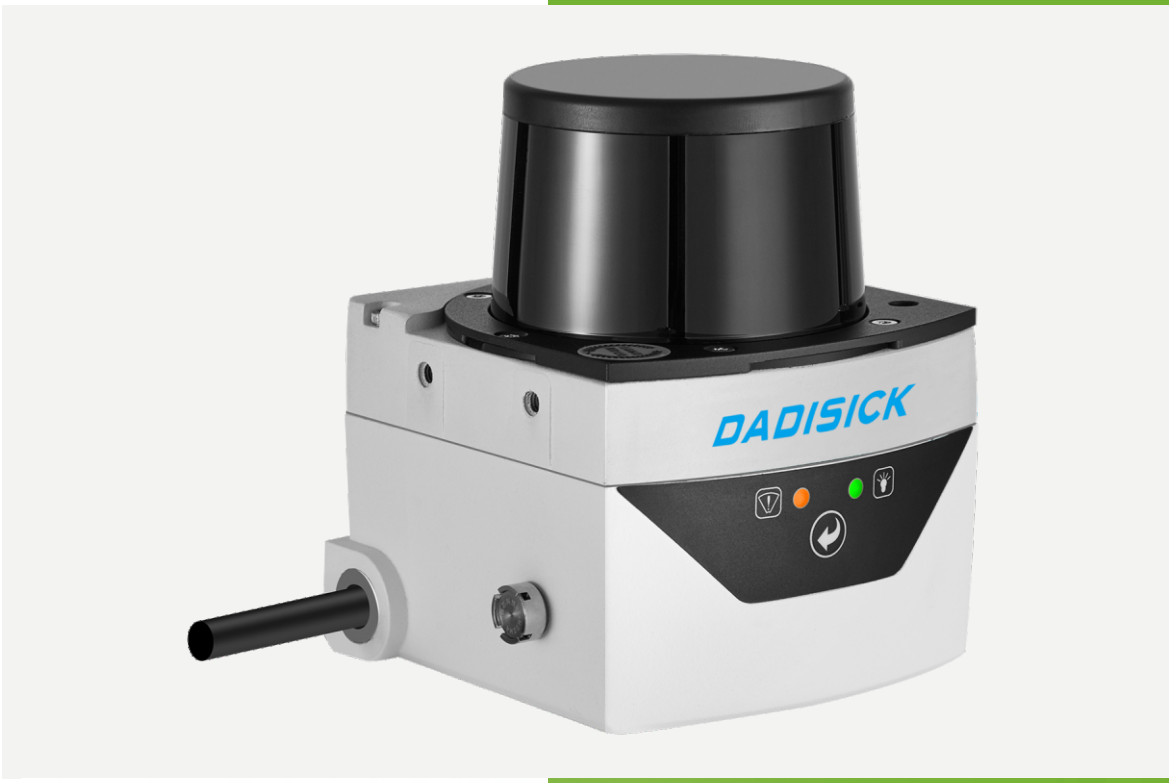
DADISICK®

SDK User Manual



V.202504

Contents

1. Document description

1.1. Prompt message

This manual provides the use and attention of the DADISICK lidar application and development of SDK. In order to be able to use this SDK safely and correctly, the user should also pay attention to:

- Complete the installation and electrical connection of DADISICK lidar products correctly;
- Follow the safety operation rules and general safety rules for lidar products in the workplace;
- The operating parameters of DADISICK lidar products are correctly configured by using lidar diagnosis and configuration software (FLIPS).

The manual is for an application development engineer

Important point	Before using this SDK to develop the DADISICK lidar products, please read the lidar product manual and the use manual carefully, familiar with the characteristics and functions of the product.

1.2. SDK description

The application development of SDK is used to develop the software of DADISICK lidar application system.

Important point	The basic equipment configuration and built-in functional configuration of DADISICK lidar products are all factory default settings. Users needs to do new setup per the actual application requirements. These configurations are checked and adjusted using FLIPS to ensure the correct operation of the lidar.

