



# Shenzhen Leishen Intelligent System Co., Ltd.

## Linux\_and\_win\_SDK Instruction Manual

### Revision History

Ver.	Revise Date	Revise Contents	Revised By	Note
v2.1.0	2023/09/27	Initial Version	LS	-
v2.2.0	2024/07/11	Add detailed descriptions	LSJ	-

Prepared By:

Reviewed By:

Approved By:

## Table of Contents

<b>1. OVERVIEW.....</b>	<b>1</b>
1.1. BRIEF INTRODUCTION.....	1
1.2. DEVELOPMENT LANGUAGE.....	1
1.3. CAUTIONS.....	1
<b>2. STRUCTURE DESCRIPTION.....</b>	<b>3</b>
<b>3. INTERFACE FUNCTION DESCRIPTIONS .....</b>	<b>4</b>
3.1. START PROGRAM AND PARSE FUNCTION .....	4
3.1.1. Set Port and IP.....	4
3.1.2. Set Callback Function and Obtain LiDAR Data .....	5
3.1.3. Start Program and Obtain LiDAR Data .....	6
3.1.4. Stop Program and Stop Obtaining LiDAR Data .....	6
3.1.5. Call Interface and Obtain One Frame of Data .....	7
3.1.6. Complete Progress of Initializing and Obtaining LiDAR Data .....	9
3.2. OBTAIN LIDAR PARAMETERS INFORMATION.....	10
3.2.1. Data Type Definition of LiDAR Parameters Information .....	10
3.2.2. Obtain LiDAR Parameter Interface .....	12
3.2.3. Complete Progress of Initializing and Obtaining LiDAR Parameters .....	13
3.2.4. Obtain Current Status Information of LiDAR Data Packet .....	16
3.2.5. Obtain Current Status Information of LiDAR Device Packet .....	17
3.3. MODIFY LIDAR PARAMETERS.....	17
3.3.1. Modify LiDAR Motor Rotating Speed .....	17
3.3.2. Modify LiDAR IP .....	18
3.3.3. Modify Destination IP.....	19
3.3.4. Modify NTP IP .....	20
3.3.5. Modify Gateway IP.....	21
3.3.6. Modify Subnet Mask IP .....	22
3.3.7. Modify Data Packet Port.....	23
3.3.8. Modify Device Packet Port.....	24
3.3.9. Modify LiDAR Clock Source .....	25
3.3.10. Modify LiDAR Work State .....	26
3.3.11. Modify LiDAR Frame Rate.....	27
3.3.12. Modify Phase Lock Switch.....	28

3.3.13.	Send UDP Configuration Packet to LiDAR.....	29
3.3.14.	Complete Progress of Initializing and Modifying LiDAR Parameters.....	29
<b>4.</b>	<b>COMPILING DESCRIPTION .....</b>	<b>31</b>
4.1.	COMPILING UNDER WINDOWS OS .....	31
4.2.	COMPILING UNDER LINUX OS .....	34

# 1. Overview

This manual is to introduce how to use the linux\_and\_win\_demo.

## 1.1. Brief Introduction

This linux\_and\_win\_SDK is for the secondary development of the lidar developed by Shenzhen Leishen Intelligent System Co., Ltd. You can obtain one frame data of the lidar according to the code parsed, and perform the secondary development.

## 1.2. Development Language

Linux\_and\_win\_SDK is based on C/C++.

## 1.3. Cautions

Table 1.1 LiDAR Default Network Configuration

	IP Address	UDP Device Packet Port Number	UDP Data Packet Port Number
Lidar	192.168.1.200	2368 (fixed)	2369 (fixed)
Computer	192.168.1.102	2369	2368

- 1) The initial IP address of the lidar is 192.168.1.200, the destination IP address is 192.168.1.102, the default destination packet port is 2368, and the device packet port is 2369. If the lidar IP is not changed, you need to set up the host computer IP as a fixed IP, 192.168.1.102, in the network connection settings, and the subnet mask as 255.255.255.0. The user can change the lidar IP address as per their actual needs, and the IP address of the host device such as host computer connected to the lidar should be modified accordingly to ensure that they are under the same network segment.
- 2) If the lidar IP address has been changed, and the computer and the lidar's IP are in different network segments when connecting the lidar to the host computer, you need to set the gateway. If the lidar IP and the computer IP are in the same network segment, you can set a different IP, for example: 192.168.1.x, with a subnet mask of 255.255.255.0. If you need to find out the lidar's Ethernet configuration information, you can use Wireshark to capture the device ARP packets for analysis after connecting the lidar to the host computer. For the feature identification of the ARP packet, see the figure below.

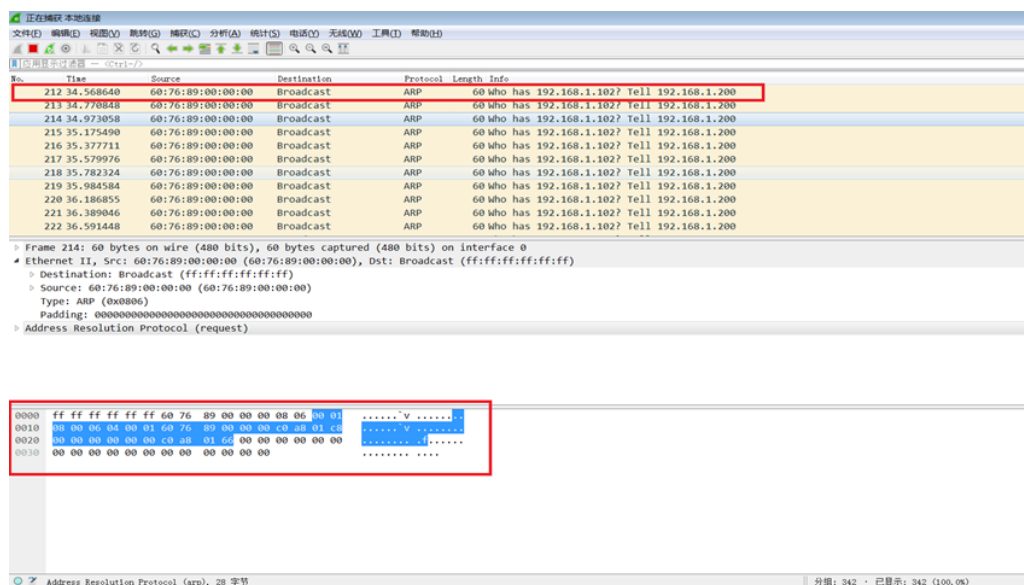









Figure 1.1 ARP packet

### Note:

- 1) Wireshark is a third-party software, and you may need to download it by yourself.  
LeiShen Intelligent bears no responsibility for any copyright and commercial disputes caused by users' use of the software.
- 2) Please disable all virtual NICs when using the SDK to prevent interference with data reception.
- 3) When developing with Visual Studio, please use the "release mode" in priority to reduce issues caused by data blocking, read and write delays.

