Lanner

Hardware Platforms for Network Computing

Lanner Platform Support Package User Guide

Version: 1.1

Date of Release: 2019-10-14

About this Document

This manual describes the overview of the various functionalities of this product and the information you need to get it ready for operation. It is intended for those who are:

- responsible for installing, administering and troubleshooting this system or information technology professionals.
- assumed to be qualified in the servicing of computer equipment, such as professional system integrators, or service personnel and technicians.

The latest version of this document can be found on Lanner's official website, available either through the product page or through the Lanner Download Center page with a login account and password.

Conventions & Icons

This document utilizes different font types and icons in order to make selected text more transparent and explicable to users. This document contains the following conventions:

Font Conventions

Example	Convention	Usage
`iptables -F`	Monospace, shaded	A command to be entered at a shell command-line
Setup page	Bold	A title of a dialog box or a page
<Enter>	Between a pair of inequality signs	A physical keyboard button
"Menu"	Between a pair of quotation marks	A menu option or a software button to be clicked
Readme.txt	In Italic	A filename or a file path
IPMI User Guide	Underlined	The name of another document or a chapter in this document

Icon Descriptions

Icon	Usage
 Note or Information	This mark indicates that there is something you should pay special attention to while using the product.
 Warning or Important	This mark indicates that there is a caution or warning and it is something that could damage your property or product.

Online Resources

To obtain additional documentation resources and software updates for your system, please visit the Lanner Download Center. As certain categories of documents are only available to users who are logged in, please be registered for a Lanner Account at http://www.lannerinc.com/ to access published documents and downloadable resources.

For troubleshooting the issues with your system, please visit the Lanner Q&A page for diagnostic procedures and troubleshooting steps.

Technical Support

In addition to contacting your distributor or sales representative, you could submit a request to our **Lanner Technical Support** at http://www.lannerinc.com/technical-support where you can fill in a support ticket to our technical support department.

Copyright and Trademarks

Documentation Feedback

Your feedback is valuable to us, as it will help us continue to provide you with more accurate and relevant documentation. To provide any feedback, comments or to report an error, please email to contact@lannerinc.com. Thank you for your time.

Contact Information

Taiwan Corporate Headquarters

Lanner Electronics Inc.

7F, No.173, Sec.2, Datong Rd. Xizhi District,

New Taipei City 22184, Taiwan

立端科技股份有限公司

221 新北市汐止區大同路二段 173 號 7 樓

T: +886-2-8692-6060

F: +886-2-8692-6101

E: contact@lannerinc.com

China

Beijing L&S Lancom Platform Tech. Co., Ltd.

Guodong LOFT 9 Layer No. 9 Huinan Road,

Huilongguan Town, Changping District, Beijing

102208 China

T: +86 010-82795600

F: +86 010-62963250

E: service@ls-china.com.cn

USA

Lanner Electronics Inc.

47790 Westinghouse Drive Fremont, CA 94539

T: +1-855-852-6637

F: +1-510-979-0689

E: sales_us@lannerinc.com

Canada

LEI Technology Canada Ltd

3160A Orlando Drive Mississauga, ON L4V 1R5

Canada

T: +1 877-813-2132

F: +1 905-362-2369

E: sales_ca@lannerinc.com

Revision History

Version	Date	Description
1.0	08/15/2019	First release
1.1	10/14/2019	Update for PSP 2.1

Table of Contents

CHAPTER 1 INTRODUCTION

Lanner PSP aims to simplify and enhance the efficiency of customer's application implementation. When developers intend to write an application that involves hardware access, they were required to fully understand the specifications to utilize the drivers. This is often being considered a time-consuming job which requires lots of related knowledge and time. In order to achieve better full-access hardware functionality, Lanner invests great effort to ease customer's development journey with the release of a suite of reliable Software APIs.

Benefits

Faster Time to Market

Lanner's API helps developers to write applications to control the hardware without knowing the hardware specs of the chipsets and driver architecture.

Reduced Project Effort

Accessing the API, developers are being given a more efficient way to utilize chipsets, communication bus, and physical I/Os. In addition, Lanner PSP also provides corresponding tools to prevent unexpected downtime causing from environment setup matters.

Enhances Lanner Platform Reliability

Lanner PSP is released after a series of reliability tests and security validations which combines manufacturing test sequences to enhance complete system reliability.

Flexible Upgrade Possibilities

Considering customer's application maintenance and upgrade tasks, Lanner PSP is designed to be flexible to update/upgrade to the module level. The simple re-initialization process will bring up the updated and upgraded modules to be functional.

Backward compatibility

Lanner grants the responsibility to control API backward support, allowing customers to worry less about the development of new products.

Operating Systems

OS supports for the current version are as below:

- CentOS 6.9/7.4/7.6
- Ubuntu 16.04.1/18.04/19.04
- Fedora 25/30

CHAPTER 2 PACKAGE INFORMATION & GUIDE

Information

This package is Lanner's platform software support package for accessing motherboard resource. The supported functions will depend on the design of the platform hardware. The PSP supports both static and dynamic mode libraries, programs. The program default is “static mode”, and “dynamic” is optional.

Normally, the functions include Watch-Dog-Timer (WDT), General Purpose IO (GPIO), Hardware Monitor (HWM), Software Reset Button (SWR), System/Status LED (SLED), Lanner Gen3 Bypass (LBP), EEPROM (EEP), Redundant Power Supply (PSU) and some external device functions such as the LCD module.

The IoT platform includes Ignition Control (IGN), G-Sensor(GSR), Global Positioning System(GPS), and Controller Area Network(CAN) .

PSP Package Installation & Building Guide

Package Content Description

Directory "iodrv"

This is a source code for IO driver, IO access libraries and utilities. Please refer to the document file “build-howto" to create execution file for your Linux system.

- `include`: all configurations and settings by hardware platform designed
- `src_drv`: the IO driver source code; the output file name is "`lmbiodrv.ko`"
- `src_lib`: the common libraries for IO driver and SMBus i2c-dev driver; the output file is `liblmbio.so`
- `src_tools`: Source code of iodrv tools is only for simple accessing without integration. The default make configure does not include it.

Directory "bin"

This is an output directory when you build the "iodrv" and "sdk" source. Please refer the "utils-howto" file for installation and usage.

- amd64: the output directory of x86_64 64-bit system
  - `driver`: the driver file "lmbiodrv.ko" location
  - `lib`:the all dynamic and static libraries location
  - `utils`:the execution files for IO access via SDK libraries (refer to the "sdkutils-howto" for installation and usage)
  - `loaddriver.sh` : script file for loading lmbiodrv.ko and i2c-dev drivers
  - `setenv.sh`: script file for setting LD_LIBRARY_PATH environment variable
  - `iodrv_tools`: a directory which will be created right after building "src_tools" is enabled
- i386: the output directory of i386/i686 32-bit system (sub-directory the same as "amd64")

Directory "sdk"

This is a source code for Lanner SDK libraries and utilities. Please refer to the document file "build-howto" to create execution file for your Linux system.

This package provides integration API library for common application interface, reducing porting effort, fast time-to-market, easy upgrading and changing platform. This sdk uses lmbiodrv.ko and common libraries, please build iodrv first. About the SDK detail description, please refer the file "Lanner SDK User Guide_Vx.x.pdf", and refer smaple code from directory "sdk/sdkutils_src".

- `include`:the SDK includes the file directory, the description and the definition of API functions.
- `src_sdk`: the source code of Lanner SDK libraries.
- `src_utils`: the source code of Lanner utilities.

Directory "hwmon"

This directory provides the configuration file for using lm-sensors tools to get hardware monitor information via **hwmon** driver. Please refer to the "sensors-howto" file to use lm-sensors tools. Some platforms use IPMI to get hardware monitor information. Please refer to the "ipmi-howto.txt" file to get the information via IPMI tools.

Building How-to

This package includes driver, common libraries, utility source code. Please build all binary files (driver, common library, utilities) for your Linux system.

Binary Output Layout

1. x86_64(64-bit) output location
   - `driver(lmbiodrv.ko)`: "bin/amd64/driver" directory.(src_drv)
   - `libraries`:"bin/amd64/lib" directory.(src_lib)(src_sdk)
   - `utilities`:"bin/amd64/utils" directory.(src_utils)

2. i386/i686(32-bit) output location
   - `driver(lmbiodrv.ko)`: "bin/i386/driver" directory (src_drv)
   - `libraries`: "bin/i386/lib" directory (src_lib)(src_sdk)
   - `utilities`:"bin/i386/utils" directory (src_utils)

Build Iodrv and SDK All Binary Files

1. `$ make clean`: to clean driver, all libraries and utilities files
2. `$ make`: to build driver, libraries and utilities based on current Linux system
   `$ make all` : to build both x86_64 and i386/i686 (32-bit environment required)
   `$ make amd64` : to build only x86_64 environment
   `$ make i386`: to build only i386/i686 environment (32-bit environment required)


Note
The driver will vary by system environment.

Build Iodrv binary files

1. `$ cd iodrv`
2. `$ make clean`: to clean driver and IO library files
   `$ make`: to build driver and IO libraries based on current Linux system
   `$ make all`: to build both x86_64 and i386/i686 (32-bit environment required)
   `$ make amd64` : to build only x86_64 environment
   `$ make i386`: to build only i386/i686 environment (32-bit environment required)


Note
The driver will vary by system environment.

Build IO driver Only

1. `$ cd iodrv/src_drv`
2. `$ make clean`: to clean driver binary
3. `$ make`: to build IO driver depending on the system

   `$ make all`: to build IO driver depending on the system

   `$ make amd64`: to build IO driver depending on the system

   `$ make i386`: to build IO driver depending on the system

Build IO library Only

1. `$ cd iodrv/src_drv`
2. `$ make clean`: to clean driver binary
3. `$ make`: to build IO driver depending on the system

   `$ make all`: to build IO driver depending on the system

   `$ make amd64`: to build IO driver depending on the system

   `$ make i386`: to build IO driver depending on the system

Build SDK Libraries and Utilities

1. `$ cd sdk`
2. `$ make clean`: to clean SDK libraries and utilities

   `$ make`: to build SDK libraries and utilities based on current Linux system

   `$ make all`: to build both x86_64 and i386/i686 (32-bit environment required)

   `$ make amd64`: to build only x86_64 environment

   `$ make i386`: to build only i386/i686 environment (32-bit environment required)

Build Utilities Only

1. `$ cd sdk/src_utils`
2. `$ make clean`: to clean all utils binary
3. `$ make`: to build utilities based on current Linux system

   `$ make all`: to build both x86_64 and i386/i686 (32-bit environment required)

   `$ make amd64`: to build only x86_64 environment

   `$ make i386`: to build only i386/i686 environment (32-bit environment required)

Note

1. Utilities always build on static mode.

2. "entry sub_directory" could build one utils

Build Dynamic Utilities Mode

1. `$ cd sdk/src_utils`
2. Edit "Makefile" and modify "export BUILD_DYNAMIC=0" to "export BUILD_DYNAMIC=1"

   `$ make`: to build utilities depending on Linux system

   `$ make all`: to build both x86_64 and i386/i686 (32-bit environment required)

   `$ make amd64`: to build only x86_64 environment

   `$ make i386`: to build only i386/i686 environment (32-bit environment required)

Note

(1) "_dynamic" will be appended to all utility file names.

(2) Please use "setsnv.sh" to set the "LD_LIBRARY_PATH" environment.

Utils How-to

Install Driver (Manual)

==========

1. `$ cd bin/amd64`: (assuming the platform to be x86_64)
2. `$ sudo su`: to enter super user mode
3. `# modprobe i2c-dev`: to install i2c-dev for SMBus access. CentOS system

   `# modprobe i2c-i801` : Ubuntu system
4. `# insmod driver/lmbiodrv.ko` : to install lmbiodrv.ko for IO access

   lmbiodrv.ko will auto-check BIOSID to see if the hardware is matched, and then parameter **idcheck=no** will skip BIOSID checking, e.g. `# insmod lmbiodrv.ko idcheck=no`
5. The driver will create **/dev/lmbiodrv** device node. If your system does not support **udev**, please use `$ dmesg | tail` to check the **lmbiodrv** major and minor number, and use `# sudo mknod dev/lmbiodrv c 10 minor#` to create device node.

Install Driver (Auto)

==========

1. `$ cd amd64` (assuming the platform to be x86_64)
2. `$ sudo su` (to enter super user mode)
3. `# . loaddriver.sh or ./loaddriver.sh`

Dynamic Program Setting

==========

The PSP default provides static type program that does not set "LD_LIBRARY_PATH". To enable “dynamic”, please refer to the "build-howto" file for building dynamic program. "_dynamic" is appended to all dynamic program names. Please set LD_LIBRARY_PATH as below:

- **Manual**:

  ```
  # cd bin/amd64
  # export LD_LIBRARY_PATH=`pwd`/lib/:$LD_LIBRARY_PATH
  ```

- **Auto**:

  ```
  # cd bin/amd64
  # . setenv.sh
  ```

  
  Note
  Uses ./setenv.sh is an invalid command.

Utils Contents & Usage

==========

After “lmbiodrv.ko” is installed, please enter "utils' directory `# cd bin/amd64/utils` and then execute them.

sdk_hwm

- **Description**: SDK hardware monitor program
- **Usage**:

  `./sdk_hwm -temp cpu1/cpu2/sys1/sys2`

  `./sdk_hwm -volt core1/core2/12v/5v/3v3/5vsb/3v3sb/vbat/psu1/psu2`

  `./sdk_hwm -rpm fan1/fan2/fan3/..../fan10`

  `./sdk_hwm -rpm fan1a/fan1b/fan2a/fan2b/....../fan10a/fan10b`

  `./sdk_hwm -callback`: use callback to detect caseopen

  `./sdk_hwm -sidname #dec`: print sensor name by **#dec**

  `./sdk_hwm -sidmsg #dec`: print sensor message by **#dec**

  `./sdk_hwm –sidlist`: print list all supports sensor ID

  `./sdk_hwm -testop`: for caseopen testing

  `./sdk_hwm –testhwm`: for hardware monitor testing

- **Example**:

  `# ./sdk_hwm -temp cpu1`: for getting CPU-1 temperature

  `# ./sdk_hwm -testop`: for testing case-open event

  `# ./sdk_hwm -testhwm`: for testing all hardware monitor and checking maximum boundary status

  
  Note
  The **hwm.conf** file is the boundary value setting file.

sdk_gpio

- **Description**: SDK General Purpose Input Output program
- **Usage**:

  `./sdk_gpio -gpo -w -data hex`: to write GPO data

  `./sdk_gpio -gpo -wpin 1/../4 -set/-reset`: to write GPO pin data

  `./sddk_gpio -gpo/-gpi -r`: to read GPO/GPI data

  `./sdk_gpio -gpi -rpin 1/../4`: to read GPI pin data

  `./sdk_gpio -callback`

- **Example**:

  `# ./sdk_gpio -gpo -w -data a` : to set GPIO4~1 data to "**1010B**"

  `# ./sdk_gpio -gpo -wpin 1 -set`: to set GPO-1 to "**high"**

  `# ./sdk_gpio -gpi -rpin 2`:to get GPI-2 status

  `# ./sdk_gpio -callback`: to simulate GPI status change events

sdk_eep

- **Description**: SDK EEPROM program
- **Usage**:

  `./sdk_eep -s0/../-s8 -byte/-word/-dword -addr #hex -read`

  `./sdk_eep -s0/../-s8 -byte/-word/-dword -addr #hex -write #hex`

  `./sdk_eep -s0/../-s8 -erase`

  `./sdk_eep -s0/../-s8 -test [-c textfile]` : for testing

- **Parameter**:

  `-s0/../-s8` : to assign slot device number. **-s0** is onboard device

  `-erase`: to erase EEPROM all content to **value 0**

  `-read/-write`: to read or write access

  `-addr #hex`: to assign data address of EEPROM device

- **Example**:

  `# ./sdk_eep -s0 -word -addr 2 -read`: to read word data from OnBoard EEPROM address **0x02**

  `# ./sdk_eep -s2 -dword -addr 14 -write ab1234`: to write dword data **0xab1234** to **Slot2** address **0x14**

  `# ./sdk_eep -s3 -test`: to test **Slot3** EEPROM write/read by *testeep.txt* file input

  `# ./sdk_eep -s3 -test -c eepfile` : to test **Slot3** EEPROM write/read by assigning file input

sdk_lbp

- **Description**: SDK Gen3 Lan Bypass program
- **Usage**:

```
./sdk_lbp -s0/../-s8 -info/-status/-save
./sdk_lbp -s0/../-s8 -sysoff/-juston/-runtime  #hex
./sdk_lbp -s0/../-s8 -p1/../-p4 -enable/-disable
./sdk_lbp -s0/../-s8 -t1/-t2 -second #dec -effect #hex
./sdk_lbp -s0/../-s8 -t1/-t2 -start/-reload/-stop
```

- **Parameter**:

`-s0/../-s8`: to assign slot device number, -**s0** is onboard device (by default)

`-p1/../-p4`: to assign pair number (runtime stage)

`-t1/-t2` : to assign time-1 or timer-2

`-enable/-disable`: to enable or disable runtime assign pair number (runtime stage)

`-info`: for getting information

`-status`: for getting bypass status

`-save`: to save value to default

`-sysoff #hex` : system-off pairs setting

`-juston#hex`: just-on pairs setting

`-runtime #hex`: runtime pairs setting

`-second #dec`: counter setting for bypass timer (1~255)

`-effect #hex`: to assign which pair bypass is to be enabled after time-out.

`-start/-reload/-stop`: to start, reload or stop timer

- **Example**:

`# ./sdk_lbp -s1 -info`: for getting **Solt1** Bypass device information

`# ./sdk_lbp -s0 -save`: to save current setting to boot default

`# ./sdk_lbp -s3 -sysoff 5`: for setting systemoff stage pairs status to "**0101B**"

`# ./sdk_lbp -s1 -t1 -second 30 -effect a`: to set Slot1 Timer 1 count to 30 seconds while enabling bypass pair4 and pair2 after time-out.

`# ./sdk_lbp -s1 -t1 -start`: to start **Solt1 Timer-1** (add new)

`# ./sdk_lbp -s1 -t1 -reload`: to reload **Solt1 Timer-1**

sdk_sled

- **Description**: SDK System/Status LED program
- **Usage**:

  `./sdk_sled -green/-red/-off`: for setting System/Status LED

  `./sdk_sled -test`: for testing
- **Example**:

  `# ./sdk_sled -green`: to set System/Status LED color to green

  `# ./sdk_sled -test`: to enter SLED test mode as green → red/amber→ off at 2-second intervals.

sdk_swr

- **Description**: SDK Software Reset Button program
- **Usage**:

  `./sdk_swr -status`: to read status

  `./sdk_swr -callback`: to use callback detection

  `./sdk_swr -test`: for testing
- **Example**:

  `# ./sdk_swr -callback`: for detection of SWR pressed event

  `# ./sdk_swr -test`: for testing the pressed state (within 5 seconds) of "Software Reset Button".

sdk_wdt

- **Description**: SDK Watch Dog Timer program
- **Usage**:

  `./sdk_wdt -config seconds`

  `./sdk_wdt -start [seconds]`

  `./sdk_wdt -reload`

  `./sdk_wdt -stop`
- **Example**:

  `# ./sdk_wdt -config 20` : to set Watch-Dog-Timer count value to 20 sec without starting up

  `# ./sdk_wdt -start 15`: to set Watch-Dog-Timer count value to 15 sec followed by a starting up

  `# ./sdk_wdt -stop`: to stop WDT countdown

  `# ./sdk_wdt -reload`: to reload WDT and re-countdown

sdk_psu

- **Description**: SDK Redundant Power Supply program
- **Usage**:

  `./sdk_psu -info/-sensor/-watts 1/2/3`

  `./sdk_psu -callback`

  `./sdk_psu -test [-c filename] (default: psu.conf)`
- **Example**:

  `# ./sdk_psu -watts 1`: for getting PSU-1 Vin/Vout, Iin/Iout and WattsIn/Out

  `# ./sdk_psu -test`: for testing all PSU information and checking max/min boundary status

sdk_lcm

- **Description**: SDK LCD module program
- **Usage**:

  `./sdk_lcm -search`: to search LCM port and baudrate

  `./sdk_lcm [-port #dec] [-baud #dec] -reset`: to reset LCD module

  `./sdk_lcm [-port #dec] [-baud #dec] -clear`: to clear LCD module screen

  `./sdk_lcm [-port #dec] [-baud #dec] -startup "starup message"`

  `--> setting LCM startup message after reset`

  `./sdk_lcm [-port #dec] [-baud #dec] -light 0/1`: to control LCM backlight Off/On

  `./sdk_lcm [-port #dec] [-baud #dec] -wrap 0/1`: to control LCM line wrap Off/On

  `./sdk_lcm [-port #dec] [-baud #dec] -bright 0/../8`: to control LCM brightness

  `./sdk_lcm [-port #dec] [-baud #dec] -curtype 0/1/2`

  `--> setting cussor type off/underline/block_blink`

  `./sdk_lcm [-port #dec] [-baud #dec] -cursor #row [#column]`: to set cursor position

  `./sdk_lcm [-port #dec] [-baud #dec] -write "string"`: to write string to LCM

  `./sdk_lcm [-port #dec] [-baud #dec] -setspeed #dec`

  `--> setting new LCM connection speed`

  `./sdk_lcm -callback`: to Callback for LCM key button

Note

If no port or speed is assigned, the default will be **/dev/ttyS1** and **115200**.

CHAPTER 3 DEFINITION

Constant and Macro Definition

```
/* WDT timer base ***/
#define BASE_SECOND       1
#define BASE_MINUTE       2

/* WDT type */
#define WDT_TYPE_UNKNOW   0
#define WDT_TYPE_SIO      1
#define WDT_TYPE_TCO      2

#define DISABLE           0
#define ENABLE            1

/* HWM Fan sequence */
#define FAN_A    1
#define FAN_B    2
```

```
/* LAN Bypass Controller Slot define */
#define SLOT_ONBOARD  0x00
#define SLOT_INDEX_1  0x01
#define SLOT_INDEX_2  0x02
#define SLOT_INDEX_3  0x03
#define SLOT_INDEX_4  0x04
#define SLOT_INDEX_5  0x05
#define SLOT_INDEX_6  0x06
#define SLOT_INDEX_7  0x07
#define SLOT_INDEX_8  0x08

/* LAN Slot modules type */
#define LAN_TYPE_UNKNOW   0
#define LAN_TYPE_COPPER   1
#define LAN_TYPE_FIBER    2

/* LAN Bypass module bit define */
#define LBP_MODBIT_SystemOff   0x01
#define LBP_MODBIT_JustOn      0x02
#define LBP_MODBIT_Runtime     0x04
#define LBP_MODBIT_TIMER1      0x08
#define LBP_MODBIT_TIMER2      0x10
#define LBP_MODBIT_TIMER3      0x20
#define LBP_MODBIT_Disconnect  0x40

/* LAN Bypass Timer Configure: action item*/
#define PAIR_NOASSIGN        0
#define PAIR_BYPASS_ENABLE   1
#define PORT_DISCONNECT      2
```

```
/***** Intrusion Item Define (bit) *****/
/* Hardware Monitor Intrusion */
#define INTRMSG_CASEOPEN      0x00000001
/* Software Reset Button Intrusion */
#define INTRMSG_SWRESET       0x00000001
/* Power Supply Intrusion */
#define INTRMSG_PSU1_INPUT    0x00000001
#define INTRMSG_PSU2_INPUT    0x00000002
#define INTRMSG_PSU1_TEMP     0x00000004
#define INTRMSG_PSU2_TEMP     0x00000008
#define INTRMSG_PSU1_VOUT     0x00000010
#define INTRMSG_PSU2_VOUT     0x00000020
#define INTRMSG_PSU1_IOUT     0x00000040
#define INTRMSG_PSU2_IOUT     0x00000080
#define INTRMSG_PSU1_FAN      0x00000100
#define INTRMSG_PSU2_FAN      0x00000200
/* Ignition KeyOn lose Intrusion */
#define INTRMSG_IGNKEYOFF     0x00000001

/* PSU Watts Select */
#define PSU_WATTS_INPUT       0
#define PSU_WATTS_OUTPUT      1

/* PSU Status Bit define */
#define PSU_FAULT_NONEOFABOVE 0x0001
#define PSU_FAULT_COMM        0x0002
#define PSU_FAULT_TEMP        0x0004
#define PSU_FAULT_VIN_UV      0x0008
#define PSU_FAULT_IOUT_OC     0x0010
#define PSU_FAULT_VOUT_OV     0x0020
#define PSU_FAULT_UINTOFF     0x0040
#define PSU_FAULT_UINTBUSY    0x0080
#define PSU_FAULT_UNKNOWN     0x0100
#define PSU_FAULT_OTHERS      0x0200
#define PSU_FAULT_FAN         0x0400
#define PSU_FAULT_POWERGOOD   0x0800
#define PSU_FAULT_MFRSPEC     0x1000
#define PSU_FAULT_INPUT       0x2000
#define PSU_FAULT_IOUT        0x4000
#define PSU_FAULT_VOUT        0x8000
```

```
/*** Ignition device functions ***/
#define FUNC_STARTUPWDT          0x0001
#define FUNC_LOWPOWER            0x0002
#define FUNC_IGNGPI              0x0004
#define FUNC_IGNGPO              0x0008
#define FUNC_IGNPOE              0x0010
#define FUNC_HEATER              0x0020

/** Ignition Power Status **/
#define STAT_CLOSEUP             0
#define STAT_POWERON_DELAY       1
#define STAT_WAIT_STARTUP        2
#define STAT_STARTUP             3
#define STAT_SHUTDOWN_DELAY      4
#define STAT_SHUTTING_DOWN       5
#define STAT_LOWPOWER_DELAY      6
#define STAT_UNKNOWN             255

/** LMB_IGN_GetParamRange define **/
#define PARAM_KeyOnDelayTime     1
#define PARAM_StartUpTime        2
#define PARAM_StartupWdtTime     3
#define PARAM_MaxFailStartupCnt  4
#define PARAM_LowmVolt           5
#define PARAM_HighmVolt          6
#define PARAM_ConfirmTime        7
#define PARAM_MaxFailShutdownCnt 8
#define PARAM_KeyOffDelayTime    9
#define PARAM_ShutdownTime       10
#define PARAM_mCelsiusMin        11
#define PARAM_mCelsiusMax        12
#define PARAM_Unknown            (PARAM_mCelsiusMax+1)

/*** Ignition Wake-up pin define ***/
#define WAKE_IGN_DI0             0x01
#define WAKE_IGN_DI1             0x02
#define WAKE_3G_MODULE           0x04
```

```
/************  G-Sensor Device ADXL345  **************/
#define  GSR_DEVID          0x00  //Device ID, read only
#define  GSR_THRESH_TAP     0x1D  //Tap threshold
#define  GSR_OFSX           0x1E  //X-axis offset
#define  GSR_OFSY           0x1F  //Y-axis offset
#define  GSR_OFSZ           0x20  //Z-axis offset
#define  GSR_DUR            0x21  //Tap duration
#define  GSR_LATENT         0x22  //Tap latency
#define  GSR_WINDOW         0x23  //Tap window
#define  GSR_THRESH_ACT     0x24  //Activity threshold
#define  GSR_THRESH_INACT   0x25  //Inactivity threshold
#define  GSR_TIME_INACT     0x26  //Inactivity time
#define  GSR_ACT_INACT_CTL  0x27  //Axis enable control for activity
                                  //and inactivity detection
#define  GSR_THRESH_FF      0x28  //Free-fall threshold
#define  GSR_TIME_FF        0x29  //Free-fall time
#define  GSR_TAP_AXES       0x2A  //Axis control for single tap/double tap
#define  GSR_ACT_TAP_STATUS 0x2B  //Source of single tap/double tap, read only
#define  GSR_BW_RATE        0x2C  //Data rate and power mode control
#define  GSR_POWER_CTL      0x2D  //Power-saving features control
#define  GSR_INT_ENABLE     0x2E  //Interrupt enable control
#define  GSR_INT_MAP        0x2F  //Interrupt mapping control
#define  GSR_INT_SOURCE     0x30  //Source of interrupts, read only
#define  GSR_DATA_FORMAT    0x31  //Data format control
#define  GSR_DATAX0         0x32  //X-Axis Data 0, read only
#define  GSR_DATAX1         0x33  //X-Axis Data 1, read only
#define  GSR_DATAY0         0x34  //Y-Axis Data 0, read only
#define  GSR_DATAY1         0x35  //Y-Axis Data 1, read only
#define  GSR_DATAZ0         0x36  //Z-Axis Data 0, read only
#define  GSR_DATAZ1         0x37  //Z-Axis Data 1, read only
#define  GSR_FIFO_CTL       0x38  //FIFO control
#define  GSR_FIFO_STATUS    0x39  //FIFO status, read only
```

