YDLidar SDK API

V1.0.0

Generated by Doxygen 1.8.11

Contents

Chapter 1

CYdLidar(YDLIDAR SDK API)

Library	CYdLidar
File	CYdLidar.h
Author	Tony [code at ydlidar com]
Source	https://github.com/ydlidar/YDLidar-SDK
Version	1.0.0
Sample	ydlidar sample[G1 G2 G4 G6 S2 X2 X4) tof sample[TG15 TG30 TG50 TX8 TX20] etlidar sample[T5 T15]

This API calls Two LiDAR interface classes in the following sections:

- YDlidarDriver
- ETLidarDriver
- C API C API

Copyright

Jump to the ::CYdLidar interface documentation.

Chapter 2

YDLIDAR DATASET

LIDAR	Model	Baudrate	Sample↩ Rate(K)	Range(m)	Frequency(HZ)	Intenstiy(bi	Single↩ Channel	voltage(↩ V)
F4	1	115200	4	0.12∼12	5∼12	false	false	4.8∼5.2
S4	4	115200	4	0.10∼8.0	5∼12 (PWM)	false	false	4.8∼5.2
S4B	4/11	153600	4	0.10∼8.0	5∼12(P↩ WM)	true(8)	false	4.8∼5.2
S2	4/12	115200	3	0.10∼8.0	4∼8(P↩ WM)	false	true	4.8∼5.2
G4	5	230400	9/8/4	0.↩ 28/0.26/0.↩ 1∼16	5∼12	false	false	4.8∼5.2
X4	6	128000	5	0.12∼10	5∼12(P↩ WM)	false	false	4.8∼5.2
X2/X2L	6	115200	3	0.10∼8.0	4∼8(P↩ WM)	false	true	4.8∼5.2
G4PRO	7	230400	9/8/4	0.↩ 28/0.26/0.↩ 1∼16	5∼12	false	false	4.8∼5.2
F4PRO	8	230400	4/6	0.12∼12	5∼12	false	false	4.8∼5.2
R2	9	230400	5	0.12∼16	5∼12	false	false	4.8∼5.2
G6	13	512000	18/16/8	0.↩ 28/0.26/0.↩ 1∼25	5∼12	false	false	4.8∼5.2
G2A	14	230400	5	0.12∼12	5∼12	false	false	4.8∼5.2
G2	15	230400	5	0.28∼16	5∼12	true(8)	false	4.8∼5.2
G2C	16	115200	4	0.1∼12	5∼12	false	false	4.8∼5.2
G4B	17	512000	10	0.12∼16	5∼12	true(10)	false	4.8∼5.2
G4C	18	115200	4	0.1∼12	5∼12	false	false	4.8∼5.2
G1	19	230400	9	0.28∼16	5∼12	false	false	4.8∼5.2
TX8	100	115200	4	0.1∼8	4∼8(P↩ WM)	false	true	4.8∼5.2
TX20	100	115200	4	0.1∼20	4∼8(P↩ WM)	false	true	4.8∼5.2
TG15	100	512000	20/18/10	0.05∼30	3∼16	false	false	4.8∼5.2
TG30	101	512000	20/18/10	0.05∼30	3∼16	false	false	4.8∼5.2
TG50	102	512000	20/18/10	0.05∼50	3∼16	false	false	4.8∼5.2

Chapter 3

FlowChart

```
st=>start: Start
op=>operation: Set Paramamters and Initialize
op1=>operation: TrunOn
tr=>operation: Try Again
op2=>operation: doProcessSimple
op3=>operation: TrunOff
op4=>operation: disconnecting
cond=>condition: success Yes or No?
cond1=>condition: success Yes or No?
cond2=>condition: success Yes or No?
cond3=>condition: LOOP Yes or No?
cond4=>condition: TryAgain Yes or No?
e=>end: End
en=>end: End

st(left)->op->cond
cond(yes)->op1->cond1
cond(no)->op3->op4->e
cond1(yes)->op2->cond3
cond3(yes)->op2
cond3(no, left)->op3->op4->e
cond1(no,right)->tr(bottom)->cond4
cond4(yes)->op3
cond4(no)->op3(right)->op4(right)->e
```

sequenceDiagram

```
sequenceDiagram
note over UserProgram: Set Paramters
note over UserProgram: Initialize SDK
UserProgram->Command: Get LiDAR Information
Command-->UserProgram: Device connected and Devce Information recevied
note over UserProgram: TurnOn
UserProgram->Command: Start LiDAR
Command-->UserProgram: LiDAR Started successfully
UserProgram->LaserScan: Get Laser Scan Data
LaserScan-->UserProgram:  Laser Scan Data Recvied
note over UserProgram: doProcessSimple
loop Laser Scan Data
  LaserScan->UserProgram: doProcessSimple
end
note over UserProgram: TurnOn
UserProgram->Command: TurnOff
note over UserProgram: disconnecting
UserProgram->Command: disconnecting
```

Chapter 4

General FAQs

I am new to the YDLIDAR SDK project, where do I start?

You have several options:

- To build YDLidar SDK your computer, start by reviewing the https://github.com/YDLIDAR/YDLidar-SDK/blob/master/README.md "README.md"
- To install and build YDLIDAR SDK on a robot Project, go to: https://github.com/YDLIDAR/YDLidar-SDK/blob/master/doc/Tutorials.md "YDLIDAR SDK quick start".

How do I send a pull request?

Sending a pull request is simple.

1. Fork the YDLidar-SDK Repository into your GitHub.
2. Create a Developer Branch in your Repository.
3. Commit your change in your Developer Branch.
4. Send the pull request from your GitHub Repository Webpage.

How do I install sdk python API separately?

Follow these steps:

1. install swig: `sudo apt-get install swig`
2. build sdk: `pyhton setup.py build`
3. install sdk: `pyhton setup.py install`

More General FAQs to follow.

Chapter 5

General FAQs_cn

请问我怎么样使用**pull request**?

使用pull request非常简单。

1. 将YDLidar-SDk Repository fork到你自己的Github中。
2. 在你的Repository中建立一个开发者 Branch。
3. 在开发者Branch中commit你做的任何的改变
4. 在你的github网页中使用pull request

参考更多的FAQs

Chapter 6

Hardware FAQs

Which types of YD LiDAR are supported by YDLIDAR-SDK?

please visit https://github.com/YDLIDAR/YDLidar-SDK/blob/master/doc/Dataset.md "this" page.