Application development SDK can develop application software including Windows, Linux and compatible system, in which the basic C++ code conforms to ANSI C++ specifications. TCP/IP communication uses IPv4 address specifications.

The example code (TryLIM) in application development SDK requirements for the runtime environment are as follows:

- operating system (OS) :
  - ✧ Microsoft Windows 7 / 8.1 / 10 (32/64 bit Chinese or English operating

system)

- ✧ Windows XP SP3 （32 bit Chinese or English operation system)
- ✧ Windows Server 2008 （64 bit Chinese or English operation system)

- ➢ Exploitation environment：Microsoft Visual Studio 2013，You can also create other development projects under the development environment according to the source code files of LIM and TryLIM.
- ➢ CPU：Intel Pentium4.3.0@1.4 GHz or above；
- ➢ Memory：1 GB or higher；
- ➢ Video Memory：256MB or higher；
- ➢ Display resolution：1024*768 or higher.

1.3. Description convention

In this manual, the following agreements have been made to simplify the narrative:

- ➢ Lidar: DADISICK Technology Limited the measurement of lidar and intelligent obstacle avoidance sensor products;
- ➢ SDK: DADISICK Technology Limited to provide the application of lidar development SDK;
- ➢ Equipment end: the lidar connected by the application software;
- ➢ Application: application software.

1.4. Application Scope of SDK

The FLIPS software features described in this manual are suitable for the following lidar products:

- ➢ ELDS2030B5-5S

1.5. Context help

This manual is intended to provide software developers with the use and precautions of SDK. Please read the chapters of this manual in sequence. The contents of this manual (in order) include:

- SDK installation and use
- SDK Overview
- LIM Overview
- Measurement data message
- Regional monitoring message
- I / O message
- Running status message
- Device configuration message
- LIM operation routine
- LIM TCP communication library

1.6. Pattern symbol description

This manual uses the following pictorial symbols to identify important considerations that require special attention when reading.

Cautious operation	Meaning : A potentially dangerous situation, if not prevented, may cause general personal injury.
Attention	Meaning : Potentially harmful conditions, if not prevented, may cause equipment damage.
Important point	Meaning : Useful advice and tips for efficient and smooth use of equipment and software.
	Meaning :

Important point	Information about important features of equipment and software.

Explanation	Meaning : Background on technical issues

Description	Meaning : Additional information

Related Reading	Meaning : Documentation that provides more information.

2. SDK installation and use

The installation and use of SDK are as follows：

- Unzip the installation package under the work path by pressing "extract to the current folder", where the "TryLIM" directory appears and the SDK user manual ;
- Prepare the lidar, configure the lidar with FLIPS according to the application requirements, and write down the IP address of the lidar. ;
- Go to the "TryLIM" directory and see the Visual Studio 2013 project file "TryLIM.sln" for TryLIM. Open the project file with Visual Studio 2013 or above ;
- Open "TryLIM.CPP" in Visual Studio 2013：
  - Change the following code to the number of I / O ports of the lidar in use：

    ```
    #define IO_OUTNUM    4
    #define IO_INNUM     4
    ```

  - Change the following code to the IP address of the lidar in use：

    ```
    char *szIP = "192.168.1.201";
    ```

- To generate and run the TryLIM project, TryLIM connects the laser lidar used and displays the output on the console. ;
- Enter the "TryLIM" directory under the "TryLIM" directory, you can see the following three folders：
  - LIM：Including lim.h and lim.cpp. for the lidar network message LIM format definition and function routine, used to develop application software on the application system platform ;

EquipmentComm：For the Windows version of the LIM TCP/IP communication library, the function is to establish and manage the TCP connection with the lidar, as well as the analysis of the LIM message. Includes the API definition header file, EquipmentComm.h and the static input library EquipmentComm.lib. And dynamic link library EquipmentComm.dll. for rapid development of applications on the Windows platform.