Structure Definition

```
typedef struct DEF_DLL_VERSION {
  uint16_t uwDllMajor;
  uint16_t uwDllMinor;
  uint16_t uwDllBuild;
  int8_t   strPlatformID[15];
  uint16_t uwBoardMajor;
  uint16_t uwBoardMinor;
  uint16_t uwBoardBuild;
}DLL_VERSION;

typedef struct DEF_WDT_INFO {
  uint8_t  ubType;
  uint16_t uwCountMax;
  uint8_t  unMinuteSupport;
}WDT_INFO;

typedef struct DEF_LBPDEV_INFO {
  uint8_t  ubSlotIndex;
  uint8_t  ubType;           //copper or fiber
  uint8_t  ubVersion[2];     //major & minor
  uint8_t  ubModules;        //modules support
  uint8_t  ubSystemOff_Pairs;//bit indicate of pair
  uint8_t  ubJustOn_Pairs;
  uint8_t  ubRuntime_Pairs;
  uint16_t uwTimer1_MaxSec;  //maximum seconds
  uint16_t uwTimer2_MaxSec;
  uint16_t uwTimer3_MaxSec;
}LBPDEV_INFO;
```

```
typedef struct DEF_PAIRS_STATUS {
  uint8_t ubSystemOff_PairsBypass;
  uint8_t ubJustOn_PairsBypass;
  uint8_t ubRuntime_PairsBypass;
  uint8_t ubJustOn_PortsDisconnect;
  uint8_t ubRuntime_PortsDisconnect;
}PAIRS_STATUS;

typedef struct DEF_LBP_TIMERCFG {
  uint8_t  ubTimerNo; //timer number
  uint16_t uwTimeSec; //timer value by second
  uint8_t ubAction;    //timeout action
  uint8_t ubEffect;    //assign pairs or ports
}LBP_TIMERCFG;

typedef struct DEF_INTRUSION_TIME {
  uint16_t uwYear;
  uint8_t ubMonth;
  uint8_t ubDay;
  uint8_t ubHour;
  uint8_t ubMinute;
  uint8_t ubSecond;
}INTRUSION_TIME;

/*** Intrusion Callback Function ***/
typedef struct DEF_INTRUSION_MSG {
  uint32_t udwOccurItem;    //Occur item
  uint32_t udwStatus;       //Occur Status
  INTRUSION_TIME stuTime;   //Occur time
}INTRUSION_MSG;
typedef void (*INTRUSION_CALLBACK)(INTRUSION_MSG);
```

```
/***GPI Changed Callback***/
typedef struct DEF_GPI_MSG {
  uint8_t ubGroup;           //GPI Group
  uint32_t udwGpis;          //Occur GPIs
  uint32_t udwStatus;        //Occur Status
  INTRUSION_TIME stuTime;    //Occur time
}GPI_MSG;
typedef void (*GPI_CALLBACK)(GPI_MSG);

/*** LCM Device Information ***/
typedef struct DEF_LCM_INFO {
  uint16_t uwModeNo;
  uint16_t uwVersion;
  uint32_t udwBaudrate;
} LCM_INFO;

/***LCM keys Callback***/
typedef struct DEF_LCMKEY_MSG {
  uint8_t  ubKeys;           //Occur Keys
  uint8_t ubStatus;          //Occur Status
  INTRUSION_TIME stuTime;    //Occur time
}LCMKEY_MSG;
typedef void (*LCMKEY_CALLBACK)(LCMKEY_MSG);
```

```
/*** Power Supply ***/
typedef struct DEF_PSU_INFO {
  uint8_t ubPsuNo;    //input parameter
  uint8_t strubMfrId[33];
  uint8_t strubMfrModel[33];
  uint8_t strubMfrSerial[33];
  uint8_t strubMfrRevision[33];
}PSU_INFO;

typedef struct DEF_PSU_WATTS {
  uint8_t  ubPsuNo;   //input parameter
  uint8_t  ubInOut;   //input parameter
  float    fVolt;
  float    fAmpere;
  float    fWatts;
}PSU_WATTS;

typedef struct DEF_PSU_SENSORS {
  uint8_t  ubPsuNo;   //input parameter
  float    fTemp_1;
  float    fTemp_2;
  uint16_t uwFanRpm;
}PSU_SENSORS;
```

```
/* Ignition device structure define */
typedef struct DEF_STARTUP_SCHEME {
  int32_t dwMaxFailStartupCnt;
  int32_t dwStartupWdtTime;
  int32_t dwKeyOnDelayTime;
  int32_t dwStartUpTime;
}STARTUP_SCHEME;

typedef struct DEF_LOWPOWER_SCHEME {
  int32_t dwDisabled;
  int32_t dwLowmVolt;     //mV
  int32_t dwHighmVolt;    //mV
  int32_t dwConfirmTime;  //second
}LOWPOWER_SCHEME;

typedef struct DEF_SHUTDOWN_SCHEME {
  int32_t dwMaxFailShutdownCnt;
  int32_t dwKeyOffDelayTime;
  int32_t dwShutdownTime;
}SHUTDOWN_SCHEME;

typedef struct DEF_HEATER_SCHEME {
  int32_t dwmCelsiusMin;    //mCelsius
  int32_t dwmCelsiusMax;    //mCelsius
}HEATER_SCHEME;
```

```
typedef struct DEF_IGN_VERSION {
  int8_t strbDeviceID[30];
  int8_t strbVersion[30];
}IGN_VERSION;

typedef struct DEF_IGN_RUNTIME_INFO {
  int32_t dwPowerState;
  int32_t dwInputVoltage;     //mV
  int32_t dwFailShutdownCnt;
  int32_t dwFailStartupCnt;
  int32_t dwShutdownFalg;
}IGN_RUNTIME_INFO;

typedef struct DEF_VALUE_RANGE {
  int32_t dwMinimum;
  int32_t dwMaximum;
  int32_t dwDefault;
}VALUE_RANGE;
```

```
/*** GPS - Global Positioning System ***/
typedef struct  DEF_RMC_DATA {
   int8_t  strLongitude[20];    //'E'= East; 'W'= West,
                                //Longitude in dddmm.mmmm format.
   int8_t  strLatitude[20];     //'N'= North; 'S'= South,
                                //Latitude in dddmm.mmmm format.
   int8_t  strSpeedKnots[10];   //Speed over ground in knots (000.0 ~ 999.9)
   int8_t  strStatus[5];        //Status, 'V'= Navigation receiver warning,
                                //'A'= Data Valid
   int8_t  strUTCTime[15];      //UTC time in hhmmss.sss format
   int8_t  strUTCDate[15];      //UTC date of position fix, ddmmyy format
   int8_t  strCourse[10];       //Course over ground in degrees (000.0 ~ 359.9)
}RMC_DATA;

typedef struct  DEF_GGA_DATA {
   int8_t  strLongitude[20];    //Longitude
   int8_t  strLatitude[20];     //Latitude
   int8_t  strAltitude[10];     //Mean sea level altitude
                                //(-9999.9 ~ 17999.9) in meter
   int8_t  strGeoidalSeparation[10];   //In meter
   int8_t  strHDOP[5];          //Horizontal dilution of precision, (00.0 ~ 99.9)
   int8_t  strQuality[5];       //GPS quality indicator
                                //  0: position fix unavailable
                                //  1: valid position fix, SPS mode
                                //  2: valid position fix, differential GPS mode
   int8_t  strUTCTime[15];      //UTC Time
   int8_t  strSatelitesUsed[5];//Number of satellites in use, (00 ~ 24)
}GGA_DATA;
```

```
typedef struct  DEF_VTG_DATA {
    int8_t  strTrueCourse[10];    //Course over ground,
                                  //degrees True (000.0 ~ 359.9)
    int8_t  strMagneticCourse[10];//Course over ground,
                                  //degrees Magnetic (000.0 ~ 359.9)
    int8_t  strSpeedKnots[10];    //Speed over ground in knots (000.0 ~ 999.9)
    int8_t  strSpeedKm[10];       //Speed over ground in kilometers per hour
                                  //(0000.0 ~ 1800.0)
    int8_t  strMode[5];           //Mode indicator, 'N'= not valid,
                                  //      'A'= Autonomous mode
                                  //      'D'= Differential mode
                                  //      'E'= Estimated (dead reckoning) mode
}VTG_DATA;

typedef struct  DEF_GLL_DATA {
    int8_t  strLongitude[20];   //Longitude
    int8_t  strLatitude[20];    //Latitude
    int8_t  strUTCTime[15];     //UTC Time
    int8_t  strStatus[5];       //Status, 'V'= Navigation receiver warning,
                                //'A'= Data Valid
}GLL_DATA;

typedef struct  DEF_GSA_DATA {
    int8_t  strMode[5];      //'M'= Manual forced 2D or 3D, 'A'= Automatic switch
  2D/3D
    int8_t  strType[5];      //fix type, '1'= fix not available, '2' = 2D, '3' =
3D
    uint8_t aryubSatelliteID[24];   //Satellite used ID list,
                                    //Maximally 24 satellites
    int8_t  strPDOP[10];     //Position dilution of precision (00.0 to 99.9)
    int8_t  strHDOP[10];     //Horizontal dilution of precision (00.0 to 99.9)
    int8_t  strVDOP[10];     //Vertical dilution of precision (00.0 to 99.9)
}GSA_DATA;
```

```
typedef struct DEF_SATELITE_INFO {
  int8_t  strSateliteID[5];
  int8_t  strElevation[5];  //Satellite elevation in degrees, (00 ~ 90)
  int8_t  strAzimuth[5];    //Satellite azimuth angle in degrees, (000 ~ 359 )
  int8_t  strSNR[5];        //Signal to Noise Ratio, in dB (00 ~ 99),
                            //Null when not tracking
}SATELITE_INFO;

typedef struct  DEF_GSV_DATA {
  int8_t   bGPStotal;              //total numbers of GPS satellites
  int8_t   bGLONASStotal;          //total numbers of GLONASS satellites
  SATELITE_INFO  stuGPS[12];       //GPS satellites
  SATELITE_INFO  stuGLONASS[12];   //GLONASS satellites
}GSV_DATA;

/*** G-Sensor X,Y,Z Axis ***/
typedef struct DEF_AXIS_RAWDATA {
  int16_t  wXaxis;
  int16_t  wYaxis;
  int16_t  wZaxis;
  int16_t  wgRange;
}AXIS_RAWDATA;
```

```
/************  CAN bus Module MT3647  **************/
typedef struct DEF_DRIVING_INFO {
  uint8_t  ubSpeedKm;
  uint8_t  ubAcceleratorPedal; //Accelerator Pedal Position (APP)
  uint8_t  ubFuelLevel;
  float    fTotalDistance;     //High Resolution Total Vehicle Distance
  int32_t  dwSERV;             //The distance which can be traveled
  uint8_t  ubTachographKm ;    //tachograph vehicle speed
  float    fBatteryVolt;
}DRIVING_INFO;

typedef struct DEF_ENGINE_INFO {
  uint16_t uwEngineRPM;
  uint8_t  ubEngineLoading;
  int16_t  wCoolantTemp;    //Engine Coolant Temperature(ECT)
  int16_t  wEIMT;           //Engine Intake Manifold 1 Temperature
  uint32_t udwTotalFuels;   //Engine Total Fuel used
  uint32_t udwTotalHours;   //Engine total hours of Operation
  uint16_t uwETBP  ;        //Engine Turbocharger Boost Pressure
  float    fEFR;            //Engine Fuel Rate
  float    fEIFE;           //Engine Instantaneous Fuel Economy
  float    fEODP;           //Engine Oil Filter Differential Pressure
}ENGINE_INFO;
```

```
/*** include file iodrv/include/iolib.h ***/
/*** Hardware Monitor Item Define ***/
typedef enum DEF_HWM_SENEORITEM {
//Temperature area
HWMID_TEMP_CPU1=0,
HWMID_TEMP_CPU2,
HWMID_TEMP_SYS1,
HWMID_TEMP_SYS2,
//Voltage area
HWMID_VCORE_CPU1,
HWMID_VCORE_CPU2,
HWMID_VOLT_P12V,
HWMID_VOLT_P5V,
HWMID_VOLT_P3V3,
HWMID_VOLT_P5VSB,
HWMID_VOLT_P3V3SB,
HWMID_VOLT_VBAT,
HWMID_VOLT_DDRCH1,
HWMID_VOLT_DDRCH2,
HWMID_VOLT_DDRCH3,
HWMID_VOLT_DDRCH4,
HWMID_VOLT_DDRCH5,
HWMID_VOLT_DDRCH6,
HWMID_VOLT_DDRCH7,
HWMID_VOLT_DDRCH8,
HWMID_VOLT_PVNN,
HWIID_VOLT_P1V05,
HWMID_VOLT_PVCCIO_CPU0,
HWMID_VOLT_PVCCIO_CPU1,
HWMID_VOLT_PVCCSA_CPU0,
HWMID_VOLT_PVCCSA_CPU1,
//Fan area
HWMID_RPM_FanCpu1,
HWMID_RPM_FanCpu2,
HWMID_RPM_FanSys1,
HWMID_RPM_FanSys2,
HWMID_RPM_Fan1A,
HWMID_RPM_Fan1B,
HWMID_RPM_Fan2A,
HWMID_RPM_Fan2B,
HWMID_RPM_Fan3A,
HWMID_RPM_Fan3B,
```

```
HWMID_RPM_Fan4A,
HWMID_RPM_Fan4B,
HWMID_RPM_Fan5A,
HWMID_RPM_Fan5B,
HWMID_RPM_Fan6A,
HWMID_RPM_Fan6B,
HWMID_RPM_Fan7A,
HWMID_RPM_Fan7B,
HWMID_RPM_Fan8A,
HWMID_RPM_Fan8B,
HWMID_RPM_Fan9A,
HWMID_RPM_Fan9B,
HWMID_RPM_Fan10A,
HWMID_RPM_Fan10B,
//Power supply area
HWMID_PSU1_STATUS,
HWMID_PSU1_VOLTIN,
HWMID_PSU1_VOLTOUT,
HWMID_PSU1_CURRENTIN,
HWMID_PSU1_CURRENTOUT,
HWMID_PSU1_POWERIN,
HWMID_PSU1_POWEROUT,
HWMID_PSU1_FAN1,
HWMID_PSU1_FAN2,
HWMID_PSU1_TEMP1,
HWMID_PSU1_TEMP2,
HWMID_PSU2_STATUS,
HWMID_PSU2_VOLTIN,
HWMID_PSU2_VOLTOUT,
HWMID_PSU2_CURRENTIN,
HWMID_PSU2_CURRENTOUT,
HWMID_PSU2_POWERIN,
HWMID_PSU2_POWEROUT,
HWMID_PSU2_FAN1,
HWMID_PSU2_FAN2,
HWMID_PSU2_TEMP1,
HWMID_PSU2_TEMP2,
//add here for new items
//
HWMID_TOTAL
}HWM_SENEORITEM;
```

Status Code

All Lanner API functions immediately return a status code from a common list of possible errors. Any function may return any of the defined status code.

```
#define ERR_Success     0
```

- **Description**: The operation is working properly.

```
#define ERR_Error     0xFFFFFFFF
```

- **Description**: Generic error message

```
#define ERR_NotExist  0xFFFFFFFE
```

- **Description**: Library file is not found/plugged, or the device does not exist.

```
#define ERR_NotOpened   0xFFFFFFFD
```

- **Description**: Library is not opened or not ready yet

```
#define ERR_Invalid     0xFFFFFFFC
```

- **Description**: The Parameter is not a valid value or out of range.

```
#define ERR_NotSupport  0xFFFFFFFB
```

- **Description**: This function is not supported in this platform.

```
#define ERR_BusyInUses  0xFFFFFFFA
```

- **Description**: Library is being used now or device IO is busy now.

```
#define ERR_BoardNotMatch 0xFFFFFFF9
```

- Description: The board library and the board do not match.

```
#define ERR_IPMI_IDLESTATE  0xFFFFFEFF
```

- **Description**: IPMI KCS interface is not in IDLE State.

```
#define ERR_IPMI_WRITESTATE 0xFFFFFEFE
```

- **Description**: IPMI KCS interface WRITE State check failure

```
#define ERR_IPMI_READSTATE  0xFFFFFEFD
```

- **Description**: IPMI KCS interface READ State check failure

```
#define ERR_IPMI_IBF0     0xFFFFFEFC
```

- **Description**: IPMI KCS interface wait IBF '0' State failure

```
#define ERR_IPMI_OBF1     0xFFFFFEFB
```

- **Description**: IPMI KCS interface wait OBF '1' State failure

CHAPTER 4 API FUNCTIONS

Library Control

LMB_DLL_Init

The **LMB_DLL_Init** function initializes the Lanner common API and board libraries.

- **Syntax**

  int32_t LMB_DLL_Init(void);

- **Parameters**

  (None)

- **Return Value**

  `ERR_Success`: Initializing library is working properly.

  `ERR_NotExist`: Board library is not found.

  `ERR_BoardNotMatch`: library and M/B do not match.

  `ERR_Error`: Initializing library failed.

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{ int32_t iRet;
     iRet = LMB_DLL_Init();
     if ( iRet == ERR_Success ) {
         printf("Initial library successful\n");
         if ( iRet == ERR_Success ) {
             printf("API Library is Ready\n");
         }
    }
    LMB_DLL_DeInit();
 return 0;
 }
```

LMB_DLL_DeInit

The **LMB_DLL_DeInit** function releases the Lanner common API library.

- **Syntax**

int32_t LMB_DLL_DeInit(void);

- **Parameters**

  (None)

- **Return Value**

  `ERR_Success`: Releasing library is working properly

  `ERR_BusyInUses`: Libray is busy or a certain process is in use

- **Remarks:** This function will auto-release board-level library.

- **Example**

  Refer to function LMB_DLL_Init example.

LMB_DLL_Version

The **LMB_DLL_Version** function loads the Lanner board-level library.

- **Syntax**

int32_t LMB_DLL_Version(DLL_VERSION* pstuDllVersion);

- **Parameters**

pstuDllVersion

[in] obtains the DLL and Board Library version information.

- **Return Value**

`ERR_Success`: This function is working properly.

`ERR_NotOpened`: The library is not ready or opened yet.

`ERR_NotSupport`: This platform does not support this function.

- **Remarks**

(None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{
int32_t iRet;
DLL_VERSION stuVer;
    LMB_DLL_Init();
    iRet = LMB_DLL_Version(&stuVer);
    if ( iRet == ERR_Success ) {
         printf("DLL Version: %d.%d.%d\n",stuVer.uwDllMajor,
                    stuVer.uwDllMinor,stuVer.uwDllBuild);
        printf("IODRV Version: %s.%d.%d.%d\n",
                    stuVer.strPlatformID, stuVer.uwBoardMajor,
                    stuVer.uwBoardMinor, stuVer.uwBoardBuild);
    }
   LMB_DLL_DeInit();
return 0;
}
```

Hardware Monitor

LMB_HWM_GetCpuTemp

The **LMB_HWM_GetCpuTemp** function reads temperature of the CPU processor.

- **Syntax**

  int32_t LMB_HWM_GetCpuTemp (uint8_t ubCpuNo, float* pfTemperature)

- **Parameters**

  ubCpuNo

  [out] selects CPU number.

  pfTemperature

  [in] the current temperature value of the CPU.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_Invalid`: The input parameter is out of range.

  `ERR_Error`: This function failed.

  `ERR_NotSupport`: This platform does not support this function.

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{
int32_t iRet ;
float fTemp =0;
     LMB_DLL_Init();
     iRet = LMB_HWM_GetCpuTemp(1, &fTemp);
     if ( iRet == ERR_Success )
                 printf("CPU 1 temperature is %f\n”, fTemp);
     LMB_DLL_DeInit();
     return 0;
 }
```

LMB_HWM_GetSysTemp

The **LMB_HWM_GetSysTemp** function reads temperature of the system.

- **Syntax**

  int32_t LMB_HWM_GetSysTemp(uint8_t ubSysNo, float* pfTemperature);

- **Parameters**

  ubSysNo

  [out] selects System sensor.

  pfTemperature

  [in] the current temperature value of the CPU.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_Invalid`: The input parameter is out of range.

  `ERR_Error`: This function failed.

  `ERR_NotSupport`: This platform does not support this function.

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{
int32_t iRet ;
float fTemp =0;
     LMB_DLL_Init();
     iRet = LMB_HWM_GetSysTemp(1, &fTemp);
     if ( iRet == ERR_Success )
                printf("SYS 1 temperature is %f\n”, fTemp);
     LMB_DLL_DeInit();
     return 0;
}
```

LMB_HWM_GetVcore

The **LMB_HWM_GetVcore** function reads the CPU Vcore voltage.

- **Syntax**

  int32_t LMB_HWM_GetVcore(uint8_t ubCpuNo, float* pfVoltage);

- **Parameters**

  ubCpuNo

  [out] selects CPU number.

  pfVoltage

  [in] obtains the current voltage of CPU Vcore.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_Invalid`: The input parameter is out of range.

  `ERR_Error`: This function failed.

  `ERR_NotSupport`: This platform does not support this function.

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{
int32_t iRet ;
float  fVolt =0;
     LMB_DLL_Init();
     iRet = LMB_HWM_GetVcore(1, &fVolt);
     if ( iRet == ERR_Success )
                printf("CPU 1 Vcore Voltage = %f\n”, fvolt);
     LMB_DLL_DeInit();
     return 0;
 }
```

LMB_HWM_Get12V

The **LMB_Get12V** function reads the positive 12V voltage.

- **Syntax**

  int32_t LMB_HWM_Get12V(float* pfVoltage);

- **Parameters**

  pfVoltage

  [in] obtains the current voltage of positive 12V.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_Error`: This function failed.

  `ERR_NotSupport`:This platform does not support this function.

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{
int32_t iRet ;
float  fVolt =0;
     LMB_DLL_Init();
     iRet = LMB_HWM_Get12V(&fVolt);
     if ( iRet == ERR_Success )
                printf("12V Voltage = %f\n”, fvolt);
     LMB_DLL_DeInit();
     return 0;
 }
```

LMB_HWM_Get5V

The **LMB_HWM_Get5V** function reads the positive 5V voltage.

- **Syntax**

  int32_t LMB_HWM_Get5V(float* pfVoltage);

- **Parameters**

  pfVoltage

  [in] obtains the current voltage of positive 5V.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_Error`: This function failed.

  `ERR_NotSupport`:This platform does not support this function.

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{
int32_t iRet ;
float  fVolt =0;
     LMB_DLL_Init();
     iRet = LMB_HWM_Get5V(&fVolt);
     if ( iRet == ERR_Success )
                printf("5V Voltage = %f\n”, fVolt);
     LMB_DLL_DeInit();
     return 0;
}
```

LMB_HWM_Get3V3

The **LMB_HWM_Get3V3** function reads the positive 3.3V voltage.

- **Syntax**

  int32_t LMB_HWM_Get3V3(float* pfVoltage);

- **Parameters**

  pfVoltage

  [in] obtains the current voltage of positive 3.3V.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_Error`: This function failed.

  `ERR_NotSupport`:This platform does not support this function.

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{
int32_t iRet ;
float  fVolt =0;
     LMB_DLL_Init();
     iRet = LMB_HWM_Get3V3(&fVolt);
     if ( iRet == ERR_Success )
                printf("3.3V Voltage = %f\n”, fVolt);
     LMB_DLL_DeInit();
     return 0;
}
```

LMB_HWM_Get5Vsb

The **LMB_HWM_Get5Vsb** function reads the positive 5V standby power.