More Hardware FAQs to follow.

Chapter 7

硬件FAQs:

YDLIDAR雷达需要什么硬件支持?

- 芯片主频大于30MHz.
- 如果芯片主频太低， 数据不能实时解析，数据将会丢失，一些角度范围会丢失，比如:Arduino UNO(16 MHz).
- 推荐最小主频大于30MHz才能实时解析雷达数据，如果是TG30这种采样率20K， 需更高的主频.
- YDLidar-SDK 不支持控制器芯片，如STM32, Arduino.

YDLIDAR雷达可以在什么样的开发板上使用?

- 雷达采样率小于6K的，开发板主频大于30MHz就可以.
- 更改采样率雷达，开发板主频大于100MHz.

YDLidar-SDK支持哪些雷达型号?

YDLidar-SDK 支持现有所有EAI标品雷达，定制版本请联系EAI

Chapter 8

FAQs

Chapter 9

Software FAQs

Can other operating systems besides Ubuntu and windows be used?

We have only tested on Ubuntu and windows which means it's the only operating system we currently officially support. Users are always welcome to try different operating systems and can share their patches with the community if they are successfully able to use them.

Can other Languages besides C/C++,Python, C# be used?

We have only tested on C/C++, Python,C# which means it's the only Languages we currently officially support. Users are always welcome to try different Languages through swig and can share their patches with the community if they are successfully able to use them.

More Software FAQs to follow.

Chapter 10

软件FAQ

除了Ubuntu和Windows之外，其他操作系统还能使用吗?

我们只对Ubuntu和Windows进行了测试，这意味着它是我们目前正式支持的操作系统。欢迎开发者尝试不同的操作系统，如果能够成功地使用它们，可以分享补丁与社区。

除了支持C/C++,Python,C::之外，其他语言还能使用吗?

我们只对C/C++,Python,C::进行了测试，这意味着它是我们目前正式支持的语言。欢迎开发者通过SWIG尝试不同的语言，如果能够成功地使用它们，可以分享补丁与社区。

更多的软件常见问题。

Chapter 11

How to Build and Debug using VSCode

Visual Studio Code (hereafter referred to as VSCode) is Microsoft's first lightweight code editor for Linux. Find below a few configuration files that allow the use of VSCode to compile and debug the YDLidar-SDK project. I will elaborate on it below, hoping to bring some help to the developers.

Compile the YDLidar-SDK project using VSCode

You could first set up the YDLidar-SDK project using the build and release document under **Build in Visual Studio Code**. Only follow the steps until the `Build the YDLidar-SDK Project in VSCode title`

In the pop-up window, select the corresponding The options are as shown below:

Chapter 12

How to Build and Install

Install CMake

The installation procedures in Ubuntu 18.04/16.04/14.04 LTS and Windows 7/10 are shown here as examples. For Ubuntu 18.04/16.04/14.04 32-bit LTS and Mac, you can get it in YDLidar-SDK wiki. YDLidar SDK requires CMake 2.8.2+ as dependencies.

Ubuntu 18.04/16.04/14.04 LTS

You can install these packages using apt:

```
1 sudo apt install cmake pkg-config
```

if you want to use python API, you need to install pyhton and swig:

```
1 sudo apt-get install python swig
2 sudo apt-get install python-pip
```

Windows 7/10

vcpkg is recommended for building the dependency libraries as follows: For the 32-bit project:

```
1 .\vcpkg install cmake
2 .\vcpkg integrate install
```

For the 64-bit project:

```
1 .\vcpkg install cmake:x64-windows
2 .\vcpkg integrate install
```

if you want to use python API, you need to install pyhton and swig: python office install swig office install

Build YDLidar-SDK

Ubuntu 18.04/16.04/14.04 LTS

In the YDLidar SDK directory, run the following commands to compile the project:

```
1 git clone https://github.com/YDLIDAR/YDLidar-SDK.git
2 cd YDLidar-SDK/build
3 cmake ..
4 make
5 sudo make install
```

Note: If already installed python and swig, `sudo make install` command will also install python API without the following operations.

python API install separtately:

The Next operation only installs the python API, if the above command has been executed, there is no need to perform the next operation.

```
1 cd YDLidar-SDK
2 pip install .
3
4 # Another method
5 python setup.py build
6 python setup.py install
```

Windows 7/10

Then, in the YDLidar SDK directory, run the following commands to create the Visual Studio solution file. Please replace [vcpkgroot] with your vcpkg installation path. Generate the 32-bit project:

```
1 cd build && \
2 cmake .. "-DCMAKE_TOOLCHAIN_FILE=[vcpkgroot]\scripts\buildsystems\vcpkg.cmake"
```

Generate the 64-bit project:

```
1 cd build && \
2 cmake .. -G "Visual Studio 15 2017 Win64"
       "-DCMAKE_TOOLCHAIN_FILE=[vcpkgroot]\scripts\buildsystems\vcpkg.cmake"
```

Note:

- For build C# API, set BUILD_CSHARP option to ON.
- You need to install Swig, When building C# API. eg:

```
1 cmake -DBUILD_CSHARP=ON .. -G "Visual Studio 15 2017 Win64"
       "-DCMAKE_TOOLCHAIN_FILE=[vcpkgroot]\scripts\buildsystems\vcpkg.cmake"
```

Compile YDLidar SDK

You can now compile the YDLidar SDK in Visual Studio. Note:

- For more windows build and Run, Please refer to How to gerenrate Vs Project by CMake
- For VS2017 or higher, Please refer to `CMake projects in visual studio`

Packaging Project

```
1 cpack
```

Run YDLidar SDK Sample

Three samples are provided in samples, which demonstrate how to configure YDLidar LiDAR units and receive the laser scan data when directly connecting YDLidar SDK to LiDAR units or by using a YDLidar Adapter board, respectively. The sequence diagram is shown as below:

Ubuntu 18.04/16.04 /14.04 LTS

For Ubuntun 18.04/16.04/14.04 LTS, run the *ydlidar_test* if connect with the Triangle LiDAR unit(s) or TOF LiDAR unit(s):

```
1 ./ydlidar_test
```

Windows 7/10

After compiling the YDLidar SDK as shown in section 4.1.2, you can find `ydlidar_test.exe` in the {YDLidar-SDK} or {YDLidar-SDK} folder, respectively, which can be run directly.

