt

Description

The function is used to add an receipt/cc/bcc address to the MMS.

Prototype	boolean ebdat_mms_add_receipt (const char* receipt, int addr_property)
Parameters	receipt: the receipt to add addr_peoperty: the type of the receipt 0=receipt,1=cc,2=bcc

Return value	the result of the adding: TRUE: successful FALSE: failed

Example

rst= ebdat_mms_add_receipt(“18602100071”, 0);

3.10.13. ebdat_mms_delete_receipt

Description

The function is used to delete an receipt/cc/bcc address to the MMS.

Prototype	int ebdat_mms_delete_receipt (const char* receipt, int receipt_type)
Parameters	receipt: the receipt to delete receipt_type: the type of the receipt 0=receipt,1=cc,2=bcc
Return value	the result of the deleting: TRUE: successful FALSE: failed

Example

rst= ebdat_mms_add_receipt(“18602100071”, 0);

3.10.14. ebdat_mms_list_receipts

Description

The function is used to list all the receipt/cc/bcc addresses from the MMS.

Prototype	int ebdat_mms_list_receipt (int receipt_type,

	ebdat_mms_list_receipt_cb list_cb, void* user_data)
Parameters	receipt_type: the type of the receipt 0=receipt,1=cc,2=bcc list_cb: the callback function for listing. user_data: the user data that will be bypassed to list_cb
Return value	The count of receipts listed

Example

```
count_of_receipts = ebdat_mms_list_receipts(0, list_cb, NULL);
```

3.10.15. ebdat_mms_save_attachment

Description

The function is used to save an attachment in the MMS to the EFS.

Prototype	int ebdat_mms_save_attachment (int attach_no, const char* file_path)
Parameters	attch_no: the attachment number file_path: the path of the file
Return value	int: the result of saving 0: successful other value: failed

Example

```
err_code = ebdat_mms_save_attachment(0, "c:\\test1.txt");
```

3.10.16. ebdat_mms_save

Description

The function is used to save the MMS to the EFS.

Prototype	int ebdat_mms_save (int box, uint32 mail_type)
Parameters	box: the box number(0 or 1) mail_type: the type of the mail to save
Return value	int: the result of saving 0: successful other value: failed

Example

```
err_code = ebdat_mms_save (0, 2);
```

3.10.17. ebdat_mms_load

Description

The function is used to load the MMS from the EFS.

Prototype	int ebdat_mms_load (int box)
Parameters	box: the box number(0 or 1)
Return value	int: the result of loading 0: successful other value: failed

Example

```
err_code = ebdat_mms_load (0);
```

3.10.18. ebdat_mms_get_attach_info

Description

The function is used to get the information of an attachment in the MMS.

Prototype	boolean ebdat_mms_get_attach_info(int fileIndex, char* filename, int filename_size, int* content_type_p, int* data_length_p)
Parameters	int fileIndex: the attachment number filename: the buffer to contain the attachment name filename_size: the size of buffer filename content_type_p: the content type data_length_p: the length of data
Return value	the result of operating TRUE: successful FALSE: failed

Example

```
rst = ebdat_mms_get_attach_info (attach_no, name, sizeof(name), &type, &size);
```

3.10.19. ebdat_mms_list_attach_info

Description

The function is used to list the information of all attachments in the MMS.

Prototype	int ebdat_mms_list_attach_info(ebdat_mms_list_attach_info_cb list_cb, void* user_data)
Parameters	list_cb: the callback function for listing user_data; the user data that will be bypassed to list_cb

Return value	the count of attachments listed

Example

count_of_attachments= ebdat_mms_get_attach_info (list_cb, NULL);

3.10.20. ebdat_mms_get_delivery_date_info

Description

The function is used to get the delivery date of the MMS.

Prototype	boolean ebdat_mms_get_delivery_date_info(int* year_p, int* month_p, int* day_p, int* hour_p, int* minute_p, int* second_p)
Parameters	year_p: the year month_p: the month day_p: the day hour_p: the hour minute_p: the minute second_p: the second
Return value	the result of operating TRUE: successful FALSE: failed

Example

dt = mms.get_delivery_date();

print("delivery_date = ", dt.year, "-", dt.month, "-", dt.day, " ", dt.hour, ":", dt.min, ":", dt.sec, "\r\n");

3.10.21. ebdat_mms_read_attachment

Description

The function is used to read an attachment in the MMS.

Prototype	int ebdat_mms_read_attachment (int attach_no, char* buf, uint32 buf_size)
Parameters	attach_no: the attachment number buf: the buffer to contain the read content buf_size: the size of buf
Return value	The length of data read

Example

```
read_len = ebdat_mms_read_attachment (attach_no, buf, sizeof(buf));
```

3.10.22. ebdat_mms_send

Description

The function is used to send MMS.

Prototype	int ebdat_mms_send (void)
Parameters	None
Return value	the result of sending 0: successful other value: failed

Example

```
rst = ebdat_mms_send ();
```

3.10.23. ebdat_mms_recv

Description

The function is used to receive MMS.

Prototype	int ebdat_mms_recv (const char* url)
Parameters	url: the MMS url to receive
Return value	the result of sending 0: successful other value: failed

Example

```
rst = ebdat_mms_recv (url);
```

3.11.sms library

3.11.1. ebdat_sms_ready

Description

The function is used to check whether the SMS module is ready.

Prototype	boolean ebdat_sms_ready(void)
Parameters	None
Return value	the result: TRUE: ready FALSE: not ready

Example

```
ready= ebdat_sms_ready();
```

3.11.2. ebdat_sms_cpms

Description

The function is used to set or query CPMS setting(AT+CPMS).

Prototype	boolean ebdat_sms_cpms(ebdat_sms_mem_input_s_type* mem_input_ptr, ebdat_sms_cpms_result_s_type* cpms_result_ptr)
Parameters	ebdat_sms_mem_input_s_type* mem_input_ptr: the memory to set. If NULL, just query the current setting. ebdat_sms_cpms_result_s_type* cpms_result_ptr: this is an output parameter which contains the current CPMS setting.
Return value	the result of operating: TRUE: successful FALSE: failed

Example

```
rst= ebdat_sms_cpms(NULL, &cpms_setting);
```

3.11.3. ebdat_sms_get_csmp

Description

The function is used to get the CSMP setting(AT+CSMP).

Prototype	boolean ebdat_sms_get_csmp(ebdat_sms_get_csmp_result_s_type* result_ptr)
Parameters	ebdat_sms_get_csmp_result_s_type* cresult_ptr: this is an output parameter which contains the current CSMP setting.
Return value	the result of operating: TRUE: successful FALSE: failed

Example

```
rst= ebdat_sms_get_csmp( &csmp_setting);
```

3.11.4. ebdat_sms_set_csmp

Description

The function is used to set the CSMP setting (AT+CSMP).

Prototype	boolean ebdat_sms_set_csmp(int fo, int vp, int pid, int dcs)
Parameters	int fo, vp, pid, dcs: the same as AT+CSMP.
Return value	the result of operating: TRUE: successful FALSE: failed

Example

```
rst= ebdat_sms_set_csmp( fo, vp, pid, dcs);
```

3.11.5. ebdat_sms_get_cnmi

Description

The function is used to get the CNMI setting (AT+CNMI).

Prototype	boolean ebdat_sms_get_cnmi(ebdat_sms_get_csnmi_result_s_type* result_ptr)
Parameters	ebdat_sms_get_cnmi_result_s_type* cresult_ptr: this is an output parameter which contains the current CNMI setting.
Return value	the result of operating: TRUE: successful FALSE: failed

Example

```
rst= ebdat_sms_get_cnmi( &cnmi_setting);
```

3.11.6. ebdat_sms_set_cnmi

Description

The function is used to set the CNMI setting (AT+CNMI).

Prototype	boolean ebdat_sms_set_cnmi(int mode, int mt, int bm, int ds, int bfr)
Parameters	int mode, mt, bm, ds, bfr: the same as AT+CNMI.
Return value	the result of operating: TRUE: successful FALSE: failed

Example

```
rst= ebdat_sms_set_cnmi( mode, mt, bm, ds, bfr);
```

3.11.7. ebdat_sms_get_csca

Description

The function is used to get the CSCA setting (AT+CSCA). The output csca string is always in ASCII mode.

Prototype	boolean ebdat_sms_get_csca(char* csca_ptr, uint32 csca_buf_size)
Parameters	char* csca_ptr: the buffer to contain result. uint32 csca_buf_size: the size of the buffer csca_ptr.
Return value	the result of operating: TRUE: successful FALSE: failed

Example

```
rst= ebdat_sms_get_csca(csca_buffer, sizeof(csca_buffer));
```

3.11.8. ebdat_sms_set_csca

Description

The function is used to set the CSCA setting (AT+CSCA).

Prototype	boolean ebdat_sms_set_csca(const char* csca_p)
Parameters	char* csca_p: the CSCA number.
Return value	the result of operating: TRUE: successful FALSE: failed

Example

```
rst= ebdat_sms_set_csca( csca_no);
```

3.11.9. ebdat_sms_get_csdh

Description

The function is used to get the CSDH setting (AT+CSDH).

Prototype	int ebdat_sms_get_csdh(void)
Parameters	None
Return value	the CSDH setting.