## 2. Structure Description

 bin	2024/7/1 16:37	文件夹	
 build	2024/7/1 16:37	文件夹	
 demo	2024/7/1 16:38	文件夹	
 doc	2024/7/1 17:04	文件夹	
 Include	2024/7/1 16:38	文件夹	
 lib	2024/7/1 16:38	文件夹	
 CMakeLists.txt	2024/7/1 16:40	TXT 文件	3 KB

bin	Compile and generate executable file	-
build	Compiled directories	-
Include	Directory of the head file and source file	-
doc	Path of description files	-
demo: usage example for how to call and start the lidar, including 3 examples	main.cpp	General example, obtain the file of the point cloud, output the point cloud data
	main_PCL.cpp	get the file of the point cloud, add the PCL visual library, display the point cloud
	main_PCL_Pcap.cpp	Obtain files for point cloud, add library for visual PCL library, example of parsing offline PCAP, and display point cloud <b>Note:</b> the offline file should be put into the folder demo/PcapPacketPath and the name of the offline packet should be modified in the main_PCL_Pcap.cpp accordingly in the source file.
CMakeLists.txt	CMake file, configure and compile project and rule	

## 3. Interface Function Descriptions

### 3.1. Start Program and Parse Function

```
GetLidarData* m_GetLidarData = new GetLidarData_LS; //the lidar to be used, initialize
the lidar model to be used
```

#### 3.1.1. Set Port and IP

```
void setPortAndIP(uint16_t mDataPort = 2368, uint16_t mDevPort = 2369, std::string
mDestIP = "192.168.1.102", std::string mLidarIP = "192.168.1.200",std::string mGroupIp =
"226.1.1.102");

/*
```

Function descriptions: initialize data packet port, device packet port, destination IP, lidar IP and multicast IP and pass in parameter so that the program can receive the lidar's data packet and device packet.

Input/output parameter:

parameter 1: mDataPort	port number of data packet	2368 (by default)
parameter 2: mDevPort	port number of device packet	2369 (by default)
parameter 3: mDestIP	destination IP	"192.168.1.102"(by default)
parameter 4: mLidarIP	lidar IP	"192.168.1.200"(by default)
parameter 5: mGroupIp	multicast IP	"226.1.1.102"(by default)
returned value: none		

call examples:     GetLidarData->setPortAndIP(2368, 2369,  
"192.168.1.102","192.168.1.200", "226.1.1.102" );

**Note:** Before the function LidarStart() starts, please call setPortAndIP () to specify the data packet parameters of the current lidar so that the program can obtain the lidar's data;

```
*/
```

### 3.1.2. Set Callback Function and Obtain LiDAR Data

```
void setCallbackFunction(FunDataPrt*);
```

```
/*
```

Function descriptions: to transfer callback function. After a frame of lidar data is parsed, the callback function is called to transmit the lidar data. After a frame of lidar data is parsed, the callback function is called to transfer the lidar data.

Input/output parameter:

parameter 1: FunDataPrt: define callback function, input callback function pointer

returned value: none

Callback function type definition

```
typedef std::function<void(std::shared_ptr<std::vector<MuchLidarData>>, int, std::string)>
FunDataPrt;
```

#### Call examples:

##### 1) Callback function definition:

```
void callbackFunction(std::shared_ptr<std::vector<MuchLidarData>>, int, std::string);
```

```
FunDataPrt fun = std::bind(callbackFunction, std::placeholders::_1, std::placeholders::_2, std::placeholders::_3);
```

Calling function Inputs callback function pointer: GetLidarData->setCallbackFunction(&fun);

##### 2) Check callback function:

```
callbackFunction(std::shared_ptr<std::vector<MuchLidarData>> PerData, int, std::string);
```

Obtain a frame of data

#### Note:

There are two ways to get the data from the lidar: one of them is to use the callback function method and the other is to use the flag bit method;

Only the callback function method is required to transmit the callback function.

```
*/
```

### 3.1.3. Start Program and Obtain LiDAR Data

```
void LidarStart();
```

```
/*
```

Function descriptions: start the program, start to get the lidar packet and device packet, parse the lidar data.

Input/output parameters:

Parameter 1: none

Returned value: none

**Call example:**

GetLidarData->LidarStart();

**Note:** before the function LidarStart() starts, please call setPortAndIP () to specify the data packet parameters of the current lidar so that the program can obtain the lidar's data;

```
*/
```

### 3.1.4. Stop Program and Stop Obtaining LiDAR Data

```
void LidarStop();
```

```
/*
```

Function description: Stop the program, stop obtaining lidar packets and device packets, stop thread parsing data.

Input/output parameters:

Parameter 1: none

Returned value: none

Call example: GetLidarData->LidarStop();

Note: none

\*/

### 3.1.5. Call Interface and Obtain One Frame of Data

```
bool getLidarPerFrameData(std::shared_ptr<std::vector<MuchLidarData>>& preFrameData,
std::string& Info);
```

/\*

Function description: obtain one frame of point cloud data

Input/output parameters: the SDK defines the information output for each data point and the point cloud information that can be obtained.

```
typedef struct _MuchLidarData
{
    float X = 0.0;           //coordinate X value
    float Y = 0.0;           // coordinate Y value
    float Z = 0.0;           // coordinate Z value
    int ID = 0;              //channel number
    float H_angle = 0.0;     //horizontal angle
    float V_angle = 0.0;     //vertical angle
    float Distance = 0.0;    // distance value
    int Intensity = 0;       // intensity value
    u_int64 Mtimestamp_nsce = 0; //timestamp
}MuchLidarData;
```

Parameter 1: preFrameData: Pass in a reference to get a frame of lidar parameters and return a frame of point cloud data;

Parameter 2: Info: Pass in string parameter to get the function call information, when the returned value is false, output the information of failure.

Returned value: bool value; true: obtain one frame of lidar data successfully, false: obtain data failed, check Info value;

#### Call examples:

1) Judge if isFrameOK is true: it marks that the program completes the parsing of one frame;

2) When isFrameOK = true, it defines that:

```
std::shared_ptr<std::vector<MuchLidarData>> m_LidarData_temp;
```

```
std::string mInfo;
```

3) call getLidarPerFrameDate(LidarData, mInfo); to obtain lidar data;

Refer to call main.cpp

```
while (true)
```

```
{
```

```
//method one, get one frame of data
```

```
if (m_GetLidarData->isFrameOK)
```

```
{
```

```
std::shared_ptr<std::vector<MuchLidarData>> m_LidarData_temp;
```

```
std::string mInfo;
```

```
if (!m_GetLidarData->getLidarPerFrameDate(m_LidarData_temp, mInfo))
```

```
{
```

```
std::cout << mInfo << std::endl;
```

```
std::this_thread::sleep_for(std::chrono::milliseconds(1));
```

```
continue;
```

```

    }

    //output the number of point cloud

    std::cout << m_LidarData_temp->size() << std::endl;

}

else

{

    std::this_thread::sleep_for(std::chrono::milliseconds(1));

}

}

```

Note: there are two methods to obtain lidar data. This is the way to use the flag bits. the lidar parameters are continuously obtained through flag bit.