- **Syntax**

  int32_t LMB_HWM_Get5Vsb(float* pfVoltage);

- **Parameters**

  pfVoltage

  [in] obtains the current voltage of positive 5V standby power.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_Error:` This function failed.

  `ERR_NotSupport`:This platform does not support this function.

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{
int32_t iRet ;
float  fVolt =0;
     LMB_DLL_Init();
     iRet = LMB_HWM_Get5Vsb(&fVolt);
     if ( iRet == ERR_Success )
                printf("5V standby Voltage = %f\n”, fvolt);
     LMB_DLL_DeInit();
     return 0;
}
```

LMB_HWM_Get3V3sb

The **LMB_HWM_Get3Vsb** function reads the positive 3V/3.3V standby power.

- **Syntax**

  int32_t LMB_HWM_Get3V3sb(float* pfVoltage);

- **Parameters**

  pfVoltage

  [in] obtains the current voltage of positive 3.3V standby power.

- **Return Value**

  `ERR_Success:` This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_Error`: This function failed.

  `ERR_NotSupport`: This platform does not support this function.

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{
int32_t iRet ;
float  fVolt =0;
     LMB_DLL_Init();
     iRet = LMB_HWM_Get3V3sb(&fVolt);
     if ( iRet == ERR_Success )
                printf("3.3V standby Voltage = %f\n”, fVolt);
     LMB_DLL_DeInit();
     return 0;
}
```

LMB_HWM_GetVbat

The **LMB_HWM_GetVbat** function reads the battery voltage.

- **Syntax**

  int32_t LMB_HWM_GetVbat(float* pfVoltage);

- **Parameters**

  pfVoltage

  [in] obtains the current voltage of the battery.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_Error`: This function failed.

  `ERR_NotSupport`: This platform does not support this function.

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{
int32_t iRet ;
float  fVolt =0;
     LMB_DLL_Init();
     iRet = LMB_HWM_GetVbat(&fVolt);
     if ( iRet == ERR_Success )
                printf("Battery Voltage = %f\n”, fVolt);
     LMB_DLL_DeInit();
     return 0;
}
```

LMB_HWM_GetVDDR

The **LMB_HWM_GetVDDR** function reads the DDRAM voltage.

- **Syntax**

  int32_t LMB_HWM_GetVDDR(int8_t bChannel, float* pfVoltage);

- **Parameters**

  bChannel

  [out] assign DDR channel.

  pfVoltage

  [in] obtains the current voltage of the DDR channel.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_Error:` This function failed.

  `ERR_NotSupport`: This platform does not support this function.

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{
int32_t iRet ;
float  fVolt =0;
int8_t bCH = 1;
     LMB_DLL_Init();
     iRet = LMB_HWM_GetVDDR(bCH, &fVolt);
     if ( iRet == ERR_Success )
                 printf("VDDR-CH%d Voltage = %f\n”,bCH, fVolt);
     LMB_DLL_DeInit();
     return 0;
}
```

LMB_HWM_GetFanSpeed

The **LMB_HWM_GetFanSpeed** function reads the fan speed which was assigned.

- **Syntax**

  int32_t LMB_HWM_GetFanSpeed(uint8_t ubFanNo, uint16_t* puwRpm);

- **Parameters**

  ubFanNo

  [out] assigns the fan index number which is described in user manual.

  puwRpm

  [in] obtains the fan speed.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_Error`: This function failed.

  `ERR_NotSupport`:This platform does not support this function.

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{
int32_t iRet ;
uint16_t wRPM =0;
uint8_t fanno = 2;
     LMB_DLL_Init();
     iRet = LMB_HWM_GetCpuFan(fanno, &wRPM);
     if ( iRet == ERR_Success )
                printf("Fan%d Speed = %d\n”,fanno, wRPM);
     LMB_DLL_DeInit();
     return 0;
}
```

LMB_HWM_GetFanSpeedEx

The **LMB_HWM_GetFanSpeedEx** function reads the fan speed which was assigned.

- **Syntax**

  int32_t LMB_HWM_GetFanSpeed(uint8_t ubFanNo, uint16_t* puwRpm, uint8_t ubFanSeq);

- **Parameters**

  ubFanNo

  [out] assigns the fan index number which is described in user manual.

  puwRpm

  [in] obtains the fan speed.

  ubFanSeq

  [out] assigns the fan sub-index number which is described in user manual.

1	FAN_A
2	FAN_B

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_Error`: This function failed.

  `ERR_NotSupport`: This platform does not support this function.

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{
int32_t iRet ;
uint16_t wRPM =0;
uint8_t fanno = 2;
     LMB_DLL_Init();
     iRet = LMB_HWM_GetFanSpeedEx(fanno, &wRPM, FAN_A);
     if ( iRet == ERR_Success )
        printf("Fan%d%c Speed = %d\n”,fanno, (uint8_t)(FAN_A+0x40), wRPM);
     LMB_DLL_DeInit();
     return 0;
}
```

LMB_HWM_GetPowerSupply

The **LMB_HWM_GetPowerSupply** function reads the power supply AC input.

- **Syntax**

  int32_t LMB_HWM_GetPowerSupply(uint8_t ubPowerNo, uint16_t* puwVoltage);

- **Parameters**

  ubPowerNo

  [out] assigns the AC power supply number which is described in user manual.

  puwVoltage

  [in] obtains the AC power voltage.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_Error`: This function failed.

  `ERR_NotSupport`: This platform does not support this function.

- **Remarks**

  (None)

- **Example**

```
 #include <stdio.h>
 #include <stdlib.h>
 #include "lmbinc.h"
 int main(int argc, char* argv[])
 {
 int32_t iRet ;
 uint16_t wVolt =0;
 uint8_t pwrno = 2;
      LMB_DLL_Init();
      iRet = LMB_HWM_GetPowerSupply(pwrno, &wVolt);
      if ( iRet == ERR_Success )
                  printf("Power-%d Voltage = %d\n”,powno, wVolt);
      LMB_DLL_DeInit();
      return 0;
}
```

LMB_HWM_CaseOpenStatus

The **LMB_HWM_CaseOpenStatus** function gets the case cover status.

- **Syntax**

  int32_t LMB_HWM_CaseOpenStatus(uint8_t *pubStatus);

- **Parameters**

  pubStatus

  [in] indicates case cover status.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported.

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{
int32_t iRet;
uint8_t bStatus= 0;
    LMB_DLL_Init();
    iRet = LMB_HWM_CaseOpenStatus(&bStatus);
    if (iRet == ERR_Success)
                printf("Case cover status is %d\n", bStatus);
    LMB_DLL_DeInit();
    return 0;
}
```

LMB_HWM_IntrCallback

The **LMB_HWM_IntrCallback** function hooks some status change callback.

- **Syntax**

  int32_t LMB_HWM_IntrCallback(INTRUSION_CALLBACK pCallback, uint16_t uwmSec);

- **Parameters**

  pCallback

  [out] hooks callback function pointer for status changed event.

  uwmSec

  [out] sets period of time for checking status changed.

  [range: 10 ~ 999ms]

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_BusyInUses`: The callback function is running now.

  `ERR_Invalid`: The parameter is out of range.

- **Remarks**

  Supports case open, software reset button, power status changed and power supply fault event (refer to "Intrusion Item Define (bit)" in Chapter 2). When pCallback is NULL pointer, this function will be disabled.

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
void hwmCallback(INTRUSION_MSG pIntr){
       printf("item=%04X,status=%04X,\
       time = %04d/%02d/%02d %02d:%02d:%02d\n",\
       pInt.udwOccurItem, pIntr.udwStatus, pIntr.stuTime.year, \
       pIntr.stuTime.month, pIntr.stuTime.day, pIntr.stuTime.hour,\
       pIntr.stuTime.minute, pIntr.stuTime.second”);
}
int main(){
  LMB_DLL_Init();
  LMB_HWM_IntrCallback(hwmCallback, 100);
  While (1) {
  ……
       }
       LMB_DLL_DeInit();
       return 0;
}
```

LMB_HWM_ClearCaseOpenStatus

The **LMB_HWM_ClearCaseOpenStatus** function resets case open status.

- **Syntax**

  int32_t LMB_HWM_ClaerCaseOpenStatus(void);

- **Parameters**

  (None)

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported.

- **Remarks**

  1. Some platforms require clearing case open status for next event detection.

  2. If the case status remains open after cleared, this means the chassis is still open.

- **Example**

  (None)

LMB_HWM_GetSensorReport

The **LMB_HWM_GetSensorReport** function reads the sensor report message.

- **Syntax**

  int32_t LMB_HWM_GetSensorReport(int32_t dwSensorID, int8_t* pbString);

- **Parameters**

  dwSensorID

  [out] designates sensor index number

  pbSring

  [in] obtains the message of the designated.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_Error`: This function failed.

  `ERR_Invalid`: The input parameter is out of range.

  `ERR_NotSupport`:This platform does not support this function.

- **Remarks**

  Please refer to Constant and Macro Defin in Chapter 2.

  `“Hardware Monitor Item Define” typedef enum DEF_HWM_SENEORITEM`

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
#include "iolib.h"
int main(int argc, char* argv[])
{
int32_t iRet ;
int8_t strMsg[30], sid;
     LMB_DLL_Init();
     memset(strMsg, 0, 30);
     sid = HWMID_TEMP_CPU1;
     iRet = LMB_HWM_GetSensorReport(sid, strMsg);
     if (iRet==ERR_Success)printf("Sensor ID=%d, message is \"%s\"\n",
                              sid, strMsg);
     else         _printf_error_message("LMB_HWM_GetSensorReport", iRet);
     LMB_DLL_DeInit();
     return 0;
 }
```

LMB_HWM_GetSensorName

The **LMB_HWM_GetSensorName** function gets the sensor name.

- **Syntax**

  int32_t LMB_HWM_GetSensorName(int32_t dwSensorID, int8_t* pbString);

- **Parameters**

  dwSensorID

  [out] designates sensor index number

  pbSring

  [in] obtains the sensor name of the designated.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_Error`: This function failed.

  `ERR_Invalid`: The input parameter is out of range.

  `ERR_NotSupport`:This platform does not support this function.

- **Remarks**

  Please refer to Constant and Macro Defin in Chapter 2.

  `“Hardware Monitor Item Define” typedef enum DEF_HWM_SENEORITEM`

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
#include "iolib.h"
int main(int argc, char* argv[])
{
int32_t iRet ;
int8_t strMsg[30], sid;
     LMB_DLL_Init();
     memset(strMsg, 0, 30);
     sid = 0; //get Id=0 sensor name
     iRet = LMB_HWM_GetSensorName(sid, strMsg);
     if (iRet==ERR_Success)printf("Sensor ID=%d, name is \"%s\"\n",sid,
 strMsg);
     else         _printf_error_message("LMB_HWM_GetSensorName", iRet);
     LMB_DLL_DeInit();
     return 0;
}
```

Watch Dog Timer

LMB_WDT_QueryInfo

The **LMB_WDT_Queryinfo** function gets Watch Dog Timer information.

- **Syntax**

  int32_t LMB_WDT_QueryInfo(`WDT_INFO* pstuWdtInfo`);

- **Parameters**

  `pstuWdtInfo`

  `[in] structure pointer for obtaining WDT information`

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_NotSupport`: This platform does not support this function.

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{ int32_t iRet;
    WDT_INFO stuWdtInfo;
    LMB_DLL_Init();
    iRet = LMB_WDT_QueryInfo(&stuWdtInfo);
    if ( iRet == ERR_Success ) {
  printf("WDT type = %d\n", stuWdtInfo.ubType);
  printf("WDT maximum count = %d\n", stuWdtInfo.uwCountMax);
  if (stuWdtInfo.unMinuteSupport)
    printf("WDT support MUNUTE and SECOND time base\n");
  else
            printf("WDT support only support SECOND base\n");
    }
    LMB_DLL_DeInit();
return 0;
}
```

LMB_WDT_Config

The **LMB_WDT_Config** function configures the Watch Dog Timer.

- **Syntax**

  int32_t LMB_WDT_Config(uint16_t uwCount, uint8_t ubTimeBase);

- **Parameters**

  uwCount

  [out] the value sets the timer count down.

  ubTimeBase

  [out] the value selects time base.

1	Second base
2	Minute base

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_Invalid`: Invalid parameter value

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_NotSupport:` This platform does not support this function.

  `ERR_BusyInUse:` This step is skipped because WDT is already starting now.

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{
uint8_t iRet ;
   LMB_DLL_Init();
   iRet = LMB_WDT_Config(200, BASE_SECOND);
   if  ( iRet == ERR_Success ) {
       printf("WDT Ready: reset system after timeout.\n");
       LMB_WDT_Start(); //starting WDT
       while(1) {
       …… Main Loop Process ……
       LMB_WDT_Tick();
   }
   else {
      printf("WDT configure failure\n");
    }
    LMB_DLL_DeInit();
    return 0;
}
```

LMB_WDT_Start

LMB_WDT_Start function starts the WDT countdown.

- **Syntax**

  int32_t LMB_WDT_Start(void);

- **Parameters**

  (None)

- **Return Value**

  `ERR_Success:` This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_NotSupport`: This platform does not support this function.

- **Remarks**

  (None)

- **Example**

  Refer to function LMB_WDT_Config example.

LMB_WDT_Stop

The **LMB_WDT_Stop** function stops the WDT countdown.

- **Syntax**

  int32_t LMB_WDT_Stop(void);

- **Parameters**

  (None)

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet

  `ERR_NotSupport`: This platform does not support this function.

- **Remarks**

  (None)

- **Example**

  Refer to function LMB_WDT_Config example.

LMB_WDT_Tick

The **LMB_WDT_Tick** function reloads the timer and then re-computes it.

- **Syntax**

  int32_t LMB_WDT_Tick(void);

- **Parameters**

  (None)

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_NotSupport`: This platform does not support this function.

- **Remarks**

  (None)

- **Example**

  Refer to function LMB_WDT_Config example.

General Purpose Input Output

LMB_GPIO_GetInfo

The **LMB_GPIO_GetInfo** function gets the GPIO information supported by this platform.

- **Syntax**

  int32_t LMB_GPIO_GetInfo(uint8_t ubGroup, uint8_t* pubGpis, uint8_t* pubGpos);

- **Parameters**

  ubGroup

  [out] selects GPIO group, [range: 1 ~ 10]

  pubGpis

  [in] obtains GPI supports pins

  pubGpos

  [in] obtains GPO supports pins

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_Error`: This function failed.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_NotSupport`: This platform does not support GPIO direction setting.

- **Remarks**

  (None)

Example

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{
int32_t iRet;
uint8_t ubGPIs=0, ubGPOs=0;
  LMB_DLL_Init();
  /** pin 0/1 set to output mode, pin 2/3 set to input mode **/
  iRet = LMB_GPIO_GetInfo(0, &ubGPIs, &ubGPOs);
  if  ( iRet == ERR_Success )   {
    printf("GPIs = %d, GPOs = %d\n”, ubGPIs, ubGPOs);
  else
            printf("Functions error \n");
  LMB_DLL_DeInit();
  return 0;
}
```

LMB_GPIO_GpoWrite

The **LMB_GPIO_GpoWrite** function writes data to GPO group.

- **Syntax**

  int32_t LMB_GPIO_GpoWrite(uint8_t ubGroup, uint32_t udwValue);

- **Parameters**

  ubGroup

  [out] selects GPIO group, if there is only one group, this parameter will be ignored.

  udwValue

  [out] represents the value for DO writing.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_Error`: This function failed.

  `ERR_NotSupport`: This function is not supported.

- **Remarks**

  DO group is dependent on hardware definition, 32 pins is maximum for one group,

  - GPO group1: GPO_11 ~ GPO_18 or GPO_101 ~ GPO_132
  - GPO group2: GPO_21 ~ GPO_28
  - No group: GPO_1 ~ GPO_4

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[] {
int32_t iRet;
uint32_t dwSize;
    LMB_DLL_Init();
    /** set DO pin0~1 is '0', Do pin 2~3 is '1' **/
    iRet = LMB_GPIO_GpoWrite(1, 0x0C);
    if  ( iRet == ERR_Success )  printf("Wrote was successful\n");
    LMB_DLL_DeInit();
    return 0;
}
```

LMB_GPIO_GpoPinWrite

The **LMB_GPIO_GpoPinWrite** function writes data to a specific pin of GPO.

- **Syntax**

  int32_t LMB_GPIO_GpoPinWrite(uint8_t ubGroup, uint8_t ubPinNo, uint8_t ubValue);

- **Parameters**

  ubGroup

  [out] selects GPO group, if there is only one group, this parameter will be ignored.

  ubPinNo

  [out] assigns a pin of GPO to write data [range: 1~32]

  ubValue

  [out] the value for write [range: 0/1] (low/high)

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_Invalid`: The assigned DO pin does not exist.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_Error`: This function failed.

  `ERR_NotSupport`: This function is not supported.

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[]) {
     LMB_DLL_Init();
     /** set GPO_1/2 is '0', GPO_3/4 is '1' **/
     LMB_GPIO_GpoPinWrite(0, 1, 0);
     LMB_GPIO_GpoPinWrite(0, 2, 0);
     LMB_GPIO_GpoPinWrite(0, 3, 1);
     LMB_GPIO_GpoPinWrite(0, 4, 1);
     LMB_DLL_DeInit();
     return 0;
 }
```

LMB_GPIO_GpoRead

The **LMB_GPIO_GpoRead** function reads data of assigned GPO group.

- **Syntax**

  int32_t LMB_GPIO_GpoRead(uint8_t ubGroup, uint32_t* pudwValue);

- **Parameters**

  ubGroup

  [out] selects GPO group, if there is only one group, this parameter will be ignored.

  pudwValue

  [in] represents the value for read; it always reads '0' when the GPO pin is invalid or absent.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_Error`: This function failed.

  `ERR_NotSupport`: This function is not supported.

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[]) {
  int32_t iRet;
  uint32_t dwData=0;
      LMB_DLL_Init();
      iRet = LMB_GPIO_GpoRead(0, &dwData);
      if (iRet == ERR_Success) printf("GPO status is %08X\n",dwData);
      LMB_DLL_DeInit();
      return 0;
}
```

LMB_GPIO_GpoPinRead

The **LMB_GPIO_GpoPinRead** function reads data of assigned GPO pin.

- **Syntax**

  int32_t LMB_GPO_GpoPinRead(uint8_t ubGroup, uint8_t ubPinNo, uint8_t* pubValue);

- **Parameters**

  ubGroup

  [out] selects GPO group, if there is only one group, this parameter will be ignored.

  ubPinNo

  [out] assigns a pin of GPO to write data [range: 1~32]

  pubValue

  [in] the value for read [range: 0/1] (low/high)

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_Error`: This function failed.

  `ERR_NotSupport`: This function is not supported.

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{
int32_t iRet;
uint8_t pin=3, bData=0;
     LMB_DLL_Init();
     iRet = LMB_GPIO_GpoPinRead(0, pin &bData);
     if (iRet == ERR_Success)
             printf("GPO pin%d status is %d\n",pin, bData);
     LMB_DLL_DeInit();
     return 0;
 }
```

LMB_GPIO_GpiRead

The **LMB_GPIO_GpiRead** function reads data of assigned GPI group.

- **Syntax**

  int32_t LMB_GPIO_GpiRead(uint8_t ubGroup, uint32_t* pudwValue);

- **Parameters**

  ubGrou

  [out] select GPI group, if there is only one group, this parameter will be ignored.

  pudwValue

  [in] represents the value for read; it always reads '0' when the GPO pin is invalid or absent.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The DIO device is not opened yet.

  `ERR_Error`: This function failed.

  `ERR_NotSupport`: This function is not supported.

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{
 int32_t iRet;
 uint32_t dwData=0;
     LMB_DLL_Init();
    iRet = LMB_GPIO_GpiRead(0, &dwData);
     if (iRet == ERR_Success)
                 printf("GPI pins status is %08X\n",dwData);
     LMB_DLL_DeInit();
     return 0;
 }
```

LMB_GPIO_GpiPinRead

The **LMB_GPIO_GpiPinRead** function reads data of assigned GPI pin.

- **Syntax**

  int32_t LMB_GPIO_GpiPinRead(uint8_t ubGroup, uint8_t ubPinNo, uint8_t* pubValue);

- **Parameters**

  ubGroup

  [out] selects GPI group, if there is only one group, this parameter will be ignored.

  ubPinNo

  [out] assigns a pin of GPI to write data [range: 1~32]

  pubValue

  [in] the value for read [range: 0/1] (low/high)

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The DIO device is not opened yet.

  `ERR_Error`: This function failed.

  `ERR_NotSupport`: This function is not supported.

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{
int32_t iRet;
uint8_t pin=3, bData=0;
    LMB_DLL_Init();
    iRet = LMB_GPIO_GpiPinRead(0, pin &bData);
    if (iRet == ERR_Success)
                printf("GPI pin%d status is %d\n",pin, bData);
    LMB_DLL_DeInit();
    return 0;
}
```

LMB_GPIO_GpiCallback

The **LMB_GPIO_GpiCallback** function hooks GPI status changed callback.

- **Syntax**

  int32_t LMB_GPIO_GpiCallback(GPI_CALLBACK pCallback, uint16_t uwmSec);

- **Parameters**

  pCallback

  [out] hooks callback function pointer for GPI status changed event.

  uwmSec

  [out] sets period of time for checking status changed.

  [range: 10 ~ 999ms]

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_BusyInUses`: The callback function is running now.

  `ERR_Invalid`: The parameter is out of range.

- **Remarks**

  Currently supports case open, software reset button and changes of power status. When pCallback is NULL pointer, this function will be disabled.

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
void gpiCallback(GPI_MSG pGpiMsg){
     printf("Gpis=%02X,status=%02X,\
     time = %04d/%02d/%02d %02d:%02d:%02d\n",\
     pGpiMsg.udwGpis, pGpiMsg.udwStatus, pGpiMsg.stuTime.year, \
     pGpiMsg.stuTime.month, pGpiMsg.stuTime.day, \
     pGpiMsg.stuTime.hour, pGpiMsg.stuTime.minute, \
     pGpiMsg.stuTime.second”);
}
int main(){
  LMB_DLL_Init();
  LMB_GPIO_GpiCallback(gpiCallback, 250);
    While (1) {
    ……
    }
     LMB_DLL_DeInit();
     return 0;
}
```

System LED

LMB_SLED_SetSystemLED

The **LMB_SLED_SetSystemLED** function sets the system LED display mode.

- **Syntax**

  int32_t LMB_SLED_SetSystemLED(uint8_t ubLedMode);

- **Parameters**

  ubLedMode

  [out] sets the LED display mode. (range 0 ~ 3).

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_Invalid`: The parameter is out of range.

  `ERR_NotSupport`: This function is not supported.

- **Remarks**

  The LED colors is dependent on the platform hardware configuration.

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{
int32_t iRet;
uint8_t sled = 1;//mode1 -> light green LED
     LMB_DLL_Init();
     iRet = LMB_SLED_SetSystemLED(sled);
     if (iRet == ERR_Success) printf("Green LED is ready\n");
     LMB_DLL_DeInit();
     return 0;
 }
```

LMB_SLED_GetSystemLED

The **LMB_SLED_GetSystemLED** function gets the system LED display mode status.

- **Syntax**

  int32_t LMB_SLED_GetSystemLED(uint8_t *pubLedMode);

- **Parameters**

  pubLedMode

  [in] obtains system LED display mode.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported.

- **Remarks**

  The LED colors is dependent on the platform hardware configuration.

- **Example**

```
#include <stdio.h>
 #include <stdlib.h>
 #include "lmbinc.h"
 int main(int argc, char* argv[])
 {
 int32_t iRet;
 uint8_t sled= 0;
     LMB_DLL_Init();
     iRet = LMB_SLED_GetSysemLED(&sled);
     if (iRet == ERR_Success) printf("LED mode is %d\n", sled);
     LMB_DLL_DeInit();
     return 0;
 }
```

LAN Bypass Control

LMB_LBP_QueryDevices

The **LMB_LBP_QueryDevices** function gets platform installed controller devices.

- **Syntax**

  int32_t LMB_LBP_QueryDevices(uint16_t* puwSlotsDev);

- **Parameters**

  puwSlotsDev

  [in] obtains the installation status of devices connected to this platform.

  bit status 0 means *empty*, status 1 means *installed*

bit 0	Onboard LBP controller
bit 1	Slot1 LBP controller status
bit 2	Slot2 LBP controller status
bit 3	Slot3 LBP controller status
bit 4	Slot4 LBP controller status
bit 5	Slot5 LBP controller status
bit 6	Slot6 LBP controller status
bit 7	Slot7 LBP controller status
bit 8	Slot8 LBP controller status
bit 9 ~ bit 15 is not using now	

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported.

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(){
int32_t iRet;
uint16_t uwSlotsDev;
LMB_DLL_Init();
LMB_LBP_QueryDevices(&uwSlotsDev);
if ( iRet == ERR_Success ) {
  printf("Current hardware devices list:\n");
  if (uwSlotsDev & 0x0001 ) printf("    SLOT_ONBOARD\n");
  if (uwSlotsDev & 0x0002 ) printf("    SLOT_INDEX_1\n");
  if (uwSlotsDev & 0x0004 ) printf("    SLOT_INDEX_2\n");
  if (uwSlotsDev & 0x0008 ) printf("    SLOT_INDEX_3\n");
  if (uwSlotsDev & 0x0010 ) printf("    SLOT_INDEX_4\n");
  if (uwSlotsDev & 0x0020 ) printf("    SLOT_INDEX_5\n");
  if (uwSlotsDev & 0x0040 ) printf("    SLOT_INDEX_6\n");
  if (uwSlotsDev & 0x0080 ) printf("    SLOT_INDEX_7\n");
  if (uwSlotsDev & 0x0100 ) printf("    SLOT_INDEX_8\n");
}
    LMB_DLL_DeInit();
    return 0;
}
```

LMB_LBP_DeviceInfo

The **LMB_LBP_DeviceInfo** function gets the controller device information.

- **Syntax**

  int32_t LMB_LBP_DeviceInfo(uint8_t ubSlot, LBPDEV_INFO *pstuDevInfo);

- **Parameters**

  ubSlot

  [in] assigns the slot for device access.

  pstuDevInfo

  [in] structure pointer for obtaining controller device information.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported.