Example

```
csdh = ebdat_sms_get_csdh ();
```

3.11.10. ebdat_sms_set_csdh

Description

The function is used to set the CSDH setting (AT+CSDH).

Prototype	boolean ebdat_sms_set_csdh(int csdh)
Parameters	int csdh: the CSDH number.
Return value	the result of operating: TRUE: successful FALSE: failed

Example

rst= ebdat_sms_set_csdh(csdh_no);

3.11.11. ebdat_sms_get_cmgf

Description

The function is used to get the AT+CMGF setting.

Prototype	int ebdat_sms_get_cmgf(void)
Parameters	None
Return value	int: the cmgf value

Example

val = ebdat_sms_get_cmgf();

3.11.12. ebdat_sms_set_cmgf

Description

The function is used to set the CMGF setting.

Prototype	boolean ebdat_sms_set_cmgf(int cmgf)
Parameters	cmgf: the cmgf value
Return value	the result of setting TRUE: successful FALSE: failed

Example

result= ebdat_sms_set_cmgf(1);

3.11.13. ebdat_sms_modify_msg_tag

Description

The function is used to modify message tag from unread to read like AT+CMGMT.

Prototype	boolean ebdat_sms_modify_tag(int index)
Parameters	int index: the index of the message

Return value	the result of operating: TRUE: successful FALSE: failed

Example

result= ebdat_sms_modify_msg_tag(1);

3.11.14. ebdat_sms_delete_msg

Description

The function is used to delete SMS.

Prototype	boolean ebdat_sms_delete_msg(int msg_index, int delflag)
Parameters	int msg_index, the index of the message to delete. int delflag: this can be set to the combination of the following value, default 0.
Return value	the result of operating: TRUE: successful FALSE: failed

Example

result= ebdat_sms_delete_msg(1, 0);

3.11.15. ebdat_sms_get_next_msg_ref

Description

The function is used to get the next message reference for later SMS writing or sending. Usually used for long SMS.

Prototype	int ebdat_sms_get_next_msg_ref()
Parameters	None
Return value	the new message reference value

Example

val = ebdat_sms_get_next_msg_ref();

3.11.16. ebdat_sms_write_pdu_msg

Description

The function is write PDU message. The AT+CMGF=0 must be set.

Prototype	int ebdat_sms_write_pdu_msg(const char* addr, char* msg_ptr, int msg_len, int stat, int* error_p)
Parameters	const char* addr : destination address char* msg_ptr : the message content. Decided by AT+CSMP and AT+CSCS. int len: the length of the actual TP data unit in octets. int stat : the state of message. This is reserved. Currently only 4(UNSENT) is valid. int* error_p: the error number of operating.
Return value	If -1, the error number is contained in error_p. Or else it is the message stored position.

Example

rst = ebdat_sms_write_pdu_msg(dest_ph_no, pdu_data_ptr, msg_len, 4, &error_code);

3.11.17. ebdat_sms_write_txt_msg

Description

The function is write TEXT message. The AT+CMGF=1 must be set.

Prototype	int ebdat_sms_write_txt_msg(const char* addr, char* msg_ptr, int msg_len, int stat, int mr, int msg_seq, int msg_total, int* error_p)
Parameters	const char* addr : destination address char* msg_ptr : the message content. Decided by AT+CSMP and AT+CSCS. int len: the length of the actual TP data unit in octets. int stat : the state of message. int mr : If long SMS, it is the message reference number. Default 0. int msg_seq : If long SMS, it is the message segment number. Default 0. int msg_total : If long SMS , it is the total number of message segments. int* error_p: the error number of operating.
Return value	If -1, the error number is contained in error_p. Or else it is the message stored position.

Example

```
rst = ebdat_sms_write_txt_msg(dest_ph_no, pdu_data_ptr, msg_len, 1, 0, 0, 0, &error_code);
```

3.11.18. ebdat_sms_send_pdu_msg

Description

The function is send PDU message. The AT+CMGF=0 must be set.

Prototype	int ebdat_sms_send_pdu_msg(const char* addr, char* msg_ptr, int msg_len, int stat, int* error_p)
Parameters	const char* addr : destination address char* msg_ptr : the message content. Decided by AT+CSMP and AT+CSCS. int len: the length of the actual TP data unit in octets. int stat : the state of message. This is reserved. Currently only 4(UNSENT) is valid. int* error_p: the error number of operating.
Return value	If -1, the error number is contained in error_p. Or else it is the message reference number.

Example

```
rst = ebdat_sms_send_pdu_msg(dest_ph_no, pdu_data_ptr, msg_len, 4, &error_code);
```

3.11.19. ebdat_sms_send_txt_msg

Description

The function is send TEXT message. The AT+CMGF=1 must be set.

Prototype	int ebdat_sms_send_txt_msg(const char* addr, char* msg_ptr, int msg_len, int stat,

	int mr, int msg_seq, int msg_total, int* error_p)
Parameters	const char* addr : destination address char* msg_ptr : the message content. Decided by AT+CSMP and AT+CSCS. int len: the length of the actual TP data unit in octets. int stat : the state of message. int mr : If long SMS, it is the message reference number. Default 0. int msg_seq : If long SMS, it is the message segment number. Default 0. int msg_total : If long SMS , it is the total number of message segments. int* error_p: the error number of operating.
Return value	If -1, the error number is contained in error_p. Or else it is the message reference number.

Example

```
rst = ebdat_sms_write_txt_msg(dest_ph_no, pdu_data_ptr, msg_len, 1, 0, 0, 0, &error_code);
```

3.11.20. ebdat_sms_decode_pdu_sms

Description

The function is to decode PDU format of SMS data.

Prototype	boolean ebdat_sms_decode_pdu_sms(byte* raw_data, int raw_data_len,

	int fmt, int tpdu_type, uint8 cscs, uint8 dcs_val_ptr, char* oa_ptr, uint8 oa_size, uint8* type_of_addr_ptr, char* timestamp_ptr, uint8 timestamp_size, uint8* fo_ptr, uint8* pid_ptr, uint8* msg_ref_ptr, uint8* seq_num_ptr, uint8* total_sm_ptr, byte* unpacked_sms_data_ptr, uint32 unpacked_sms_buf_size, uint32 unpacked_sms_data_len, char* discharge_time_ptr, uint8 discharge_time_size, uint8 status_ptr, uint32 command_ptr)
Parameters	raw_data: the PDU format SMS data raw_data_len: the length of raw_data fmt: the format of SMS(SMS_FMT_GW_PP) tpdu_type: the type of PDU(SUBMIT/DELIVERY) cscs: the CSCS value dcs_val_ptr: the DCS value pointer

	oa_ptr: the oa pointer oa_size: the size of oa_ptr type_of_addr_ptr: the type_of_addr value pointer timestamp_ptr: the timerstamp pointer timestamp_size: the size of timestamp buffer fo_ptr: fo pointer pid_ptr: the pid pointer msg_ref_ptr: the message reference pointer seq_num_ptr: the sequence number pointer total_sm_ptr: the total_sm pointer unpacked_sms_data_ptr: the unpacked sms data pointer unpacked_sms_buf_size: the size of unpacked_sms_data_ptr unpacked_sms_data_len: the length of unpacked sms data contained in unpacked_sms_data_ptr discharge_time_ptr: the discharge time pointer discharge_time_size: the size of discharge_time_ptr status_ptr: the status pointer command_ptr: the command pointer
Return value	the result of operating TRUE: successful FALSE: failed

Example

```
uint8 dcs_val;
char oa_addr[40];
uint8 type_of_addr_ptr;
char timestamp[25];
uint8 fo;
```

```
uint8 pid;
uint8 msg_ref;
uint8 seq_num;
uint8 total_sm;
byte unpacked_sms_data[200];
uint32 unpacked_sms_data_len;
char discharge_time[25];
uint8 status = 0;
uint32 command = 0;
memset(discharge_time, 0, sizeof(discharge_time));
memset(timestamp, 0, sizeof(timestamp));
memset(unpacked_sms_data, 0, sizeof(unpacked_sms_data));
if (ebdat_sms_decode_pdu_sms((byte*)raw_sms_data,
    sms_data_len*2,
    SMS_FMT_GW_PP,
    tpdu_type,
    EBA_ALPHA_IRA,
    &dcs_val,
    oa_addr,
    sizeof(oa_addr),
    &type_of_addr_ptr,
    timestamp,
    sizeof(timestamp),
    &fo,
    &pid,
    &msg_ref,
    &seq_num,
    &total_sm,
    unpacked_sms_data,
```

```
        sizeof(unpacked_sms_data),
        &unpacked_sms_data_len,
        discharge_time,
        sizeof(discharge_time),
        &status,
        &command))
    {}
```

3.11.21. ebdat_sms_cmss

Description

The function is used to send SMS in the store.

Prototype	boolean ebdat_sms_cmss(uint32 msg_index, boolean long_sms, uint32* send_mr_or_reason_ptr)
Parameters	msg_index: the index of message in the store long_sms: whether it is a long SMS send_mr_or_reason_ptr: the message reference(if successful) or the failure reason code(if failed)
Return value	the result of operating TRUE: successful FALSE: failed

Example

```
rst = ebdat_sms_cmss(1, FALSE, &mr_or_reason);
```

3.12.voice call library

3.12.1. ebdat_voice_call_initiate

Description

The function is used to initiate voice call.