\*/

### 3.1.6. Complete Progress of Initializing and Obtaining LiDAR Data

(Refer to ./demo/main.cpp) Obtain data through flag bit

```

GetLidarData* m_GetLidarData = new GetLidarData_LS;           //initialize the lidar
model

m_GetLidarData->setPortAndIP(2368, 2369, "192.168.1.102","192.168.1.200",
"226.1.1.102" );//set parameters

m_GetLidarData->LidarStart();           //start program to obtain lidar data and parse

while (true)

{

    if (m_GetLidarData->isFrameOK)

```

```

{

    std::shared_ptr<std::vector<MuchLidarData>> m_LidarData_temp;

    std::string mInfo;

    if (!m_GetLidarData->getLidarPerFrameDate(m_LidarData_temp, mInfo))

    {

        std::cout << mInfo << std::endl;

        std::this_thread::sleep_for(std::chrono::milliseconds(1));

        continue;

    }

    //output the number of point cloud

    std::cout << m_LidarData_temp->size() << std::endl;

}

else

{

    std::this_thread::sleep_for(std::chrono::milliseconds(1));

}

}

m_GetLidarData->LidarStop();           //stop program and stop parsing lidar data

```

Note: It currently only outputs the number of points per frame of the lidar. To check the data type of MuchLidarData, it requires to obtain the information of each point.

## 3.2. Obtain LiDAR Parameters Information

### 3.2.1. Data Type Definition of LiDAR Parameters Information

It defines the data type of the lidar output parameter information, the default value is -1: and

judge the main information of the returned value or unit.

```
typedef struct _LidarStateParam
```

```
{

    std::string LidarIP = "-1,-1,-1,-1";    //lidar IP

    std::string ComputerIP = "-1,-1,-1,-1";    //destination IP

    std::string NtpIP = "-1,-1,-1,-1";        //NTP IP

    std::string GatewayIP = "-1,-1,-1,-1";    //gateway IP

    std::string SubnetMaskIP = "-1,-1,-1,-1"; //subnet mask IP

    c MacAddress = "-1,-1,-1,-1,-1,-1";    //MAC address of lidar: 50 3E 7C ** ** **

    int DataPort = -1;                      //data packet port

    int DevPort = -1;                       //device packet port


    float ReceiverTemperature = -1.0;        //lidar temperature, unit: °

    float ReceiverHighVoltage = -1.0;        //receiver board, unit: V

    float MotorSpeed = -1;                   //motor rotating speed

    int FrameRateMode = -1;                  //frame rate mode    mode: 0、1、2


    int PTP_State = -1;                      //PTP status:    1: lost; 0: not lost

    int GPS_State = -1;                      //GPS status    1: lost; 0: not lost

    int PPS_State = -1;                      //PPS status:    1: lost; 0: not lost

    int StandbyMode = -1;                    // standby status    0: normal 1: standby

    int Clock_Source = -1;                   //clock source:    0:GPS; 1: PTP_L2;
                                           2:NTP; 3:PTP_UDPv4
}
```

```

int   PhaseLockedSwitch = -1;           /phase lock:      0: disable;    1: enable

float PhaseLockedAngle = -1;           // phase lock    phase lock angle  unit: °

int   PhaseLockedState = -1;           // phase locks status:  0: unlocked;  1:
locked

std::string FaultCode = "-1";          //fault code      4 bytes display by bit, each bit
represents a fault condition

double RunningTime = -1;               //running time    unit: hour

}LidarStateParam;

```

Note: The default parameter is -1, if the other items have values, but one item is -1, it means that the lidar does not have such parameter output.

### 3.2.2. Obtain LiDAR Parameter Interface

```
bool getLidarParamState(LidarStateParam& mLidarStateParam, std::string& InfoString);
```

/\*

Function description: obtain returned value of lidar internal parameters

Input/output parameters:

Parameter 1: mLidarStateParam: Pass in a reference to get customized data type of lidar parameters and return currently acquired lidar parameters

Parameter 2: InfoString: Pass in string parameter to get the function call information, when the returned value is false, output the information of acquisition failure.

Returned value: bool value; true: obtain one frame of lidar data successfully, false: obtain data failed, check Info value

#### Call examples:

1) Parameters type definition:

```
LidarStateParam mLidarStateParam;
```

```
std::string mInfo1;
```

2) Call interface to pass in reference and output

```
if (!m_GetLidarData->getLidarParamState(mLidarStateParam, mInfo1))

{

    std::cout << mInfo1 << std::endl;

}

else

{

    std::cout << "LidarIP    =    " << mLidarStateParam.LidarIP << std::endl;

}
```