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(){
int32_t iRet;
LBPDEV_INFO LbpDevInfo;
LMB_DLL_Init();
iRet = LMB_LBP_DeviceInfo(xi, &LbpDevInfo);
if ( iRet == ERR_Success ) {
     printf("===== Lan Bypass Controller information =====\n");
     printf("Slot index is %d\n", LbpDevInfo.ubSlotIndex);
     if  ( LbpDevInfo.ubType == LAN_TYPE_COPPER )  printf("Controller Type is Copper\n");
      else if ( LbpDevInfo.ubType == LAN_TYPE_FIBER  )printf("Controller Type is Fiber\n");
     else               printf("Controller Type is unknown\n");
     printf("Version: %d.%d\n",LbpDevInfo.ubVersion[0], LbpDevInfo.ubVersion[1]);
     printf("  Modules: %02X\n",LbpDevInfo.ubModules);
     printf("  SystemOff Pairs: %02X\n",LbpDevInfo.ubSystemOff_Pairs);
     printf("  JustOn Pairs: %02X\n",LbpDevInfo.ubJustOn_Pairs);
     printf("  Runtime Pairs: %02X\n",LbpDevInfo.ubRuntime_Pairs);
     printf("  Timer1 MaxSec: %d seconds\n",LbpDevInfo.uwTimer1_MaxSec);
     printf("  Timer2 MaxSec: %d seconds\n",LbpDevInfo.uwTimer2_MaxSec);
     printf("  Timer3 MaxSec: %d seconds\n",LbpDevInfo.uwTimer3_MaxSec);
}
LMB_DLL_DeInit();
return 0;
}
```

LMB_LBP_FactoryReset

The **LMB_LBP_FactoryReset** function clears setting of the device configuration.

- **Syntax**

  int32_t LMB_LBP_FactoryReset(uint8_t ubSlot);

- **Parameters**

  ubSlot

  [in] assigns the slot for device access.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported.

- **Remarks**

  This parameter will reset all configurations to factory default settings.

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(){
int32_t iRet;
LMB_DLL_Init();
LMB_LBP_FactoryReset();
if ( iRet == ERR_Success ) {
  printf("Lan Bypass Control was set to factory default\n");
}
LMB_DLL_DeInit();
return 0;
}
```

LMB_LBP_SaveConfig

The **LMB_LBP_SaveConfig** function saves the configuration to NVRAM as default.

- **Syntax**

  int32_t LMB_LBP_SaveConfig(uint8_t ubSlot);

- **Parameters**

  ubSlot

  [in] assigns the slot for device access.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported.

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(){
int32_t iRet;
LMB_DLL_Init();
LMB_LBP_SaveConfig();
if ( iRet == ERR_Success ) {
  printf("Lan Bypass Control was saved to NVRAM successful\n");
}
    LMB_DLL_DeInit();
    return 0;
}
```

LMB_LBP_GetPairsStatus

The **LMB_LBP_GetPairsStatus** function gets all pairs and ports current status.

- **Syntax**

  int32_t LMB_LBP_GetPairsStatus(uint8_t ubSlot, PAIRS_STATUS *pstuPairsStatus);

- **Parameters**

  ubSlot

  [in] assigns the slot for device access.

  pstuPairsStatus

  [in] a pointer of PAIRS_STATUS structure

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported.

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(){
int32_t iRet;
PAIRS_STATUS stuPairsStatus
LMB_DLL_Init();
iRet =  LMB_LBP_GetPairsStatus(1, &PairsStatus);
if ( iRet == ERR_Success )  {Weilong
  printf("===========GetPairsStatus==============\n");
  printf("Slot index is %d\n", PairsStatus.ubSlotIndex);
  printf("SystemOff_PairsBypass: %02X\n",
         PairsStatus.ubSystemOff_PairsBypass);
  printf("JustOn_PairsBypass: %02X\n",
                        PairsStatus.ubJustOn_PairsBypass);
  printf("Runtime_PairsBypass: %02X\n",
                        PairsStatus.ubRuntime_PairsBypass);
  printf("JustOn_PortsDisconnect: %02X\n",
                        PairsStatus.ubJustOn_PortsDisconnect);
  printf("ubRuntime_PortsDisconnect: %02X\n",
                        PairsStatus.ubRuntime_PortsDisconnect);
}
    LMB_DLL_DeInit();
    return 0;
}
```

LMB_LBP_SetPairBypass

The **LMB_LBP_SetPairBypass** function sets the LAN bypass pair new status.

- **Syntax**

  int32_t LMB_LBP_SetPairBypass(uint8_t ubSlot, uint8_t ubPairNo, uint8_t ubEnable);

- **Parameters**

  ubSlot

  [out] assigns the slot for device access.

  ubPairNo

  [out] assigns a pair to set bypass function.

  ubEnable

  [out] enables or disables bypass function.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported.

- **Remarks**

  This function will update pair status immediately in Runtime stage.

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(){
int32_t iRet;
LMB_DLL_Init();
iRet =  LMB_LBP_SetPairBypass(1, 1, ENABLE);
if ( iRet == ERR_Success )  {
  printf("Set Slot 1, Pair 1, Lan Bypass Enable\n");
}
    LMB_DLL_DeInit();
    return 0;
}
```

LMB_LBP_SetAllPairs

The **LMB_LBP_SetAllPairs** function sets the all pairs new status.

- **Syntax**

  int32_t LMB_LBP_SetAllPairs(uint8_t ubSlot, uint8_t ubStatus);

- **Parameters**

  ubSlot

  [out] assigns the slot for device access.

  ubStatus

  [out] new status for all pairs.

- **Return Value**

  `ERR_Success`: This function is working properly.

  ERR_NotOpened: The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported.

- **Remarks**

  This function will update all pair status immediately in Runtime stage.

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(){
int32_t iRet;
uint8_t ubStatus = 0x03;  //pari1 & pair2 bypass enable
LMB_DLL_Init();
iRet =  LMB_LBP_SetAllPairs(1, ubStatus);
if ( iRet == ERR_Success )  {
  printf("Set Slot 1, Pair 1&2, Lan Bypass Enable\n");
  printf("Set Slot 1, Pair 3&4, Lan Bypass Disable\n");
}
    LMB_DLL_DeInit();
    return 0;
}
```

LMB_LBP_SetJustOnPairs

The **LMB_LBP_SetJustOnPairs** function sets the all pairs new status after Just-On stat.

- **Syntax**

  int32_t LMB_LBP_SetJustOnPairs(uint8_t ubSlot, uint8_t ubStatus);

- **Parameters**

  ubSlot

  [out] assigns the slot for device access.

  ubStatus

  [out] new status for all pairs.

- **Return Value**

  `ERR_Success`: This function is working properly.

  ERR_NotOpened: The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported.

- **Remarks**

  This function will update all pair status immediately in Runtime stage.

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(){
int32_t iRet;
uint8_t ubStatus = 0x03;  //pari1 & pair2 bypass enable
LMB_DLL_Init();
iRet =  LMB_LBP_SetJustOnPairs(1, ubStatus);
if ( iRet == ERR_Success )  {
  printf("Set Slot 1, Pair 1&2, Just-On stat Lan Bypass Enable\n");
  printf("Set Slot 1, Pair 3&4, Just-On stat Lan Bypass Disable\n");
}
    LMB_DLL_DeInit();
    return 0;
}
```

LMB_LBP_SetSystemOffPairs

The **LMB_LBP_SetSystemOffPairs** function sets the all pairs new status when system-off stat

- **Syntax**

  int32_t LMB_LBP_SetSystemOffPairs(uint8_t ubSlot, uint8_t ubStatus);

- **Parameters**

  ubSlot

  [out] assigns the slot for device access.

  ubStatus

  [out] new status for all pairs.

- **Return Value**

  `ERR_Success`: This function is working properly.

  ERR_NotOpened: The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported.

- **Remarks**

  This function will update all pair status immediately in Runtime stage.

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(){
int32_t iRet;
uint8_t ubStatus = 0x03;  //pari1 & pair2 bypass enable
LMB_DLL_Init();
iRet =  LMB_LBP_SetSystemOffPairs(1, ubStatus);
if ( iRet == ERR_Success )  {
  printf("Set Slot 1, Pair 1&2, System-Off stat Lan Bypass Enable\n");
  printf("Set Slot 1, Pair 3&4, System-Off stat Lan Bypass Disable\n");
}
    LMB_DLL_DeInit();
    return 0;
}
```

LMB_LBP_SetPortDisconnect

The **LMB_LBP_SetPortDisconnect** function sets the fiber port disconnect status.

- **Syntax**

  int32_t LMB_LBP_SetPortDisconnect(uint8_t ubSlot, uint8_t ubPortNo, uint8_t ubEnable);

- **Parameters**

  ubSlot

  [in] assigns the slot for device access.

  ubPortNo

  [out] assigns a port to set disconnect function.

  ubEnable

  [out] enables or disables fiber port disconnection.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported.

- **Remarks**

  This function will update fiber port connection status immediately in Runtime stage.

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(){
int32_t iRet;
uint8_t ubStatus = 0x03;  //pari1 & pair2 bypass enable
LMB_DLL_Init();
iRet =  LMB_LBP_SetPortDisconnect(2, 1, ENABLE );
if ( iRet == ERR_Success )  {
  printf("Set Slot 2, Fiber port 1, Lan Disconnect Enable\n");     }
    LMB_DLL_DeInit();
    return 0;
}
```

LMB_LBP_SetAllPortsDisconnect

The **LMB_LBP_SetAllPortsDisconnect** function sets all fiber port disconnected.

- **Syntax**

  int32_t LMB_LBP_SetAllPortsDisconnect(uint8_t ubSlot, uint8_t ubStatus);

- **Parameters**

  ubSlot

  [in] assigns the slot for device access

  ubStatus

  [out] assigns new status to set all fiber ports disconnected.

- **Return Value**

  `ERR_Success`: This function is working properly.

  ERR_NotOpened: The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported.

- **Remarks**

  This function will update fiber port connection status immediately in Runtime stage.

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(){
int32_t iRet;
uint8_t ubStatus = 0x0C;  //port3 port4 disconnect enable
LMB_DLL_Init();
iRet =  LMB_LBP_SetAllPortsDisconnect(2, ubStatus);
if ( iRet == ERR_Success )  {
  printf("Set Slot 2, Fiber port 3 & 4, Lan Disconnect Enable\n");  }
    LMB_DLL_DeInit();
    return 0;
}
```

LMB_LBP_TimerConfig

The **LMB_LBP_TimerConfig** function configures Timer count and timeout action.

- **Syntax**

  int32_t LMB_LBP_TimerConfig(uint8_t ubSlot, LBP_TIMERCFG stuLbpTimerCfg);

- **Parameters**

  ubSlot

  [in] assigns the slot for device access.

  stuLbpTimerCfg

  [out] structure for configuring LAN bypass timer

- **Return Value**

  `ERR_Success`: This function is working properly.

  ERR_NotOpened: The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported.

- **Remarks**

  1. If timer second value is larger than the allowed maximum value, the allowed maximum value will be used.
  2. Timer3 counts step by 5 seconds, which means timer count will ignore the remainder that cannot be divided by 5 before setting.

Example

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(){
int32_t iRet;
uint8_t bSlot=1;
LMB_DLL_Init();
printf("===========LMB_LBP_TimerConfig==============\n");
stuTimerCfg.ubTimerNo = 1;
stuTimerCfg.uwTimeSec = 10;
stuTimerCfg.ubAction = PAIR_BYPASS_ENABLE;
stuTimerCfg.ubEffect = 0x08; //effect pair
printf("Slot=%d, Timer=%d, Second=%d, Action=%d, Pair/Port Effect %02X\n",
          bSlot, stuTimerCfg.ubTimerNo, stuTimerCfg.uwTimeSec,
          stuTimerCfg.ubAction, stuTimerCfg.ubEffect);
iRet = LMB_LBP_TimerConfig(bSlot, stuTimerCfg);
if ( iRet == ERR_Success ) {
     printf("---> set Slot1 Time1 10 seconds timeout pair4 bypass enable\n");
     LMB_LBP_TimerStart(bSlot, stuTimerCfg.ubTimerNo);
     printf("Wait 5 seconds....... tick 1\n");   sleep(5);
     LMB_LBP_TimerTick(bSlot, stuTimerCfg.ubTimerNo);
     printf("Wait 5 seconds....... tick 2\n");   sleep(5);
     LMB_LBP_TimerTick(bSlot, stuTimerCfg.ubTimerNo);
     printf("----> stop timer\n");
     LMB_LBP_TimerStop(bSlot, stuTimerCfg.ubTimerNo);
     printf("hit <Enter> to start timer !!!\n");   getchar();
     LMB_LBP_TimerStart(bSlot, stuTimerCfg.ubTimerNo);
     printf("Wait 12 seconds....... wait timeout\n");
   sleep(12);
}
else
  printf("Error ===> pairs setting may be not exist\n");
LMB_DLL_DeInit();
return 0;
}
```

LMB_LBP_TimerStart

The **LMB_LBP_TimerStart** starts the controller timer function.

- **Syntax**

  int32_t LMB_LBP_TimerStart(uint8_t ubSlot, uint8_t ubTimerNo);

- **Parameters**

ubSlot

[in] assigns the slot for device access.

ubTimerNo

[out] assigns timer of LAN bypass controller.

- **Return Value**

`ERR_Success`: This function is working properly.

ERR_NotOpened: The library is not ready or opened yet.

`ERR_NotSupport`: This function is not supported by this platform.

- **Remarks**

(None)

- **Example**

Refer to function LMB_LBP_TimerrConfig example.

LMB_LBP_TimerStop

The **LMB_LBP_TimerStop** stops the controller timer function.

- **Syntax**

  int32_t LMB_LBP_TimerStop(uint8_t ubSlot, uint8_t ubTimerNo);

- **Parameters**

ubSlot

[in] assigns the slot for device access.

ubTimerNo

[out] assigns timer of LAN bypass controller.

- **Return Value**

`ERR_Success`: This function is working properly.

ERR_NotOpened: The library is not ready or opened yet.

`ERR_NotSupport`: This function is not supported by this platform.

- **Remarks**

(None)

- **Example**

Refer to function LMB_LBP_TimerrConfig example.

LMB_LBP_TimerTick

The **LMB_LBP_TimerTick** function reloads the timer then re-computes it.

- **Syntax**

  int32_t LMB_LBP_TimerTick(uint8_t ubSlot, uint8_t ubTimerNo);

- **Parameters**

  ubSlot

  [in] assigns the slot for device access.

  ubTimerNo

  [out] assigns timer of LAN bypass controller.

- **Return Value**

  `ERR_Success`: This function is working properly.

  ERR_NotOpened: The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported by this platform.

- **Remarks**

  (None)

- **Example**

  Refer to function LMB_LBP_TimerrConfig example.

MB_LBP_TimerStatus

The **LMB_LBP_TimerStatus** function gets the Timer current status.

- **Syntax**

  int32_t LMB_LBP_TimerStatus(uint8_t ubSlot, uint8_t ubTimerNo, uint8_t *pubStatus);

- **Parameters**

  ubSlot

  [in] assigns the slot for device access.

  ubTimerNo

  [out] assigns timer of LAN bypass controller.

  pubStatus

  [in] obtains current timer status.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported by this platform.

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(){
int32_t iRet;
uint8_t ubStatus;
LMB_DLL_Init();
iRet =  LMB_LBP_TimerStatus(2, 1, &ubStatus);
if ( iRet == ERR_Success )  {
  if (ubStatus ==0 )    printf("Slot2, Timer1, Stopped\n");
  else if (ubStatus==1 ) printf("Slot2, Timer1, Running\n");
  else if (ubStatus==2 ) printf("Slot2, Timer1, Expired\n");
  else          printf("Unknown status\n");
    }
    LMB_DLL_DeInit();
    return 0;
}
```

Serial EEPROM

LMB_EEP_QueryDevices

The **LMB_EEP_QueryDevices** function gets controller devices installed on this platform.

- **Syntax**

  int32_t LMB_EEP_QueryDevices(uint16_t* puwSlotsDev);

- **Parameters**

  puwSlotsDev

  [in] obtains the status of devices installed on this platform.

- **Return Value**

  `ERR_Success`: This function is working properly.

  ERR_NotOpened: The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported by this platform.

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(){
int32_t iRet;
uint16_t uwSlotsDev;
LMB_DLL_Init();
LMB_EEP_QueryDevices(&uwSlotsDev);
if ( iRet == ERR_Success ) {
  printf("Current EEPROM devices list:\n");
  if (uwSlotsDev & 0x0001 ) printf("    SLOT_ONBOARD\n");
  if (uwSlotsDev & 0x0002 ) printf("    SLOT_INDEX_1\n");
  if (uwSlotsDev & 0x0004 ) printf("    SLOT_INDEX_2\n");
  if (uwSlotsDev & 0x0008 ) printf("    SLOT_INDEX_3\n");
  if (uwSlotsDev & 0x0010 ) printf("    SLOT_INDEX_4\n");
  if (uwSlotsDev & 0x0020 ) printf("    SLOT_INDEX_5\n");
  if (uwSlotsDev & 0x0040 ) printf("    SLOT_INDEX_6\n");
  if (uwSlotsDev & 0x0080 ) printf("    SLOT_INDEX_7\n");
  if (uwSlotsDev & 0x0100 ) printf("    SLOT_INDEX_8\n");
}
    LMB_DLL_DeInit();
    return 0;
}
```

LMB_EEP_WriteByte

The **LMB_EEP_WriteByte** function writes byte data to EEPROM device.

- **Syntax**

  int32_t LMB_EEP_WriteByte(uint8_t ubDeviceNo, uint32_t udwAddr, uint8_t ubData);

- **Parameters**

  ubDeviceNo

  [out] assigns the Serial EEPROM device.

  udwAddr

  [out] assigns the location to write.

  ubData

  [out] byte data for writing

- **Return Value**

  `ERR_Success`: This function is working properly.

  ERR_NotOpened: The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported.

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(){
int32_t iRet;
uint8_t aryData[0x20];
LMB_DLL_Init();
    //erase onboard eeprom
    LMB_EEP_Erase(0);
    //Write data
    printf("Write addr.0x00,data 0x41\n");
   LMB_EEP_WriteByte(0, 0, 0x41);
    printf("Write addr.0x04,data 0x3132\n");
   LMB_EEP_WriteWord(0, 0x04, 0x3132);
    printf("Write addr.0x08, data 0x61626364\n");
   LMB_EEP_WriteDWord(0, 0x08, 0x61626364);
    printf("Write addr.0x10, string AbCdEfgHiJk\n")
LMB_EEP_WriteBlock(0, 0x10, 11, "AbCdEfgHiJk" );
//Read data
   LMB_EEP_ReadByte(0, 0, &aryData[0]);
   printf("Read Addr.0x00,data=%02X\n", aryData[0]);
   LMB_EEP_ReadWord(0, 4, (uint16_t*)&aryData[4]);
   printf("Read Addr.0x04,data=%04X\n", *(uint16_t*)&aryData[4]);
    LMB_EEP_ReadDWord(0, 0x08, (uint32_t*)&aryData[8]);
   printf("Read Addr.0x08,data=%08X\n",*(uint32_t*)&aryData[8]);
   LMB_EEP_ReadBlock(0, 0x10, 11,  &aryData[0x10]);
   printf("Read Addr.0x10,data=%s\n", &aryData[0x10]);
   LMB_DLL_DeInit();
   return 0;
}
```

LMB_EEP_WriteWord

The **LMB_EEP_WriteWord** function writes word data to EEPROM device.

- **Syntax**

  int32_t LMB_EEP_WriteWord(uint8_t ubDeviceNo, uint32_t udwAddr, uint16_t uwData);

- **Parameters**

  ubDeviceNo

  [out] assigns the Serial EEPROM device.

  udwAddr

  [out] assigns the location to write.

  uwData

  [out] word data for writing

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported.

- **Remarks**

  (None)

- **Example**

  Refer to function LMB_EEP_WriteByte example.

LMB_EEP_WriteDWord

The **LMB_EEP_WriteDWord** function writes dword data to EEPROM device.

- **Syntax**

  int32_t LMB_EEP_WriteDWord(uint8_t ubDeviceNo, uint32_t udwAddr, uint32_t udwData);

- **Parameters**

  ubDeviceNo

  [out] assigns the Serial EEPROM device.

  udwAddr

  [out] assigns the location to write.

  udwData

  [out] doubles word data for writing.

- **Return Value**

  `ERR_Success`: This function is working properly.

  ERR_NotOpened: The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported.

- **Remarks**

  (None)

- **Example**

  Refer to function LMB_LBP_TimerrConfig example.

LMB_EEP_WriteBlock

The **LMB_EEP_WriteBlock** function writes block data to EEPROM device.

- **Syntax**

  int32_t LMB_EEP_WriteBlock(uint8_t ubDeviceNo, uint16_t udwAddr, uint16_t uwLength, uint8_t* pubBlock);

- **Parameters**

  ubDeviceNo

  [out] assigns the Serial EEPROM device.

  udwAddr

  [out] assigns the location to write.

  uwLength

  [out] doubles word data for writing.

  pubBlock

  [out] byte block data for writing.

- **Return Value**

  `ERR_Success`: This function is working properly.

  ERR_NotOpened: The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported.

- **Remarks**

  (None)

- **Example**

  Refer to function LMB_EEP_WriteByte example.

LMB_EEP_ReadByte

The **LMB_EEP_ReadByte** function writes byte data from EEPROM device.

- **Syntax**

  int32_t LMB_EEP_ReadByte(uint8_t ubDeviceNo, uint32_t udwAddr, uint8_t* pubData);

- **Parameters**

  ubDeviceNo

  [out] assigns the Serial EEPROM device.

  udwAddr

  [out] assigns the location to write.

  pubData

  [in] obtains byte data for reading.

- **Return Value**

  `ERR_Success`: This function is working properly.

  ERR_NotOpened: The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported.

- **Remarks**

  (None)

- **Example**

  Refer to function LMB_EEP_WriteByte example.

LMB_EEP_ReadWord

The **LMB_EEP_ReadWord** function reads word data from EEPROM device.

- **Syntax**

  int32_t LMB_EEP_ReadWord(uint8_t ubDeviceNo, uint32_t udwAddr, uint16_t* puwData);

- **Parameters**

  ubDeviceNo

  [out] assigns the Serial EEPROM device.

  udwAddr

  [out] assigns the location to write.

  puwData

  [in] obtains word data for reading.

- **Return Value**

  `ERR_Success`: This function is working properly.

  ERR_NotOpened: The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported.

- **Remarks**

  (None)

- **Example**

  Refer to function LMB_EEP_WriteByte example.

LMB_EEP_ReadDWord

The **LMB_EEP_ReadDWord** function reads dword data from EEPROM device.

- **Syntax**

  int32_t LMB_EEP_ReadDWord(uint8_t ubDeviceNo, uint32_t udwAddr, uint32_t* pudwData);

- **Parameters**

  ubDeviceNo

  [out] assigns the Serial EEPROM device.

  udwAddr

  [out] assigns the location to write.

  pudwData

  [in] obtains dword data for reading.

- **Return Value**

  `ERR_Success`: This function is working properly.

  ERR_NotOpened: The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported.

- **Remarks**

  (None)

- **Example**

  Refer to function LMB_EEP_WriteByte example.

LMB_EEP_ReadBlock

The **LMB_EEP_ReadBlock** function reads block data from EEPROM device.

- **Syntax**

  int32_t LMB_EEP_ReadBlock(uint8_t ubDeviceNo, uint32_t udwAddr, uint16_t uwLength, uint8_t* pubBlock);

- **Parameters**

  ubDeviceNo

  [out] assigns the Serial EEPROM device.

  udwAddr

  [out] assigns the location to write.

  uwLength

  [out] obtains dword data for reading..

  pubBlock

  [in] obtains byte block data for reading.

- **Return Value**

  `ERR_Success`: This function is working properly.

  ERR_NotOpened: The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported.

- **Remarks**

  (None)

- **Example**

  Refer to function LMB_EEP_WriteByte example.

LMB_EEP_Erase

The **LMB_EEP_Erase** function erases EPROM device.

- **Syntax**

  int32_t LMB_EEP_Erase(uint8_t ubDeviceNo);

- **Parameters**

  ubDeviceNo

  [out] assigns the Serial EEPROM device.

- **Return Value**

  `ERR_Success`: This function is working properly.

  ERR_NotOpened: The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported.

- **Remarks**

  Erase EEPROM all data to 0x00 value

- **Example**

  Refer to function LMB_EEP_WriteByte example.

LCD Module

LMB_LCM_DeviceOpen

The **LMB_LCM_DeviceOpen** function opens and connects LCD module.

- **Syntax**

  int32_t LMB_LCM_DeviceOpen(void);

- **Parameters**

  (None)

- **Return Value**

  `ERR_Success`: This function is working properly.

  ERR_NotOpened: The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported.

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(){
int32_t iRet;
uint8_t aryData[0x20];
  LMB_DLL_Init();
  iRet = LMB_LCM_DeviceOpen();
  if ( iRet != ERR_Success ) {
      printf("LCD Module not exist\n");
      return -1;
  }
  … … … … … …
  LMB_LCM_DeviceClose();
  LMB_DLL_DeInit();
  return 0;
}
```

LMB_LCM_DeviceClose

The **LMB_LCM_DeviceClose** function closes and disconnects LCD module.

- **Syntax**

  int32_t LMB_LCM_DeviceClode(void);

- **Parameters**

  (None)

- **Return Value**

  `ERR_Success`: This function is working properly.

  ERR_NotOpened: The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported.

- **Remarks**

  (None)

- **Example**

  Refer to function LMB_EEP_WriteByte example.

LMB_LCM_DeviceInfo

The **LMB_LCM_DeviceInfo** function gets platform LCM device information.

- **Syntax**

  int32_t LMB_LCM_DeviceInfo(LCM_INFO* pstuLcmInfo);

- **Parameters**

  pstuLcmInfo

  [in] obtains this LCM device information.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported.

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(){
int32_t iRet;
LCM_INFO stuLcmInfo;
    LMB_DLL_Init();
    LMB_LCM_DeviceOpen();
    iRet = LMB_LCM_DeviceInfo(&stuLcmInfo);
    if (iRet == ERR_Success ){
        printf("LCM Mode No. is %04X\n", stuLcmInfo.uwModeNo);
        printf("LCM Firmware Ver. is %04X\n", stuLcmInfo.uwVersion);
        printf("LCM Speed =%d\n", stuLcmInfo.udwBaudrate);
     }
    LMB_LCM_DeviceClose();
    LMB_DLL_DeInit();
    return 0;
}
```

LMB_LCM_LightCtrl

The **LMB_LCM_LightCtrl** function sets LCD module light on/off status.