Prototype	int ebdat_voice_call_initiate (char* ph_no, int dial_attri_mask)
Parameters	char* ph_no: the phone number to dial. int dial_attri_mask: default 0. EBDAT_VOICE_CALL_INIT_MASK_NONE=0 EBDAT_VOICE_CALL_INIT_MASK_ENABLE_CCUG = 4 EBDAT_VOICE_CALL_INIT_MASK_DISABLE_CCUG=8 EBDAT_VOICE_CALL_INIT_MASK_ACTIVE_CLIR=16 EBDAT_VOICE_CALL_INIT_MASK_SUPRESS_CLIR=32
Return value	The call id.

Example

```
call_id= ebdat_voice_call_initiate("15021309668", EBDAT_VOICE_CALL_INIT_MASK_NONE);
```

3.12.2. ebdat_voice_call_end

Description

The function is used to end voice call.

Prototype	boolean ebdat_voice_call_end (int call_id)
Parameters	int call_id: the call ID.
Return value	The result of operating:

	TRUE: successful. FALSE: failed.

Example

```
rst = ebdat_voice_call_end(call_id);
```

3.12.3. ebdat_voice_call_answer_call

Description

The function is used to answer voice call.

Prototype	boolean ebdat_voice_call_answer_call (int call_id)
Parameters	int call_id: the call ID.
Return value	The result of operating: TRUE: successful. FALSE: failed.

Example

```
rst = ebdat_voice_call_answer_call(call_id);
```

3.12.4. ebdat_voice_call_answer_all

Description

The function is used to answer all incoming/waiting voice call.

Prototype	int ebdat_voice_call_answer_all (int *answered_call_ids_ptr, int ids_ary_size)
Parameters	int* answered_call_ids_ptr: the answered call ID array. int ids_ary_size; the size of the answered_call_ids_ptr buffer.
Return value	How many calls answered.

Example

```
int answered_call_ids[15];
```

```
memset(answered_call_ids, -1, sizeof(answered_call_ids));

rst = ebdat_voice_call_answer_all(&answered_call_ids, sizeof(answered_call_ids));
```

3.12.5. ebdat_voice_call_list

Description

The function is used to list all voice calls (similar to AT+CLCC).

Prototype	int ebdat_voice_call_list (ebdat_voice_call_list_callback_func_type cb, void* user_data)
Parameters	ebdat_voice_call_list_callback_func_type cb: the call back function. void* user_data: the user data.
Return value	How many calls listed.

Example

```
num_calls = ebdat_voice_call_list(cb_func, NULL);
```

3.12.6. ebdat_voice_call_get_state

Description

The function is used to get voice call state.

Prototype	lvoicelib_external_call_state_e_type ebdat_voice_call_get_state(int call_id)
Parameters	int call_id: the call ID.
Return value	The state of call.

Example

```
state = ebdat_voice_call_get_state(call_id);
```

3.12.7. ebdat_voice_call_send_dtmf

Description

The function is used to send DTMF.

Prototype	boolean ebdat_voice_call_send_dtmf(int call_id, const char* dtmf, uint8 durationx100)
Parameters	int call_id: the call ID. const char* dtmf: the DTMF string to send. uint8 durationx100: reserved value. Currently it is always 1.
Return value	The result of operating: TRUE: successful. FALSE: failed.

Example

rst = ebdat_voice_call_send_dtmf(call_id, "82#988*", 1);

3.12.8. ebdat_voice_call_chld

Description

The function is used to do CHLD operation like AT+CHLD.

Prototype	boolean ebdat_voice_call_chld(int chld_cmd, int call_id)
Parameters	int chld_cmd: 0 to 6. int call_id: the call ID.
Return value	The result of operating: TRUE: successful. FALSE: failed.

Example

rst = ebdat_voice_call_chld(EBDAT_VOICE_CALL_CHLD_CMD_2X, call_id);

3.12.9. ebdat_voice_call_id2seq

Description

The function is used to convert call ID to external sequence number showed in AT+CLCC.

Prototype	int ebdat_voice_call_id2seq(int call_id)
Parameters	int call_id: the call ID.
Return value	int seq_no: the <idX> of AT+CLCC.

Example

```
seq = ebdat_voice_call_id2seq(call_id);
```

3.12.10. ebdat_voice_call_seq2id

Description

The function is used to convert external sequence number showed in AT+CLCC to inner call ID.

Prototype	int ebdat_voice_call_seq2id(int seq)
Parameters	int seq_no: the <idX> of AT+CLCC.
Return value	the converted call ID.

Example

```
call_id = ebdat_voice_call_seq2id(seq);
```

3.13.phonebook library

3.13.1. ebdat_pb_ready

Description

The function is used to check whether the phonebook module is ready.

Prototype	boolean ebdat_pb_ready (void)
Parameters	None
Return value	Whether the PB module is ready: TRUE: successful FALSE: failed.

Example

```
ready= ebdat_pb_ready();
```

3.13.2. ebdat_pb_get_storage_info

Description

The function is used to get the PB storage information.

Prototype	boolean ebdat_pb_get_storage_info (int storage_dev, ebdat_pb_storage_info_s_type *result_ptr)
Parameters	int storage_dev: the PB storage device. ebdat_pb_storage_info_s_type* result_ptr: the result to contain the storage information.
Return value	The result of operation: TRUE: successful FALSE: failed.

Example

```
rst= ebdat_pb_get_storage_info(EBDAT_PB_DEV_SM, &storage_info);
```

3.13.3. ebdat_pb_write

Description

The function is used to write item to PB storage.

Prototype	int ebdat_pb_write (int storage_dev, int rec_index, const char* ph_no, const char* accname)
Parameters	int storage_dev: the PB storage device. int rec_index: the position to save item. const char* ph_no: the phone number to write. const char8 accname: the name to write.
Return value	The position of saved item.

Example

```
index= ebdat_pb_write(EBDAT_PB_DEV_SM, EBDAT_INVALID_PB_INDEX, “15021309668”, “sj”);
```

3.13.4. ebdat_pb_read

Description

The function is used to read item from PB storage.

Prototype	int ebdat_pb_read (int storage_dev, int start_index, int end_index, ebdat_pb_read_find_cb_func_type cb, void* user_data)
Parameters	int storage_dev: the PB storage device. int start_index: the start position to read. int end_index: the end position to read ebdat_pb_read_find_cb_func_type cb: the callback function of reading. void* user_data: the user data.
Return value	How many PB items read.

Example

```
read_count = ebdat_pb_read(EBDAT_PB_DEV_SM, 1, 10, read_func, NULL);
```

3.13.5. ebdat_pb_findname

Description

The function is used to find item by name from PB storage.

Prototype	int ebdat_pb_findname (int storage_dev, const char* accname, int match_type, boolean case_sensitive, ebdat_pb_read_find_cb_func_type cb, void* user_data)
Parameters	int storage_dev: the PB storage device. const char* accname: the name to find. int match_type: the match type. 0 to 2. boolean case_sensitive: whether case sensitive. ebdat_pb_read_find_cb_func_type cb: the callback function of finding.

	void* user_data: the user data.
Return value	How many PB items found.

Example

found_count = ebdat_pb_findname (EBDAT_PB_DEV_SM, “sj”,EBDAT_PB_MATCH_TYPE_EXACT , TRUE, find_func, NULL);

3.13.6. ebdat_pb_findphone

Description

The function is used to find item by phone number from PB storage.

Prototype	int ebdat_pb_findname (int storage_dev, const char* ph_no, int match_type, ebdat_pb_read_find_cb_func_type cb, void* user_data)
Parameters	int storage_dev: the PB storage device. const char* ph_no: the phone number to find. int match_type: the match type. 0 to 3. ebdat_pb_read_find_cb_func_type cb: the callback function of finding. void* user_data: the user data.
Return value	How many PB items found.

Example

found_count = ebdat_pb_findphone (EBDAT_PB_DEV_SM, “15021309668”,EBDAT_PB_MATCH_TYPE_INTELLIGENT, find_func, NULL);

3.14.pin library

3.14.1. ebdat_pin_get_remain_info

Description

The function is used to get the PIN status.

Prototype	boolean ebdat_pin_get_remain_info (ebdat_pin_remain_info_s_type* remain_info_p)
Parameters	ebdat_pin_remain_info_s_type: the PIN information.
Return value	The result of operating: TRUE: successful. FALSE: failed.

Example

```
rst = ebdat_pin_get_remain_info(&info);
```

3.14.2. ebdat_pin_verify

Description

The function is used to verify PIN.

Prototype	boolean ebdat_pin_verify (ebdat_pin_id_e_type pin_id, const char* pin)
Parameters	ebdat_pin_id_e_type pin_id: the pin ID. const char* pin: the pin code.
Return value	The result of operating: TRUE: successful. FALSE: failed.

Example

```
rst = ebdat_pin_verify(EBDAT_PIN_1, “1234”);
```

3.14.3. ebdat_pin_change

Description

The function is used to change PIN.