Refer to call in the main.cpp

### 3.2.3. Complete Progress of Initializing and Obtaining LiDAR Parameters

```
GetLidarData* m_GetLidarData = new GetLidarData_LS;           //initialize lidar model

m_GetLidarData->setPortAndIP(2368, 2369, "192.168.1.102","192.168.1.200",
"226.1.1.102" );//set parameters

m_GetLidarData->LidarStart();           //start program to obtain lidar data and parse

//obtain parameters once only

LidarStateParam mLidarStateParam;

std::string mInfo1;

std::this_thread::sleep_for(std::chrono::milliseconds(2000));    // It needs to wait 2 seconds
after starting the lidar to get the lidar data.

if (!m_GetLidarData->getLidarParamState(mLidarStateParam, mInfo1)) //Get           radar
parameter call    Judge returned dvalue
```

```
{
    std::cout << mInfo1 << std::endl;           //print failure information
}

else

{

std::cout << "***** Lidar Parameters Display Start *****" << std::endl;

std::cout.width(20); std::cout << "LidarIP    =    " << mLidarStateParam.LidarIP << std::endl;

std::cout.width(20); std::cout << "ComputerIP  =    " << mLidarStateParam.ComputerIP <<
std::endl;

std::cout.width(20); std::cout << "GatewayIP   =    " << mLidarStateParam.GatewayIP <<
std::endl;

std::cout.width(20); std::cout << "SubnetMaskIP=    " << mLidarStateParam.SubnetMaskIP <<
std::endl;

std::cout.width(20); std::cout << "DataPort    =    " << mLidarStateParam.DataPort << std::endl;

std::cout.width(20); std::cout << "DevPort     =    " << mLidarStateParam.DevPort << std::endl;


std::cout.width(20); std::cout << "ReceiverTemperature    =    " <<
mLidarStateParam.ReceiverTemperature << std::endl;

std::cout.width(20); std::cout << "ReceiverHighVoltage    =    " <<
mLidarStateParam.ReceiverHighVoltage << std::endl;

std::cout.width(20); std::cout << "MotorSpeed    =    " << mLidarStateParam.MotorSpeed <<
std::endl;

std::cout.width(20); std::cout << "PTP_State     =    " << mLidarStateParam.PTP_State <<
```

```

std::endl;

std::cout.width(20); std::cout << "GPS_State    =    " << mLidarStateParam.GPS_State <<
std::endl;

std::cout.width(20); std::cout << "PPS_State    =    " << mLidarStateParam.PPS_State <<
std::endl;

std::cout.width(20); std::cout << "StandbyMode =    " << mLidarStateParam.StandbyMode <<
std::endl;

std::cout.width(20); std::cout << "Clock_Source =    " << mLidarStateParam.Clock_Source <<
std::endl;

std::cout.width(20); std::cout << "PhaseLockedSwitch    =    " <<
mLidarStateParam.PhaseLockedSwitch << std::endl;

std::cout.width(20); std::cout << "PhaseLockedState =    " <<
mLidarStateParam.PhaseLockedState << std::endl;

std::cout.width(20); std::cout << "FaultCode    =    " << mLidarStateParam.FaultCode <<
std::endl;

std::cout.width(20); std::cout << "RunningTime =    " << mLidarStateParam.RunningTime <<
std::endl;

std::cout << "" << "***** Lidar Paramet<< std::cout.width(20)ers Display End *****"
<< std::endl << std::endl;

}

```

Note: since the interval between device packets is 1 second, you need to start ;LidarStart() after more than 1 second to get the data. To ensure the acquisition of the lidar parameter, it is set as 2 seconds.

```
// Loop to get parameter
```

```
While(true)
```

```
{
```

```
    LidarStateParam mLidarStateParam;
```

```
    std::string mInfo1;
```

```
    if (!m_GetLidarData->getLidarParamState(mLidarStateParam, mInfo1))
```

```
    {
```

```
        std::cout << mInfo1 << std::endl;
```

```
    }
```

```
    else
```

```
    {
```

```
        std::cout << "ReceiverTemperature    =    " <<
```

```
        mLidarStateParam.ReceiverTemperature << std::endl;
```

```
    }
```

```
    std::this_thread::sleep_for(std::chrono::milliseconds(1000))    //sleep for 1 second;
```

```
}
```

### 3.2.4. Obtain Current Status Information of LiDAR Data Packet

```
std::string getDataPacketState();
```

```
/*
```

Function descriptions: Whether there are any exceptions in the currently obtained data packet, determine whether there are any errors in the data obtained by the program.

Input/output parameters:

Parameter 1: none

Returned value: std::string: return string description information

Call example: std::string infoStr = getDataPacketState();

```
std::cout << "infoStr = " << infoStr << std::endl;
```

Note: After starting the program to get the lidar data LidarStart(), call the interface to get the lidar parameters.

```
*/
```

### 3.2.5. Obtain Current Status Information of LiDAR Device Packet

```
std::string getDevPacketState();
```

```
/*
```

Function descriptions: Whether there is any exceptions in the currently acquired device packet, and determine whether the program acquires the device packet information normally.

Input/output parameters:

Parameter 1: none

Returned value: std::string: return description of string information

Call examples: std::string infoStr = getDevPacketState();

```
std::cout << "infoStr = " << infoStr << std::endl;
```

Note: after start the program LidarStart() to obtain lidar data, call interface to obtain lidar parameters.

## 3.3. Modify LiDAR Parameters

### 3.3.1. Modify LiDAR Motor Rotating Speed

```
bool setLidarRotateSpeed(int SpeedValue, std::string& InfoString);
```

```
/*
```

Function descriptions: modify lidar rotating speed parameters;

Input/output parameters:

Parameter 1: SpeedValue rotating speed, support 300, 600, 1200 (represents 5 Hz, 10 Hz, 20 Hz)

Parameter 2: InfoString: pass in string parameter and obtain settings information, when returned value is false, it outputs settings failure information

Returned value: bool value; true: successful rotating speed setting; false: settings failed; check information value.

Call example:

```
std::string& InfoString; //create read-back information
string
GetLidarData->setLidarRotateSpeed(300, InfoString); //call interface, set
300 RPM (5 Hz)
GetLidarData->sendPackUDP(); // call interface, send UDP
packet to lidar
```

#### Note:

- 1) After calling setLidarRotateSpeed (), it appears "This version of Lidar does not support \*\*\*!!!", which means the function call is not supported by this lidar (no such function);
- 2) After calling setLidarRotateSpeed(), it requires calling sendPackUDP(), so that the UDP packet can be sent to lidar to modify lidar parameters;
- 3) It doesn't apply to 1550 nm(LS series) as these don't support this function.

\*/

### 3.3.2. Modify LiDAR IP

```
bool setLidarIP(std::string IPString, std::string& InfoString);
```

/\*

Input/output parameters:

Parameter 2: InfoString: pass in string parameter and obtain settings information,  
returned value is false, it outputs settings failure information;

Call examples:

Note: 1, After call `setLidarIP()`, it requires to call `sendPackUDP()` so that the UDP packet can be sent to lidar to modify lidar parameters

\* /

### 3.3.3. Modify Destination IP

/ \*

Input/output parameters:

Parameter 2: InfoString: pass in string parameter and obtain settings information.

when returned value is false, it outputs settings failure information;

Returned value: bool value; true: rotating speed setting successful; false: rotating speed setting failed, check Info value;

Call example:

```
std::string& InfoString; //create read-back
information string

GetLidarData->setComputerIP("192.168.1.102", InfoString); //call interface,
set IP

GetLidarData->sendPackUDP(); //call interface, send
UDP packet to lidar
```

Note: After call setComputerIP ( ), it requires to call sendPackUDP() so that the UDP packet can be sent to lidar to modify lidar parameters;

\*/

### 3.3.4. Modify NTP IP

```
bool setNTP_IP(std::string IPString, std::string& InfoString);
```

/\*

Function descriptions: modify NTP parameters;

Input/output parameters:

Parameter 1: IPString IP value requires to be modified, such as  
"192.168.1.102"

Parameter 2: InfoString: pass in string parameter, obtain settings information ,  
when returned value is false, output settings failure information

Returned value: bool value; true: rotating speed setting successful; false: rotating speed setting failed, check Info value;

Call example:

```

std::string& InfoString;                                //create read-back
information string

GetLidarData->setNTP_IP("192.168.1.102", InfoString);      //call interface, set IP

GetLidarData->sendPackUDP();                             //call interface, send UDP
packet to lidar

```

Call:

- 1: After call setNTP\_IP ( ) ;it shows "This version of Lidar does not support \*\*\*\*!!!" , which means the function call is not supported by this lidar (no such function);
- 2: After call setNTP\_IP( ), it requires to call sendPackUDP() so that the UDP packet can be sent to lidar to modify lidar parameters;

\*/

### 3.3.5. Modify Gateway IP

```
bool setGatewayIP(std::string IPString, std::string& InfoString);
```

/\*

Function descriptions: modify gateway IP parameters;

Input/output parameters:

Parameter 1: IPString      IP requires to be modified, such as "192.168.1.254"

Parameter 2: InfoString:      pass in string parameter, obtain settings information,  
when returned value is false, output settings failure information.