Then you can see SDK initializing the information as below:

Then you can see SDK Scanning the information as below:

Connect to the specific LiDAR units

Samples we provided will connect all the LiDAR device in you USB in default.There are two ways to connect the specific units:

- run sample with input options in serial port.
- run sample with input options in network.

Program Options

We provide the following program options for connecting the specific units :

```
1 [Please select the lidar baudrate:]: input LiDAR BaudRate.
2 [Whether the Lidar is one-way communication[yes/no]:]: Whether The Current LiDAR is single-channel.
3 [Whether the Lidar is a TOF Lidar [yes/no]:]: Whether The Current LiDAR is TOF Type LiDAR.
4 [Please enter the lidar scan frequency[5-12]:]:input LiDAR Scan Frequency.
```

Here is the example:

```
1 __   ______  _     ___ ____     _    ____
2 \ \ / /  _ \| |   |_ _|  _ \   / \  |  _ \
3  \ V /| | | | |    | || | | |/ _ \ | |_) |
4   | | | |_| | |___ | || |_| / ___ \|  _ <
5   |_| |____/|_____|___|____/_/   \_\_| \_\
6
7 Baudrate:
8 0. 115200
9 1. 128000
10 2. 153600
11 3. 230400
12 4. 512000
13 Please select the lidar baudrate:4
14 Whether the Lidar is one-way communication[yes/no]:no
15 Whether the Lidar is a TOF Lidar [yes/no]:yes
16 Please enter the lidar scan frequency[5-12]:10
```

Python Run

```
1 cd python/examples
2 # Console
3 python tof_test.py
4 # If it's a drawing
5 pip install numpy
6 pip install matplotlib
7 python plot_tof_test.py
```

Build in Visual Studio Code

Install VSCode

The easiest way to install for Debian/Ubuntu based distributions is to download from https://code.↩visualstudio.com and install the .deb package (64-bit) either through the graphical software center if it's available or through the command line with:

```
1 sudo dpkg -i <file>.deb
2 sudo apt-get install -f # Install dependencies
```

Start VSCode

Start VSCode with the following command:

```
1 code
```

Open the YDLidar-SDK project in VSCode

Use the keyboard shortcut ∗∗(Ctrl+K Ctrl+O)∗∗ to open the YDLidar-SDK project.

Build the YDLidar-SDK project in VSCode

Use the keyboard shortcut ∗∗(Ctrl+Shift+B)∗∗ to build the YDLidar-SDK project.

Run all unit tests for the YDLidar-SDK project in VSCode

Select the "Tasks->Run Tasks..." menu command and click "run all unit tests for the YDLidar-SDK project" from a popup menu to check the code style for the YDLidar-SDK project.

Run a code style check task for the YDLidar-SDK project in VSCode

Select the "Tasks->Run Tasks..." menu command and click "code style check for the YDLidar-SDK project" from a popup menu to check the code style for the YDLidar-SDK project.

Clean the YDLidar-SDK project in VSCode

Select the "Tasks->Run Tasks..." menu command and click "clean the YDLidar-SDK project" from a popup menu to clean the YDLidar-SDK project.

Chapter 13

How to create a pull request

You can follow the standard `github approach` to contribute code to YDLidar-SDK. Here is a sample setup:

- Fork a new repo with your GitHub username.
- Set up your GitHub personal email and user name

```
1 git config user.name "XXX"
2 git config user.email "XXX@[XXX.com]"
```

- Clone your fork (Please replace "USERNAME" with your GitHub user name.)

```
1 (Use SSH) git clone git@github.com:USERNAME/YDLidar-SDK.git
2 (Use HTTPS) git clone https://github.com/USERNAME/YDLidar-SDK.git
```

- Add YDLidar-SDK repository as upstream

```
1 (Use SSH) git remote add upstream git@github.com:YDLIDAR/YDLidar-SDK.git
2 (Use HTTPS) git remote add upstream https://github.com/YDLIDAR/YDLidar-SDK.git
```

- Confirm that the upstream branch has been added

```
1 git remote -v
```

- Create a new branch, make changes and commit

```
1 git checkout -b  "my_dev"
```

- Sync up with the YDLIDAR/YDLidar-SDK repo

```
1 git pull --rebase  upstream master
```

- Push local developments to your own forked repository

```
1 git push -f -u origin "my_dev"
```

- Generate a new pull request between "YDLIDAR/YDLidar-SDK:master" and "forked repo:my_dev"
- Collaborators will review and merge the commit (this may take some time, please be patient)

Thanks a lot for your contributions!

Chapter 14

How to create a udev rules

Introduction

The serial port is used under Linux. The serial port number will change with the insertion order of multiple serial ports. This problem can be solved by setting the serial port alias.

Create The New UDEV Rules

Create a new `ydlidar_ports.rules` file and write the corresponding serial port rules to the file.

Create new udev file

```
1 sudo gedit /etc/udev/rules.d/ydlidar_ports.rules
```

or

```
1 sudo vim /etc/udev/rules.d/ydlidar_ports.rules
```

Query serial port number through udevadm

```
1 udevadm info -a -n /dev/ttyUSB0 | grep KERNELS
```

result as follows:

```
1 udevadm info -a -n /dev/ttyUSB1 | grep KERNELS
```

result as follows:

Create UDEV Permission Rule For tty Devices

Write the first KERNELS queried above into the new `ydlidar_ports.rules` file. Add these two following rules in it.

```
1 {ydlidar_ports.rules}
2 SUBSYSTEM=="tty", KERNELS=="1-1:1.0", SYMLINK+="ydlidar", MODE="0666", GROUP:="dialout"
3 SUBSYSTEM=="tty", KERNELS=="1-2:1.0", SYMLINK+="ydlidar1", MODE="0666", GROUP:="dialout"
```

Restart The UDEV Service

Save the file and close it. Then as root, tell systemd-udevd to reload the rules files (this also reloads other databases such as the kernel module index), by running.

```
1 sudo udevadm control --reload
```

and

```
1 sudo service udev reload
2 sudo service udev restart
```

Note: If it doesn't work, plug and unplug the USB or restart the computer

You can query the corresponding results with the following command

```
1 ls -l /dev/ydlidar*
2
3 lrwxrwxrwx 1 root dialout 7 Feb 17 13:27 /dev/ydlidar -> ttyUSB0
4 lrwxrwxrwx 1 root dialout 7 Feb 17 13:27 /dev/ydlidar1 -> ttyUSB1
```

Chapter 15

Introduction

The Visual Studio version recommended by YDLidar-SDK to use is Visual Studio 2017. This document decribes steps to run YDLidar-SDK on Visual Studio 2017.