3. SDK Overview

Important point

- ✧ This chapter gives a brief description of the concepts, functions, usage methods and important terms of SDK. Please read this chapter carefully before using SDK.
- ✧ Before using this SDK to develop the DADISICK lidar products, please read the lidar product manual and the use manual carefully, familiar with the characteristics and functions of the product.

In order to facilitate the rapid prototyping development of user application, we also provide LIM TCP communication library in the form of Windows dynamic link library. This document gives a complete description of the API of this library. Users can choose to use this library, or they can write their own TCP communication code, and use LIM messages to interact with lidar.

SDK uses the ANSI C++ language for six files, including：

Table 3.1 SDK file description

Content	Path	File	Explain
LIM Message	TryLIM/LIM	lim.h	LIM Message format definition
		lim.cpp	LIM routine
LIM TCP Communication library	TryLIM/EquipmentComm	EquipmentComm.h	API
		EquipmentComm.lib	derive library file
		EquipmentComm.dll	dynamic link library
TryLIM	TryLIM	TryLIM.CPP	LIM sample programs
	.	TryLIM.sln	Sample program VS13.0 project file

When using, you need to introduce header files and export library files according to the path you actually store. The dynamic link library files and the final executable files can be placed in the same directory；

Acronym contrast：

- ✧ **LIM :** Lidar Interaction Message
- ✧ **LMD :** Lidar Measurement Data
- ✧ **RSSI :** Received Signal Strength Indication ---The RSSI data of the measured target reflect the ability of the target to reflect the laser pulse emitted by the lidar.

4. LIM overview

The format of **LIM** is defined in lim.h and its basic structure is as follows

LIM_HEAD	Extended_Data(Extended data, optional)

Where **LIM_HEAD** is the basic component of LIM, and the data structure is as follows:

```
#define LIM_TAG                 0xF5EC96A5      // Header identification code
#define LIM_VER                 0x01000000      // Message version
#define LIM_DATA_LEN             4              //  LIM_HEAD basic data length in 32-bit WORD

// LIM_HEAD structure
typedef struct
{
    unsigned int   TAG;                 // Header identification code
    unsigned int   VER;                 // Message version
    unsigned int   nCID;                // Link number, as described below
    unsigned int   nCode;               // message code
    unsigned int   Data[LIM_DATA_LEN]; // Message basic data
    unsigned int   nLIMLen;             // The total length of the message, including extended numbers, is shown below
    unsigned int   CheckSum;            // check sum
} LIM_HEAD;
```

nCID explanation：

1. When multiple clients or applications connect to a single lidar at the same time, **nCID** is the unique and invariant **ID** of each client's **TCP** connection；
2. After establishing a **TCP** connection with the lidar, the client specifies its corresponding **nCIDs** in the **LIM** message sent to the lidar, which is also used for each message returned by the lidar to the client.。

```
// LIM Code：message code，Carried by LIM_HEAD.nCode
#define LIM_CODE_HB                 10      // heartbeat
#define LIM_CODE_HBACK              11      // Heartbeat recovery

#define LIM_CODE_LMD                901     // LMD：Lidar measurement data
#define LIM_CODE_LMD_RSSI           911     // LMD-RSSI：Lidar measurement data with
                                            reflectivity
#define LIM_CODE_START_LMD          1900    // Start LMD
#define LIM_CODE_STOP_LMD           1902    // Stop LMD

#define LIM_CODE_FMSIG_QUERY        1912    // Query area monitoring signal
#define LIM_CODE_FMSIG              1911    // Regional monitoring signal

#define LIM_CODE_IOREAD             1920    // Read the I / O terminal state
#define LIM_CODE_IOSET              1922    // Setting the I/O terminal status (only for
                                            the output terminal)
#define LIM_CODE_IOSET_RELEASE      1924    // Deactivate the I / O terminal state
#define LIM_CODE_IOSTATUS           1921    // I / O state

#define LIM_CODE_ALARM              9001    // Running state alarm
#define LIM_CODE_DISALARM           9003    // State alarm release

#define LIM_CODE_LDBCONFIG          111     // Device configuration information
#define LIM_CODE_START_LDBCONFIG    110     // Initiate device configuration
information broadcast
#define LIM_CODE_STOP_LDBCONFIG     112     // Stop broadcast of device
configuration information
#define LIM_CODE_GET_LDBCONFIG      114     // Query device configuration
information
```

The LIM types supported by SDK are shown in Table 4.1 LIM.