- **Syntax**

  int32_t LMB_LCM_LightCtrl(uint8_t ubOnOff);

- **Parameters**

  ubOnOff

  [out] controls the LCM light status. [range: 0/1] (Off/On)

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported.

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(){
int32_t iRet;
LCM_INFO stuLcmInfo;
    LMB_DLL_Init();
    LMB_LCM_DeviceOpen();
    iRet = LMB_LCM_LightCtrl(DISABLE);
    if (iRet == ERR_Success ) printf("LCM Light Off\n");
    sleep(2);//delay 2 seconds
    iRet = LMB_LCM_LightCtrl(ENABLE);
    if (iRet == ERR_Success ) printf("LCM Light On\n");
    LMB_LCM_DeviceClose();
    LMB_DLL_DeInit();
    return 0;
}
```

LMB_LCM_SetCursor

The **LMB_LCM_SetCursor** function writes string to LCD module.

- **Syntax**

  int32_t LMB_LCM_SetCursor(uint8_t ubColumn, uint8_t ubRow);

- **Parameters**.

  ubColumn

  [out] assigns column value of LCM display cursor.

  ubRow

  [out] assigns row value of LCM display cursor.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported.

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(){
int32_t iRet;
    LMB_DLL_Init();
     LMB_LCM_DeviceOpen();
     iRet = LMB_LCM_SetCursor(5,1); //column 5, row 1
     if (iRet == ERR_Success )
         printf("LCM set cursor to column 5, row 1\n");
    LMB_LCM_DeviceClose();
     LMB_DLL_DeInit();
     return 0;
}
```

LMB_LCM_WriteString

The **LMB_LCM_WriteString** function writes string to LCD module.

- **Syntax**

  int32_t LMB_LCM_WriteString(int8_t* pstrubString);

- **Parameters**

  pstrubString

  [out] message string to LCM display.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported.

- **Remarks**

  Please refer to Appendix: LCD Module Character Table

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(){
int32_t iRet;
    LMB_DLL_Init();
     LMB_LCM_DeviceOpen();
     iRet = LMB_LCM_WriteString("123ABCD");
     if (iRet == ERR_Success )
         printf("Write string to LCM is successful\n");
    LMB_LCM_DeviceClose();
     LMB_DLL_DeInit();
     return 0;
}
```

LMB_LCM_DisplayClear

The **LMB_LCM_DisplayClear** function clears LCM display.

- **Syntax**

  int32_t LMB_LCM_DisplayClear(void);

- **Parameters**

  (None)

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported.

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
 int main(){
int32_t iRet;
    LMB_DLL_Init();
    LMB_LCM_DeviceOpen();
    iRet = LMB_LCM_DisplayClear();
    if (iRet == ERR_Success ) printf("Clear display successful\n");
    LMB_LCM_DeviceClose();
    LMB_DLL_DeInit();
    return 0;
}
```

LMB_LCM_Brightness

The **LMB_LCM_Brightness** function sets the brightness of LCD module.

- **Syntax**

  int32_t LMB_LCM_Brightness(uint8_t ubBrightness);

- **Parameters**

  ubBrightness

  [out] sets brightness value of LCM. [range 1 ~ 8]

  When the value is out of range, it will be automatically set to maximum or minimum value.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported.

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
 int main(){
int32_t iRet;
    LMB_DLL_Init();
     LMB_LCM_DeviceOpen();
     iRet = LMB_LCM_Brightness(1);
     if (iRet == ERR_Success ) printf("Set brightness step 8\n");
     iRet = LMB_LCM_Brightness(8);
     if (iRet == ERR_Success ) printf("Set brightness step 1\n");
    LMB_LCM_DeviceClose();
     LMB_DLL_DeInit();
     return 0;
}
```

LMB_LCM_Reset

The **LMB_LCM_Reset** function resets the LCD module.

- **Syntax**

  int32_t LMB_LCM_Reset(void);

- **Parameters**

  (None)

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported.

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(){
int32_t iRet;
    LMB_DLL_Init();
     LMB_LCM_DeviceOpen();
     iRet = LMB_LCM_Reset();
     if (iRet == ERR_Success ) printf("Soft Reset LCM\n");
    LMB_LCM_DeviceClose();
     LMB_DLL_DeInit();
     return 0;
}
```

LMB_LCM_SetSpeed

The **LMB_LCM_SetSpeed** function sets communication baud rate of LCM.

- **Syntax**

  int32_t LMB_LCM_SetSpeed(uint32_t udwBaudrate);

- **Parameters**

  udwBaudrate

  [out] sets communication baudrate of LCM.

  Supports speed: 115200, 57600, 38400, 19200, 9600

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported.

- **Remarks**

  After connection speed is changed, the application needs to close LCM device and then re-open LCM device again.

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
 int main(){
int32_t iRet;
LMB_DLL_Init();
     LMB_LCM_DeviceOpen();
     iRet = LMB_LCM_SetSpeed(115200);
     if (iRet == ERR_Success ) printf("set LCM speed is 115200\n");
    LMB_LCM_DeviceClose();
     LMB_DLL_DeInit();
     return 0;
 }
```

LMB_LCM_WrapCtrl

The **LMB_LCM_WrapCtrl** function sets LCM line wrap format.

- **Syntax**

  int32_t LMB_LCM_WrapCtrl(uint8_t ubOnOff);

- **Parameters**

  ubOnOff

  [out] controls wrap's on/off status. [range: 0/1] (Off/On)

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported.

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(){
int32_t iRet;
    LMB_DLL_Init();
     LMB_LCM_DeviceOpen();
     iRet = LMB_LCM_WrapCtrl(DISABLE);
     if (iRet == ERR_Success ) printf("Line wrap is off\n");
     iRet = LMB_LCM_WrapCtrl(ENABLE);
     if (iRet == ERR_Success ) printf("Line wrap is on\n");
    LMB_LCM_DeviceClose();
     LMB_DLL_DeInit();
     return 0;
}
```

LMB_LCM_CursorModeCtrl

The **LMB_LCM_CursorModeCtrl** function sets LCM cursor format.

- **Syntax**

  int32_t LMB_LCM_CursorModeCtrl(uint8_t ubCursorMode);

- **Parameters**

  ubCursorMode

  [out] control cursor display mode.

0	Off
1	Underline
2	Blinking Block

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported.

- **Remarks**

  Setting cursor display type will automatically restore the cursor to HOME(1,1) position.

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
 int main(){
int32_t iRet;
    LMB_DLL_Init();
     LMB_LCM_DeviceOpen();
     iRet = LMB_LCM_CursorModeCtrl(2);
     if (iRet == ERR_Success ) printf("Cursor is blinking & block\n");
     iRet = LMB_LCM_CursorModeCtrl(1);
     if (iRet == ERR_Success ) printf("Cursor is underline\n");
    LMB_LCM_DeviceClose();
     LMB_DLL_DeInit();
     return 0;
}
```

LMB_LCM_KeysStatus

The **LMB_LCM_KeysStatus** function reads LCM key status.

- **Syntax**

  int32_t LMB_LCM_KeysStatus(uint8_t * pubKeys);

- **Parameters**

  pubKeys

  [in] obtains keys status of LCD module.

Bit 0	Key 1 pressed
Bit 1	Key 2 pressed
Bit 2	Key 3 pressed
Bit 3	Key 4 pressed

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported.

  `ERR_Error`: No key is pressed.

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(){
int32_t iRet;
uint8_t ubKeys;
    LMB_DLL_Init();
     LMB_LCM_DeviceOpen();
     iRet = LMB_LCM_KeysStatus(&ubKeys);
     if (iRet == ERR_Success )
             printf("key was pressed: %02X\n",ubKeys);
    LMB_LCM_DeviceClose();
     LMB_DLL_DeInit();
     return 0;
}
```

LMB_LCM_KeysCallback

The **LMB_LCM_KeysCallback** function hooks keys event callback of LCM.

- **Syntax**

  int32_t LMB_LCM_KeysCallback(LCMKEY_CALLBACK pCallback, uint16_t uwmSec);

- **Parameters**

  pCallback

  [out] hooks callback function pointer for LCM keys status changed event.

  uwmSec

  [out] setting period of time for checking changed status.

  [range: 10 ~ 999ms]

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_BusyInUses`: The callback function is running now.

  `ERR_Invalid`: The parameter is out of range.

- **Remarks**

  Currently supports case open, software reset button and changes of power status. When pCallback is NULL pointer, this function will be disabled.

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
void lcmCallback(LCMKEY_MSG pLcmMsg){
     printf("Keys=%02X,status=%02X,\
     time = %04d/%02d/%02d %02d:%02d:%02d\n",\
     pLcmMsg.ubKeys, pLcmMsg.ubStatus, pLcmMsg.stuTime.year, \
     pLcmMsg.stuTime.month, pLcmMsg.stuTime.day, \
     pLcmMsg.stuTime.hour, pLcmMsg.stuTime.minute, \
     pLcmMsg.stuTime.second”);
}
int main(){
LMB_DLL_Init();
LMB_LCM_KeysCallback(lcmCallback, 150);
While (1) {
……
     }
     LMB_DLL_DeInit();
     return 0;
}
```

LMB_LCM_StartupMsg

The **LMB_LCM_StartupMsg** function sets default message after LCM reset.

- **Syntax**

  int32_t LMB_LCM_StartupMsg(uint8_t * pubMsg, uint8_t ubLength);

- **Parameters**

  pubMessage

  [out] default displays message after LCM reset.

  ubLength

  [out] message string length.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported.

- **Remarks**

  The string maximum length depends on LCD module, normally 40 characters.

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
 int main(){
int32_t iRet;
uint8_t *pstrMsg = (uint8_t*)"Lanner Electronics Inc.";
    LMB_DLL_Init();
     LMB_LCM_DeviceOpen();
     LMB_LCM_StartupMsg(pstrMsg, strlen((char*)pstrMsg);
     LMB_LCM_Reset();
    LMB_LCM_DeviceClose();
     LMB_DLL_DeInit();
     return ;
 }
```

LMB_LCM_SearchPort

The **LMB_LCM_SearchPort** function gets the LCM current connected port and speed.

- **Syntax**

  int32_t LMB_LCM_SearchPort(int8_t *pbLcmPort, int32_t *pdwSpeed);

- **Parameters**

  pbLcmPort

  [in] obtains the connected port of LCM.

  pdwSpeed

  [in] obtains the current connected speed.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported.

  `ERR_NotExist`: This function is not enabled or does not exist.

- **Remarks**

  The string maximum length depends on LCD module, normally 40 characters.

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(){
int32_t iRet;
int8_t strLcmPort[50];
int32_t dwSpeed=0;
    LMB_DLL_Init();
    iRet = LMB_LCM_SearchPort(strLcmPort, &dwSpeed);
    if ( iRet == ERR_Sucess) printf("port=%s, speed=%d\n",strLcmPort, dwSpeed);
    LMB_DLL_DeInit();
    return ;
}
```

LMB_LCM_OpenPort

The **LMB_LCM_OpenPort** function opens the LCM device with path and assigned speed.

- **Syntax**

  int32_t LMB_LCM_OpenPort(int8_t * pbLcmPort, int32_t dwSpeed);

- **Parameters**

  pbLcmPort

  [out] assigns the connected port of LCM.

  dwSpeed

  [out] assigns the connected speed.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported.

  `ERR_NotExist`: This function is not enabled or does not exist.

- **Remarks**

  The string maximum length depends on LCD module, normally 40 characters.

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(){
int32_t iRet;
int8_t strLcmPort[50];
int32_t dwSpeed=0;
    LMB_DLL_Init();
    Strcpy(strLcmPort, "/dev/ttyS1");
    dwSpeed=115200;
    iRet = LMB_LCM_OpenPort(strLcmPort, &dwSpeed);
    if ( iRet == ERR_Sucess) printf("LCM connected OK\n");
    LMB_DLL_DeInit();
    return ;
}
```

Power Supply

LMB_PSU_QueryDevices

The **LMB_PSU_QueryDevices** function gets platform power supply devices.

- **Syntax**

  int32_t LMB_PWR_QueryDevices(uint16_t* puwPsuDev);

- **Parameters**

  puwPsuDev

  [in] obtains this platform devices installed status.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported.

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
 int main(){
int32_t iRet;
uint16_t uwPsuDev;
    LMB_DLL_Init();
     iRet = LMB_PSU_QueryDevices(&uwPsuDev);
     if (iRet == ERR_Success )
             printf("PSU devices = %02X\n", uwPsuDev);
     LMB_DLL_DeInit();
     return 0;
 }
```

LMB_PSU_DeviceInfo

The **LMB_PSU_DeviceInfo** function gets power supply information.

- **Syntax**

  int32_t LMB_PSU_DeviceInfo(PSU_INFO* pstuPsuInfo);

- **Parameters**

  pstuPsuInfo

  [in/out] obtains this power supply device information.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened:` The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported.

  `ERR_Invalid`: The parameter is out of range.

- **Remarks**

  The “ubPsuNo” parameter of PSU_INFO needs to assign PSU number.

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(){
int32_t iRet;
PSU_INFO stuPsuInfo;
    LMB_DLL_Init();
     stuPsuInfo.ubPsuNo = 1;
     iRet = LMB_PSU_DeviceInfo(&stuPsuInfo);
     if (iRet == ERR_Success ) {
        printf(“PSU 1 Device information\n”);
        printf(“ MFRID = %s\n”, stuPsuInfo.strubMfrId);
        printf(“ MFRMODEL = %s\n”, stuPsuInfo.strubMfrModel);
        printf(“ MFRSERIAL = %s\n”, stuPsuInfo.strubMfrSerial);
    }
     LMB_DLL_DeInit();
     return 0;
}
```

LMB_PSU_SensorInfo

The **LMB_PSU_SensorInfo** function gets temperature and fan speed information.

- **Syntax**

  int32_t LMB_PSU_SensorInfo(PSU_SENSORS* pstuPsuSensors);

- **Parameters**

  pstuPsuSensors

  [in/out] obtains this power supply sensors information.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported.

  `ERR_Invalid`: The parameter is out of range.

- **Remarks**

  The “ubPsuNo” parameter of PSU_SENSORS needs to assign PSU number.

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(){
int32_t iRet;
PSU_SENSORS stuPsuSensors;
    LMB_DLL_Init();
     stuPsuSensors.ubPsuNo = 1;
     iRet = LMB_PSU_SensorInfo(&stuPsuSensors);
     if (iRet == ERR_Success ) {
        printf(“PSU 1 Sensor information\n”);
        printf(“ Temp-1 = %4.1f\n”, stuPsuSensors.fTemp_1);
        printf(“ Temp-2 = %4.1f\n”, stuPsuSensors.fTemp_2);
        printf(“ FanSpeed = %d\n”, stuPsuSensors.uwFanRpm);
    }
     LMB_DLL_DeInit();
     return 0;
}
```

LMB_PSU_WattsInfo

The **LMB_PSU_WattsInfo** function gets power supply Watts information.

- **Syntax**

  int32_t LMB_PSU_WattsInfo(PSU_WATTS* pstuPsuWatts);

- **Parameters**

  pstuPsuWatts

  [in/out] obtains this power supply Watts, Volt and Amperes information.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported.

- **Remarks**

  The “ubPsuNo” parameter of PSU_WATTS needs to assign PSU number.

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(){
int32_t iRet;
PSU_WATTS stuPsuWatts;
    LMB_DLL_Init();
     stuPsuWatts.ubPsuNo = 1;
     stuPsuWatts.ubPsuNo = PSU_WATTS_OUTPUT;
     iRet = LMB_PSU_WattsInfo(&stuPsuWatts);
     if (iRet == ERR_Success ) {
        printf(“PSU 1 Output Watts information\n”);
        printf(“ Volts = %4.3f\n”, stuPsuWatts.fVolts);
        printf(“ Amperes = %4.3f\n”, stuPsuWatts.fAmperes);
        printf(“ Watts = %4.3f\n”, stuPsuWatts.Watts);
    }
     LMB_DLL_DeInit();
     return 0;
}
```

LMB_PSU_Status

The **LMB_PSU_Status** function gets power supply status information.

- **Syntax**

  int32_t LMB_PSU_Status(uint8_t ubPsuNo, uint16_t* puwStatus);

- **Parameters**

  ubPsuNo

  [out] assigns the PSU number.

  puwStatus

  [in] obtains this power supply status word.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_NotSupport`: This function is not supported.

- **Remarks**

  Please refer to the "PSU Status Bit define" inChapter 2.

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
 int main(){
int32_t iRet;
 uint16_t uwStatus =0;
    LMB_DLL_Init();
     iRet = LMB_PSU_Status(1, &uwStatus);
     if (iRet == ERR_Success ) {
        printf("PSU 1 status Status = %04X\n", uwStatus);
    }
     LMB_DLL_DeInit();
     return 0;
}
```

LMB_PSU_IntrCallback

The **LMB_PSU_IntrCallback** function hooks power supply alarm callback.

- **Syntax**

  int32_t LMB_PSU_IntrCallback(INTRUSION_CALLBACK pCallback, uint16_t uwmSec);

- **Parameters**

  pCallback

  [out] hooks callback function pointer for status changed event.

  uwmSec

  [out] setting period of time for checking status changed

  [range: 10 ~ 999ms]

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_BusyInUses`: The callback function is running now.

  `ERR_Invalid`: The parameter is out of range.

- **Remarks**

  Supports case open, software reset button, changes of power status and power supply fault event (refer to "Intrusion Item Define (bit)" in Chapter 2). When pCallback is NULL pointer, this function will be disabled.

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
void psuCallback(INTRUSION_MSG pIntr){
     printf("item=%04X,status=%04X,\
     time = %04d/%02d/%02d %02d:%02d:%02d\n",\
     pInt.udwOccurItem, pIntr.udwStatus, pIntr.stuTime.year, \
     pIntr.stuTime.month, pIntr.stuTime.day, pIntr.stuTime.hour,\
     pIntr.stuTime.minute, pIntr.stuTime.second”);
}
int main(){
    LMB_DLL_Init();
    LMB_PSU_IntrCallback(psuCallback, 100);
    While (1) {
    ……
     }
     LMB_DLL_DeInit();
     return 0;
}
```

Software Reset Button

LMB_SWR_GetStatus

The **LMB_SWR_GetStatus** function gets software reset button status.

- **Syntax**

  int32_t LMB_SWR_GetStatus(uint8_t *pubStatus);

- **Parameters**

  pubStatus

  [in] indicates software reset button status.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotSupport`: This function is not supported.

  `ERR_Error`: general function error

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{
int32_t iRet;
uint8_t bStatus= 0;
    LMB_DLL_Init();
    iRet = LMB_SW_GetStatus(&bStatus);
    if (iRet == ERR_Success)
            printf("Software reset button status is %d\n", bStatus);
    LMB_DLL_DeInit();
    return 0;
}
```

LMB_SWR_IntrCallback

The **LMB_SWR_IntrCallback** function hooks some status change callback.

- **Syntax**

  int32_t LMB_SWR_IntrCallback(INTRUSION_CALLBACK pCallback, uint16_t uwmSec);

- **Parameters**

  pCallback

  [out] hooks callback function pointer for status changed event.

  uwmSec

  [out] sets period of time for checking changed status .

  [range: 10 ~ 999ms]

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_BusyInUses`: The callback function is running now.

  `ERR_Invalid`: The parameter is out of range.

- **Remarks**

  Supports case open, software reset button and changes of power status and power supply fault event (refer to "Intrusion Item Define (bit)" in Chapter 2). When pCallback is NULL pointer, this function will be disabled.

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
void swrCallback(INTRUSION_MSG pIntr){
     printf("item=%04X,status=%04X,\
     time = %04d/%02d/%02d %02d:%02d:%02d\n",\
     pInt.udwOccurItem, pIntr.udwStatus, pIntr.stuTime.year, \
     pIntr.stuTime.month, pIntr.stuTime.day, pIntr.stuTime.hour,\
     pIntr.stuTime.minute, pIntr.stuTime.second”);
}
int main(){
    LMB_DLL_Init();
    LMB_SWR_IntrCallback(swrCallback, 100);
    While (1) {
    ……
    }
     LMB_DLL_DeInit();
     return 0;
}
```

Power over Ethernet

LMB_POE_Query_Devices

The **LMB_POE_QueryDevices** function queries the PoE supported of LAN ports.

- **Syntax**

  int32_t LMB_POE_QueryDevices(uint32_t *pudwPorts);

- **Parameters**

  pudwPorts

  [in] obtains PoE supported port of LAN ports number.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotSupport`: This function is not supported.

  `ERR_Error`: general function error

- **Remarks**

  bit0 indicates the LAN1, bit1 indicates LAN2, in a similar fashion bit31 indicates LAN32.

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{ int32_t iRet;
  uint32_t udwPorts=0;
    LMB_DLL_Init();
    iRet = LMB_POE_QueryDevices(&udwPorts);
    if ( iRet == ERR_Success ) printf("PoE Ports = %08X\n", udwPorts);
    LMB_DLL_DeInit();
    return 0;
}
```

`

LMB_POE_SetPortPower

The **LMB_POE_SetPortPower** function enables/disables the PoE port power.

- **Syntax**

  int32_t LMB_POE_SetPortPower(uint8_t ubPort, uint8_t ubEnable);

- **Parameters**

  ubPort

  [out] assigns LAN port number.

  ubEnable

  [out] controls power enabling or disabling.

0	Power Off
1	Power On by auto

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_Invalid`: The parameter is out of range or not a PoE LAN port.

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{ int32_t iRet;
    LMB_DLL_Init();
    iRet = LMB_POE_SetPortPower(1, 0);
    if ( iRet == ERR_Success ) printf("LAN1 Port Power Off\n");
    iRet = LMB_POE_SetPortPower(3, 1);
    if ( iRet == ERR_Success ) printf("LAN3 Port Power On by Auto\n");
   LMB_DLL_DeInit();
   return 0;
}
```

LMB_POE_GetPortStatus

The **LMB_POE_GetPortStatus** function gets the PoE port power status

- **Syntax**

  int32_t LMB_POE_GetPortStatus(uint8_t ubPort, uint32_t* pudwStatus);

- **Parameters**

  ubPort

  [out] assigns LAN port number.

  pudwStatus

  [in] obtains PoE LAN port power status

bit stat 0	PoE power is disabled
bit stat 1	PoE power is enabled by auto

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotOpened`: The library is not ready or opened yet.

  `ERR_Invalid`: The parameter is out of range or not a PoE LAN port.

- **Remarks**

  bit0 indicates the LAN1, bit1 indicates LAN2, in a similar fashion bit31 indicates LAN32

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{ int32_t iRet;
  uint32_t dwStatus =0;
    LMB_DLL_Init();
    iRet = LMB_POE_GetPortStatus(3, &dwStatus);
    if ( iRet == ERR_Success )
           printf("LAN3 Port Power status = %0x8X\n", dwStatus);
    iRet = LMB_POE_ GetPortStatus(255, &dwStatus);
    if ( iRet == ERR_Success )
           printf("All LAN Ports Power status = %0x8X\n", dwStatus);
   LMB_DLL_DeInit();
   return 0;
}
```

Ignition

LMB_IGN_OpenPort

The **LMB_IGN_OpenPort** function opens the ignition device port with 57600 bps.

- **Syntax**

  int32_t LMB_IGN_OpenPort(int8_t* pbPortPath);

- **Parameters**

  pbPortPath

  [out] assigns the ignition device path

- **Return Value**

`ERR_Success`: This function is working properly.

`ERR_NotSupport`: This function is not supported.

`ERR_NotExist`: The device does not exist.

`ERR_Error`: general function error

- **Remarks**

  None

- **Example**

  Please refer to the LMB_IGN_xxx functions.

LMB_IGN_OpenPort

The **LMB_IGN_OpenPort** function closes the ignition device port.

- **Syntax**

  int32_t LMB_IGN_ClosePort(void);

- **Parameters**

  None

- **Return Value**

`ERR_Success`: This function is working properly.

`ERR_NotSupport`: This function is not supported.

`ERR_NotExist`: The device does not exist.

`ERR_Error`: general function error

- **Remarks**

  None

- **Example**

  Please refer to the LMB_IGN_xxx functions.

LMB_IGN_VersionInfo

The **LMB_IGN_VersionInfo** function gets the ignition device-ID and version information.

- **Syntax**

  int32_t LMB_IGN_VersionInfo(IGN_VERSION* pstuIgnVersion);

- **Parameters**

  pstuIgnVersion

  [in] pstuIgnVersion->strbDeviceID obtains the device-ID message

  pstuIgnVersion->strbVersion obtains the version message

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotSupport`: This function is not supported.

  `ERR_Error`: general function error

  `ERR_NotOpened`: device port not opened yet

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{ int32_t iRet;
  IGN_VERSION stuIgnVersion;
    LMB_DLL_Init();
    iRet = LMB_IGN_OpenPort("/dev/ttyS4");
    if ( iRet != ERR_Success ) {
      printf("Device port not exist, return %d\n", iRet);
      return -1;
    }
    iRet = LMB_IGN_VersionInfo(&stuIgnVersion);
    if ( iRet == ERR_Success ) {
        printf("Ignition device-ID = %s\n", stuIgnVersion.strbDeviceID);
        printf("         version   = %s\n", stuIgnVersion.strbVersion);
    }
    LMB_IGN_ClosePort();
    LMB_DLL_DeInit();
    return 0;
}
```

LMB_IGN_FuncSupport

The **LMB_IGN_FuncSupport** function gets function support from this ignition device.