Prototype	boolean ebdat_pin_change(ebdat_pin_id_e_type pin_id, const char* old_pin, const char* new_pin)

Parameters	ebdat_pin_id_e_type pin_id: the pin ID. const char*old_ pin: the old pin code. const char* new_pin: the new pin code.
Return value	The result of operating: TRUE: successful. FALSE: failed.

Example

rst = ebdat_pin_change(EBDAT_PIN_1, “1234”, “4321”);

3.14.4. ebdat_pin_unblock

Description

The function is used to unblock PIN.

Prototype	boolean ebdat_pin_change(ebdat_pin_id_e_type pin_id, const char* puk, const char* pin)
Parameters	ebdat_pin_id_e_type pin_id: the pin ID. const char*puk: the PUK code. const char* pin: the new pin code.
Return value	The result of operating: TRUE: successful. FALSE: failed.

Example

rst = ebdat_pin_unblock(EBDAT_PIN_1, “12345678”, “1234”);

3.14.5. ebdat_pin_enable

Description

The function is used to enable of disable PIN.

Prototype	boolean ebdat_pin_enable(ebdat_pin_id_e_type pin_id, const char* pin, boolean enable)

Parameters	ebdat_pin_id_e_type pin_id: the pin ID. const char* pin: the pin code. boolean enable: enable or disable the PIN. TRUE: enable. FALSE: disable.
Return value	The result of operating: TRUE: successful. FALSE: failed.

Example

```
rst = ebdat_pin_enable(EBDAT_PIN_1, “1234”, TRUE);
```

3.15.LUA library

3.15.1. ebdat_lua_register_api_handler

Description

The function is used register extended LUA API handler.

Prototype	boolean ebdat_lua_register_api_handler (uint32 api_no, ebdat_lua_api_handler_func_type func)
Parameters	uint32 api_no: the API no. ebdat_lua_api_handler_func_type func: the handler function.
Return value	The result of operationg: TRUE: successful. FALSE: failed.

Example

```
rst = ebdat_lua_register_api_handler(api_no, handler);
```

3.15.2. ebdat_lua_deregister_api_handler

Description

The function is used deregister extended LUA API handler.

Prototype	boolean ebdat_lua_deregister_api_handler (uint32 api_no)
Parameters	uint32 api_no: the API no.
Return value	The result of operationg: TRUE: successful. FALSE: failed.

Example

```
rst = ebdat_lua_deregister_api_handler(api_no);
```

3.15.3. ebdat_lua_set_evt

Description

The function is used to set event for LUA module.

Prototype	boolean ebdat_lua_set_evt (int lua_thread_index, int evt, int evt_p1, int evt_p2, int evt_p3, double evt_clock)
Parameters	int lua_thread_index: the LUA thread index. int evt: the event ID. int evt_p1: the first parameter of event. int evt_p2: the second parameter of event. int evt_p3: the third parameter of event. double evt_clock: the clock.
Return value	The result of operationg: TRUE: successful. FALSE: failed.

Example

rst = ebdat_lua_set_evt(lua_thread_index, 10, 0, 0, 0, 0);

3.15.4. ebdat_lua_signal_notify

Description

The function is used to set signal for LUA module.

Prototype	void ebdat_lua_set_evt (int lua_thread_index, uint8 sig_mask)
Parameters	int lua_thread_index: the LUA thread index. uint8 sig_mask: the mask of the signals.
Return value	None

Example

ebdat_lua_signal_notify (lua_thread_index, 0x10);

3.15.5. ebdat_lua_set_ready

Description

The function is used to tell LUA module that LUA API extension module of EBDAT is ready.

Prototype	void ebdat_lua_set_ready (boolean ready)
Parameters	boolean ready: whether ready or not. TRUE: ready. FALSE: not ready.
Return value	None

Example

rst = ebdat_lua_set_ready (TRUE);

3.16.network library

3.16.1. ebdat_network_get_creg

Description

The function is used to get the CREG value.

Prototype	int ebdat_network_get_creg (void)
Parameters	None
Return value	The value of CREG

Example

```
creg_value= ebdat_network_get_creg();
```

3.16.2. ebdat_network_get_cgreg

Description

The function is used to get the CGREG value.

Prototype	int ebdat_network_get_cgreg (void)
Parameters	None
Return value	The value of CGREG

Example

```
cgreg = ebdat_network_get_cgreg();
```

3.16.3. ebdat_network_get_cnsmod

Description

The function is used to get the CNSMOD value.

Prototype	int ebdat_network_get_cnsmod (void)
Parameters	None
Return value	The value of CNSMOD

Example

```
cnsmod = ebdat_network_get_cnsmod();
```

3.16.4. ebdat_network_get_csq

Description

The function is used to get the CSQ value.

Prototype	int ebdat_network_get_csq(void)
Parameters	None
Return value	The value of CSQ

Example

```
csq = ebdat_network_get_csq();
```

3.17.socket library

This is the non-blocking socket library. For blocking socket library,please refer to bsocket library.

3.17.1. ebdat_ps_open_ps_network

Description

The function is used to open PS network(PDP Activation).

Prototype	ebdat_ps_op_result_e_type ebdat_ps_open_ps_network (uint8 cid, ebdat_ps_op_net_cb_fcn net_cb_func, ebdat_ps_sock_cb_fcn sock_cb_func, void* sock_cb_user_data, sint15* net_handle_ptr)
Parameters	uint8 cid: the profile number. ebdat_ps_op_net_cb_fcn net_cb_func: the PS network event callback function. ebdat_ps_sock_cb_fcn sock_cb_func: the socket event callback function. void* sock_cb_user_data: the user data that will be passed to net_cb_func and sock_cb_func. sint15* net_handle_ptr: the network handle returned.
Return value	The result of operating.

Example

```
rst = ebdat_ps_open_ps_network(cid, net_cb_func, sock_cb_func, user_data, &net_handle);
```

3.17.2. ebdat_ps_close_ps_network

Description

The function is used to close PS network(PDP Deactivation).

Prototype	ebdat_ps_op_result_e_type ebdat_ps_close_ps_network (int16 net_handle)
Parameters	int16 net_handle: the network handle.
Return value	The result of operating.

Example

```
rst = ebdat_ps_close_ps_network(net_handle);
```

3.17.3. ebdat_ps_release_ps_netlib

Description

The function is used to release PS network library. This must be called after PDP deactivation succeeds.

Prototype	ebdat_ps_op_result_e_type ebdat_ps_release_ps_netlib (int16 net_handle)
Parameters	int16 net_handle: the network handle.
Return value	The result of operating.

Example

```
rst = ebdat_ps_release_ps_netlib(net_handle);
```

3.17.4. ebdat_dns_get_host_entry

Description

The function is used to resolve IP address for domain name.

Prototype	ebdat_ps_op_result_e_type ebdat_dns_get_host_entry(const char* host, struct in_addr* const ip_address, ebdat_dns_event_callback_fcn dns_cb, void* user_data)

Parameters	const char* host: the domain name to resolve. Struct in_addr* const ip_address: the resolved IP address. ebdat_dns_event_callback_fcn: the DNS resolve event callback function. void* user_data: the user data.
Return value	The result of operating.

Example

rst = ebdat_dns_get_host_entry("www.myaddress.com", &ip_address, my_dns_cb, NULL);

3.17.5. ebdat_sock_create

Description

The function is used to create socket.

Prototype	int16 ebdat_sock_create (int net_handle, byte family, byte type, byte protocol)
Parameters	int16 net_handle: the handle of the PS network. byte family: the family of the stack. byte type: the type of the socket byte protocol: the protocol of the socket.
Return value	The result of operating.

Example

int sock_fd = ebdat_sock_create(net_handle, family, type, protocol);

3.17.6. ebdat_sock_connect

Description

The function is used to connect to TCP server.

Prototype	ebdat_ps_op_result_e_type ebdat_sock_connect (int16 sfd, struct sockaddr_in* addr, const size_t saddrsiz)

Parameters	int16 sock_fd: the handle of the socket. struct sockaddr_in* addr: the address. const size_t saddrsiz: the size of the address.
Return value	The result of operating.

Example

```
rst = ebdat_sock_connect(sock_fd, addr, sizeof(sockaddr_in));
```

3.17.7. ebdat_sock_close

Description

The function is used to close TCP connection.

Prototype	ebdat_ps_op_result_e_type ebdat_sock_close (const int16 sock_fd)
Parameters	int16 sock_fd: the handle of the socket.
Return value	The result of operating.

Example

```
rst = ebdat_sock_close(sock_fd);
```

3.17.8. ebdat_sock_send

Description

The function is used to send data on TCP connection.

Prototype	ebdat_ps_op_result_e_type ebdat_sock_send (const int16 sock_fd, byte* buf, uint32 len, int* bytes_sent_p)
Parameters	int16 sock_fd: the handle of the socket. byte* buf: the buffer of data to send. uint32 len: the length of data to send. int* bytes_sent_p: the length of data sent.
Return value	The result of operating.

Example

rst = ebdat_sock_send(sock_fd, buf, len, &bytes_sent);

3.17.9. ebdat_sock_recv

Description

The function is used to receive data on TCP connection.

Prototype	ebdat_ps_op_result_e_type ebdat_sock_recv (const int16 sock_fd, byte* buf, uint32 len, int* bytes_recv_p)
Parameters	int16 sock_fd: the handle of the socket. byte* buf: the buffer to contain received data. uint32 len: the size of buffer. int* bytes_recv_p: the length of data received.
Return value	The result of operating.