Returned value:    bool value; true: rotating speed settings successful; false: rotating  
speed settings failed, check Info value.

Call example:

```
std::string& InfoString;                                //create read-back
```

information string

```

        GetLidarData->setGatewayIP("192.168.1.254", InfoString);           //call interface,
set IP
        GetLidarData->sendPackUDP();                                       //call interface, send
        UDP packet to the lidar

```

Note:

1: call setGatewayIP ( ) ;it shows "This version of Lidar does not support \*\*\*\*!!!", which means the function call is not supported by this lidar (no such function);

2: After call setGatewayIP ( ) , it requires to call sendPackUDP(), the UDP packet can be sent to lidar to modify lidar parameters;

\*/

### 3.3.6. Modify Subnet Mask IP

```
bool setSubnetMaskIP(std::string IPString, std::string& InfoString);
```

/\*

Function descriptions: modify subnet mask, IP parameters;

Input/output parameters:

Parameter 1: IPString      IP requires to be modified, such as "255.255.255.0"

Parameter 2: InfoString:    pass in string parameter, obtain settings information.

When returned value is false, output settings failure information.

Returned value:    bool value; true: rotating speed settings successful; false: rotating speed settings failed, check Info value.

Call example:

```

        std::string& InfoString;                                           // create read-back
        information string

```

```
GetLidarData->setSubnetMaskIP("192.168.1.254", InfoString);    // call interface,
set IP
```

```
GetLidarData->sendPackUDP();                                ///call interface, send
UDP packet to the lidar
```

Note: 1: call setSubnetMaskIP ( ) ;it shows "This version of Lidar does not support \*\*\*\*!!!", which means the function call is not supported by this lidar (no such function);

2: After call setSubnetMaskIP ( ) , it requires to call sendPackUDP(), the UDP packet can be sent to lidar to modify lidar parameters;

```
*/
```

### 3.3.7. Modify Data Packet Port

```
bool setDataPort(int PortNum, std::string& InfoString);
```

```
/*
```

Functions descriptions: modify data packet port;

Input/output parameters:

Parameter 1: PortNum,: data packet port value requires to be modified, such as 2368

Parameter 2: InfoString: pass in string parameter, obtain settings information, when returned value is false, output settings failure information.

Returned value: bool value; true: rotating speed settings successful; false: rotating speed settings failed, check Info value.

Call examples:

```
std::string& InfoString;                                //create read-back information
string
```

```
GetLidarData->setDataPort(2368, InfoString);            //call interface, set IP
```

```
GetLidarData->sendPackUDP();                            //call interface, send UDP
```

packet to the lidar

**Note 1:** After call `setDataPort()`, it requires to call `sendPackUDP()`, so that the UDP packet can be sent to lidar to modify lidar parameters;

\*/

### 3.3.8. Modify Device Packet Port

```
bool setDevPort(int PortNum, std::string& InfoString);
```

/\*

Functions descriptions: modify device packet port;

Input/output parameters:

Parameter 1: PortNum,: the data packet port requires to be modified, such as 2369;

Parameter 2: InfoString: pass in string parameter, obtain settings information, when the returned value is false, output the settings failure information;

Returned value: bool value, true: setting rotating speed successful; false: setting rotating speed failed, check Info value;

Call example:

```
std::string& InfoString;           // create read-back information
string
GetLidarData->setDevPort(2369, InfoString);    // call interface, set IP
GetLidarData->sendPackUDP();                  //call interface, send UDP
packet to sent
```

**Note: 1:** After call `setDevPort()`, it requires to call `sendPackUDP()`, so that the UDP packet can be sent to lidar to modify lidar parameters;

**\***

### 3.3.9. Modify LiDAR Clock Source

```
bool setLidarSoureSelection(int StateValue, std::string& InfoString)
```

/ \*

### Functions descriptions: modify lidar clock source

Input/output parameters:

Parameter 1: StateValue values require to be modified: 0:GPS; 1: PTP\_L2;  
2:NTP; 3:PTP UDPV4;

Parameter 2: InfoString: pass in string parameter, obtain settings information,  
when the returned value is false, output the settings failure information;

Returned value: bool value, true: setting rotating speed successful; false: setting rotating speed failed, check Info value.

Call example:

```
std::string& InfoString; //create read-back information
string

GetLidarData->setLidarSoureSelection(0, InfoString); //call interface, set

GetLidarData->sendPackUDP(); //call interface to send
UDP packet to lidar
```

Note: 1: After call `setLidarSourceSelection()`; it shows "This version of Lidar does not support \*\*\*!!!" , which means the function call is not supported by this lidar (no such function);

2: After call `setLidarSourceSelection()`, it requires to call `sendPackUDP()`, so that the UDP packet can be sent to lidar to modify lidar parameters.

\* /

```
bool setLidarWorkState(int StateValue, std::string& InfoString)
```

Function descriptions: modify lidar work state: normal mode or low power mode (only send device packet without data packet, lidar doesn't emit laser).

Parameter 1: StateValue      the value requires to be modified, 0: normal mode, power mode;

Parameter 2: InfoString: pass in string parameter, obtain settings information,  
when the returned value is false, output the settings failure information;

Returned value: bool value, true: setting rotating speed successful; false: setting rotating speed failed, check Info value.

```
std::string& InfoString; //create read-back information
string
GetLidarData->setLidarWorkState(0, InfoString); //call interface, set
GetLidarData->sendPackUDP(); //call interface, send UDP
packet to lidar
```

Note: 1: After call `setLidarWorkState ( )`; it shows “This version of Lidar does not support \*\*\*!!!”, which means the function call is not supported by this lidar (no such function);

2: After call `setLidarWorkState()`, it requires to call `sendPackUDP()`, so that the UDP packet can be sent to lidar to modify lidar parameters.