Table 4.1 LIM

classify	Message code	Meaning	Transmitter	Response Message
Heartbeat	LIM_CODE_HB	Heartbeat	client	LIM_CODE_HBACK
	LIM_CODE_HBACK	Heartbeat recovery	device	None
LMD data	LIM_CODE_LMD	Measured data	device	None
	LIM_CODE_LMD_RSSI	With RSSI measurement data	device	None
	LIM_CODE_START_LMD	Start LMD	client	LIM_CODE_LMD LIM_CODE_LMD_RSSI
	LIM_CODE_STOP_LMD	Stop LMD	client	None
Area monitor	LIM_CODE_FMSIG_QUERY	Query area monitoring signal	client	LIM_CODE_FMSIG
	LIM_CODE_FMSIG	Regional monitoring signal	device	None
I/O	LIM_CODE_IOREAD	Read the I /O terminal state	client	LIM_CODE_IOSTATUS
	LIM_CODE_IOSET	Setting the I/O terminal status (only for the output terminal)	client	LIM_CODE_IOSTATUS

	LIM_CODE_IOSET_RELEASE	Deactivate the I / O terminal state	client	LIM_CODE_IOSTATUS
	LIM_CODE_IOSTATUS	I / O state	device	None
Running status	LIM_CODE_ALARM	Running status alarm	device	None
	LIM_CODE_DISALARM	State alarm release	device	None
Equipment configure	LIM_CODE_LDBCONFIG	Device configuration information	device	None
	LIM_CODE_START_LDBCONFIG	Initiate device configuration information broadcast	client	None
	LIM_CODE_STOP_LDBCONFIG	Stop broadcast of device configuration information	client	None
	LIM_CODE_GET_LDBCONFIG	Query device configuration information	client	LIM_CODE_LDBCONFIG

point

- ✧ The TCP SOCKET(2112) port number used by lidar to interact with the application system is Lim ;
- ✧ The application needs to send a heartbeat message (within 5 seconds) to the radar device and receive a

	heartbeat response; if the heartbeat is not sent in time, the device will interrupt the TCP connection ; ✧ heartbeat processing is embedded in the communication library provided by the SDK , and if the application software uses the communication library to communicate with the Lidar , the heartbeat processing is not required。

5. Measurement data message

Message

classify	message code	Meaning	Transmitter	Response Message
LMD data	LIM_CODE_LMD	Measurement data	device	None
	LIM_CODE_LMD_RSSI	Measurement data with RSSI	device	None
	LIM_CODE_START_LMD	Start LMD	client	LIM_CODE_LMD LIM_CODE_LMD_RSSI
	LIM_CODE_STOP_LMD	Stop LMD	client	None

Explanation：

- After receiving the LIM_CODE_START_LMD message from the application side, the device side starts sending LIM_CODE_LMD or LIM_CODE_LMD_RSSI message to the application side continuously.
- Device receives LIM_CODE_STOP_LMD message from client and stops sending LIM_CODE_LMD orLIM_CODE_LMD_RSSI to client ;
- The measured data of LIM_CODE_LMD and LIM_CODE_LMD_RSSI are carried by Extended_Data.