Example

rst = ebdat_sock_recv(sock_fd, buf, len, &bytes_rcvd);

3.17.10. ebdat_sock_sendto

Description

The function is used to send data on UDP connection.

Prototype	ebdat_ps_op_result_e_type ebdat_sock_sendto (const int16 sock_fd, byte* buf, uint32 len, int* bytes_sent_p, struct sockaddr_in* destaddr_p)
Parameters	int16 sock_fd: the handle of the socket. byte* buf: the buffer of data to send. uint32 len: the length of data to send. int* bytes_sent_p: the length of data sent. struct sockaddr_in* destaddr_p: the destination address.

Return value	The result of operating.

Example

rst = ebdat_sock_sendto(sock_fd, buf, len, &bytes_sent, &dest);

3.17.11. ebdat_sock_recvfrom

Description

The function is used to receive data on TCP connection.

Prototype	ebdat_ps_op_result_e_type ebdat_sock_recvfrom (const int16 sock_fd, byte* buf, uint32 len, int* bytes_recv_p, struct sockaddr_in* fromaddr_p)
Parameters	int16 sock_fd: the handle of the socket. byte* buf: the buffer to contain received data. uint32 len: the size of buffer. int* bytes_recv_p: the length of data received. struct sockaddr_in* fromaddr_p: the from address.
Return value	The result of operating.

Example

rst = ebdat_sock_recvfrom(sock_fd, buf, len, &bytes_rcvd, &fromaddr);

3.17.12. ebdat_sock_bind

Description

The function is used to bind address to socket.

Prototype	ebdat_ps_op_result_e_type ebdat_sock_recvfrom (const int16 sock_fd, struct sockaddr_in* addr, const size_t saddrsiz)
Parameters	int16 sock_fd: the handle of the socket. struct sockaddr_in* addr: the address. Const size_t: the size of address

Return value	The result of operating.

Example

rst = ebdat_sock_bind(sock_fd,&address, sizeof(struct sockaddr_in));

3.17.13. ebdat_sock_listen

Description

The function is used to listen using TCP socket.

Prototype	ebdat_ps_op_result_e_type ebdat_sock_listen (const int16 sock_fd)
Parameters	int16 sock_fd: the handle of the socket.
Return value	The result of operating.

Example

rst = ebdat_sock_listen(sock_fd);

3.17.14. ebdat_sock_accept

Description

The function is used to accept incoming TCP client.

Prototype	int16 ebdat_sock_accept (const int16 sock_fd, struct sockaddr_in* peer_addr_p)
Parameters	int16 sock_fd: the handle of the socket. struct sockaddr_in* peer_addr_p: the accepted client address.
Return value	The accepted TCP client handle.

Example

client_sock_fd = ebdat_sock_accept(sock_fd, &client_addr);

3.17.15. ebdat_sock_setsockopt

Description

The function is used to set option for socket.

Prototype	boolean ebdat_sock_setsockopt (const int16 sock_fd, int level, int optname, void* optval, uint32 optlen)
Parameters	int16 sock_fd: the handle of the socket. int level: the level. int optname; the operation name void* optval: the operation parameter. uint32 optlen: the size of optval.
Return value	the result of operating. TRUE: successful FALSE: failed.

Example

```
rst = ebdat_sock_setsockopt(sock_fd, level, optname, optval, sizeof(optval));
```

3.17.16. ebdat_sock_getsockopt

Description

The function is used to get option for socket.

Prototype	boolean ebdat_sock_getsockopt (const int16 sock_fd, int level, int optname, void* optval, uint32 optlen)
Parameters	int16 sock_fd: the handle of the socket. int level: the level. int optname; the operation name void* optval: the operation parameter. uint32 optlen: the size of optval.
Return value	the result of operating. TRUE: successful FALSE: failed.

Example

rst = ebdat_sock_getsockopt(sock_fd, level, optname, optval, sizeof(optval));

3.17.17. ebdat_sock_get_sock_name

Description

The function is used to get option for socket.

Prototype	boolean ebdat_sock_get_sock_name (const int16 sock_fd, struct sockaddr* addr_p, uint16* addrlen_p)
Parameters	int16 sock_fd: the handle of the socket. struct sockaddr* addr_p: the address uint16* addrlen_p: the size of address
Return value	the result of operating. TRUE: successful FALSE: failed.

Example

rst = ebdat_sock_get_sock_name(sock_fd, &addr, sizeof(struct sockaddr));

3.17.18. ebdat_sock_async_select

Description

The function is used to select the cared socket event.

Prototype	ebdat_ps_op_result_e_type ebdat_sock_async_select (int16 sock_fd, int32 event_mask)
Parameters	int16 sock_fd: the socket handle. int32 event_mask: the mask of the events.
Return value	The result of operating.

Example

rst = ebdat_sock_async_select(sock_fd,

EBDAT_SOCK_READ_EVENT|EBDAT_SOCK_CLOSE_EVENT);

3.17.19. ebdat_sock_async_deselect

Description

The function is used to deselect the cared socket event.

Prototype	ebdat_ps_op_result_e_type ebdat_sock_async_deselect (int16 sock_fd, int32 event_mask)
Parameters	int16 sock_fd: the socket handle. int32 event_mask: the mask of the events.
Return value	The result of operating.

Example

```
rst = ebdat_sock_async_select(sock_fd, EBDAT_SOCK_READ_EVENT|EBDAT_SOCK_CLOSE_EVENT);
```

3.18.bsocket library

This is the blocking socket library. For non-blocking socket library, please refer to socket library.

3.18.1. ebdat_bpsnetwork_open

Description

The function is used to open PS network(PDP Activation).

Prototype	int ebdat_bpsnetwork_open (uint8 cid, int timeout)
Parameters	uint8 cid: the profile number. int timeout: the maximum milliseconds to wait for PDP activation.
Return value	The network handle. -1 means failure.

Example

```
net_handle = ebdat_bpsnetwork_open(cid, 30000);
```

3.18.2. ebdat_bpsnetwork_close

Description

The function is used to close PS network(PDP Deactivation).

Prototype	boolean ebdat_bpsnetwork_close (int net_handle)
Parameters	int net_handle: the network handle.
Return value	the result of operating: TRUE: successful FALSE: failed

Example

```
rst = ebdat_bpsnetwork_close(net_handle);
```

3.18.3. ebdat_bpsnetwork_status

Description

The function is used to get the status of PS network.

Prototype	ebdat_bpsnetwork_status_type ebdat_bpsnetwork_status (int net_handle)
Parameters	int net_handle: the network handle.
Return value	the status of network.

Example

```
status = ebdat_bpsnetwork_status(net_handle);
```

3.18.4. ebdat_bpsnetwork_dorm

Description

The function is used to let the PS network enter or leave dormancy state.

Prototype	boolean ebdat_bpsnetwork_dorm (int net_handle, boolean dorm_or_resume)
Parameters	int net_handle: the network handle.

	boolean dorm_or_resume: enter or leave dormancy state. TRUE: enter dormancy state FALSE: leave dormancy state
Return value	The result of operating(reserved).

Example

```
ebdat_bpsnetwork_dorm(net_handle, TRUE);
```

3.18.5. ebdat_bpsnetwork_resolve_dns

Description

The function is used to resolve IP address for domain name using DNS.

Prototype	boolean ebdat_bpsnetwork_resolve_dns (const char* domain_name, int cid, char* out_ip_address, uint32 out_ip_addr_size)
Parameters	char* domain_name: the domain name to resolve int cid: the cid of AT+CGSOCKCONT to use to activate PDP context. char* out_ip_address: the buffer to contain IP address string. uint32 out_ip_addr_size: the size of buffer out_ip_address.
Return value	The result of operating: TRUE: successful FALSE: failed

Example

```
ebdat_bpsnetwork_resolve_dns("www.baidu.com", 1, ip_str_buf, sizeof(ip_str_buf));
```

3.18.6. ebdat_bpsnetwork_local_ip

Description

The function is used to get the IPv4 address of PS network.

Prototype	boolean ebdat_bpsnetwork_local_ip (int net_handle, char* out_ip_address, uint32 out_ip_addr_size)
Parameters	int net_handle: the network handle. char* out_ip_address: the buffer to contain IP address string. uint32 out_ip_addr_size: the size of buffer out_ip_address.
Return value	The result of operating: TRUE: successful FALSE: failed

Example

```
ebdat_bpsnetwork_local_ip(net_handle, 1, ip_str_buf, sizeof(ip_str_buf));
```

3.18.7. ebdat_bpsnetwork_primary_dns

Description

The function is used to get the primary DNS address of PS network.

Prototype	boolean ebdat_bpsnetwork_primary_dns (int net_handle, char* out_ip_address, uint32 out_ip_addr_size)
Parameters	int net_handle: the network handle. char* out_ip_address: the buffer to contain IP address string. uint32 out_ip_addr_size: the size of buffer out_ip_address.
Return value	The result of operating: TRUE: successful FALSE: failed

Example

```
ebdat_bpsnetwork_primary_dns(net_handle, 1, ip_str_buf, sizeof(ip_str_buf));
```

3.18.8. ebdat_bpsnetwork_change_primary_dns

Description

The function is used to get the secondary DNS address of PS network.