26

### 3.3.11. Modify LiDAR Frame Rate

```
bool setFrameRateMode(int StateValue, std::string& InfoString)
```

/ \*

Function descriptions: modify lidar's frame rate.

Input/output parameters:

Parameter 1: StateValue      the value requires to be modified, 0: normal

frame rate: 1: 50% frame rate; 2: 25% frame rate;

Parameter 2: InfoString: pass in string parameter, obtain settings information,  
when the returned value is false, output the settings failure information;

Returned value: bool value, true: setting rotating speed successful; false: setting rotating speed failed, check Info value.

Call examples:

```
std::string& InfoString; // create read-back information
string

GetLidarData->setFrameRateMode(0, InfoString); //call interface, set

GetLidarData->sendPackUDP(); //call interface, send UDP
packet to lidar
```

Note: 1: After call setFrameRateMode ( ) ;it shows “This version of Lidar does not support \*\*\*!!!”, which means the function call is not supported by this lidar (no such function);

2: After call `setFrameRateMode()`, it requires to call `sendPackUDP()`, so that the UDP packet can be sent to lidar to modify lidar parameters.

3: It only applies to 1550 nm (LS series) lidar.

\* /

```
bool setPhaseLockedSwitch(int StateValue, std::string& InfoString)
```

Function descriptions: modify lidar's phase lock switch.

Parameter 1: StateValue      the value requires to be modified. 0: off; 1: on.

Returned value: bool value, true: setting rotating speed successful; false: setting rotating speed failed, check Info value.

```
std::string& InfoString; // create read-back information

string
```

```
GetLidarData->setPhaseLockedSwitch(0, InfoString); //call interface, set
phase lock switch
```

```
GetLidarData->sendPackUDP();           // call interface, send UDP
packet to lidar
```

Note: 1: After call `setPhaseLockedSwitch()`; it shows "This version of Lidar does not support \*\*\*!!!" , which means the function call is not supported by this lidar (no such function);

2: After call `setPhaseLockedSwitch()`, it requires to call “`sendPackUDP()`” so that the UDP packet can be sent to lidar to modify lidar parameters;

3: Currently it only applies to LS series lidar.

28

### 3.3.13. Send UDP Configuration Packet to LiDAR

```
bool sendPackUDP()
```

```
/*
```

Function descriptions: send configuration packet to modify lidar parameters

Input/output parameters:

Parameter 1: none

Returned value: none

Call examples:

```
GetLidarData->sendPackUDP();           //call interface, send UDP
packet to lidar
```

Note: 1: After call the functions above to modify interface, it requires to call "sendPackUDP()" so that the UDP packet can be sent to lidar to modify lidar parameters;

2: It supports to call sendPackUDP() to send a unified packet to modify the parameters after modify multiple parameters.

```
*/
```

### 3.3.14. Complete Progress of Initializing and Modifying LiDAR Parameters

```
GetLidarData* m_GetLidarData = new GetLidarData_LS;           // initialize the lidar model
```

```
m_GetLidarData->setPortAndIP(2368, 2369, "192.168.1.102","192.168.1.200",
"226.1.1.102" );//set parameters
```

```
m_GetLidarData->LidarStart();           // start program to obtain lidar data and parse
```

```
std::this_thread::sleep_for(std::chrono::milliseconds(2000));           // It needs to wait 2
seconds after starting the lidar to get the device packet
```

```
std::string mInfo;
```

```
m_GetLidarData->setLidarRotateSpeed(600, mInfo);

m_GetLidarData->setLidarIP("192.168.1.200", mInfo);

m_GetLidarData->setComputerIP("192.168.1.102", mInfo);

m_GetLidarData->setDataPort(2368, mInfo);

m_GetLidarData->setDevPort(2369, mInfo);

m_GetLidarData->setLidarSoureSelection(0, mInfo);

m_GetLidarData->setLidarWorkState(0, mInfo);


m_GetLidarData->sendPackUDP();           //send UDP packet

m_GetLidarData->LidarStop();             // stop program and stop parsing lidar data
```

Note: As the interval between device packets is 1 second, you need to get the device packet after more than 1s after the program starts ;LidarStart(). To ensure that the parameters can be set normally, please standby for 2s.

## 4. Compiling Description

### 4.1. Compiling under Windows OS

Use cmake-gui to configure compiling project. For the source code directory, please select "CMakeList" and the directory that source code file belongs to (linux\_and\_win\_demo), for the compiling directory, please select the "build" file that it belongs to.

名称	修改日期	类型	大小
bin	2024/7/1 16:37	文件夹	
build	2024/7/1 16:37	文件夹	
demo	2024/7/1 16:38	文件夹	
doc	2024/7/1 17:04	文件夹	
Include	2024/7/1 16:38	文件夹	
lib	2024/7/1 16:38	文件夹	
CMakeLists.txt	2024/7/1 16:40	TXT 文件	3 KB

Figure 4.1 Files in the source code directory

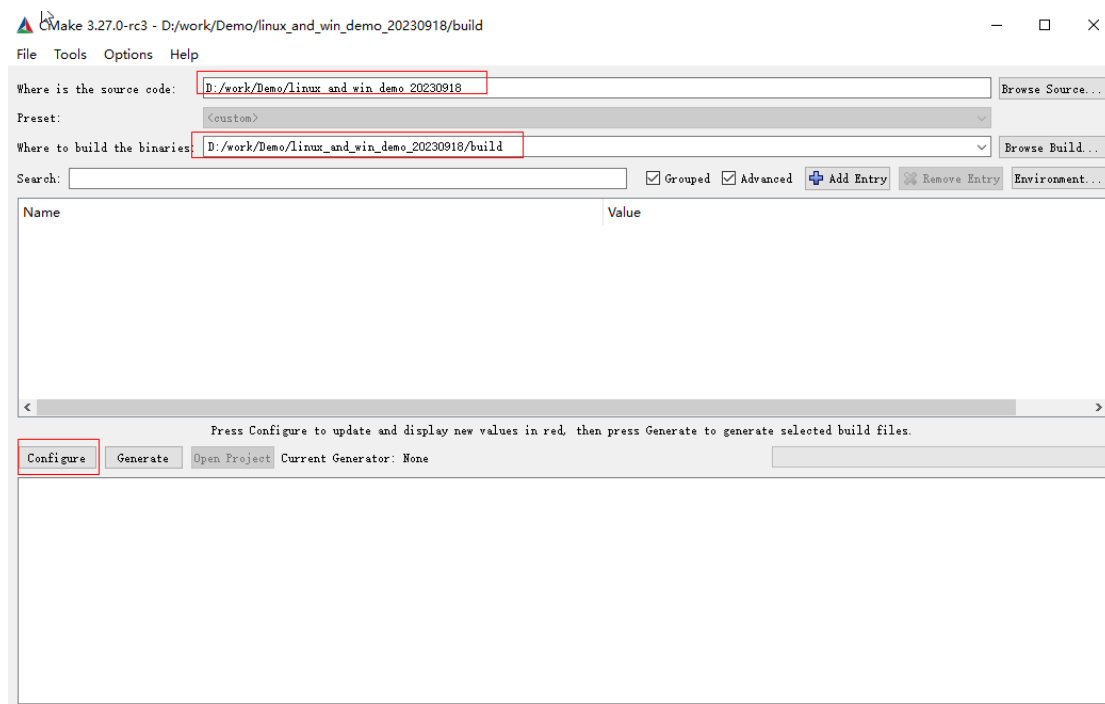



Figure 4.2 Cmake-gui configuration

Click the "configure" button, and select the compiler that is under use.