The YDLidar-SDK version used in this document is the lastest release version which is 1.0.0. And this document focuses on How to Install Software, and conforms to the steps and rules provided by YDLidar-SDK.

Download YDLidar-SDK

please refer to https://github.com/YDLIDAR/YDLidar-SDK/blob/master/docs/quickstart/ydlidar_sdk_software_↩ installation_guide.md "YDLidar-SDK Software Installation", download YDLidar-SDK version 1.0.0 source code onto the computer.

Install CMake

Please follow the official guide to install the cmake.

Build and Run YDLIDAR SDK

Please refer to https://github.com/YDLIDAR/YDLidar-SDK/blob/master/doc/howto/how_to_build_and_release.md "How to build and release".

Chapter 16

Github访问慢解决方案

浏览器打开如下网站

http://github.global.ssl.fastly.net.ipaddress.com/

找到对应IP地址，例如：151.101.xx.xx

浏览器打开另外一个网站

http://github.com.ipaddress.com/

找到对应IP地址。例如：192.30.xx.xx

编辑hosts文件

```
1 sudo vim /etc/hosts
```

在文件中加入如下两行

```
1 192.30.xx.xx github.com
2 151.101.xx.xx github.global.ssl.fastly.net
```

如果使用mac，还需更新DNS缓存

```
1 sudo dscacheutil -flushcache
```

Chapter 17

Howto Guides

Build

- How to build and install
- How to build and debug using VSCode

Contribution

- How to create a pull request

Others

- How to create a udev rules

Chinese versions

- How to install ubuntu
- How to solve slow pull from cn

Chapter 18

Quick Start Guides

YDLidar-SDK 1.0.0

- YDLidar-SDK 1.0.0 quick start
- YDLidar-SDK 1.0.0 quick start cn
- YDLidar-SDK 1.0.0 hardware system installation guide
- YDLidar-SDK 1.0.0 quick start developer

Others

- YDLidar-SDK software installation guide

Chapter 19

Software Overview of YDLidar-SDK

YDLidar-SDK has been initiated to provide an open, comprehensive, and reliable software platform for its partners in the robot, Large screen interaction and mapping industries.

YDLIDAR SDK Software Installation

This section includes:

- Download the YDLidar-SDK Release Package
- Run YDLidar-SDK

Before getting started, please make sure you have installed Linux or Windows.

∗New - Git∗

git Installation

ubuntu 14.04 / 16.04 / 18.04

```
1 sudo apt-get install -y git
```

Windows

Installation

Download YDLidar-SDK Source

1. Download YDLidar-SDK source code from the github source and check out the correct branch:

   “‘ git clone git@github.com:YDLIDAR/YDLidar-SDK.git cd YDLidar-SDK git checkout [release_branch↩ _name] “‘

Run YDLidar-SDK

Please refer to https://github.com/YDLIDAR/YDLidar-SDK/blob/master/doc/howto/how_to_build_and_release.md "How to build and release"

Chapter 20

YDLIDAR SDK Documents

Quick Start Guide

README - A hardware and software guide to setting up YDLidar-SDK, segregated by versions

Communication Protocol

YDLIDAR SDK Communication Protocol - All you need to know about YDLiDAR-SDK Communication Protocol.

API

YDLIDAR SDK API for Developers - All you need to know about YDLiDAR-SDK API

Howto Guides

README - Brief technical solutions to common problems that developers face during the installation and use of the YDLiDAR-SDK

Tutorials

Tutorials-Quick Tutorials.

FAQs

README - Commonly asked questions about YDLidar-SDK's setup

Chapter 21

Examining the Simple Lidar Tutorial

Description: This tutorial examines running the simple lidar tutorial. Tutorial Level: BEGINNER Previous Tutorial: Writing a Simple Lidar Tutorial (c)(python) (writing_lidar_tutorial_c++.md "c++")

Table of Contents

Running the Lidar Turorial

In the last tutorial we made a tutorial called "lidar_tutorial". Let's run it:

```
1 ./lidar_tutorial               (C++)(C)
2 python lidar_tutorial.py       (Python)
```

You will see something similar to:

```
YDLidar SDK initializing
YDLidar SDK has been initialized
[YDLIDAR]:SDK Version: 1.0.0
LiDAR successfully connected
[YDLIDAR]:Lidar running correctly ! The health status: good
[YDLIDAR] Connection established in [/dev/ttyUSB0][230400]:
Firmware version: 1.3
Hardware version: 1
Model: G4
Serial: 2020010200010001
[YDLIDAR INFO] Current Scan Frequency: 10.000000Hz
LiDAR init success!
[YDLIDAR]:Fixed Size: 900
[YDLIDAR]:Sample Rate: 9K
[YDLIDAR INFO] Current Sampling Rate : 9K
[YDLIDAR INFO] Now YDLIDAR is scanning ......
Scan received[1582955714469712000]: 964 ranges is [9.342144]Hz
Scan received[1582955714607423000]: 954 ranges is [9.429421]Hz
Scan received[1582955714742073000]: 949 ranges is [9.485030]Hz
Scan received[1582955714875723000]: 946 ranges is [9.513148]Hz
Scan received[1582955715008873000]: 943 ranges is [9.547170]Hz
Scan received[1582955715141423000]: 938 ranges is [9.593015]Hz
Scan received[1582955715273253000]: 933 ranges is [9.651109]Hz
Scan received[1582955715404003000]: 930 ranges is [9.686395]Hz
Scan received[1582955715534183000]: 928 ranges is [9.698230]Hz
Scan received[1582955715664261000]: 919 ranges is [9.794232]Hz
Scan received[1582955715792611000]: 906 ranges is [9.936508]Hz
```

When you are done, press Ctrl-C to terminate both the lidar tutorial.