Prototype	boolean ebdat_bpsnetwork_change_primary_dns (int net_handle, char* ip_address)
Parameters	int net_handle: the network handle. char* ip_address: the buffer to contain IP address string.
Return value	The result of operating: TRUE: successful FALSE: failed

Example

```
ebdat_bpsnetwork_change_primary_dns(net_handle, 1, ip_str_buf);
```

3.18.9. ebdat_bpsnetwork_secondary_dns

Description

The function is used to get the secondary DNS address of PS network.

Prototype	boolean ebdat_bpsnetwork_secondary_dns (int net_handle, char* out_ip_address, uint32 out_ip_addr_size)
Parameters	int net_handle: the network handle. char* out_ip_address: the buffer to contain IP address string. uint32 out_ip_addr_size: the size of buffer out_ip_address.
Return value	The result of operating: TRUE: successful FALSE: failed

Example

ebdat_bpsnetwork_secondary_dns(net_handle, 1, ip_str_buf, sizeof(ip_str_buf));

3.18.10. ebdat_bpsnetwork_change_secondary_dns

Description

The function is used to get the secondary DNS address of PS network.

Prototype	boolean ebdat_bpsnetwork_change_secondary_dns (int net_handle, char* ip_address)
Parameters	int net_handle: the network handle. char* ip_address: the buffer to contain IP address string.
Return value	The result of operating: TRUE: successful FALSE: failed

Example

ebdat_bpsnetwork_change_secondary_dns(net_handle, 1, ip_str_buf);

3.18.11. ebdat_bpsnetwork_get_mtu

Description

The function is used to get the MTU value of PS network.

Prototype	int ebdat_bpsnetwork_get_mtu (int net_handle)
Parameters	int net_handle: the network handle.
Return value	The result of operating: >0: successful <=0: failed

Example

mtu = ebdat_bpsnetwork_get_mtu(net_handle);

3.18.12. ebdat_bpsnetwork_config_keepalive_parameter

Description

The function is used to configure TCP keep alive parameters.

Prototype	boolean ebdat_bpsnetwork_config_keepalive_parameter (int max_ka_check_times, int ka_check_interval_minutes)
Parameters	int max_ka_check_times: the times to check for keep alive. int ka_check_interval_minutes: the interval in minute to check.
Return value	The result of operating: TRUE: successful FALSE: failed

Example

```
ebdat_bpsnetwork_config_keepalive_parameter(max_ka_check_times, ka_check_interval_minutes);
```

3.18.13. ebdat_bpsnetwork_config_tcp_retran_parameter

Description

The function is used to configure TCP retransmit parameters.

Prototype	boolean ebdat_bpsnetwork_config_tcp_retran_parameter (int max_backoff, int tcp_min_rto)
Parameters	int max_backoff: The maximum backoff value. int tcp_min_rto: the minimum milliseconds to resend the TCP packet.
Return value	The result of operating: TRUE: successful FALSE: failed

Example

ebdat_bpsnetwork_config_tcp_retran_parameter(10, 10000);

3.18.14. ebdat_bpsnetwork_config_dns_timeout_parameter

Description

The function is used to configure DNS query timeout parameters.

Prototype	boolean ebdat_bpsnetwork_config_dns_timeout_parameter (int dns_max_net_retries, int dns_net_timeout, int dns_max_query_retires)
Parameters	int dns_max_net_retries: the maximum times to retry opening the PS network. int dns_net_timeout: the timeout in millisecond for opening PS network. int dns_max_query_retries: the maximum query times for DNS UDP query.
Return value	The result of operating: TRUE: successful FALSE: failed

Example

ebdat_bpsnetwork_config_dns_timeout_parameter(0, 30000, 5);

3.18.15. ebdat_bsock_create

Description

The function is used to create socket.

Prototype	int ebdat_bsock_create (int net_handle, byte family, byte type, byte protocol)
Parameters	int16 net_handle: the handle of the PS network. byte family: the family of the stack. byte type: the type of the socket

	byte protocol: the protocol of the socket.
Return value	The socket handle created.

Example

```
int sock_fd = ebdat_bsock_create(net_handle, family, type, protocol);
```

3.18.16. ebdat_bsock_connect

Description

The function is used to connect to TCP server.

Prototype	boolean ebdat_bsock_connect (int sfd, struct sockaddr_in* addr, const size_t saddrsiz, int timeout, boolean* sock_released_p)
Parameters	int sock_fd: the handle of the socket. struct sockaddr_in* addr: the address. const size_t saddrsiz: the size of the address. int timeout: the timeout value in millisecond. boolean* sock_released_p: When returns FALSE, this fileld indicates whether
Return value	The result of operating. TRUE: successful FALSE: failed

Example

```
rst = ebdat_bsock_connect(sock_fd, addr, sizeof(sockaddr_in), 30000, &released);
```

3.18.17. ebdat_bsock_close

Description

The function is used to close TCP connection.

Prototype	boolean ebdat_bsock_close (const int sock_fd)
Parameters	int sock_fd: the handle of the socket.

Return value	The result of operating. TRUE: successful FALSE: failed

Example

```
rst = ebdat_bsock_close(sock_fd);
```

3.18.18. ebdat_bsock_send

Description

The function is used to send data on TCP connection.

Prototype	int ebdat_bsock_send (const int sock_fd, byte* buf, uint32 len, int timeout, int* error_p)
Parameters	int sock_fd: the handle of the socket. byte* buf: the buffer of data to send. uint32 len: the length of data to send. int timeout: the timeout value for sending in millisecond. int * error_p: If returns <=0, this field indicates the error cause.
Return value	the bytes of data sent.

Example

```
rst = ebdat_bsock_send(sock_fd, buf, len, &bytes_sent, 120000, &err_code);
```

3.18.19. ebdat_bsock_recv

Description

The function is used to receive data on TCP connection.

Prototype	int ebdat_sock_recv (const int sock_fd, byte* buf, uint32 len, int timeout, int* error_p)
Parameters	int sock_fd: the handle of the socket.

	byte* buf: the buffer to contain received data. uint32 len: the size of buffer. int timeout: the timeout value for receiving in millisecond. int * error_p: If returns <=0, this field indicates the error cause.
Return value	the length of data received.

Example

```
rst = ebdat_bsock_recv(sock_fd, buf, len, &bytes_rcvd, 120000, &err_code);
```

3.18.20. ebdat_bsock_sendto

Description

The function is used to send data on UDP connection.

Prototype	int ebdat_bsock_sendto (const int sock_fd, byte* buf, uint32 len, struct sockaddr_in* destaddr_p, int timeout, int* error_p)
Parameters	int sock_fd: the handle of the socket. byte* buf: the buffer of data to send. uint32 len: the length of data to send. struct sockaddr_in* destaddr_p: the destination address. int timeout: the timeout value for sending in millisecond. int * error_p: If returns <=0, this field indicates the error cause.
Return value	the length of data sent.

Example

```
rst = ebdat_bsock_sendto(sock_fd, buf, len, &dest, 30000, &error_code);
```

3.18.21. ebdat_bsock_recvfrom

Description

The function is used to receive data on TCP connection.

Prototype	int ebdat_sock_recvfrom (const int sock_fd, byte* buf, uint32 len, int* bytes_recv_p, struct sockaddr_in* fromaddr_p)
Parameters	int sock_fd: the handle of the socket. byte* buf: the buffer to contain received data. uint32 len: the size of buffer. struct sockaddr_in* fromaddr_p: the from address. int timeout: the timeout value for receiving in millisecond. int * error_p: If returns <=0, this field indicates the error cause.
Return value	the length of data received.

Example

```
rst = ebdat_bsock_recvfrom(sock_fd, buf, len, &bytes_rcvd, &fromaddr,30000,&err_code);
```

3.18.22. ebdat_bsock_bind

Description

The function is used to bind address to socket.

Prototype	boolean ebdat_sock_recvfrom (const int sock_fd, struct sockaddr_in* addr, const size_t saddrsiz)
Parameters	int sock_fd: the handle of the socket. struct sockaddr_in* addr: the address. const size_t: the size of address
Return value	the result of operating. TRUE: successful

	FALSE: failed

Example

```
rst = ebdat_bsock_bind(sock_fd,&address, sizeof(struct sockaddr_in));
```

3.18.23. ebdat_bsock_listen

Description

The function is used to listen using TCP socket.

Prototype	ebdat_ps_op_result_e_type ebdat_sock_listen (const int sock_fd, int backlog)
Parameters	int sock_fd: the handle of the socket. int backlog: how many connections can be queued for a listening socket. It's range is from 1 to 3.
Return value	The result of operating. TRUE: successful FALSE: failed

Example

```
rst = ebdat_bsock_listen(sock_fd, 3);
```

3.18.24. ebdat_bsock_accept

Description

The function is used to accept incoming TCP client.

Prototype	int ebdat_bsock_accept (const int sock_fd, struct sockaddr_in* peer_addr_p, int timeout, int* error_p)
Parameters	int sock_fd: the handle of the socket. struct sockaddr_in* peer_addr_p: the accepted client address. int timeout: the timeout value in millisecond. int error_p: if failed, this field indicates the error cause.

Return value	The accepted TCP client handle.