? X



Specify the generator for this project

Visual Studio 12 2013

Optional platform for generator(if empty, generator uses: Win32)

x64

Optional toolset to use (argument to -T)

☒ Use default native compilers  
☐ Specify native compilers  
☐ Specify toolchain file for cross-compiling  
☐ Specify options for cross-compiling

Finish Cancel

Figure 4.3 Select the compiler

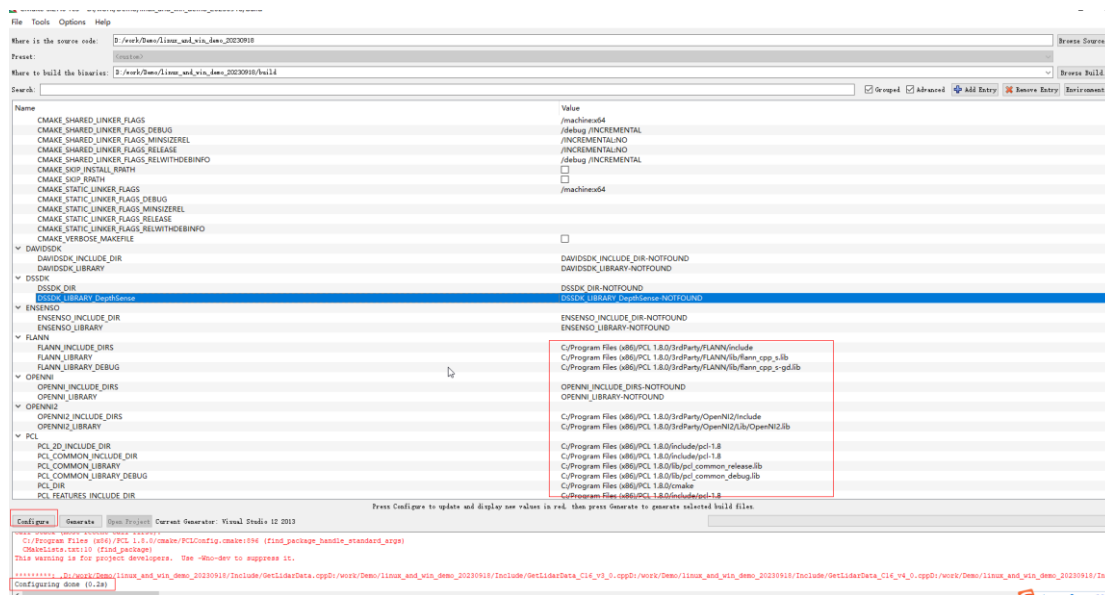


Figure 4.4 Configure PCL

Configure the PCL installation path, and click "Configure" button. When the "Configuration done" prompts, the configuration is finished.

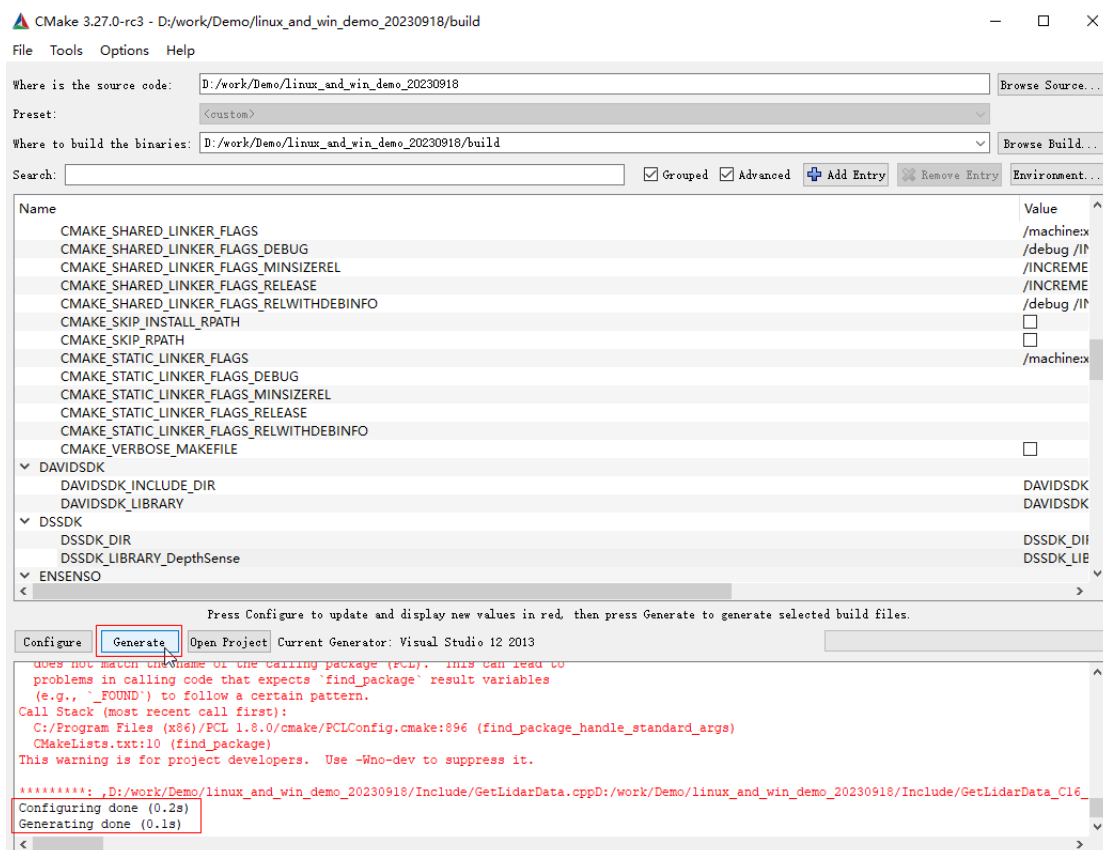


Figure 4.5 "Generate" the project

Click "Generate" to generate project, when the "Generating done" prompts, the configuration is finished. Open the build directory to find the generated project.