Message structure

LIM_HEAD					Extended Data
nCode	Data				
	0	1	2	3	
LIM_CODE_LMD	0	0	0	0	**LMD_INFO** + LMD Measured data
LIM_CODE_LMD_RSSI	0	0	0	0	**LMD_INFO** + LMD_RSSI Measured data
LIM_CODE_START_LMD	0	0	0	0	None
LIM_CODE_STOP_LMD	0	0	0	0	None

LMD_INFO

```
//LMD_INFO structure
typedef struct
{
    unsigned int    nRange;          // range。Unit: cm
    int             nBAngle;         // The starting angle of the measured data, may be negative.
Unit: 1 / 1000 degrees
    int             nEAngle;         // The end angle of measuring data, may be negative . Unit : 1 /
1000 degrees
    unsigned int    nAnglePrecision; // Angular accuracy. Unit 1: 1000 degrees.
    unsigned int    nRPM;            // Scanning frequency. Unit: RPM (r / min)
    unsigned int    nMDataNum;       // Measure the number of data,
                                     Calculated according to nBAnglen, EAngle and nAnglePrecision.
} LMD_INFO;

typedef unsigned short LMD_D_Type;              // Measuring distance data. Unit: cm
typedef unsigned short LMD_D_RSSI_Type;         //The RSSI value of the measured target is 0-1000)
```

Explanation：

1. The calculation of nMDataNum is as follows：

   nMDataNum = (nEAngle - nBAngle)/nAnglePrecision + 1

LMD measurement data

LMD_D_Type, …, LMD_D_Type

An array of distance measurement data of type LMD_D_Type for LMD_INFO. nMDataNum, the scanning angle corresponding to the i^{th} measurement data is:

LMD_INFO.nBAngle + i* LMD_INFO.nAnglePrecision

i = 0, …, LMD_INFO. nMDataNum-1

LMD_RSSI measurement data

LMD_D_Type, …, …LMD_D_Type	LMD_D_RSSI_Type, …, …LMD_D_RSSI_Type

An array of distance measurement data of type LMD_D_Type for LMD_INFO. nMDataNum. An array of RSSI measurement data of type LMD_D_RSSI_Type with a scan angle corresponding to the i^{th} measurement data of type LMD_INFO. nMDataNum is:

LMD_INFO.nBAngle + i* LMD_INFO.nAnglePrecision

i = 0, …, LMD_INFO. nMDataNum-1

6. Regional monitoring message

Message

classify	Message code	Meaning	transmitter	Response Message
area monitor	LIM_CODE_FMSIG_QUERY	Query area monitoring signal	client	LIM_CODE_FMSIG
	LIM_CODE_FMSIG	Area monitoring signal	device	None

Explanation：

- Device receives LIM_CODE_FMSIG_QUERY message sent by client and then responds to LIM_CODE_FMSIG message to client;
- In below situation，Device sends LIM_CODE_FMSIG messages to client actively:
  - The state of the monitoring signal is changed because of the change of scene or target;
  - Due to forced control (withdrawal / forced alarm) caused by the "alarm" monitoring signal state change.

Message structure

LIM_HEAD					Extended Data
nCode	Data				
	0	1	2	3	
LIM_CODE_FMSIG_QUERY	Number of regional groups[1]	0	0	0	None
LIM_CODE_FMSIG	Number of regional groups[1]	bit0: Alarm signal status[2] bit1: General status of alarm signal[3] bit2: Early Warning Signal Status[2] bit3: General status of early warning signal status[3] bit4: Attention signal status[2] bit5: Attention the total state of the signal[3] bit6~bit31: 0	0	0	None

Explanation:

1. Number of regional groups starting from 0;
2. "0" means the signal is invalid, "1" means the signal is valid;
3. The total state of the corresponding monitoring signal for all activated monitoring area groups ("or") of all states.