Note: ERROR

- ./lidar_tutorial: error while loading shared libraries: libydlidar_sdk.so: cannot open shared object file: No such file or directory if the above error occurs, the operation is as follows:

```
1 cat /etc/ld.so.conf
2         include ld.so.conf.d/*.conf
3 ## ydlidar_sdk library path added to ld.so.conf.
4 echo "/usr/local/lib" >> /etc/ld.so.conf
5 sudo ldconfig
```

OR

```
1 ## ydlidar_sdk library path added to LD_LIBRARY_PATH
2 export LD_LIBRARY_PATH=/usr/local/lib:$LD_LIBRARY_PATH
```

Chapter 22

WritingLidarTutorial(c++)

Description: This tutorial covers how to write a lidar tutorial in C++. Tutorial Level: BEGINNER Next Tutorial: Examining the simple lidar tutorial

Table of Contents

Writing a Simple lidar tutorial (C++)

Description: This tutorial covers how to write a LiDAR data console program in C++. Tutorial Level: BEGINNER

create beginner_tutorials directories

```
1 mkdir beginner_tutorials
2 cd beginner_tutorials
```

Create the lidar_tutorial.cpp file within the beginner_tutorials project and paste the following inside it:

https://github.com/YDLIDAR/ydlidar_tutorials/blob/master/cpp_tutorials/lidar↩
_tutorial/lidar_tutorial.cpp

```
1 {c++}
2 #include "CYdLidar.h"
3 #include <string>
4 using namespace std;
5 using namespace ydlidar;
6
7 #if defined(_MSC_VER)
8 #pragma comment(lib, "ydlidar_sdk.lib")
9 #endif
10
11 int main(int argc, char *argv[]) {
12   // init system signal
13   ydlidar::os_init();
14
```

```
15  CYdLidar laser;
16  ////////////////////string property/////////////////
17  /// Lidar ports
18  std::map<std::string, std::string> ports = ydlidar::lidarPortList();
19  std::string port = "/dev/ydlidar";
20  if(!ports.empty()) {
21      port = ports.begin()->second;
22  }
23  /// lidar port
24  laser.setlidaropt(LidarPropSerialPort, port.c_str(), port.size());
25  /// ignore array
26  std::string ignore_array;
27  ignore_array.clear();
28  laser.setlidaropt(LidarPropIgnoreArray, ignore_array.c_str(),
29                    ignore_array.size());
30
31  ////////////////////int property/////////////////
32  /// lidar baudrate
33  int optval = 230400;
34  laser.setlidaropt(LidarPropSerialBaudrate, &optval, sizeof(int));
35  /// tof lidar
36  optval = TYPE_TRIANGLE;
37  laser.setlidaropt(LidarPropLidarType, &optval, sizeof(int));
38  /// device type
39  optval = YDLIDAR_TYPE_SERIAL;
40  laser.setlidaropt(LidarPropDeviceType, &optval, sizeof(int));
41  /// sample rate
42  optval = 9;
43  laser.setlidaropt(LidarPropSampleRate, &optval, sizeof(int));
44  /// abnormal count
45  optval = 4;
46  laser.setlidaropt(LidarPropAbnormalCheckCount, &optval, sizeof(int));
47
48  ////////////////////bool property/////////////////
49  /// fixed angle resolution
50  bool b_optvalue = false;
51  laser.setlidaropt(LidarPropFixedResolution, &b_optvalue, sizeof(bool));
52  /// rotate 180
53  laser.setlidaropt(LidarPropReversion, &b_optvalue, sizeof(bool));
54  /// Counterclockwise
55  laser.setlidaropt(LidarPropInverted, &b_optvalue, sizeof(bool));
56  b_optvalue = true;
57  laser.setlidaropt(LidarPropAutoReconnect, &b_optvalue, sizeof(bool));
58  /// one-way communication
59  b_optvalue = false;
60  laser.setlidaropt(LidarPropSingleChannel, &b_optvalue, sizeof(bool));
61  /// intensity
62  b_optvalue = false;
63  laser.setlidaropt(LidarPropIntenstiy, &b_optvalue, sizeof(bool));
64  /// Motor DTR
65  b_optvalue = false;
66  laser.setlidaropt(LidarPropSupportMotorDtrCtrl, &b_optvalue, sizeof(bool));
67
68  ////////////////////float property/////////////////
69  /// unit: °
70  float f_optvalue = 180.0f;
71  laser.setlidaropt(LidarPropMaxAngle, &f_optvalue, sizeof(float));
72  f_optvalue = -180.0f;
73  laser.setlidaropt(LidarPropMinAngle, &f_optvalue, sizeof(float));
74  /// unit: m
75  f_optvalue = 16.f;
76  laser.setlidaropt(LidarPropMaxRange, &f_optvalue, sizeof(float));
77  f_optvalue = 0.1f;
78  laser.setlidaropt(LidarPropMinRange, &f_optvalue, sizeof(float));
79  /// unit: Hz
80  f_optvalue = 10.f;
81  laser.setlidaropt(LidarPropScanFrequency, &f_optvalue, sizeof(float));
82
83  // initialize SDK and LiDAR
84  bool ret = laser.initialize();
85  if (ret) {//success
86    //Start the device scanning routine which runs on a separate thread and enable motor.
87    ret = laser.turnOn();
88  } else {
89    fprintf(stderr, "%s\n", laser.DescribeError());
90    fflush(stderr);
91  }
92
93  // Turn On success and loop
94  while (ret && ydlidar::os_isOk()) {
95    LaserScan scan;
96    if (laser.doProcessSimple(scan)) {
97      fprintf(stdout, "Scan received[%llu]: %u ranges is [%f]Hz\n",
98              scan.stamp,
99              (unsigned int)scan.points.size(), 1.0 / scan.config.scan_time);
100     fflush(stdout);
101   } else {
```

```
102       fprintf(stderr, "Failed to get Lidar Data\n");
103       fflush(stderr);
104     }
105   }
106   // Stop the device scanning thread and disable motor.
107   laser.turnOff();
108   // Uninitialize the SDK and Disconnect the LiDAR.
109   laser.disconnecting();
110   return 0;
111 }
```

The Code Explained

Now, let's break the code down.

```
1 {c++}
2 #include "CYdLidar.h"
```

CYdLidar.h is a convenience include that includes all the headers necessary to use the most common public pieces of the YDLIDAR SDK.

```
1 {c++}
2   ydlidar::os_init();
```

Initialize system signal. install a SIGINT handler which provides Ctrl-C handling

```
1 {c++}
2   CYdLidar laser;
```

Create a handle to this Lidar.

```
1 {c++}
2  ////////////////////string property/////////////////
3   /// Lidar ports
4   std::map<std::string, std::string> ports = ydlidar::lidarPortList();
5   std::string port = "/dev/ydlidar";
6   if(!ports.empty()) {
7       port = ports.begin()->second;
8   }
```