- **Syntax**

  int32_t LMB_IGN_FuncSupport (uint16_t* puwFunc);

- **Parameters**

  puwFunc

  [in] obtains functions support from the ignition device

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotSupport`: This function is not supported.

  `ERR_Error`: general function error

  `ERR_NotOpened`: device port not opened yet

- **Remarks**

  Please refer to "Ignition Device Functions" in chapter 2.

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{ int32_t iRet;
  uint32_t uwFunc=0 ;
    LMB_DLL_Init();
    iRet = LMB_IGN_OpenPort("/dev/ttyS4");
    if ( iRet != ERR_Success ) {
      printf("Device port not exist, return %d\n", iRet);
      return -1;
    }
    iRet = LMB_IGN_FuncSupport(&uwFunc);
    if ( iRet == ERR_Success ) {
      printf("Get IGN Functions Support Info.=0x%04X\n", uwIGN_Func);
      printf("    Startup WatchDogTimer -------> ");
      if (uwFunc & (uint16_t)FUNC_STARTUPWDT ) printf("Enabled\n");
      else           printf("\033[1;36mDisabled\033[0m\n");
      printf("    Low-Power Shutdown Control --> ");
      if (uwFunc & (uint16_t)FUNC_LOWPOWER ) printf("Enabled\n");
      else           printf("\033[1;36mDisabled\033[0m\n");
      printf("    Digital Input ----------------> ");
      if (uwFunc & (uint16_t)FUNC_IGNGPI ) printf("Enabled\n");
      else           printf("\033[1;36mDisabled\033[0m\n");
      printf("    Digital Output ---------------> ");
      if (uwFunc & (uint16_t)FUNC_IGNGPO ) printf("Enabled\n");
      else           printf("\033[1;36mDisabled\033[0m\n");
      printf("    Power over Ethernet ---------> ");
      if (uwFunc & (uint16_t)FUNC_IGNPOE ) printf("Enabled\n");
      else           printf("\033[1;36mDisabled\033[0m\n");
      printf("    Heater Control ---------------> ");
      if (uwFunc & (uint16_t)FUNC_HEATER ) printf("Enabled\n");
      else           printf("\033[1;36mDisabled\033[0m\n");
    }
    LMB_IGN_ClosePort();
    LMB_DLL_DeInit();
    return 0;
}
```

LMB_IGN_SetStartupScheme

The **LMB_IGN_ SetStartupScheme** function configures system startup procedure when the Key is at "On" state.

- **Syntax**

  int32_t LMB_IGN_ SetStartupScheme(STARTUP_SCHEME stuStartupScheme);

- **Parameters**

  stuStartupScheme

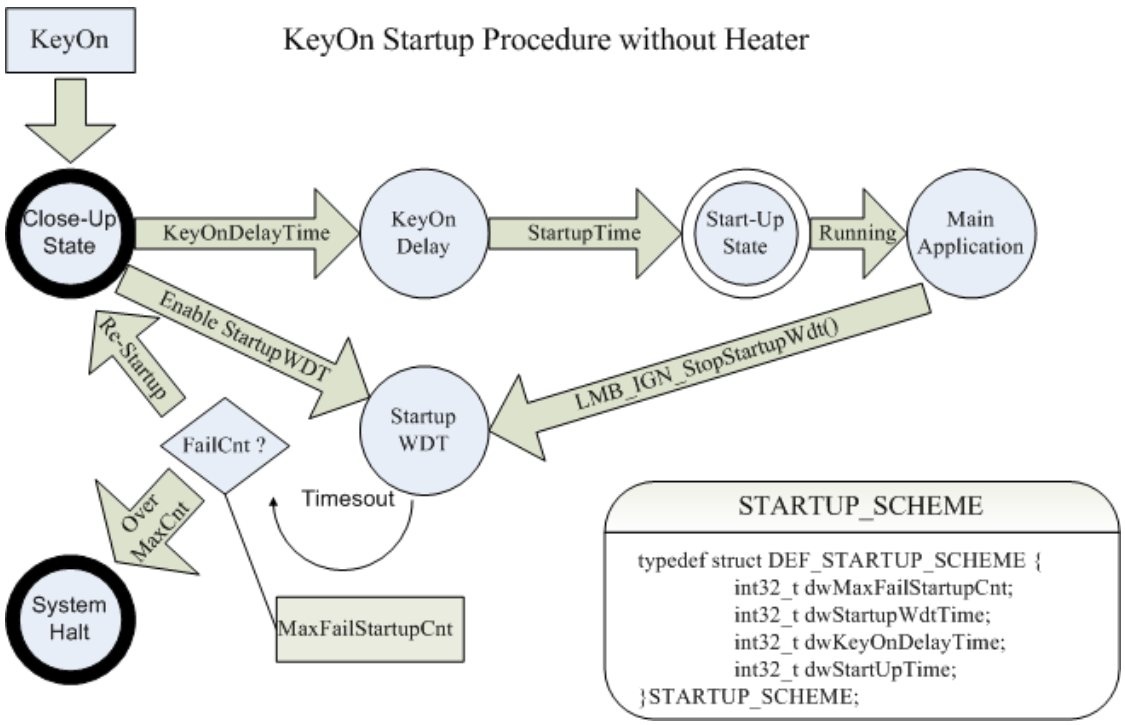
  [out] stuStartupScheme .dwMaxFailStartupCnt sets the maximum count when WDT timeout
  stuStartupScheme .dwStartupWdtTime sets the WDT value of startup
  stuStartupScheme .dwKeyOnDelayTime sets delay time after Key-On state
  stuStartupScheme .dwStartUpTime sets system booting time then entry "Start-Up" state

- **Return Value**

  `ERR_Success`: This function is working properly.
  `ERR_NotSupport`: This function is not supported.
  `ERR_Error`: general function error
  `ERR_NotOpened`: device port not opened yet

- **Remarks**

  StartupWdtTime is dependent on platform hardware setting, please refer to **錯誤! 找不到參照來源。** function.



- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{ int32_t iRet;
  STARTUP_SCHEME stuStartupScheme;
    LMB_DLL_Init();
    iRet = LMB_IGN_OpenPort("/dev/ttyS4");
    if ( iRet != ERR_Success ) {
      printf("Device port not exist, return %d\n", iRet);
      return -1;
    }
    iRet = LMB_IGN_GetStartupScheme(&stuStartupScheme);
    if ( iRet == ERR_Success )  {
        printf("Get StaruUp Time: MaxFailStartupCnt=%d, StartupWdtTime=%d\n",
            stuStartupScheme.dwMaxFailStartupCnt,
  stuStartupScheme.dwStartupWdtTime);
        printf("               PoweronDelay=%d, StartupTime=%d\n",
            stuStartupScheme.dwKeyOnDelayTime, stuStartupScheme.dwStartUpTime);
    }
    else    printf("<Error> LMB_IGN_GetStartupScheme return %d\n", iRet);
    srand(time(NULL));
    stuStartupScheme.dwKeyOnDelayTime = (int32_t)(rand() % 256);
    stuStartupScheme.dwStartUpTime = (int32_t)(rand() % 640000) ;
    stuStartupScheme.dwMaxFailStartupCnt = (int32_t)(rand() % 1024);
    stuStartupScheme.dwStartupWdtTime = (int32_t)(rand() % 640000);
    printf("Set StaruUp Time: MaxFailStartupCnt=%d, StartupWdtTime=%d\n",
          stuStartupScheme.dwMaxFailStartupCnt, stuStartupScheme.dwStartupWdtTime);
    printf("               KeyOnDelayTime=%d, StartupTime=%d\n",
          stuStartupScheme.dwKeyOnDelayTime, stuStartupScheme.dwStartUpTime);
    iRet = LMB_IGN_SetStartupScheme(stuStartupScheme);
    if ( iRet != ERR_Success )
          printf("<Error> LMB_IGN_SetStartupScheme return %d\n", iRet);
    iRet = LMB_IGN_GetStartupScheme(&stuStartupScheme);
    if ( iRet == ERR_Success ) {
        printf("Get StaruUp Time: MaxFailStartupCnt=%d, StartupWdtTime=%d\n",
          stuStartupScheme.dwMaxFailStartupCnt,stuStartupScheme.dwStartupWdtTime);
        printf("               KeyOnDelayTime=%d, StartupTime=%d\n",
            stuStartupScheme.dwKeyOnDelayTime, stuStartupScheme.dwStartUpTime);
    }
```

```
  else  printf("<Error> LMB_IGN_GetStartupScheme return %d\n", iRet);
  LMB_IGN_ClosePort();
  LMB_DLL_DeInit();
  return 0;
}
```

LMB_IGN_GetStartupScheme

The **LMB_IGN_ GetStartupScheme** function gets the current settings of system startup procedure.

- **Syntax**

  int32_t LMB_IGN_ SetStartupScheme(STARTUP_SCHEME* pstuStartupScheme);

- **Parameters**

  pstuStartupScheme

  [in] pstuStartupScheme ->dwMaxFailStartupCnt obtains the maximum count of fail booting

  pstuStartupScheme ->dwStartupWdtTime obtains WDT value of startup procedure

  pstuStartupScheme ->dwKeyOnDelayTime obtains Key-On delay time value

  pstuStartupScheme ->dwStartUpTime obtains system booting time then enters "Start-Up" state after Key-On delay time.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotSupport`: This function is not supported.

  `ERR_Error`: general function error

  `ERR_NotOpened`: device port not opened yet

- **Remarks**

  (None)

- **Example**

  Please refer to "LMB_IGN_SetStartupScheme" example.

LMB_IGN_SetShutdownScheme

The **LMB_IGN_SetShutdownScheme** function configures shutdown procedure when the Key is at "ACC"/"OFF" state.

- **Syntax**

  int32_t LMB_IGN_SetShutdownScheme(SHUTDOWN_SCHEME stuShutdownScheme);

- **Parameters**

  stuShutdownScheme

  [out] stuShutdownScheme.dwMaxFailShutdownCnt sets the maximum count when system shutdown is an abnormal mode.

  stuShutdownScheme.dwKeyOffDelayTime sets delay time after Key-ACC/Off state

  stuShutdownScheme.dwShutdownTime sets system shutdown time then enters "Close-Up" state

- **Return Value**

  `ERR_Success`: This function is working properly.

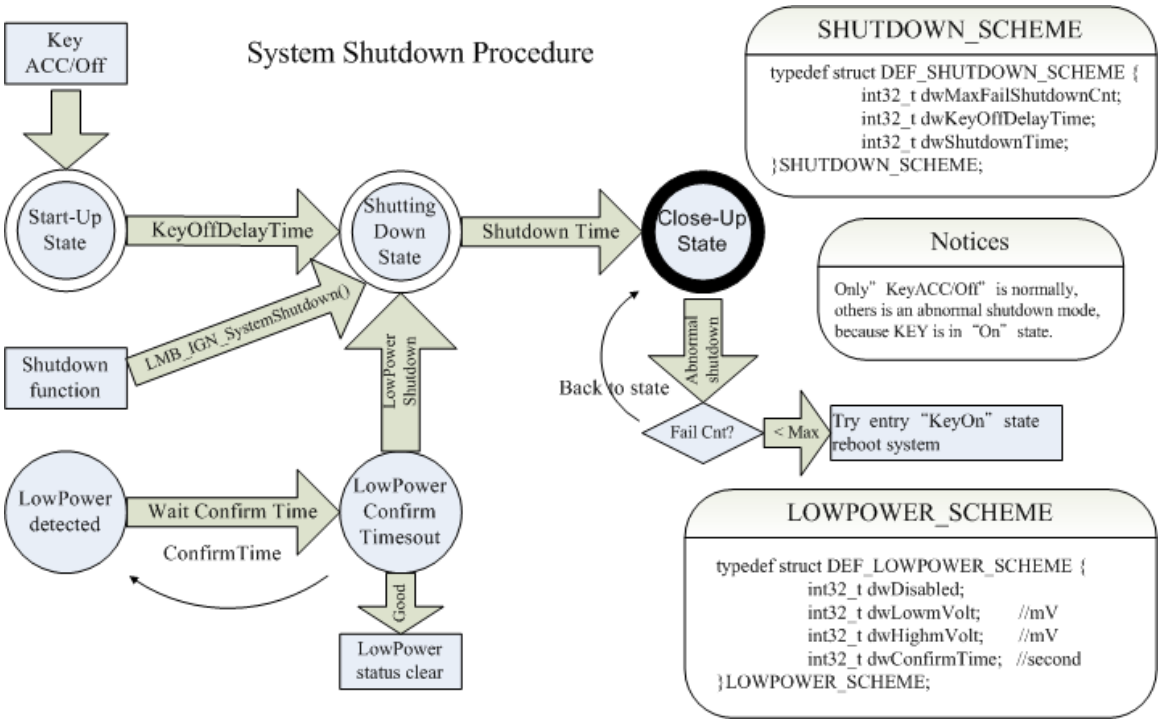
  `ERR_NotSupport`: This function is not supported.

  `ERR_Error`: general function error

  `ERR_NotOpened`: device port not opened yet

- **Remarks**

  If the Key remains in "On" state after the system shuts down, this is an abnormal shutdown mode.



- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{ int32_t iRet;
  SHUTDOWN_SCHEME stuShutdownScheme;
    LMB_DLL_Init();
    iRet = LMB_IGN_OpenPort("/dev/ttyS4");
    if ( iRet != ERR_Success ) {
      printf("Device port not exist, return %d\n", iRet);
      return -1;
    }
    iRet = LMB_IGN_GetShutdownScheme(&stuShutdownScheme);
    if ( iRet == ERR_Success )
          printf("Get Shutdown Time: MaxFailShutdownCnt=%d, ShutdownDelay=%d,
            ShutdownTime=%d\n", stuShutdownScheme.dwMaxFailShutdownCnt,
            stuShutdownScheme.dwKeyOffDelayTime,stuShutdownScheme.dwShutdownTime);
    else  printf("<Error> LMB_IGN_GetShutdownScheme return %d\n", iRet);
    stuShutdownScheme.dwKeyOffDelayTime = (int32_t)(rand() % 256);
    stuShutdownScheme.dwShutdownTime = (int32_t)(rand() % 640000) ;
    stuShutdownScheme.dwMaxFailShutdownCnt = (int32_t)(rand() % 256) ;
    printf("Set Shutdown Time: MaxFailShutdownCnt=%d, ShutdownDelay=%d,
      ShutdownTime=%d\n", stuShutdownScheme.dwMaxFailShutdownCnt,
      stuShutdownScheme.dwKeyOffDelayTime,stuShutdownScheme.dwShutdownTime);
    iRet = LMB_IGN_SetShutdownScheme(stuShutdownScheme);
    if ( iRet != ERR_Success )
            printf("<Error> LMB_IGN_SetShutdownScheme return %d\n", iRet);
    iRet = LMB_IGN_GetShutdownScheme(&stuShutdownScheme);
    if ( iRet == ERR_Success )
        printf("Get Shutdown Time: MaxFailShutdownCnt=%d, ShutdownDelay=%d,
            ShutdownTime=%d\n", stuShutdownScheme.dwMaxFailShutdownCnt,
            stuShutdownScheme.dwKeyOffDelayTime, stuShutdownScheme.dwShutdownTime);
      else  printf(<Error> "LMB_IGN_GetShutdownScheme return %d\n", iRet);
    LMB_IGN_ClosePort();
    LMB_DLL_DeInit();
    return 0;
}
```

LMB_IGN_GetShutdownScheme

The **LMB_IGN_GetShutdownScheme** function gets the settings value of shutdown procedure.

- **Syntax**

  int32_t LMB_IGN_GetShutdownScheme(SHUTDOWN_SCHEME* pstuShutdownScheme);

- **Parameters**

  pstuShutdownScheme

  [in] pstuShutdownScheme->dwMaxFailShutdownCnt obtains the maximum value of an abnormal shutdown.

  pstuShutdownScheme->dwKeyOffDelayTime obtains delay time value when Key-ACC/Off

  pstuShutdownScheme->dwShutdownTime obtains system shutdown time value

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotSupport`: This function is not supported.

  `ERR_Error`: general function error

  `ERR_NotOpened`: device port not opened yet

- **Remarks**

  (None)

- **Example**

  Please refer to "LMB_IGN_SetShutdownScheme" example.

LMB_IGN_SetLowPowerScheme

The **LMB_IGN_SetLowPowerScheme** function configures shutdown procedure when lowpower occurs

- **Syntax**

  int32_t LMB_IGN_SetLowPowerScheme (LOWPOWER_SCHEME stuLowPowerScheme);

- **Parameters**

  stuLowPowerScheme

  [out] stuLowPowerScheme.dwDisabled disables low power scheme

  stuLowPowerScheme.dwLowmVolt sets low voltage to trigger the low-power event

  stuLowPowerScheme.dwHighmVol sets high voltage to trigger the low-power event

  stuLowPowerScheme.dwConfirmTime sets confirmation time when the event occurs.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotSupport`: This function is not supported.

  `ERR_Error`: general function error

  `ERR_NotOpened`: device port not opened yet

- **Remarks**

  Please refer to diagram "System Shutdown Procedure" for low-power event scheme.

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{ int32_t iRet;
  LOWPOWER_SCHEME stuLowPowerScheme;
    LMB_DLL_Init();
    iRet = LMB_IGN_OpenPort("/dev/ttyS4");
    if ( iRet != ERR_Success ) {
      printf("Device port not exist, return %d\n", iRet);
      return -1;
    }
    iRet = LMB_IGN_GetLowPowerScheme(&stuLowPowerScheme);
    if ( iRet == ERR_Success )  {
          printf("Get LowPower Event Env.: Minimum=%dmV, Maximum=%dmV\n",
                  stuLowPowerScheme.dwLowmVolt, stuLowPowerScheme.dwHighmVolt);
          printf("                    ConfirmTime=%d, LowPowerDisable=%d\n",
                  stuLowPowerScheme.dwConfirmTime, stuLowPowerScheme.dwDisabled);
    }
    else  {
        printf("LMB_IGN_GetPwrSuddenScheme return %d\n", iRet);
    }
    stuLowPowerScheme.dwLowmVolt= (uint32_t)(rand() % 12000) ;
    stuLowPowerScheme.dwHighmVolt = (uint32_t)(rand() % 100000) ;
    stuLowPowerScheme.dwConfirmTime = (uint32_t)(rand() % 256) ;
    printf("Set LowPower Event Env.: Minimum=%dmV, Maximum=%dmV, ConfirmTime=%d\n",
              stuLowPowerScheme.dwLowmVolt, stuLowPowerScheme.dwHighmVolt,
              stuLowPowerScheme.dwConfirmTime);
    iRet = LMB_IGN_SetLowPowerScheme(stuLowPowerScheme);
    if ( iRet != ERR_Success ) printf("LMB_IGN_SetLowPowerScheme return %d\n", iRet);
    iRet = LMB_IGN_GetLowPowerScheme(&stuLowPowerScheme);
    if ( iRet == ERR_Success )  {
          printf("Get LowPower Event Env.: Minimum=%dmV, Maximum=%dmV\n",
                  stuLowPowerScheme.dwLowmVolt, stuLowPowerScheme.dwHighmVolt);
          printf("                    ConfirmTime=%d, LowPowerDisable=%d\n",
                  stuLowPowerScheme.dwConfirmTime, stuLowPowerScheme.dwDisabled);
    }
```

```
    else  {
        printf("LMB_IGN_GetPwrSuddenScheme return %d\n", iRet);
    }
    LMB_IGN_ClosePort();
    LMB_DLL_DeInit();
    return 0;
}
```

LMB_IGN_GetLowPowerScheme

The **LMB_IGN_GetLowPowerScheme** function gets setting value of the low-power scheme.

- **Syntax**

  int32_t LMB_IGN_GetLowPowerScheme(LOWPOWER_SCHEME* pstuLowPowerScheme);

- **Parameters**

  pstuLowPowerScheme

  [in] pstuLowPowerScheme->dwDisabled obtains software disabled status
  pstuLowPowerScheme->dwLowmVolt obtains low voltage setting value
  pstuLowPowerScheme->dwHighmVolt obtains high voltage setting value
  pstuLowPowerScheme->dwConfirmTime obtains confirm time setting value

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotSupport`: This function is not supported.

  `ERR_Error`: general function error

  `ERR_NotOpened`: device port not opened yet

- **Remarks**

  (None)

- **Example**

  Please refer to "LMB_IGN_SetLowPowerScheme" example.

LMB_IGN_LoadFactorySetting

The **LMB_IGN_LoadFactorySetting** function loads the factory setting to the working environment.

- **Syntax**

  int32_t LMB_IGN_LoadFactorySetting(void);

- **Parameters**

  (None)

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotSupport`: This function is not supported.

  `ERR_Error`: general function error

  `ERR_NotOpened`: device port not opened yet

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{ int32_t iRet;
    LMB_DLL_Init();
    iRet = LMB_IGN_OpenPort("/dev/ttyS4");
    if ( iRet != ERR_Success ) {
      printf("Device port not exist, return %d\n", iRet);
      return -1;
    }
    iRet = LMB_IGN_LoadFactorySetting();
    if ( iRet == ERR_Success )   printf("Load Factory Setting is OK\n");
    else                 printf("LMB_IGN_LoadFactorySetting return %d\n", iRet);
    LMB_IGN_ClosePort();
    LMB_DLL_DeInit();
    return 0;
}
```

LMB_IGN_SaveToDefault

The **LMB_IGN_SaveToDefault** function saves the environment to device rom area for initial loading.

- **Syntax**

  int32_t LMB_IGN_SaveToDefault (void);

- **Parameters**

  (None)

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotSupport`: This function is not supported.

  `ERR_Error`: general function error

  `ERR_NotOpened`: device port not opened yet

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{ int32_t iRet;
    LMB_DLL_Init();
    iRet = LMB_IGN_OpenPort("/dev/ttyS4");
    if ( iRet != ERR_Success ) {
      printf("Device port not exist, return %d\n", iRet);
      return -1;
    }
    iRet = LMB_IGN_SaveToDefault();
    if ( iRet == ERR_Success )   printf("Save to Default is OK\n");
    else         printf("LMB_IGN_SaveToDefault return %d\n", iRet);
    LMB_IGN_ClosePort();
    LMB_DLL_DeInit();
    return 0;
}
```

LMB_IGN_SystemShutdown

The **LMB_IGN_SystemShutdown** function will turn off the system with an abnormal shutdown status.

- **Syntax**

  int32_t LMB_IGN_SystemShutdown(void);

- **Parameters**

  (None)

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotSupport`: This function is not supported.

  `ERR_Error`: general function error

  `ERR_NotOpened`: device port not opened yet

- **Remarks**

  This function will cause an abnormal shutdown status and increases the count of the failed shutdown.

  If the count is lower than the maximum value of shutdown scheme, the system will auto-reboot.

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{ int32_t iRet;
    LMB_DLL_Init();
    iRet = LMB_IGN_OpenPort("/dev/ttyS4");
    if ( iRet != ERR_Success ) {
      printf("Device port not exist, return %d\n", iRet);
      return -1;
    }
    iRet = LMB_IGN_SystemShutdown();
    if ( iRet == ERR_Success )
              printf("the system will turn off abnormal shutdown) ......\n");
    else
              printf("LMB_IGN_SystemShutdown return error %d\n", iRet);
    LMB_IGN_ClosePort();
    LMB_DLL_DeInit();
    return 0;
}
```

LMB_IGN_SetWakeupInput

The **LMB_IGN_SetWakeupInput** function controls the system wake-up input when the system is turned off and the "Key" on "On" state.

- **Syntax**

  int32_t LMB_IGN_SetWakeupInput (uint8_t ubInput);

- **Parameters**

  ubInput

  [out] sets the wakeup pin "enable" or "disable" wake up trigger, when the system is turned off and the "Key" on "On" state

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotSupport`: This function is not supported.

  `ERR_Error`: general function error

  `ERR_NotOpened`: device port not opened yet

- **Remarks**

  Please refer to "Wake-up pin define" section in chapter 2 for the wakeup pin define.

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{ int32_t iRet;
  uint8_t ubData;
    LMB_DLL_Init();
    iRet = LMB_IGN_OpenPort("/dev/ttyS4");
    if ( iRet != ERR_Success ) {
      printf("Device port not exist, return %d\n", iRet);
      return -1;
    }
    ubData = (uint8_t)(rand() % 8) ;
    printf("Set Wakeup Input=0x%02X\n", ubData);
    iRet = LMB_IGN_SetWakeupInput(ubData);
    if ( iRet != ERR_Success ) printf("LMB_IGN_SetWakeupInput return %d\n", iRet);
    iRet = LMB_IGN_GetWakeupInput(&ubData);
      if ( iRet == ERR_Success )   {
        printf("Get Wakeup Input=0x%02X\n", ubData);
        printf("    IGN_DI0 Wakeup Enable= %d\n", ubData & WAKE_IGN_DI0);
        printf("    IGN_DI1 Wakeup Enable= %d\n", (ubData & WAKE_IGN_DI1)>>1);
        printf("    3G module Wakeup Enable= %d\n", (ubData & WAKE_3G_MODULE)>>2);
```

```
        }
        else        printf("LMB_IGN_GetWakeupInput", iRet);
    LMB_IGN_ClosePort();
    LMB_DLL_DeInit();
    return 0;
}
```

LMB_IGN_GetWakeupInput

The **LMB_IGN_SetWakeupInput** function gets the system wakeup input status.

- **Syntax**

  int32_t LMB_IGN_GetWakeupInput (uint8_t* pubInput);

- **Parameters**

  pubInput

  [in] obtains the wakeup pin “enable” or “disable” status

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotSupport`: This function is not supported.

  `ERR_Error`: general function error

  `ERR_NotOpened`: device port not opened yet

- **Remarks**

  For the wakeup pin definition, please refer to chapter 2 “Wake-up pin define” section.

- **Example**

  Please refer to “LMB_IGN_SetWakeupInput” example.

LMB_IGN_GetKeyOnStat

The **LMB_IGN_GetKeyOnStat** function gets the "KeyOn" state.

- **Syntax**

  int32_t LMB_IGN_GetKeyOnStat (uint8_t* pubStat);

- **Parameters**

  pubStat

  [in] obtains the Key on "On" state or not

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotSupport`: This function is not supported.