7. I/O Message

Message

Classify	Message code	Meaning	transmitter	Response Message
I/O	LIM_CODE_IOREAD	Read the I / O terminal state	client	LIM_CODE_IOSTATUS
	LIM_CODE_IOSET	Setting the I/O terminal status (only for the output terminal)	client	LIM_CODE_IOSTATUS
	LIM_CODE_IOSET_RELEASE	Deactivate the I / O terminal state	client	LIM_CODE_IOSTATUS
	LIM_CODE_IOSTATUS	I / O terminal state	device	None

Explanation：

- After device receives the LIM_CODE_IOSET message from client, it performs a setup operation on the I / O output port. In response to the LIM_CODE_FMSIG message to the client, the area monitoring function suspends the output of the monitoring signal through the I / O output port. The control of the I / O output port is controlled by client over the network until the LIM_CODE_IOSET_RELEASE message sent by client is received. The area monitoring function recovers and outputs the monitoring signal through the I / O output port;
- In below situation, Device initiatively sends LIM_CODE_IOSTATUS message to client:
  - The input state of the I / O input port changes;
  - The state of the I / O output port is caused by the monitoring signal output by the region monitoring function through the I / O output port. Including the "alarm" monitoring signal state change caused by forced control (withdrawal / mandatory alarm), so that the state of the OUT2A changes.
  - After device receives the LIM_CODE_IOSET_RELEASE message, the area monitoring function resumes to output the monitoring signal through the I / O

output port. And causes a change in the state of the I / O output port.

Message Structure

LIM_HEAD					Extended Data
nCode	Data				
	0	1	2	3	
LIM_CODE_IOREAD	0	0	0	0	None
LIM_CODE_IOSET	Port Status[1]	Output port number[2]	0	0	None
LIM_CODE_IOSET_RELEASE	0	0			
LIM_CODE_IOSTATUS	IO status[3]	0			

Explanation：

1. "0" means "off / low level" and "1" means "through / high level";
2. The output port number starts at 0;
3. I/O status：
   - ✧ bit_i：Output port state，i = 0, …, OUT_Num-1，OUT_Num is number of output ports；
   - ✧ bit_j：Output port state，j = OUT_Num, …, OUT_Num+ IN_Num -1，IN_Num is number of input ports；
   - ✧ bit_k：0，k = OUT_Num+ IN_Num, …, 31。

8. Running status message

Message

classify	Message code	Meaning	transmitter	Response Message
Running status	LIM_CODE_ALARM	Running state alarm	device	None
	LIM_CODE_DISALARM	State alarm release	device	None

Explanation：

- The device sends theLIM_CODE_ALARM message to the client actively when the alarm occurs in the running state, and each kind of alarm sends the alarm message separately. When the alarm is lifted, the LIM_CODE_DISALARM message is sent to the client, and each type of alarm sends the alarm release message separately.

Message Structure

LIM_HEAD					Extended Data
nCode	Data				
	0	1	2	3	
LIM_CODE_ALARM	Alarm code	0	0	0	None
LIM_CODE_DISALARM	Alarm code	0	0	0	None

Alarm code

```
#define LIM_DATA_ALARMCODE_INTERNAL          1      // internal error
#define LIM_DATA_ALARMCODE_Occluded          101    // The lens hood are blocked or too dirty.
#define LIM_DATA_ALARMCODE_High_Temperature  1001   // High temperature alarm
#define LIM_DATA_ALARMCODE_Low_Temperature   1002   // Low temperature alarm
```

9. Device configuration message

Message

classify	Message code	Meaning	transmitter	Response Message
equipment configure	LIM_CODE_LDBCONFIG	Device configuration information	device	None
	LIM_CODE_START_LDBCONFIG	Initiate device configuration information broadcast	client	None
	LIM_CODE_STOP_LDBCONFIG	Stop broadcast of device configuration information	client	None
	LIM_CODE_GET_LDBCONFIG	Query device configuration information	client	LIM_CODE_LDBCONFIG

Point

When the Ethernet port of the lidar is switched on. Actively broadcast their configuration information to the broadcast address LIM_DT_IP (237.1.1.200) and the UDP port numberLIM_DT_PORT (2111) , LIM_DT_IP and LIM_DT_PORT are defined in lim.h.