Query avaliable Lidar ports.

```
1 {c++}
2   /// lidar port
3   laser.setlidaropt(LidarPropSerialPort, port.c_str(), port.size());
4   /// ignore array
5   std::string ignore_array;
6   ignore_array.clear();
7   laser.setlidaropt(LidarPropIgnoreArray, ignore_array.c_str(),
8                     ignore_array.size());
```

Set Lidar string property paramters.

```c++
  ////////////////////int property/////////////////
  /// lidar baudrate
  int optval = 230400;
  laser.setlidaropt(LidarPropSerialBaudrate, &optval, sizeof(int));
  /// tof lidar
  optval = TYPE_TRIANGLE;
  laser.setlidaropt(LidarPropLidarType, &optval, sizeof(int));
  /// device type
   optval = YDLIDAR_TYPE_SERIAL;
   laser.setlidaropt(LidarPropDeviceType, &optval, sizeof(int));
   /// sample rate
   optval = 9;
   laser.setlidaropt(LidarPropSampleRate, &optval, sizeof(int));
   /// abnormal count
   optval = 4;
   laser.setlidaropt(LidarPropAbnormalCheckCount, &optval, sizeof(int));
```

Set Lidar string int paramters.

```c++
  ////////////////////bool property/////////////////
  /// fixed angle resolution
  bool b_optvalue = false;
  laser.setlidaropt(LidarPropFixedResolution, &b_optvalue, sizeof(bool));
  /// rotate 180
  laser.setlidaropt(LidarPropReversion, &b_optvalue, sizeof(bool));
  /// Counterclockwise
  laser.setlidaropt(LidarPropInverted, &b_optvalue, sizeof(bool));
   b_optvalue = true;
   laser.setlidaropt(LidarPropAutoReconnect, &b_optvalue, sizeof(bool));
   /// one-way communication
   b_optvalue = false;
   laser.setlidaropt(LidarPropSingleChannel, &b_optvalue, sizeof(bool));
   /// intensity
   b_optvalue = false;
   laser.setlidaropt(LidarPropIntenstiy, &b_optvalue, sizeof(bool));
   /// Motor DTR
   b_optvalue = false;
   laser.setlidaropt(LidarPropSupportMotorDtrCtrl, &b_optvalue, sizeof(bool));
```

Set Lidar bool property paramters.

```c++
  ////////////////////float property/////////////////
  /// unit: °
  float f_optvalue = 180.0f;
  laser.setlidaropt(LidarPropMaxAngle, &f_optvalue, sizeof(float));
  f_optvalue = -180.0f;
  laser.setlidaropt(LidarPropMinAngle, &f_optvalue, sizeof(float));
  /// unit: m
  f_optvalue = 16.f;
   laser.setlidaropt(LidarPropMaxRange, &f_optvalue, sizeof(float));
   f_optvalue = 0.1f;
   laser.setlidaropt(LidarPropMinRange, &f_optvalue, sizeof(float));
   /// unit: Hz
   f_optvalue = 10.f;
   laser.setlidaropt(LidarPropScanFrequency, &f_optvalue, sizeof(float));
```

Set Lidar float property paramters.

```c++
  // initialize SDK and LiDAR
  bool ret = laser.initialize();
```

Initialize the SDK and LiDAR.

`initialize` will return false if:

- Serial port does not correspond to the actual Lidar.

- Serial port does not have read and write permissions.
- Lidar baud rate settings error.
- Incorrect Lidar type setting.

```
1 {c++}
2   if (ret) {//success
3     //Start the device scanning routine which runs on a separate thread and enable motor.
4     ret = laser.turnOn();
5   } else {
6     fprintf(stderr, "%s\n", laser.DescribeError());
7     fflush(stderr);
8   }
```

Start the device scanning routine which runs on a separate thread and enable motor.

`turnOn` will return false if:

- Lidar stall.
- Lidar power suppy is unstable.

```
1 {c++}
2   // Turn On success and loop
3   while (ret && ydlidar::os_isOk()) {
```

By `ydlidar::os_init()` will install a SIGINT handler which provides Ctrl-C handling which will cause `ydlidar::os_isOk()` to return false if that happens.

`ydlidar::os_isOk()` will return false if:

- a SIGINT is received (Ctrl-C)
- ydlidar::os_shutdown() has been called by another part of the application.

Once `ydlidar::os_isOk()` returns false, Loop exit.

```
1 {c++}
2     LaserScan scan;
3     if (laser.doProcessSimple(scan)) {
4       fprintf(stdout, "Scan received[%llu]: %u ranges is [%f]Hz\n",
5               scan.stamp,
6               (unsigned int)scan.points.size(), 1.0 / scan.config.scan_time);
7       fflush(stdout);
8     } else {
9       fprintf(stderr, "Failed to get Lidar Data\n");
10       fflush(stderr);
11     }
```

Get the LiDAR Scan Data.

```
1 {c++}
2   // Stop the device scanning thread and disable motor.
3   laser.turnOff();
```

Stop the device scanning thread and disable motor.

```
1 {c++}
2   // Uninitialize the SDK and Disconnect the LiDAR.
3   laser.disconnecting();
```

Uninitialize the SDK and Disconnect the LiDAR.

Building your project

You need to create a CMakeLists.txt file.

The generated CMakeLists.txt should look like this: https://github.com/YDLIDAR/ydlidar_tutorials/blob/master/cpp_tutorials/lidar_tutorial/CMakeLists.txt

```
1 cmake_minimum_required(VERSION 2.8)
2 PROJECT(lidar_tutorial)
3 set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++11")
4 add_definitions(-std=c++11) # Use C++11
5
6
7 #Include directories
8 include_directories(
9      ${CMAKE_SOURCE_DIR}
10 )
11 ############## YDLIDAR SDK START####################################
12 #find ydlidar_sdk package
13 find_package(ydlidar_sdk REQUIRED)
14 #Include directories
15 include_directories(
16   ${YDLIDAR_SDK_INCLUDE_DIRS}
17 )
18
19 #link library directories
20 link_directories(${YDLIDAR_SDK_LIBRARY_DIRS})
21
22 add_executable(${PROJECT_NAME} lidar_tutorial.cpp)
23
24 #Link your project to ydlidar_sdk library.
25 target_link_libraries(${PROJECT_NAME} ${YDLIDAR_SDK_LIBRARIES})
26
27 ############## YDLIDAR SDK END####################################
```

This will create one executable, lidar_tutorial, which by default will go into package directory of your build space.