Example

client_sock_fd = ebdat_bsock_accept(sock_fd, &client_addr, 120000, &error_code);

3.18.25. ebdat_bsock_select

Description

The function is used to set careful event masks for a socket.

Prototype	boolean ebdat_bsock_select (int16 sock_fd, int32 event_mask)
Parameters	int16 sock_fd: the handle of the socket. int event_mask: 1: write_event 2: read_event 4: close_event 8: accept_event
Return value	the result of operating. TRUE: successful FALSE: failed.

Example

rst = ebdat_bsock_select(sock_fd, EBDAT_SOCK_READ_EVENT | EBDAT_SOCK_CLOSE_EVENT);

3.18.26. ebdat_bsock_deselect

Description

The function is used to deselect the event that the socket doesn't care for.

Prototype	boolean ebdat_bsock_select (int16 sock_fd, int32 event_mask)
Parameters	int16 sock_fd: the handle of the socket.

	int event_mask: 1: write_event 2: read_event 4: close_event 8: accept_event
Return value	the result of operating. TRUE: successful FALSE: failed.

Example

```
rst = ebdat_bsock_deselect(sock_fd, EBDAT_SOCK_READ_EVENT);
```

3.18.27. ebdat_bsock_keepalive_socket

Description

The function is used to set KEEP ALIVE parameter for a socket.

Prototype	boolean ebdat_bsock_keepalive_socket (int16 sock_fd, boolean keepalive, int idle_time)
Parameters	int16 sock_fd: the handle of the socket. boolean keepalive: whether KEEP ALIVE takes effect. int idle_time: reserved value, current it should be 1.
Return value	the result of operating. TRUE: successful FALSE: failed.

Example

```
rst = ebdat_bsock_keepalive_socket(sock_fd, TRUE, 1);
```

3.18.28. ebdat_bsock_get_sock_name

Description

The function is used to get option for socket.

Prototype	boolean ebdat_bsock_get_sock_name (const int16 sock_fd, struct sockaddr* addr_p, uint16* addrlen_p)
Parameters	int16 sock_fd: the handle of the socket. struct sockaddr* addr_p: the address uint16* addrlen_p: the size of address
Return value	the result of operating. TRUE: successful FALSE: failed.

Example

```
rst = ebdat_bsock_get_sock_name(sock_fd, &addr, sizeof(struct sockaddr));
```

3.19.gps library

3.19.1. ebdat_gps_start

Description

The function is used to start GPS device.

Prototype	boolean ebdat_gps_start (ebdat_gps_start_mode_type_enum mode)
Parameters	mode: the starting mode
Return value	the result of operating TRUE: successful FALSE: failed

Example

```
result = ebdat_gps_start(EBDAT_GPS_START_MODE_COLD);
```

3.19.2. ebdat_gps_stop

Description

The function is used to stop GPS device.

Prototype	boolean ebdat_gps_stop (void)
Parameters	None
Return value	the result of operating TRUE: successful FALSE: failed

Example

```
result = ebdat_gps_stop();
```

3.19.3. ebdat_gps_get_fix_info

Description

The function is used to get GPS fix information.

Prototype	void ebdat_gps_get_fix_info (char* gps_info)
Parameters	gps_info: the output parameter that contain the GPS information
Return value	None

Example

```
char fix_info[512];
memset(fix_info, 0, sizeof(fix_info));
ebdat_gps_get_fix_info(fix_info);
```

3.19.4. ebdat_gps_get_nema_info

Description

The function is used to get GPS fix information by NMEA data type.

Prototype	int ebdat_gps_get_nema_info (ebdat_nmea_e_type nmea_type, char* nmea)
Parameters	nmea_type: the NMEA type nmea: the output parameter that contains the GPS information
Return value	The length of data contained in nmea variable.

Example

```
char nmea_info[512];
memset(nmea_info, 0, sizeof(nmea_info));
ebdat_gps_get_nmea_info(EBDAT_NMEA_GGA, nmea_info);
```

3.19.5. ebdat_gps_set_mode

Description

The function is used to set GPS mode.

Prototype	boolean ebdat_gps_set_mode (ebdat_session_operation_e_type op)
Parameters	op: the GPS session operation mode.
Return value	the result of operating TRUE: successful FALSE: failed

Example

```
ebdat_gps_set_mode(EBDAT_SESSION_OPERATION_STANDALONE_ONLY);
```

3.19.6. ebdat_gps_get_mode

Description

The function is used to get GPS mode.

Prototype	ebdat_session_operation_e_type ebdat_gps_get_mode (void)

Parameters	None
Return value	The GPS session operation mode

Example

```
mode = ebdat_gps_get_mode();
```

3.19.7. ebdat_gps_delete_info

Description

The function is used to delete GPS information. This function must be called after stopping GPS device.

Prototype	boolean ebdat_gps_delete_info(void)
Parameters	None
Return value	the result of operating TRUE: successful FALSE: failed

Example

```
rst = ebdat_gps_delete_info();
```

3.20.gpio library

3.20.1. ebdat_gpio_config

Description

The function is used to configure GPIO direction.

Prototype	boolean ebdat_gpio_config (uint8 gpio, ebdat_gpio_direction_t gpio_io)
Parameters	gpio: the number of GPIO gpio_io: the direction of GPIO
Return value	the result of operating TRUE: successful

	FALSE: failed

Example

```
result = ebdat_gpio_config(1, EBDAT_GPIO_OUTPUT);
```

3.20.2. ebdat_gpio_out

Description

The function is used to set GPIO value.

Prototype	boolean ebdat_gpio_out (uint8 gpio, ebdat_gpio_value_t gpio_val)
Parameters	gpio: the number of GPIO gpio_val: the value of GPIO
Return value	the result of operating TRUE: successful FALSE: failed

Example

```
result = ebdat_gpio_out(1, EBDAT_GPIO_LOW_VALUE);
```

3.20.3. ebdat_gpio_get

Description

The function is used to get GPIO value.

Prototype	ebdat_gpio_value_t ebdat_gpio_out (uint8 gpio)
Parameters	gpio: the number of GPIO
Return value	The value of GPIO

Example

```
gpio_val = ebdat_gpio_get(1);
```

3.20.4. ebdat_gpio_func_set

Description

The function is used to work instead of using AT+CGFUNC.

Prototype	boolean ebdat_gpio_out (uint8 gpio_func, boolean bEnabled)
Parameters	gpio_func: the number of GPIO function bEnabled: whether enable the GPIO for this pin.
Return value	the result of operating TRUE: successful FALSE: failed

Example

```
rst = ebdat_gpio_func_set(1, TRUE);
```

3.20.5. ebdat_gpio_isr_set

Description

The function is used to set a GPIO as ISR.

Prototype	boolean ebdat_gpio_isr_set (uint8 gpio_no, ebdat_gpio_int_detect_t detect, ebdat_gpio_int_pol_t polarity)
Parameters	gpio_no: the number of GPIO detect: detect type. polarity: the polarity type for triggering
Return value	the result of operating TRUE: successful FALSE: failed

Example

```
rst = ebdat_gpio_isr_set(1, EBDAT_DETECT_LEVEL, EBDAT_ACTIVE_LOW);
```

3.20.6. ebdat_gpio_debounce_set

Description

The function is used to set the debounce value when a GPIO is configured as ISR.

Prototype	boolean ebdat_gpio_debounce_set (uint8 gpio_no, uint32 debounce_ms)
Parameters	gpio_no: the number of GPIO debounce_ms: the duration of debounce in milliseconds
Return value	the result of operating TRUE: successful FALSE: failed

Example

```
rst = ebdat_gpio_debounce_set(1, 10);
```

3.21.spi library

3.21.1. ebdat_spi_init

Description

The function is used to initialize the SPI interface. This function should only be called once after power up.

Prototype	void ebdat_spi_init (void)
Parameters	None
Return value	None

Example

```
ebdat_spi_init();
```

3.21.2. ebdat_spi_set_clk_info

Description

The function is used to set SPI interface clock information.

Prototype	void ebdat_spi_set_clk_info (ebdat_spi_clk_pol_t clk_pol, ebdat_spi_clk_mode_t clk_md, ebdat_spi_trans_mode_t trans_md)
Parameters	clk_pol: the clock polarity clk_md: the clock mode trans_md: the clock phase parameter
Return value	None

Example

```
ebdat_spi_set_clk_info(SPI_CLK_IDLE_LOW, SPI_CLK_NORMAL, SPI_OUTPUT_FIRST_MODE);
```

3.21.3. ebdat_spi_set_cs

Description

The function is used to set SPI interface chip selector information.

Prototype	void ebdat_spi_set_cs (ebdat_spi_cs_mode_t cs_md, ebdat_spi_cs_pol_t cs_pol)
Parameters	cs_md: CS mode clk_pol: the clock polarity
Return value	None

Example

```
ebdat_spi_set_cs(SPI_CS_DEASSERTE, SPI_CS_ACTIVE_LOW);
```

3.21.4. ebdat_spi_set_frequency

Description

The function is used to set SPI interface frequency information.