  `ERR_Error`: general function error

  `ERR_NotOpened`: device port not opened yet

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{ int32_t iRet;
  uint8_t ubPinStat =0;
    LMB_DLL_Init();
    iRet = LMB_IGN_OpenPort("/dev/ttyS4");
    if ( iRet != ERR_Success ) {
      printf("Device port not exist, return %d\n", iRet);
      return -1;
    }
      iRet = LMB_IGN_GetKeyOnStat(&ubPinStat);
      if ( iRet == ERR_Success ) printf("Get KeyOn State=0x%02X\n", ubPinStat);
      else                       printf("LMB_IGN_GetKeyOnStat return %d\n", iRet);
    LMB_IGN_ClosePort();
    LMB_DLL_DeInit();
    return 0;
  }
```

LMB_IGN_StopStartupWdt

The **LMB_IGN_StopStartupWdt** function stops the Startup Watch-dog Timer.

- **Syntax**

  int32_t LMB_IGN_StopStartupWdt (void);

- **Parameters**

  (None)

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotSupport`: This function is not supported.

  `ERR_Error`: general function error

  `ERR_NotOpened`: device port not opened yet

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{ int32_t iRet;
  uint16_t uwIGN_Func =0 ;
    LMB_DLL_Init();
    iRet = LMB_IGN_OpenPort("/dev/ttyS4");
    if ( iRet != ERR_Success ) {
      printf("Device port not exist, return %d\n", iRet);
      return -1;
    }
    LMB_IGN_FuncSupport(&uwIGN_Func);
    if  ( uwIGN_Func & FUNC_STARTUPWDT ) {
        iRet = LMB_IGN_StopStartupWdt();
        if ( iRet != ERR_Success )printf("LMB_IGN_StopStartupWdt return %d\n", iRet);
        int32_t dwSeconds = (uint8_t)(rand() % 640000) ;
        printf("Set Reload StartupWdt Time=%d\n", dwSeconds);
        iRet = LMB_IGN_ReloadStartupWdt(dwSeconds);
        if ( iRet != ERR_Success )printf("LMB_IGN_ReloadStartupWdt return %d\n", iRet);
      }
    else {
        printf("<-- Hardware not support Startup WDT function -->\n");
    }
```

```
  LMB_IGN_ClosePort();
  LMB_DLL_DeInit();
  return 0;
}
```

LMB_IGN_ReloadStartupWdt

The **LMB_IGN_ReloadStartupWdt** function reloads the Startup Watch-dog-Timer and restart counts down.

- **Syntax**

  int32_t LMB_IGN_ReloadStartupWdt (int32_t dwSeconds);

- **Parameters**

  dwSeconds

  [out] reload startupWDT with new countdown value

  if dwSeconds = -1, the IGN device will load dwStartupWdtTime of STARTUP_SCHEME

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotSupport`: This function is not supported.

  `ERR_Error:` general function error

  `ERR_NotOpened`: device port not opened yet

- **Remarks**

  dwSeconds

  [in] obtains the Key on "On" state or not

- **Example**

  Please refer to "LMB_IGN_StopStartupWdt" example.

LMB_IGN_RuntimeInfo

The **LMB_IGN_RuntimeInfo** function gets the information of the device runtime state.

- **Syntax**

  int32_t LMB_IGN_RuntimeInfo (IGN_RUNTIME_INFO* pstuRuntimeInfo);

- **Parameters**

  pstuRuntimeInfo

  [in] pstuRuntimeInfo->dwPowerState obtains the current device state.

  pstuRuntimeInfo->dwInputVoltage obtains the system input voltage.

  pstuRuntimeInfo->dwFailShutdownCnt obtains the count of abnormal shutdown.

  pstuRuntimeInfo->dwFailStartupCnt obtains the count of failed startup.

  pstuRuntimeInfo->dwShutdownFalg indicates the use of "SHUTDOWN" command.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotSupport`: This function is not supported.

  `ERR_Error`: general function error

  `ERR_NotOpened`: device port not opened yet

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{ int32_t iRet;
  IGN_RUNTIME_INFO stuRuntimeInfo;
    LMB_DLL_Init();
    iRet = LMB_IGN_OpenPort("/dev/ttyS4");
    if ( iRet != ERR_Success ) {
      printf("Device port not exist, return %d\n", iRet);
      return -1;
    }
    iRet = LMB_IGN_RuntimeInfo(&stuRuntimeInfo);
      if ( iRet == ERR_Success )   {
        printf("Get RuntimeInfo PowerState=%d\n", stuRuntimeInfo.dwPowerState);
        printf("              InputVoltage=%dmV\n", stuRuntimeInfo.dwInputVoltage);
        printf("           FailShutdownCnt=%d\n",
stuRuntimeInfo.dwFailShutdownCnt);
        printf("            FailStartupCnt=%d\n", stuRuntimeInfo.dwFailStartupCnt);
        printf("         SoftShutdownFlag=%d\n", stuRuntimeInfo.dwShutdownFalg);
```

```
        }
        else        printf("LMB_IGN_RuntimeInfo return %d\n", iRet);
    LMB_IGN_ClosePort();
    LMB_DLL_DeInit();
    return 0;
}
```

LMB_IGN_GetDigitalPins

The **LMB_IGN_GetDigitalPins** function gets the IGN device GPI/DI and GPO/DO pins.

- **Syntax**

  int32_t LMB_IGN_GetDigitalPins (uint32_t* pudwOutPins, uint32_t* pudwInPins);

- **Parameters**

  pudwOutPins

  [in] obtains the IGN device GPO/DO pins

  pudwInPins

  [in] obtains the IGN device GPI/DI pins

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotSupport`: This function is not supported.

  `ERR_Error`: general function error

  `ERR_NotOpened`: device port not opened yet

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{ int32_t iRet;
  uint16_t uwIGN_Func =0;
    LMB_DLL_Init();
    iRet = LMB_IGN_OpenPort("/dev/ttyS4");
    if ( iRet != ERR_Success ) {
      printf("Device port not exist, return %d\n", iRet);
      return -1;
    }
    uint32_t udwOutPins=0, udwInPins=0, udwDioPins, udwDioStat ;
    LMB_IGN_FuncSupport(&uwIGN_Func);
    iRet = LMB_IGN_GetDigitalPins(&udwOutPins, &udwInPins);
    if ( iRet == ERR_Success ) printf("Get DIO support Pins:   DigitalOutPins=0x%08X,
                          DigitalInPins=0x%08X\n", udwOutPins, udwInPins);
    else         printf("LMB_IGN_GetDigitalPins return %d\n", iRet);
    if ( uwIGN_Func & FUNC_IGNGPO ) {
        udwDioPins = udwOutPins ;
        udwDioStat = (uint32_t)(rand() % udwOutPins) ;
        printf("---> Set DigitalOut, PinAssigns=0x%04X, DititalOut=0x%04X\n",
                                      udwDioPins, udwDioStat);
        iRet = LMB_IGN_SetDigitalOut(  udwDioPins, udwDioStat);
        if ( iRet != ERR_Success ) printf("LMB_IGN_SetDigitalOut return %d\n", iRet);
        sleep(1); // GPIO_DELAY_CAPACITOR
        udwDioStat=0;
        iRet = LMB_IGN_GetDigitalOut( udwDioPins, &udwDioStat);
        if ( iRet == ERR_Success ) printf("    Get DigitalOut, PinAssigns=0x%04X,
                                DititalOut=0x%04X\n", udwDioPins, udwDioStat);
        else         printf("LMB_IGN_GetDigitalOut return %d\n", iRet);
    }
    if ( uwIGN_Func & FUNC_IGNGPI ) {
        udwDioPins = udwInPins ;
        udwDioStat=0;
        iRet = LMB_IGN_GetDigitalIn( udwDioPins, &udwDioStat);
        if ( iRet == ERR_Success ) printf("    Get DigitalIn , PinAssigns=0x%04X,
                                DititalOut=0x%04X\n", udwDioPins, udwDioStat);
        else         printf("LMB_IGN_GetDigitalIn return %d\n", iRet);
```

```
    }
    LMB_IGN_ClosePort();
    LMB_DLL_DeInit();
    return 0;
}
```

LMB_IGN_SetDigitalOut

The **LMB_IGN_SetDigitalOut** function sets the GPO/DO status of the IGN device.

- **Syntax**

  int32_t LMB_IGN_SetDigitalOut (uint32_t udwPinAssign, uint32_t udwDigitalOut);

- **Parameters**

  udwPinAssign

  [out] assigns the GPO/DO pins to write the new status

  udwDigitalOut

  [out] sets GPO/DO new status through the assigned pin

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotSupport`: This function is not supported.

  `ERR_Error`: general function error

  `ERR_NotOpened`: device port not opened yet

- **Remarks**

  (None)

- **Example**

  Please refer to "LMB_IGN_GetDigitalPins" example

LMB_IGN_GetDigitalIn

The **LMB_IGN_GetDigitalIn** function gets the GPO/DO status of the IGN device.

- **Syntax**

  int32_t LMB_IGN_GetDigitalIn (uint32_t udwPinAssign, uint32_t* pudwDigitalIn);

- **Parameters**

  udwPinAssign

  [out] assigns the GPI/DI pins to read the current status

  udwDigitalIn

  [in] obtains GPI/DI current status through the assigned pin

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotSupport`: This function is not supported.

  `ERR_Error`: general function error

  `ERR_NotOpened`: device port not opened yet

- **Remarks**

  (None)

- **Example**

  Please refer to "LMB_IGN_GetDigitalPins" example

LMB_IGN_QueryPoePorts

The **LMB_IGN_QueryPoePorts** function gets the IGN device Power-Over-Ethernet information.

- **Syntax**

  int32_t LMB_IGN_QueryPoePorts (uint32_t* pudwPorts);

- **Parameters**

  pudwPorts

  [in] obtains PoE ports of the IGN device supported.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotSupport`: This function is not supported.

  `ERR_Error`: general function error

  `ERR_NotOpened`: device port not opened yet

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{ int32_t iRet;
  uint16_t uwIGN_Func =0;
  uint32_t udwPoePorts = 0, udwPoePower = 0 ;
    LMB_DLL_Init();
    iRet = LMB_IGN_OpenPort("/dev/ttyS4");
    if ( iRet != ERR_Success ) {
      printf("Device port not exist, return %d\n", iRet);
      return -1;
    }
    LMB_IGN_FuncSupport(&uwIGN_Func);
    if ( uwIGN_Func & FUNC_IGNPOE ) {
        iRet = LMB_IGN_QueryPoePorts(&udwPoePorts);
        if ( iRet == ERR_Success ) printf("Get IGN-POE Ports=0x%04X\n", udwPoePorts);
        else         printf("LMB_IGN_QueryPoePorts return %d\n", iRet);
        printf("Set DigitalPOE, PortAssigns=0x%04X, PowerEnable=0x%04X\n",
                                                udwPoePorts, udwPoePower);
        iRet = LMB_IGN_SetPoePower( udwPoePorts, udwPoePower);
        if ( iRet != ERR_Success ) printf("LMB_IGN_SetPoePower return %d\n", iRet);
        udwPoePower=0;
        iRet = LMB_IGN_GetPoePower(udwPoePorts, &udwPoePower);
        if ( iRet == ERR_Success ) printf("Get DigitalPOE, PortAssigns=0x%04X,
                          PowerEnable=0x%04X\n", udwPoePorts, udwPoePower);
        else         printf("LMB_IGN_GetPoePower return %d\n", iRet);
        udwPoePorts = (uint32_t)(rand() % 1024) ;
        udwPoePower = (uint32_t)(rand() % 1024) ;
        printf("Set DigitalPOE, PortAssigns=0x%04X, PowerEnable=0x%04X,
          (valid=0x%04X)\n", udwPoePorts, udwPoePower, udwPoePorts & udwPoePower);
        iRet = LMB_IGN_SetPoePower( udwPoePorts, udwPoePower);
        if ( iRet != ERR_Success ) printf("LMB_IGN_SetPoePower return %d\n", iRet);
        udwPoePorts = 0x3FF ;
        udwPoePower=0;
        iRet = LMB_IGN_GetPoePower(udwPoePorts, &udwPoePower);
       if ( iRet == ERR_Success ) printf("Get DigitalPOE, PortAssigns=0x%04X,
            PowerEnable=0x%04X\n", udwPoePorts, udwPoePower);
       else         printf("LMB_IGN_GetPoePower return %d\n", iRet);
```

```
  }
  else  printf("<-- Hardware not support Power over Ethernet function -->\n");
  LMB_IGN_ClosePort();
  LMB_DLL_DeInit();
  return 0;
}
```

LMB_IGN_SetPoePower

The **LMB_IGN_SetPoePower** function sets the PoE power status of the IGN device.

- **Syntax**

  int32_t LMB_IGN_SetPoePower(uint32_t udwPortAssign, uint32_t udwPowerEnable);

- **Parameters**

  udwPortAssign

  [out] assigns the PoE ports to control the power state

  udwPowerEnable

  [out] sets PoE ports power state through the assigned pin

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotSupport`: This function is not supported.

  `ERR_Error`: general function error

  `ERR_NotOpened`: device port not opened yet

- **Remarks**

  (None)

- **Example**

  Please refer to "LMB_IGN_QueryPoePorts" example

LMB_IGN_GetPoePower

The **LMB_IGN_GetPoePower** function gets the PoE power status of the IGN device.

- **Syntax**

  int32_t LMB_IGN_GetPoePower(uint32_t udwPortAssign, uint32_t* pudwPowerStatus);

- **Parameters**

  udwPortAssign

  [out] assigns the PoE ports to get the power state

  udwPowerStatus

  [in] obtains PoE ports power state through the assigned pin

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotSupport`: This function is not supported.

  `ERR_Error`: general function error

  `ERR_NotOpened`: device port not opened yet

- **Remarks**

  (None)

- **Example**

  Please refer to "LMB_IGN_QueryPoePorts" example

LMB_IGN_GetParamRange

The **LMB_IGN_GetParamRange** function gets the IGN device Power-Over-Ethernet information.

- **Syntax**

  int32_t LMB_IGN_GetParamRange(uint8_t ubParam, VALUE_RANGE* pstuValueRange);

- **Parameters**

  ubParam

  [out] assigns parameter item

  (refer chapter 2 "LMB_IGN_GetParamRange define " section)

  pstuValueRange

  [in] pstuValueRange->dwMinimum obtains the minimum value of this parameter.

  pstuValueRange->dwMaximum obtains the maximum value of this parameter.

  pstuValueRange-> dwDefault obtains the factory value of this parameter.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotSupport`: This function is not supported.

  `ERR_Error`: general function error

  `ERR_NotOpened`: device port not opened yet

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{ int32_t iRet;
int8_t aryParamString[][30]={
"PARAM_KeyOnDelayTime(Sec.) ",
"PARAM_StartUpTime(Sec.)    ",
"PARAM_StartupWdtTime(Sec.) ",
"PARAM_MaxFailStartupCnt    ",
"PARAM_LowmVolt(mV)        ",
"PARAM_HighmVolt(mV)       ",
"PARAM_ConfirmTime(Sec.)    ",
"PARAM_MaxFailShutdownCnt   ",
"PARAM_KeyOffDelayTime(Sec.)",
"PARAM_ShutdownTime(Sec.)   ",
"PARAM_mCelsiusMin(milli)   ",
"PARAM_mCelsiusMax(milli)   "
};
    LMB_DLL_Init();
    iRet = LMB_IGN_OpenPort("/dev/ttyS4");
    if ( iRet != ERR_Success ) {
      printf("Device port not exist, return %d\n", iRet);
      return -1;
    }
    for (int xi=1 ; xi<PARAM_Unknown ; xi++) {
        iRet = LMB_IGN_GetParamRange(xi, &stuValueRange);
        if ( iRet == ERR_Success )  {
          printf("%s: Minimum=%d\t, Maximum=%7d\t, Default=%7d\n",
aryParamString[xi-1]
                stuValueRange.dwMinimum, stuValueRange.dwMaximum,
                stuValueRange.dwDefault);
        }
        else  printf("LMB_IGN_GetParamRange return %d\n", iRet);
      }
    LMB_IGN_ClosePort();
    LMB_DLL_DeInit();
  return 0;
}
```

LMB_IGN_SetHeaterScheme

The **LMB_IGN_SetHeaterScheme** function sets the heater control range of the IGN device.

- **Syntax**

  int32_t LMB_IGN_SetHeaterScheme(HEATER_SCHEME stuHeaterScheme);

- **Parameters**

  stuHeaterScheme

  [out] stuHeaterScheme.dwmCelsiusMin sets the lowest temperature to start the heater.
  stuHeaterScheme.dwmCelsiusMax sets the highest temperature to stop the heater.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotSupport`: This function is not supported.

  `ERR_Error`: general function error

  `ERR_NotOpened`: device port not opened yet

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{ int32_t iRet;
  uint16_t uwIGN_Func =0;
    LMB_DLL_Init();
    iRet = LMB_IGN_OpenPort("/dev/ttyS4");
    if ( iRet != ERR_Success ) {
      printf("Device port not exist, return %d\n", iRet);
      return -1;
    }
    LMB_IGN_FuncSupport(&uwIGN_Func);
    if ( uwIGN_Func & FUNC_HEATER ) {
        HEATER_SCHEME stuHeaterScheme;
        iRet = LMB_IGN_GetHeaterScheme(&stuHeaterScheme);
        if ( iRet == ERR_Success ) printf("Get HeaterTemperature, mCelsiusMin=%d,
               mCelsiusMax=%d\n", stuHeaterScheme.dwmCelsiusMin,
               stuHeaterScheme.dwmCelsiusMax);
        else   printf("LMB_IGN_GetHeaterScheme return %d\n", iRet);
        stuHeaterScheme.dwmCelsiusMin = (int32_t)(rand() % 24000) ;
        stuHeaterScheme.dwmCelsiusMax = (int32_t)(rand() % 31000) ;
        printf("--> Set HeaterTemperature, mCelsiusMin=%d, mCelsiusMax=%d\n",
              stuHeaterScheme.dwmCelsiusMin, stuHeaterScheme.dwmCelsiusMax);
        iRet = LMB_IGN_SetHeaterScheme( stuHeaterScheme);
        if ( iRet != ERR_Success ) printf("LMB_IGN_SetHeaterScheme return %d\n", iRet);
        iRet = LMB_IGN_GetHeaterScheme(&stuHeaterScheme);
        if ( iRet == ERR_Success ) printf("Get HeaterTemperature, mCelsiusMin=%d,
              mCelsiusMax=%d\n", stuHeaterScheme.dwmCelsiusMin,
              stuHeaterScheme.dwmCelsiusMax);
        else  printf("LMB_IGN_GetHeaterScheme return %d\n", iRet);
      }
      else    printf("<-- Hardware not support Heater control function -->\n");
      }
    LMB_IGN_ClosePort();
    LMB_DLL_DeInit();
  return 0;
    }
```

LMB_IGN_GetHeaterScheme

The **LMB_IGN_GetHeaterScheme** function gets the heater control range of the IGN device.

- **Syntax**

  int32_t LMB_IGN_GetHeaterScheme(HEATER_SCHEME* pstuHeaterScheme);

- **Parameters**

  pstuHeaterScheme

  [in] pstuHeaterScheme->dwmCelsiusMin obtains the lowest temperature

  pstuHeaterScheme->dwmCelsiusMax obtains the highest temperature

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotSupport`: This function is not supported.

  `ERR_Error`: general function error

  `ERR_NotOpened`: device port not opened yet

- **Remarks**

  (None)

- **Example**

  Please refer to "LMB_IGN_SetHeaterScheme" example

LMB_IGN_IntrCallback

The **LMB_IGN_IntrCallback** function hooks "KeyOn" status change callback.

- **Syntax**

  int32_t LMB_IGN_IntrCallback(INTRUSION_CALLBACK pCallback, uint16_t uwmSec);

- **Parameters**

  pCallback

  [out] hooks callback function pointer for status changed events.

  uwmSec

  [out] sets period of time for checking changes of status

  [range: 10 ~ 999ms]

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotSupport`: This function is not supported.

  `ERR_BusyInUses`: The callback function is running now.

  `ERR_Invalid`: The parameter is out of range.

  `ERR_NotOpened`: device port not opened yet

- **Remarks**

  Supports case open, software reset button, changes of power status and power supply fault event (refer to "Intrusion Item Define (bit)" in Chapter 2). When pCallback is NULL pointer, this function will be disabled.

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
void ignCallback(INTRUSION_MSG pIntr){
     printf("item=%04X,status=%04X,\
     time = %04d/%02d/%02d %02d:%02d:%02d\n",\
     pInt.udwOccurItem, pIntr.udwStatus, pIntr.stuTime.year, \
     pIntr.stuTime.month, pIntr.stuTime.day, pIntr.stuTime.hour,\
     pIntr.stuTime.minute, pIntr.stuTime.second”);
}
int main(){
  LMB_DLL_Init();
    iRet = LMB_IGN_OpenPort(“/dev/ttyS4”);
    if ( iRet != ERR_Success ) {
      printf("Device port not exist, return %d\n", iRet);
      return -1;
 }
  LMB_IGN_IntrCallback(ignCallback, 100);
  While (1) {
  ……
 }
LMB_IGN_ClosePort();
LMB_DLL_DeInit();
return 0;
 }
```

Global Positioning System

LMB_GPS_OpenPort

The **LMB_GPS_OpenPort** function opens the GPS device port with 9600 bps.

- **Syntax**

  int32_t LMB_GPS_OpenPort(int8_t* pbPortPath);

- **Parameters**

  pbPortPath

  [out] assigns the GPS device path

- **Return Value**

`ERR_Success`: This function is working properly.

`ERR_NotSupport`: This function is not supported.

`ERR_NotExist`: The device does not exist.

`ERR_Error`: general function error

- **Remarks**

  None

- **Example**

  Please refer to the LMB_GPS_Getxxx functions.

LMB_GPS_ClosePort

The **LMB_GPS_ClosePort** function closes the GPS device port.

- **Syntax**

  int32_t LMB_GPS_OpenPort(void);

- **Parameters**

  None

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotSupport`: This function is not supported.

  `ERR_NotExist`: The device does not exist.

  `ERR_Error`: general function error

- **Remarks**

  None

- **Example**

  Please refer to the LMB_GPS_Getxxx functions.

LMB_GPS_GetGNRMC

The **LMB_GPS_GetGNRMC** function gets the 'Recommended Minimum Specific GNSS Data' information.

- **Syntax**

  int32_t LMB_GPS_GetGNRMC(RMC_DATA* pstuRmcData);

- **Parameters**

  pstuRmcData

  [in] pstuRmcData->strLongitude obtains the longitude information

  'E'= East; 'W'= West, Longitude in dddmm.mmmm format.

  pstuRmcData->strLatitude obtains the latitude information

  'N'= North; 'S'= South, Latitude in ddmm.mmmm format.

  pstuRmcData->strSpeedKnots obtains the speed of knot

  pstuRmcData->strStatus obtains the status of GPS information

  'V'= Navigation receiver warning, 'A'= Data valid

  pstuRmcData->strUTCTime obtains the time of UTC

  UTC time in hhmmss.sss format (000000.000 ~ 235959.999)

  pstuRmcData->strUTCDate obtains the date of UTC

  UTC date of position fix, ddmmyy format

  pstuRmcData->strCourse obtains the degrees of Course over ground

- **Return Value**

`ERR_Success`: This function is working properly.

`ERR_NotSupport`: This function is not supported.

`ERR_Error`: general function error

`ERR_NotOpened`: device port not opened yet

- **Remarks**

  1 knot = 1.852 km/h

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{ int32_t iRet;
  RMC_DATA stuRmcData ;
    LMB_DLL_Init();
    iRet = LMB_GPS_OpenPort("/dev/ttyS4");
    if ( iRet != ERR_Success ) {
      printf("Device port not exist, return %d\n", iRet);
      return -1;
    }
    iRet = LMB_GPS_GetGNRMC(&stuRmcData);
    if ( iRet != ERR_Success )  printf("LMB_GPS_GetGNRMC return %d\n", iRet);
    else {
      printf("-----------   GNRMC Data -----------------\n");
      printf("strStatus =%s\n", stuRmcData.strStatus);
      printf("strLongitude =%s\n", stuRmcData.strLongitude);
      printf("strLatitude =%s\n", stuRmcData.strLatitude);
      printf("strSpeedKnots =%s\n", stuRmcData.strSpeedKnots);
      printf("strCourse =%s\n", stuRmcData.strCourse);
      printf("strUTCTime =%s\n", stuRmcData.strUTCTime);
      printf("strUTCDate =%s\n", stuRmcData.strUTCDate);
    }
    LMB_GPS_ClosePort();
    LMB_DLL_DeInit();
    return 0;
}
```

LMB_GPS_GetGNGGA

The **LMB_GPS_GetGNGGA** function gets the 'Global Positioning System Fix Data' information.

- **Syntax**

  int32_t LMB_GPS_GetGNGGA(GGA_DATA* pstuGgaData);

- **Parameters**

  pstuGgaData

  [in] pstuGgaData->strLongitude obtains the longitude information
  'E'= East; 'W'= West, Longitude in dddmm.mmmm format.
  pstuGgaData->strLatitude obtains the latitude information
  'N'= North; 'S'= South, Latitude in ddmm.mmmm format.
  pstuGgaData->strAltitude obtains the sea level altitude in meter
  pstuGgaData->strGeoidalSeparation obtains the geoidal separation In meter
  pstuGgaData->strHDOP obtains the Horizontal dilution of precision
  pstuGgaData->strQuality obtains the GPS quality indicator
  pstuGgaData->strUTCTime obtains the time of UTC
  pstuGgaData->strSatelitesUsed obtains the number of satellites in use

- **Return Value**

`ERR_Success`: This function is working properly.
`ERR_NotSupport`: This function is not supported.
`ERR_Error`: general function error
`ERR_NotOpened`: device port not opened yet

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{ int32_t iRet;
  GGA_DATA stuGgaData ;
    LMB_DLL_Init();
    iRet = LMB_GPS_OpenPort("/dev/ttyS4");
    if ( iRet != ERR_Success ) {
      printf("Device port not exist, return %d\n", iRet);
      return -1;
    }
    iRet = LMB_GPS_GetGNGGA(&stuGgaData);
    if ( iRet != ERR_Success )  printf("LMB_GPS_GetGNGGA return %d\n", iRet);
    else {
      printf("-----------   GNGGA Data -----------------\n");
      printf("strStatus =%s\n", stuRmcData.strStatus);
      printf("strLongitude =%s\n", stuGgaData.strLongitude);
      printf("strLatitude =%s\n", stuGgaData.strLatitude);
      printf("strAltitude =%s\n", stuGgaData.strAltitude);
      printf("strGeoidalSeparation =%s\n",
      stuGgaData.strGeoidalSeparation);
      printf("strHDOP =%s\n", stuGgaData.strHDOP);
      printf("strQuality =%s\n", stuGgaData.strQuality);
      printf("strUTCTime =%s\n", stuGgaData.strUTCTime);
      printf("strSatelitesUsed =%s\n", stuGgaData.strSatelitesUsed);
    }
    LMB_GPS_ClosePort();
    LMB_DLL_DeInit();
    return 0;
}
```

LMB_GPS_GetGNVTG

The **LMB_GPS_GetGNVTG** function gets the '*Course Over Ground and Ground Speed*' information.