Linux:

- `YDLIDAR_SDK_LIBRARIES` includes `ydlidar_sdk pthread rt`
- If you need the pthread library at the end of the compilation flag, you need to put `YDLIDAR_SDK_LIBRARIES` at the end.

you can use the following variable to depend on all necessary targets:

```
1 target_link_libraries(${PROJECT_NAME} ${YDLIDAR_SDK_LIBRARIES})
```

Now run cmake:

```
1 # In your project directory
2 mkdir build
3 cd build
4 cmake ..
5 make j4
```

Now that you have written a simple lidar tutorial, let's examine the simple lidar tutorial.

Chapter 23

WritingLidarTutorial(C)

Description: This tutorial covers how to write a lidar tutorial in C. Tutorial Level: BEGINNER Next Tutorial: Examining the simple lidar tutorial

Table of Contents

Writing a Simple lidar tutorial (C)

Description: This tutorial covers how to write a LiDAR data console program in C. Tutorial Level: BEGINNER

create beginner_tutorials directories

```
1 mkdir beginner_tutorials
2 cd beginner_tutorials
```

Create the lidar_tutorial.cpp file within the beginner_tutorials project and paste the following inside it:

https://github.com/YDLIDAR/ydlidar_tutorials/blob/master/c_tutorials/lidar_tutorial/lidar_tutorial.c

```
//
// The MIT License (MIT)
//
// Copyright (c) 2019 EAIBOT. All rights reserved.
//
// Permission is hereby granted, free of charge, to any person obtaining a copy
// of this software and associated documentation files (the "Software"), to deal
// in the Software without restriction, including without limitation the rights
// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
// copies of the Software, and to permit persons to whom the Software is
// furnished to do so, subject to the following conditions:
//
// The above copyright notice and this permission notice shall be included in
// all copies or substantial portions of the Software.
```

```
//
// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
// SOFTWARE.
//

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#ifdef WIN32
#include <windows.h>
#else
#include <unistd.h>
#endif
#include "ydlidar_sdk.h"

#if defined(_MSC_VER)
#pragma comment(lib, "ydlidar_sdk.lib")
#endif

int main(int argc, const char *argv[]) {

    os_init();
    YDLidar *laser = lidarCreate();
    //string prop
    char port[50] = "/dev/ydlidar";
    LidarPort ports;
    int size = lidarPortList(&ports);
    int i = 0;
    for(i =0; i < size; i++) {
        printf("port: %s\n", ports.port[i].data);
        strcpy(port, ports.port[i].data);
    }
    setlidaropt(laser, LidarPropSerialPort, port, sizeof(port));
    strcpy(port, "");
    setlidaropt(laser, LidarPropIgnoreArray,port, sizeof(port));

    //int prop
    int i_optvalue = 512000;
    setlidaropt(laser, LidarPropSerialBaudrate, &i_optvalue, sizeof(int))
      ;
    i_optvalue = TYPE_TOF;
    setlidaropt(laser, LidarPropLidarType, &i_optvalue, sizeof(int));
    i_optvalue = YDLIDAR_TYPE_SERIAL;
    setlidaropt(laser, LidarPropDeviceType, &i_optvalue, sizeof(int));
    i_optvalue = 20;
    setlidaropt(laser, LidarPropSampleRate, &i_optvalue, sizeof(int));

    //bool prop
    bool b_optval = true;
    setlidaropt(laser, LidarPropAutoReconnect, &b_optval, sizeof(bool));
    b_optval = false;
    setlidaropt(laser, LidarPropSingleChannel, &b_optval, sizeof(bool));
    setlidaropt(laser, LidarPropIntenstiy, &b_optval, sizeof(bool));
    setlidaropt(laser, LidarPropInverted, &b_optval, sizeof(bool));
    setlidaropt(laser, LidarPropReversion, &b_optval, sizeof(bool));
    setlidaropt(laser, LidarPropSupportMotorDtrCtrl, &b_optval,
      sizeof(bool));
    setlidaropt(laser, LidarPropFixedResolution, &b_optval, sizeof(bool)
      );

    //float prop
    float f_optval = 10.f;
    setlidaropt(laser, LidarPropScanFrequency, &f_optval, sizeof(float));
    f_optval = 180.0f;
    setlidaropt(laser, LidarPropMaxAngle, &f_optval, sizeof(float));
    f_optval = -180.0f;
    setlidaropt(laser, LidarPropMinAngle, &f_optval, sizeof(float));
    f_optval = 64.f;
    setlidaropt(laser, LidarPropMaxRange, &f_optval, sizeof(float));
    f_optval = 0.05f;
    setlidaropt(laser, LidarPropMinRange, &f_optval, sizeof(float));

    getlidaropt(laser, LidarPropSerialBaudrate,&i_optvalue, sizeof(int));
    printf("baudrate: %d\n", i_optvalue);

    bool ret = initialize(laser);
    if(ret) {
        ret = turnOn(laser);
    }

    LaserFan scan;
    LaserFanInit(&scan);
```

```
    while (ret && os_isOk()) {
        if(doProcessSimple(laser, &scan)) {
            fprintf(stdout, "Scan received[%llu]: %u ranges is [%f]Hz\n",
                    scan.stamp,
                    (unsigned int)scan.npoints, 1.0 / scan.config.
      scan_time);
            fflush(stdout);
        } else {
            fprintf(stderr, "Failed to get Lidar Data\n");
            fflush(stderr);
        }
    }
    LaserFanDestroy(&scan);
    turnOff(laser);
    disconnecting(laser);
    lidarDestroy(&laser);
    return 0;

}
```

The Code Explained

Now, let's break the code down.

```
1 {c++}
2 #include "ydlidar_sdk.h"
```

ydlidar_sdk.h is a convenience include that includes all the headers necessary to use the most common public pieces of the YDLIDAR SDK.

```
os_init();
```

Initialize system signal. install a SIGINT handler which provides Ctrl-C handling

```
YDLidar *laser = lidarCreate();
```