Message Structure

LIM_HEAD					Extended Data
nCode	Data				
	0	1	2	3	
LIM_CODE_LDBCONFIG	0	0	0	0	ULDINI_Type
LIM_CODE_START_LDBCONFIG	0	0	0	0	None
LIM_CODE_STOP_LDBCONFIG	0	0	0	0	None

LIM_CODE_GET_LDBCONFIG	0	0	0	0	None

ULDINI_Type

```
#define ULDINI_MAX_ATTR_STR_LEN                    0x20        // Length of character array in
ULDINI_Type // ULDINI_Type structure
typedef struct
{
    // Product information
    char szType[ULDINI_MAX_ATTR_STR_LEN];          // Product model
    char szManufacturer[ULDINI_MAX_ATTR_STR_LEN];  // Manufacturer
    char szReleaseDate[ULDINI_MAX_ATTR_STR_LEN];   // Date of manufacture
    char szSerialNo[ULDINI_MAX_ATTR_STR_LEN];      // Serial Number
    // line configuration
    char szMAC[ULDINI_MAX_ATTR_STR_LEN];       // MAC address
    char szIP[ULDINI_MAX_ATTR_STR_LEN];        // IP address
    char szMask[ULDINI_MAX_ATTR_STR_LEN];      // Subnet mask
    char szGate[ULDINI_MAX_ATTR_STR_LEN];      // Default gateway
    char szDNS[ULDINI_MAX_ATTR_STR_LEN];       // DNS server
    // measurement parameters
    int nMR;            // Range
    int nESAR;          // Equipment angle range
    int nESA[2];        // Device angle starting termination angle
    int nSAR;           // Effective angle range
    int nSA[2];         // effective angle of initial termination angle
    int nSAV;           // Scanning angular velocity (degree / s)
    int nSAP;           // Scanning angle resolution
    int nPF;            // Measuring frequency (HZ)
} ULDINI_Type;
```

10. LIM operation routine

Point	The LIM operation routine in SDK is also a basic example program for application development using LIM. Reading carefully is helpful to speed up the development.

SDK provides some basic **LIM** operation routines, the source code see lim.cpp / lim.h these routines are briefly described in this section.

```
unsigned int LIM_CheckSum(LIM_HEAD * _lim);

// Calculate the checksum for the LIM package .
Parameter：
    _lim：lim message
Return value：
    proof test value
```

Explanation：

1. When **LIM** groups packets and sends them, it is used to calculate the **CheckSum** field;

2. When **LIM** packets are received, they are used to check **LIM** packets.

```
void* LIM_ExData(LIM_HEAD* _lim);

//Gets the memory address of the Extended_Data of the LIM packets
Parameter：
    _lim:lim message
Returned value  ：
    Extended_Data pointer
```

Explanation：

1. When **LIM** exists in Extended_Data, such as **LMD** data, you can use LIM_ExData to obtain **LMD** measurements
2. **LIM_HEAD** and **Extended_Data** are contiguous on memory addresses. So the memory address of the **Extended_Data** is equivalent to the memory address of the(void*)(_lim + 1).

```
LMD_INFO* LMD_Info(LIM_HEAD* _lim);

//Get the memory address of the LMD_INFO
Parameter：
    _lim：lim message
Return value：
    LMD_INFO pointer
```

```
LMD_D_Type* LMD_D(LIM_HEAD* _lim);
//Gets the memory address of the measurement data
Parameter：
    _lim：lim message
return value：
    LMD_D_Type data pointer
```

Explanation：

1. For the **LMD** message, the address of the **LMD_INFO** is equal to (LMD_INFO*)LIM_ExData(_lim) ;
2. The memory address of the measurement data of the **LMD** message is the same as the (LMD_D_Type*)(LMD_Info(_lim) + 1).

```
bool LIM_Pack(LIM_HEAD*& _lim, unsigned int_cid, unsigned int _code, unsigned int* _data = NULL,
                    unsigned int_ext_data_len = 0, void* _ext_data = NULL);
//LIM packet group, often used for LIM message sending
Parameter：
    _lim：The pointer to the returned LIM message, allocated memory by the LIM_Pack, to be released
by the user
    _cid：Connection number, see Explanation in LIM_HEAD
    _code：LIM message code
    _data：Message data related to the message code, optional
    _ext_data_len：Additional data length
    _ext_data：Memory address of additional data
return value：
    true：succeed
    false：failure
```

```
bool LIM_Copy(LIM_HEAD*&_dlim, LIM_HEAD* _slim);
//Copy LIM message
Parameter：
    _dlim：The pointer to the destination LIM, allocated memory by the LIM_Copy, to be released by the
end-user.
```