- **Syntax**

  int32_t LMB_GPS_GetGNVTG(VTG_DATA* pstuVtgData);

- **Parameters**

  pstuVtgData

  [in] pstuVtgData->strTrueCourse obtains the course over ground, degrees True (000.0 ~ 359.9)
  pstuVtgData->strMagneticCourse obtains the course over ground,
  degrees Magnetic (000.0 ~ 359.9)
  pstuVtgData->strSpeedKnots obtains the speed over ground in knots
  pstuVtgData->strSpeedKm obtains the speed over ground in kilometers per hour
  pstuVtgData->strMode obtains the mode indicator 'N'= not valid, 'A'= Autonomous mode
  'D'= Differential mode, 'E'= Estimated (dead reckoning) mode

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotSupport`: This function is not supported.

  `ERR_Error`: general function error

  `ERR_NotOpened`: device port not opened yet

- **Remarks**

  1 knot = 1.852 km/h

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{ int32_t iRet;
  VTG_DATA stuVtgData ;
    LMB_DLL_Init();
    iRet = LMB_GPS_OpenPort("/dev/ttyS4");
    if ( iRet != ERR_Success ) {
      printf("Device port not exist, return %d\n", iRet);
      return -1;
    }
    iRet = LMB_GPS_GetGNVTG(&stuVtgData);
    if ( iRet != ERR_Success )  printf("LMB_GPS_GetGNVTG return %d\n", iRet);
        else {
          printf("-----------  GNVTG Data -----------------\n");
          printf("strTrueCourse =%s\n", stuVtgData.strTrueCourse);
          printf("strMagneticCourse =%s\n", stuVtgData.strMagneticCourse);
          printf("strSpeedKnots =%s\n", stuVtgData.strSpeedKnots);
          printf("strSpeedKm =%s\n", stuVtgData.strSpeedKm);
          printf("strMode =%s\n", stuVtgData.strMode);
        }
    LMB_GPS_ClosePort();
    LMB_DLL_DeInit();
    return 0;
}
```

LMB_GPS_GetGNGLL

The **LMB_GPS_GetGNGLL** function gets the '*Geographic Position – Latitude/Longitude*' information.

- **Syntax**

  int32_t LMB_GPS_GetGNGLL(GLL_DATA* pstuGllData);

- **Parameters**

  pstuGllData

  [in] pstuGllData->strLongitude obtains the longitude information

  'E'= East; 'W'= West, Longitude in dddmm.mmmm format.

  pstuGllData->strLatitude obtains the latitude information

  'N'= North; 'S'= South, Latitude in ddmm.mmmm format.

  pstuGllData->strUTCTime obtains the time of UTC

  UTC time in hhmmss.sss format (000000.000 ~ 235959.999)

  pstuGllData->strStatus obtains the status of GPS information

  'V'= Navigation receiver warning, 'A'= Data Valid

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotSupport`: This function is not supported.

  `ERR_Error`: general function error

  `ERR_NotOpened`: device port not opened yet

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{ int32_t iRet;
  GLL_DATA stuGllData ;
    LMB_DLL_Init();
    iRet = LMB_GPS_OpenPort("/dev/ttyS4");
    if ( iRet != ERR_Success ) {
      printf("Device port not exist, return %d\n", iRet);
      return -1;
    }
    iRet = LMB_GPS_GetGNGLL(&stuGllData);
    if ( iRet != ERR_Success )  printf("LMB_GPS_GetGNGLL return %d\n", iRet);
    else {
      printf("-----------   GNGLL Data -----------------\n");
      printf("strLongitude =%s\n", stuGllData.strLongitude);
      printf("strLatitude =%s\n", stuGllData.strLatitude);
      printf("strUTCTime =%s\n", stuGllData.strUTCTime);
      printf("strStatus =%s\n", stuGllData.strStatus);
    }
    LMB_GPS_ClosePort();
    LMB_DLL_DeInit();
    return 0;
}
```

LMB_GPS_GetGNGSA

The **LMB_GPS_GetGNGSA** function gets the 'GNSS DOP and Active Satellites' information.

- **Syntax**

  int32_t LMB_GPS_GetGNGSA(GSA_DATA* pstuGsaData);

- **Parameters**

  pstuGsaData

  [in] pstuGsaData->strMode obtains the GPS of mode

  'M'= Manual forced 2D or 3D, 'A'= Automatic switch 2D/3D.

  pstuGsaData->strType obtains the fix type, '1'= fix not available, '2' = 2D, '3' = 3D

  pstuGsaData->aryubSatelliteID obtains the Satellite used ID list, Maximally 24 satellites

  pstuGsaData->strPDOP obtains the position dilution of precision (00.0 to 99.9)

  pstuGsaData->strHDOP obtains the horizontal dilution of precision (00.0 to 99.9)

  pstuGsaData->strVDOP obtains the vertical dilution of precision (00.0 to 99.9)

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotSupport`: This function is not supported.

  `ERR_Error:` general function error

  `ERR_NotOpened`: device port not opened yet

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{ int32_t iRet;
  GSA_DATA stuGsaData ;
    LMB_DLL_Init();
    iRet = LMB_GPS_OpenPort("/dev/ttyS4");
    if ( iRet != ERR_Success ) {
      printf("Device port not exist, return %d\n", iRet);
      return -1;
    }
    iRet = LMB_GPS_GetGNGSA(&stuGsaData);
    if ( iRet != ERR_Success )  err_printf("LMB_GPS_GetGNGSA", iRet);
    else {
        printf("-----------  GNGSA Data -----------------\n");
        if ( strcmp(stuGsaData.strType , "1") == 0)  {
            printf("<Warning> GPS Fix not available\n");
        }
        else {
            printf("strMode =%s\n", stuGsaData.strMode);
            printf("strType =%s\n", stuGsaData.strType);
            printf("strPDOP =%s\n", stuGsaData.strPDOP);
            printf("strHDOP =%s\n", stuGsaData.strHDOP);
            printf("strVDOP =%s\n", stuGsaData.strVDOP);
            printf("Active Satellites =");
            for( int xi=0 ; xi<24 ; xi++ ) {
            if( stuGsaData.aryubSatelliteID[xi]!= 255 )
                  printf(" %d,", stuGsaData.aryubSatelliteID[xi]);
            }
            printf("\n");
        }
    }
    LMB_GPS_ClosePort();
    LMB_DLL_DeInit();
    return 0;
}
```

LMB_GPS_GetGPGSV

The **LMB_GPS_GetGPGSV** function gets the 'GNSS Satellites in View' information.

- **Syntax**

  int32_t LMB_GPS_GetGPGSV(GSV_DATA* pstuGsvData);

- **Parameters**

  pstuGsvData

  [in] pstuGsvData->bGPStotal obtains the total number of GPS satellites

  pstuGsvData->bGLONASStotal obtains the total number of GLONASS satellites

  pstuGsvData->stuGPS obtains the GPS satellites information with SATELITE_INFO structure

  pstuGsvData->stuGLONASS obtains the GLONASS satellites information with SATELITE_INFO structure

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotSupport`: This function is not supported.

  `ERR_Error`: general function error

  `ERR_NotOpened`: device port not opened yet

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{ int32_t iRet;
  GSV_DATA stuGsvData ;
    LMB_DLL_Init();
    iRet = LMB_GPS_OpenPort("/dev/ttyS4");
    if ( iRet != ERR_Success ) {
      printf("Device port not exist, return %d\n", iRet);
      return -1;
    }
    iRet = LMB_GPS_GetGPGSV(&stuGsvData);
    if ( iRet != ERR_Success )  err_printf("LMB_GPS_GetGPGSV", iRet);
    else {
        printf("-----------   GPGSV Data -----------------\n");
        printf("bGPStotal =%d\n", stuGsvData.bGPStotal);
        printf("\tID\tElevation\tAzimuth\tSNR\n");
        printf("\t=============================================\n");
```

```
        for ( int xi=0 ; xi<stuGsvData.bGPStotal ; xi++) {
              printf("\t%s\t%s\t\t%s\t%s\n",
              stuGsvData.stuGPS[xi].strSateliteID,
              stuGsvData.stuGPS[xi].strElevation,
              stuGsvData.stuGPS[xi].strAzimuth,
              stuGsvData.stuGPS[xi].strSNR);
        }
        printf("bGLONASStotal =%d\n", stuGsvData.bGLONASStotal);
        printf("\tID\tElevation\tAzimuth\tSNR\n");
        printf("\t================================================\n");
        for ( int xi=0 ; xi<stuGsvData.bGLONASStotal ; xi++) {
            printf("\t%s\t%s\t\t%s\t%s\n",
            stuGsvData.stuGLONASS[xi].strSateliteID,
            stuGsvData.stuGLONASS[xi].strElevation,
            stuGsvData.stuGLONASS[xi].strAzimuth,
            stuGsvData.stuGLONASS[xi].strSNR);
        }
    }
    LMB_GPS_ClosePort();
    LMB_DLL_DeInit();
    return 0;
}
```

G-Sensor

LMB_GSR_GetAxisData

The **LMB_GSR_GetAxisData** function gets the X-Axis, Y-Axis and Z-Axis data.

- **Syntax**

  int32_t LMB_GSR_GetAxisData(AXIS_RAWDATA* pstuRawData);

- **Parameters**

  pstuGsvData

  [in] pstuRawData->wXaxis obtains the X-Axis data.

  pstuRawData->wYaxis obtains the Y-Axis data s

  pstuRawData->wZaxis obtains the Z-Axis data

  pstuRawData->wgRange obtains the X/Y/Z Axis range with ±g

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotSupport`: This function is not supported.

  `ERR_Error`: general function error

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{ int32_t iRet;
  AXIS_RAWDATA stuRawData;
    LMB_DLL_Init();
    iRet = LMB_GSR_GetAxisData(&stuRawData);
        if (  iRet == ERR_Success ) {
          printf("stuRawData.wRange= ±%dg\n", stuRawData.wgRange);
          switch (stuRawData.wgRange ) {
          case 2:
            fmgstep = (float)2/(float)255;
            break;
          case 4:
            fmgstep = (float)4/(float)255;
            break;
          case 8:
            fmgstep = (float)8/(float)255;
            break;
          case 16:
            fmgstep = (float)16/(float)255;
            break;
          }
          fXmg = (float)stuRawData.wXaxis * fmgstep;
          fYmg = (float)stuRawData.wYaxis * fmgstep;
          fZmg = (float)stuRawData.wZaxis * fmgstep;

          printf("Raw=%d\t, X-Axis= %03.8f\n", stuRawData.wXaxis, fXmg);
          printf("Raw=%d\t, Y-Axis= %03.8f\n", stuRawData.wYaxis, fYmg);
          printf("Raw=%d\t, Z-Axis= %03.8f\n", stuRawData.wZaxis, fZmg);
        }
    else  printf("LMB_GSR_GetAxisData return %d", iRet);
    LMB_DLL_DeInit();
    return 0;
}
```

LMB_GSR_GetAxisOffset

The **LMB_GSR_GetAxisOffset** function gets the X-Axis, Y-Axis and Z-Axis offset data.

- **Syntax**

  int32_t LMB_GSR_GetAxisOffset(AXIS_RAWDATA* pstuRawData);

- **Parameters**

  pstuGsvData

  [in] pstuRawData->wXaxis obtains the offset data of X-Axis.

  pstuRawData->wYaxis obtains the offset data of Y-Axis.

  pstuRawData->wZaxis obtains the offset data of Z-Axis.

  pstuRawData->wgRange not used

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotSupport`: This function is not supported.

  `ERR_Error`: general function error

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{ int32_t iRet;
  AXIS_RAWDATA stuRawData;
    LMB_DLL_Init();
    iRet = LMB_GSR_GetAxisOffset(&stuRawData);
    if (  iRet == ERR_Success ) {
          printf("Offset X-Axis=%d\n", stuRawData.wXaxis);
          printf("Offset Y-Axis=%d\n", stuRawData.wYaxis);
          printf("Offset Z-Axis=%d\n", stuRawData.wZaxis);
    }
    else  printf("LMB_GSR_GetAxisOffset return %d", iRet);
    LMB_DLL_DeInit();
    return 0;
}
```

LMB_GSR_GetRegData

The **LMB_GSR_GetRegData** function gets register raw data from the G-Sensor device.

- **Syntax**

  int32_t LMB_GSR_GetRegData(uint8_t ubReg, uint8_t* pubData)

- **Parameters**

  ubReg

  [out] assigns the register number.

  pubData

  [in] obtains the raw data of G-Sensor register.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotSupport`: This function is not supported.

  `ERR_Error`: general function error

- **Remarks**

  About the registers, please refers to "G-Sensor Device ADXL345)" in Chapter 2

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{ int32_t iRet;
    LMB_DLL_Init();
    iRet = LMB_GSR_GetRegData(GSR_DEVID, &ubData);
    if (  iRet == ERR_Success ) printf("GSR_DEVID = 0x%02X\n", ubData);
    else   printf("LMB_GSR_GetRegData return %d\n", iRet);
    LMB_DLL_DeInit();
    return 0;
}
```

LMB_GSR_SetRegData

The **LMB_GSR_SetRegData** function sets register raw data to the G-sensor device.

- **Syntax**

  int32_t LMB_GSR_SetRegData(uint8_t ubReg, uint8_t ubData)

- **Parameters**

  ubReg

  [out] assigns the register number.

  ubData

  [out] writes data to the G-Sensor register.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotSupport`: This function is not supported.

  `ERR_Error`: general function error

- **Remarks**

  About the registers, please refer to "G-Sensor Device ADXL345)" in Chapter 2.

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{ int32_t iRet;
    LMB_DLL_Init();
    iRet = LMB_GSR_SetRegData(GSR_OFSX, ubData);
    if (iRet == ERR_Success) printf("Write GSR_OFSX 0x%02X ---> OK\n", ubData);
    else    printf("LMB_GSR_SetRegData return %d\n", iRet);
    LMB_DLL_DeInit();
    return 0;
}
```

CAN Bus Module

LMB_CAN_OpenPort

The **LMB_CAN_OpenPort** function opens the CAN device port with 9600 bps.

- **Syntax**

  int32_t LMB_CAN_OpenPort(int8_t* pbPortPath);

- **Parameters**

  pbPortPath

  [out] assigns the CAN device path

- **Return Value**

`ERR_Success`: This function is working properly.

`ERR_NotSupport`: This function is not supported.

`ERR_NotExist`: The device does not exist.

`ERR_Error`: general function error

- **Remarks**

  None

- **Example**

  Please refer to the LMB_CAN_xxx functions.

LMB_CAN_ClosePort

The **LMB_CAN_closePort** function closes the CAN device port

- **Syntax**

  int32_t LMB_CAN_OpenPort(void);

- **Parameters**

  None

- **Return Value**

`ERR_Success`: This function is working properly.

`ERR_NotSupport`: This function is not supported.

`ERR_NotExist`: The device does not exist.

`ERR_Error`: general function error

- **Remarks**

  None

- **Example**

  Please refer to the LMB_CAN_xxx functions.

LMB_CAN_ModuleInfo

The **LMB_CAN_ModuleInfo** function gets the version information of the CAN module.

- **Syntax**

  int32_t LMB_CAN_ModuleInfo(int8_t* strbModelInfo);

- **Parameters**

  strbModelInfo

  [in] obtains the version information of the CAN module.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotSupport`: This function is not supported.

  `ERR_Error`: general function error

  `ERR_NotOpened`: device port not opened yet

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{ int32_t iRet;
  int8_t strbModuleInfo[30];
    LMB_DLL_Init();
    iRet = LMB_CAN_OpenPort("/dev/ttyS5");
    if ( iRet != ERR_Success ) {
      printf("Device port not exist, return %d\n", iRet);
      return -1;
    }
    iRet = LMB_CAN_ModuleInfo(strbModulelInfo);
    if ( iRet == ERR_Success )  printf("CAN Module = %s\n", strbModuleInfo);
    else   printf("LMB_CAN_ModuleInfo return %d\n", iRet);
    LMB_CAN_Close();
    LMB_DLL_DeInit();
    return 0;
}
```

LMB_CAN_DrivingInfo

The **LMB_CAN_DrivingInfo** function gets the driving information from the CAN module.

- **Syntax**

  int32_t LMB_CAN_DrivingInfo(DRIVING_INFO* pstuDrivingInfo);

- **Parameters**

  pstuDrivingInfo

  [in] pstuDrivingInfo->ubSpeedKm obtains the car speed .

  pstuDrivingInfo->ubAcceleratorPedal obtains the “Accelerator Pedal Position”.

  pstuDrivingInfo->ubFuelLevel obtains the fuel level.

  pstuDrivingInfo->fTotalDistance otbains the “High Resolution Total Vehicle Distance”

  pstuDrivingInfo->dwSERV obtains the “The distance which can be traveled”

  pstuDrivingInfo->ubTachographKm obtains the “tachograph vehicle speed”

  pstuDrivingInfo->fBatteryVolt obtains the battery voltage.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotSupport`: This function is not supported.

  `ERR_Error`: general function error

  `ERR_NotOpened`: device port not opened yet

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{ int32_t iRet;
  DRIVING_INFO stuDrivingInfo;
    LMB_DLL_Init();
    iRet = LMB_CAN_OpenPort("/dev/ttyS5");
    if ( iRet != ERR_Success ) {
      printf("Device port not exist, return %d\n", iRet);
      return -1;
    }
    iRet = LMB_CAN_DrivingInfo(&stuDrivingInfo);
      if ( iRet == ERR_Success )  {
        printf("-----> LMB_CAN_DrivingInfo\n");
        printf("ubSpeedKm = %dKm\n", stuDrivingInfo.ubSpeedKm);
        printf("ubAcceleratorPedal = %d%%\n",
                     stuDrivingInfo.ubAcceleratorPedal);
        printf("ubFuelLevel = %d%%\n", stuDrivingInfo.ubFuelLevel);
        printf("fTotalDistance = %.3fKm\n", stuDrivingInfo.fTotalDistance);
        printf("dwSERV = %dKm\n", stuDrivingInfo.dwSERV);
        printf("ubTachographKm = %dKm\n", stuDrivingInfo.ubTachographKm);
        printf("fBatteryVolt = %.1fV\n", stuDrivingInfo.fBatteryVolt);
      }
      else       printf("LMB_CAN_DrivingInfo return %d\n", iRet);
    LMB_CAN_Close();
    LMB_DLL_DeInit();
    return 0;
}
```

LMB_CAN_EngineInfo

The **LMB_CAN_EngineInfo** function gets the engine information from the CAN module.

- **Syntax**

  int32_t LMB_CAN_EngineInfo(ENGINE_INFO* pstuEngineInfo);

- **Parameters**

  pstuDrivingInfo

  [in] pstuEngineInfo->uwEngineRPM obtains the engine rpm speed .
  pstuEngineInfo->ubEngineLoading obtains the “Engine Percent Load At Current Speed”.
  pstuEngineInfo->wCoolantTemp otbains the “Engine Coolant Temperature(ECT)”
  pstuEngineInfo->wEIMT obtains the “Engine Intake Manifold 1 Temperature”
  pstuEngineInfo->udwTotalFuels obtains the “Engine Total Fuel used”
  pstuEngineInfo->udwTotalHours obtains the “Engine total hours of Operation”.
  pstuEngineInfo->uwETBP obtains the “Engine Turbocharger Boost Pressure.”
  pstuEngineInfo->fEFR obtains the “Engine Fuel Rate.”
  pstuEngineInfo->fEIFE obtains the “Engine Instantaneous Fuel Economy.”
  pstuEngineInfo->fEODP obtains the “Engine Oil Filter Differential Pressurevoltage.”

- **Return Value**

  `ERR_Success`: This function is working properly.
  `ERR_NotSupport`: This function is not supported.
  `ERR_Error`: general function error
  `ERR_NotOpened`: device port not opened yet

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{ int32_t iRet;
  ENGINE_INFO stuEngineInfo;
    LMB_DLL_Init();
    iRet = LMB_CAN_OpenPort("/dev/ttyS5");
    if ( iRet != ERR_Success ) {
      printf("Device port not exist, return %d\n", iRet);
      return -1;
    }
    iRet = LMB_CAN_EngineInfo(&stuEngineInfo);
      if ( iRet == ERR_Success )  {
        printf("-----> LMB_CAN_EngineInfo\n");
        printf("uwEngineRPM = %d rpm\n", stuEngineInfo.uwEngineRPM);
        printf("ubEngineLoading = %d%%\n", stuEngineInfo.ubEngineLoading);
        printf("wCoolantTemp = %d degC\n", stuEngineInfo.wCoolantTemp);
        printf("wEIMT = %d degC\n", stuEngineInfo.wEIMT);
        printf("udwTotalFuels = %d Liter\n", stuEngineInfo.udwTotalFuels);
        printf("udwTotalHours = %d Hours\n", stuEngineInfo.udwTotalHours);
        printf("uwETBP = %d kPa\n", stuEngineInfo.uwETBP);
        printf("fEFR = %.3f Liter/Hour\n", stuEngineInfo.fEFR);
        printf("fEIFE = %.3f Km/Liter\n", stuEngineInfo.fEIFE);
        printf("fEODP = %0.3f kPa\n", stuEngineInfo.fEODP);

    }
    else         erintf("LMB_CAN_EngineInfo return %d\n", iRet);
    LMB_CAN_Close();
    LMB_DLL_DeInit();
    return 0;
}
```

LMB_CAN_VehicleID

The **LMB_CAN_VehicleID** function gets the Vehicle identification number from the CAN module.

- **Syntax**

  int32_t LMB_CAN_VehicleID(int8_t* strbVehicleID);

- **Parameters**

  strbVehicleID

  [in] obtains the Vehicle identification number.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotSupport`: This function is not supported.

  `ERR_Error`: general function error

  `ERR_NotOpened`: device port not opened yet

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{ int32_t iRet;
  int8_t strbVehicleID[30];
    LMB_DLL_Init();
    iRet = LMB_CAN_OpenPort("/dev/ttyS5");
    if ( iRet != ERR_Success ) {
      printf("Device port not exist, return %d\n", iRet);
      return -1;
    }
    iRet = LMB_CAN_VehicleID(strbVehicleID);
    if ( iRet == ERR_Success )  printf("strbVehicleID=%s\n", strbVehicleID);
    else        printf("LMB_CAN_VehicleID return %d\n", iRet);
    LMB_CAN_Close();
    LMB_DLL_DeInit();
    return 0;
}
```

LMB_CAN_RawData

The **LMB_CAN_RawData**function gets the all raw data from the CAN module.

- **Syntax**

  int32_t LMB_CAN_ RawData(uint8_t* aryubRawData, uint8_t* pubAmount);

- **Parameters**

  aryubRawData

  [in] obtains the all raw data.

  pubAmount

  [in] obtains the total byte counter of the raw data.

- **Return Value**

  `ERR_Success`: This function is working properly.

  `ERR_NotSupport`: This function is not supported.

  `ERR_Error`: general function error

  `ERR_NotOpened`: device port not opened yet

- **Remarks**

  (None)

- **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include "lmbinc.h"
int main(int argc, char* argv[])
{ int32_t iRet, xindex=0;
  uint8_t aryubRawData[256], ubAmount=0;
    LMB_DLL_Init();
    iRet = LMB_CAN_OpenPort("/dev/ttyS5");
    if ( iRet != ERR_Success ) {
      printf("Device port not exist, return %d\n", iRet);
      return -1;
    }
    iRet = LMB_CAN_RawData(aryubRawData, &ubAmount);
      if ( iRet == ERR_Success )  {
          for (int xj=0; xj<ubAmount ; xj++ ) {
              printf("%02X ",aryubRawData[xj]);
              xindex++;
              if ( xindex >= 18 ) {printf("\n");   xindex=0;  }
            }
            if ( xindex !=0 ) printf("\n");
        }
        else        printf("LMB_CAN_RawData return %d\n", iRet);
```

```
    LMB_CAN_Close();
    LMB_DLL_DeInit();
    return 0;
}
```

APPENDIX

Parameter & Variable Naming Rule

bValue : 8-bit size variable with sign
ubValue: 8-bit size variable without sign
wValue: 16-bit size variable with sign
uwValue: 16-bit size variable without sign
dwValue: 32-bit size variable with sign
udwValue: 32-bit size variable without sign
fValue: float type variable with sign
dlValue: 64-bit size variable with sign
udlValue: 64-bit size variable without sign

pbValue : pointer of 8-bit size variable with sign
pubValue: pointer of 8-bit size variable without sign
pwValue: pointer of 16-bit size variable with sign
puwValue: pointer of 16-bit size variable without sign
pdwValue: pointer of 32-bit size variable with sign
pudwValue: pointer of 32-bit size variable without sign
pfValue: float type variable with sign
pdlValue: pointer of 64-bit size variable with sign
pudlValue: pointer of 64-bit size variable without sign

strbString: 8bit with sign string or array pointer
strubString: 8bit without sign string or array pointer
strwString: 16bit with sign string or array pointer
struwString: 16bit without sign string or array pointer
strdwString: 32bit with sign string or array pointer
strudwString: 32bit without sign string or array pointer

stuStruct: structure name
pstuStruct: pointer of structure name

LCD Module Character Table

NO.7066-0A