Prototype	void ebdat_spi_set_frequency (uint32 nMinFreq, uint32 nMaxFreq, uint32 nDeassertionTime)
Parameters	nMinFreq: minimum frequency

	nMaxFreq: maximum frequency nDeassertionTime: in master mode, configures the minimum time to wait between transfer units in nanoseconds
Return value	None

Example

ebdat_spi_set_frequency(10000, 500000, 100);

3.21.5. ebdat_spi_set_param

Description

The function is used to set SPI interface parameters.

Prototype	void ebdat_spi_set_param (uint8 nNumBits, ebdat_spi_input_packing_t input_packing, ebdat_spi_output_unpacking_t output_packing)
Parameters	nNumBits: configures the number of bits to use per transfer unit input_packing: specifies whether data should be packed into the user input buffer output_packing: specifies whether data should be unpacked from the user output buffer
Return value	None

Example

ebdat_spi_set_param(8, SPI_INPUT_PACKING_ENABLED, SPI_OUTOUT_PACKING_ENABLED);

3.21.6. ebdat_spi_write_data

Description

The function is used to write data to SPI slave device.

Prototype	void ebdat_spi_write_data (uint8* data, uint32 len)
Parameters	data: the data to write

	len: the length of data to write
Return value	None

Example

ebdat_spi_write_data(data, len);

3.21.7. ebdat_spi_write_uint32_data

Description

The function is used to write data to SPI slave device.

Prototype	void ebdat_spi_write_uint32_data (uint32* data, uint32 len)
Parameters	data: the data to write len: the length of data to write
Return value	None

Example

ebdat_spi_write_uint32_data(data, len);

3.21.8. ebdat_spi_write_reg

Description

The function is used to write data to SPI register.

Prototype	void ebdat_spi_write_reg (uint32 reg, uint32 reg_val, uint32 len)
Parameters	reg: teh register to write reg_val: the data to write len: the length of data to write
Return value	None

Example

ebdat_spi_write_reg(reg, reg_val, len);

3.21.9. ebdat_spi_read_reg

Description

The function is used to read data from SPI register.

Prototype	void ebdat_spi_read_reg (uint32 reg, uint8* read_buf, uint32 len)
Parameters	reg: teh register to read Read_buf: the buffer to contain read data len: the length of data to read
Return value	None

Example

```
ebdat_spi_read_reg(reg, read_buf, len);
```

3.22.pm library

3.22.1. ebdat_pm_auto_pwr_off_set

Description

The function is used to set the voltage for automatical power off.

Prototype	boolean ebdat_pm_auto_pwr_off_set (uint32 low_voltage)
Parameters	low_voltage: the low voltage
Return value	the result of operating TRUE: successful FALSE: failed

Example

```
rst = ebdat_pm_auto_pwr_off_set(low_voltage);
```

3.22.2. ebdat_pm_auto_pwr_off_get

Description

The function is used to get the voltage for automatical power off.

Prototype	uint32 ebdat_pm_auto_pwr_off_get (void)
Parameters	None
Return value	the low voltage

Example

```
low_voltage = ebdat_pm_auto_pwr_off_get();
```

3.22.3. ebdat_pm_enable_low_voltage_alarm

Description

The function is used to set the alarm voltage.

Prototype	void ebdat_pm_enable_low_voltage_alarm(uint32 low_voltage)
Parameters	low_voltage: the low voltage to alarm
Return value	None

Example

```
ebdat_pm_enable_low_voltage_alarm(low_voltage);
```

3.22.4. ebdat_pm_disable_low_voltage_alarm

Description

The function is used to disable low voltage alarm function.

Prototype	void ebdat_pm_disable_low_voltage_alarm(void)
Parameters	None
Return value	None

Example

ebdat_pm_disable_low_voltage_alarm();

3.22.5. ebdat_pm_enable_auto_pwr_on

Description

The function is used to enable automatical power on function.

Prototype	void ebdat_pm_enable_auto_pwr_on(void)
Parameters	None
Return value	None

Example

ebdat_pm_enable_auto_pwr_on();

3.22.6. ebdat_pm_disable_auto_pwr_on

Description

The function is used to disable automatical power on function.

Prototype	void ebdat_pm_disable_auto_pwr_on(void)
Parameters	None
Return value	None

Example

ebdat_pm_disable_auto_pwr_on();

3.22.7. ebdat_pm_enable_auto_pwr_off

Description

The function is used to enable automatical power off function.

Prototype	boolean ebdat_pm_enable_auto_pwr_off(uint8 hour, uint8 minute, boolean bRepeated

	)
Parameters	hour: the hour value for automatical power off Minute: the minute value for automatical power off bRepeated: does it take effect permanently
Return value	the result of operating TRUE: successful FALSE: failed

Example

```
rst = ebdat_pm_enable_auto_pwr_off(hour, minute, repeatable);
```

3.22.8. ebdat_pm_disable_auto_pwr_off

Description

The function is used to disable automatical power off function.

Prototype	void ebdat_pm_disable_auto_pwr_off(void)
Parameters	None
Return value	None

Example

```
ebdat_pm_disable_auto_pwr_off();
```

3.23.misc hardware library

3.24.1. ebdat_adc_read

Description

The function is used to read ADC value.

Prototype	int ebdat_adc_read (ebdat_adc_t adc_type)
Parameters	adc_type: the type of ADC
Return value	The ADC value read

Example

```
val = ebdat_adc_read(EBDAT_ADC_TEMP_TYPE);
```

3.24.2. ebdat_vaux_set

Description

The function is used to set voltage value for the LDO.

Prototype	boolean ebdat_vaux_set (uint16 level)
Parameters	level: the level of voltage
Return value	the result of operating TRUE: successful FALSE: failed

Example

```
rst = ebdat_vaux_set(level);
```

3.24.3. ebdat_vaux_get

Description

The function is used to get voltage value for the LDO.

Prototype	uint16 ebdat_vaux_get (void)
Parameters	None
Return value	The voltage value for the LDO

Example

```
level = ebdat_vaux_get();
```

3.24.4. ebdat_vaux_switch

Description

The function is used to enable or disable the LDO.

Prototype	boolean ebdat_vaux_switch (boolean bEnabled)

Parameters	bEnabled: whether enable or disable the LDO
Return value	the result of operating TRUE: successful FALSE: failed

Example

```
rst = ebdat_vaux_switch(TRUE);
```

3.24.5. ebdat_vaux_get_state

Description

The function is used check the state of LDO.

Prototype	boolean ebdat_vaux_get_state (void)
Parameters	None
Return value	whether enable or disable the LDO

Example

```
state = ebdat_vaux_get_state();
```

3.24.6. ebdat_i2c_read

Description

The function is used to read IIC slave device.

Prototype	boolean ebdat_i2c_read (byte slave_addr, byte req, byte* read_buf, int read_len)
Parameters	slave_addr: the address of slave_address req: the request read_buf: the read buffer read_len: the length of data to read
Return value	the result of operating TRUE: successful

	FALSE: failed

Example

```
rst = ebdat_i2c_read(slave_addr, req, read_buf, read_len);
```

3.24.7. ebdat_i2c_write

Description

The function is used to write IIC slave device.

Prototype	boolean ebdat_i2c_write (byte slave_addr, byte req, byte* write_buf, int write_len)
Parameters	slave_addr: the address of slave_address req: the request write_buf: the write buffer write_len: the length of data to write
Return value	the result of operating TRUE: successful FALSE: failed

Example

```
rst = ebdat_i2c_write(slave_addr, req, write_buf, write_len);
```

4. Setup Developing Environment

4.1. Install developing tools

Following is the steps to setup the building environment for EBDAT:

- Install RVCT22 Build593 to C:\Apps\RVDS210
- Install cygwin software to C:\cygwin
- Install ACtivePerl to C:\Perl
- Install python-2.6.2 to C:\Python26, and add a variable PYTHON to the system environment, it's value is C:\Python26.

4.2. Burn customer developed application to the module

Following is the steps to burn customer EBDAT application to the module:

- Copy the customer EBDAT application to the module

Use EFS-Explorer tool to copy customer EBDAT application(.elf file) to the module (C:\ directory on the module).

- Use AT+CEBDAT to bring the customer application into effect

For example: AT+CEBDAT="ebdat_cust_entry.elf"

- Reset the module
- Enable autorun character of EBDAT

If customer needs to run the customer application on powering up, the AT+CEBDATAUTORUN=1 needs to be used. If customer needs to test the customer application autorun function, the AT+CEBDATAUTORUN=1,<max_autorun_times> can be used, for example, AT+CEBDATAUTORUN=1,3, the module will run the application automatically in next three power cycles.

Appendix

A Related Documents

SN	Document name	Remark
[1]	SIMCOM_SIM5360_ATC_EN_VX.X.doc	

B Terms and Abbreviations

Abbreviation	Description
API	Application Programming Interface
CPU	Central Processing Unit
LIB	Library
OS	Operating System
PDU	Protocol Data Unit
RAM	Random-Access Memory
ROM	Read-Only Memory
UMTS	Universal Mobile Telecommunications System
USIM	Universal Subscriber Identity Module
WCDMA	Wideband Code Division Multiple Access

Contact Us

Shanghai SIMCom Wireless Solutions Ltd.

Add: Building A, SIM Technology Building, No.633, Jinzhong Road, Changning District 200335

Tel: +86 21 3252 3300

Fax: +86 21 3252 3301

URL: http:/www.sim.com/wm/