_slim：Pointer to source LIM return value： true：succeed false：failure

voidLIM_Release(LIM_HEAD*&_lim);
//release LIM message
Parameter：
_lim:Address of the LIM message pointer, released as NULL

11. LIM TCP communication library

Important point

- The purpose of **LIM TCP** communication library is to make it easy for users to write lidar applications quickly. It is not mandatory. According to the structure of the **LIM** message, the user can complete the **TCP** communication and the **LIM** message analysis with the lidar.
- The heartbeat mechanism is embedded in the communication library, and the user is not required to handle the heartbeat.
- TryLIM.cpp is an example code that communicates with lidar using communication library. Reading carefully is helpful to understand how to use communication library.

Get the library version number

```
int __stdcall GetEquipmentCommVersion();
return value：
    Communication library version number
```

Library initialization

```
bool __stdcall EquipmentCommInit(int_paddr, EQCOMMDataCallBack _feqdata,
                                 EQCOMMStateCallBack _feqstate);
Parameter：
    _paddr：User data, which is passed back as Parameter to the callback function
    _feqdata：Receive data callback function
    _feqstate：Connection state callback function
return value：
    true：succeed
    false：failure
```

Closing the communication library

```
bool __stdcall EquipmentCommDestory();
return value：
    true：succeed
    false：failure
```

Open one device connection

```
bool __stdcall OpenEquipmentComm(int_cid, char* _ip, int_port);
```

Parameter：

_cid：Connection number, see explanation in LIM_HEAD

_ip：IP address of lidar equipment

_port：Connect port, should be LIM_USER_PORT

return value：

true：succeed

false：failure

Close one device connection

```
bool __stdcall CloseEquipmentComm(int_cid);
```

Parameter：

_cid：Connection number, see explanation in LIM_HEAD

return value：

true：succeed

false：failure

Send LIM message

```
bool __stdcall SendLIM(int _cid, void* _lim, int _lim_len);
```

Parameter：

_cid：Connection number, see explanation in LIM_HEAD

_lim：LIM message

_lim_len：LIM message length, LIM_HEAD.nLIMLen

return value：

true：succeed

false：failure

Receive data callback function

```
void (CALLBACK *EQCOMMDataCallBack)( int _cid, unsigned int _lim_code, void* _lim,
                                     int _lim_len, int _paddr)
```

Parameter：

_cid：Connection number, see explanation in LIM_HEAD

_lim_code：is LIM_HEAD.nCode

_lim：LIM message

_lim_len：LIM message length is LIM_HEAD.nLIMLen

_paddr：User data，That is the_paddrParameter passed in when EquipmentCommInit is called

Connection state callback function

```
//state code
#define EQCOMM_STATE_OK                2001    //Communication database connected to
lidar successfully
#define EQCOMM_STATE_ERR               2002    //Communication database connected to
lidar failure
#define EQCOMM_STATE_LOST              2003    //Lidar connection disconnection

void (CALLBACK *EQCOMMSTATECallBack)(int _cid, unsigned int _state_code, char* _ip,
                                     int _port, int _paddr)
```

Parameter：

_cid：Connection number

_state_code：Connection status code（EQCOMM_STATE_XXX）

_ip：IP address of the connection

_port：port number to connect，it is LIM_USER_PORT

_paddr：User data，That is the_paddrParameter passed in when EquipmentCommInit is called

DADISICK®

DADISICK TECHNOLOGY LIMITED

ru.dadisick.com