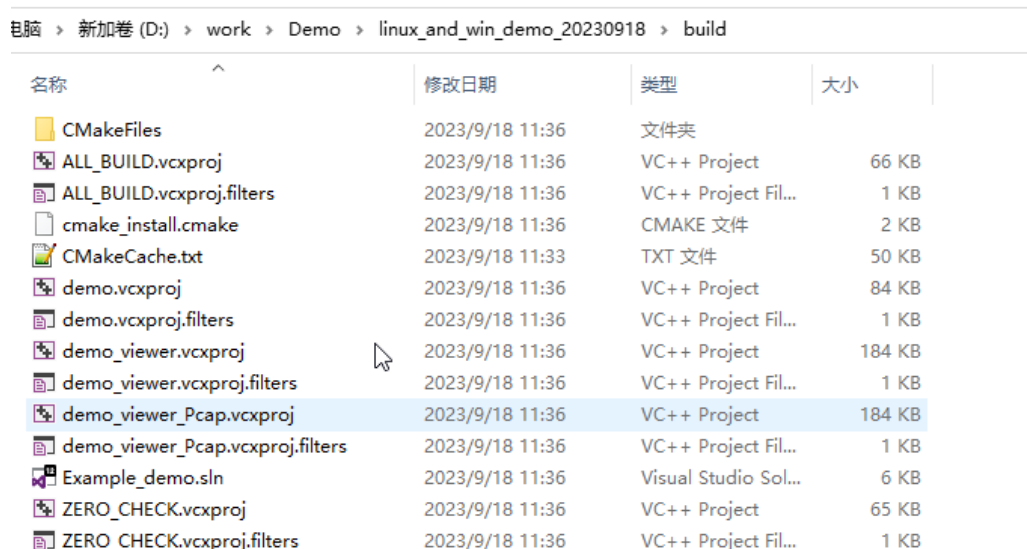


Figure 4.6 The generated project in the build directory

Click the "Open Project" button or open the \*.sln file in the compiled working directory to open the project.

Change the project to "Release mode" (recommended) and the corresponding 32/64-bit mode, and compile the corresponding project as needed.

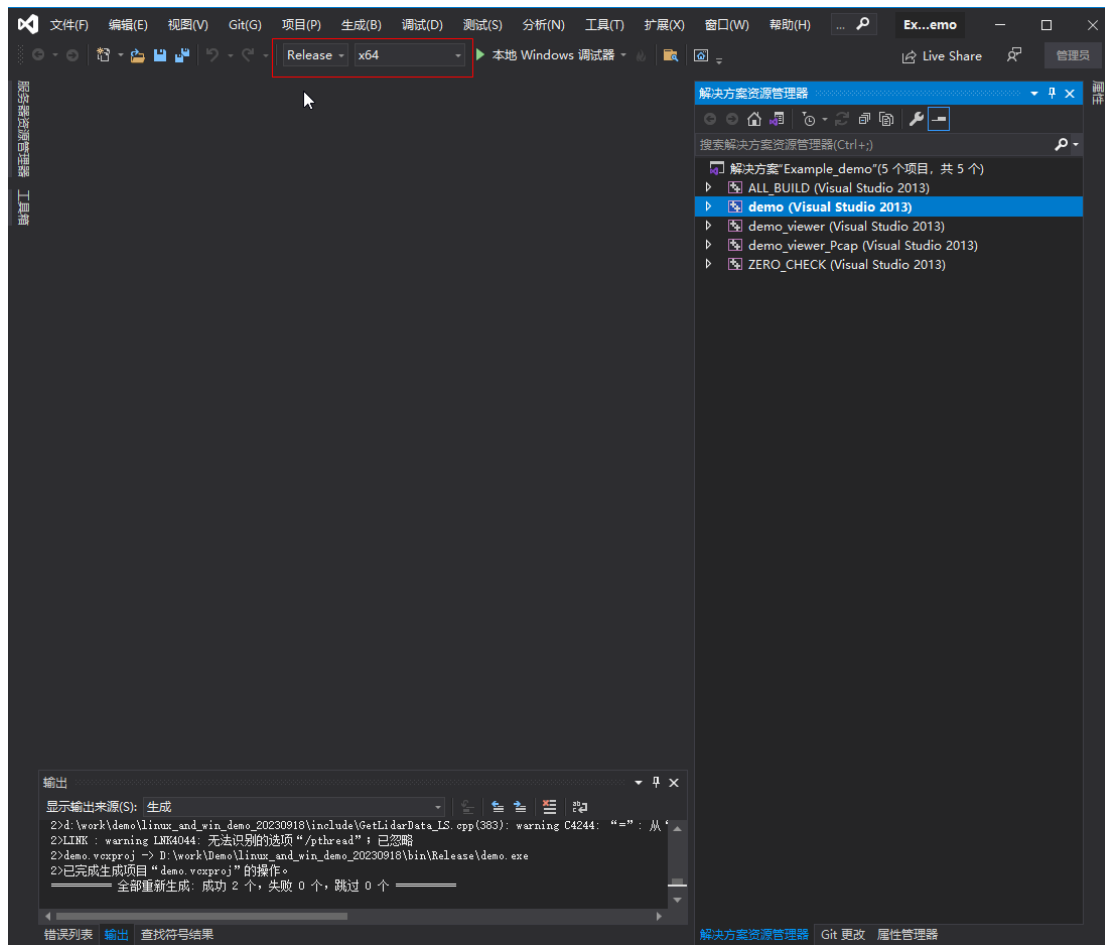


Figure 4.7 Compile the corresponding project

此电脑 > 新加卷 (D:) > work > Demo > linux\_and\_win\_demo\_20230918 > bin > Release

名称	修改日期	类型	大小
demo.exe	2023/9/18 11:57	应用程序	103 KB
demo_viewer.exe	2023/9/18 11:57	应用程序	5,013 KB
demo_viewer_Pcap.exe	2023/9/18 11:58	应用程序	5,012 KB

Figure 4.8 Generates the corresponding sample program in the bin directory

## 4.2. Compiling under Linux OS

### ## Dependency

If the PCL is not installed, please install it, pcl : sudo apt-get install libpcl-dev

Refer to: <https://github.com/PointCloudLibrary/pcl>

# # compile and run

Compiling progress under Linux OS:

Switch to the directory where the folder is located

```
cd build
```

```
cmake ..
```

```
make -j4
```

Switch to the directory of the file that generated the executable file

```
cd ../bin
```

```
obtain data demo: ./demo
```

```
visualized demo_viewer: ./demo_viewer
```

```
visualized offline pcap: demo_viewer_Pcap: ./demo_demo_viewer_Pcap
```

Note: The offline files should be put into . /demo/PcapPacketPath folder and modify the name of the offline package correspondingly in the source file main\_PCL\_Pcap.cpp.