Create a handle to this Lidar.

```
//string prop
char port[50] = "/dev/ydlidar";
LidarPort ports;
int size = lidarPortList(&ports);
int i = 0;
for(i =0; i < size; i++) {
    printf("port: %s\n", ports.port[i].data);
    strcpy(port, ports.port[i].data);
}
```

Query avaliable Lidar ports.

```
setlidaropt(laser, LidarPropSerialPort, port, sizeof(port));
strcpy(port, "");
setlidaropt(laser, LidarPropIgnoreArray,port, sizeof(port));
```

Set Lidar string property paramters.

```
//int prop
int i_optvalue = 512000;
setlidaropt(laser, LidarPropSerialBaudrate, &i_optvalue, sizeof(int));
i_optvalue = TYPE_TOF;
setlidaropt(laser, LidarPropLidarType, &i_optvalue, sizeof(int));
i_optvalue = YDLIDAR_TYPE_SERIAL;
setlidaropt(laser, LidarPropDeviceType, &i_optvalue, sizeof(int));
i_optvalue = 20;
setlidaropt(laser, LidarPropSampleRate, &i_optvalue, sizeof(int));
```

Set Lidar string int paramters.

```
//bool prop
bool b_optval = true;
setlidaropt(laser, LidarPropAutoReconnect, &b_optval, sizeof(bool));
b_optval = false;
setlidaropt(laser, LidarPropSingleChannel, &b_optval, sizeof(bool));
setlidaropt(laser, LidarPropIntenstiy, &b_optval, sizeof(bool));
setlidaropt(laser, LidarPropInverted, &b_optval, sizeof(bool));
setlidaropt(laser, LidarPropReversion, &b_optval, sizeof(bool));
setlidaropt(laser, LidarPropSupportMotorDtrCtrl, &b_optval, sizeof(
      bool));
setlidaropt(laser, LidarPropFixedResolution, &b_optval, sizeof(bool));
```

Set Lidar bool property paramters.

```
//float prop
float f_optval = 10.f;
setlidaropt(laser, LidarPropScanFrequency, &f_optval, sizeof(float));
f_optval = 180.0f;
setlidaropt(laser, LidarPropMaxAngle, &f_optval, sizeof(float));
f_optval = -180.0f;
setlidaropt(laser, LidarPropMinAngle, &f_optval, sizeof(float));
f_optval = 64.f;
setlidaropt(laser, LidarPropMaxRange, &f_optval, sizeof(float));
f_optval = 0.05f;
setlidaropt(laser, LidarPropMinRange, &f_optval, sizeof(float));
```

Set Lidar float property paramters.

```
// initialize SDK and LiDAR
bool ret = initialize(laser);
```

Initialize the SDK and LiDAR.

`initialize` will return false if:

- Serial port does not correspond to the actual Lidar.
- Serial port does not have read and write permissions.
- Lidar baud rate settings error.
- Incorrect Lidar type setting.

```
if(ret) {
  ret = turnOn(laser);
}
```

Start the device scanning routine which runs on a separate thread and enable motor.

`turnOn` will return false if:

- Lidar stall.
- Lidar power suppy is unstable.

```
// Turn On success and loop
LaserFan scan;
LaserFanInit(&scan);
while (ret && os_isOk()) {
```

By `os_init()` will install a SIGINT handler which provides Ctrl-C handling which will cause `os_isOk()` to return false if that happens.

`os_isOk()` will return false if:

- a SIGINT is received (Ctrl-C)
- ydlidar::os_shutdown() has been called by another part of the application.

Once `os_isOk()` returns false, Loop exit. Note:

- `LaserFan` need to be initialized with `LaserFanInit`
- After `LaserFan` leaves the Scope, it need to be destroyed with `LaserFanDestroy`, otherwisw it will leak memory.

```
if(doProcessSimple(laser, &scan)) {
    fprintf(stdout, "Scan received[%llu]: %u ranges is [%f]Hz\n",
            scan.stamp,
            (unsigned int)scan.npoints, 1.0 / scan.config.scan_time);
    fflush(stdout);
} else {
    fprintf(stderr, "Failed to get Lidar Data\n");
    fflush(stderr);
}
```

Get the LiDAR Scan Data.

```
LaserFanDestroy(&scan);
```

Destroy LaserFan, Free up memory. Note:

- After `LaserFan` leaves the Scope, it need to be destroyed with `LaserFanDestroy`, otherwisw it will leak memory.

```
// Stop the device scanning thread and disable motor.
turnOff(laser);
```

Stop the device scanning thread and disable motor.

```
// Uninitialize the SDK and Disconnect the LiDAR.
disconnecting(laser);
```

Uninitialize the SDK and Disconnect the LiDAR.

```
1 lidarDestroy(&laser);
```

Destroy YDLidar, Free up memory.

Building your project

You need to create a CMakeLists.txt file.

The generated CMakeLists.txt should look like this: https://github.com/YDLIDAR/ydlidar_tutorials/blob/master/c_tutorials/lidar_tutorial/CMakeLists.txt

```
1 cmake_minimum_required(VERSION 2.8)
2 PROJECT(lidar_tutorial C)
3
4 #Include directories
5 include_directories(
6      ${CMAKE_SOURCE_DIR}
7 )
8 ############## YDLIDAR SDK START#####################################
9 #find ydlidar_sdk package
10 find_package(ydlidar_sdk REQUIRED)
11 #Include directories
12 include_directories(
13   ${YDLIDAR_SDK_INCLUDE_DIRS}
14 )
15 #link library directories
16 link_directories(${YDLIDAR_SDK_LIBRARY_DIRS})
17
18 add_executable(${PROJECT_NAME} lidar_tutorial.c)
19 #Link your project to ydlidar_sdk library.
20 target_link_libraries(${PROJECT_NAME} ${YDLIDAR_SDK_LIBRARIES} -lstdc++ -lm)
21
22 ############## YDLIDAR SDK END#####################################
```

This will create one executable, lidar_tutorial, which by default will go into package directory of your build space.

ydlidar_sdk dependent libraries

- C++ standard library.
- math library.

Note:

- GCC CCLDFLAGS requires "-lstdc++ -lm".

you can use the following variable to depend on all necessary targets:

```
1 target_link_libraries(${PROJECT_NAME} ${YDLIDAR_SDK_LIBRARIES} -lstdc++ -lm)
```

Now run cmake:

```
1 # In your project directory
2 mkdir build
3 cd build
4 cmake ..
5 make j4
```

Now that you have written a simple lidar tutorial, let's examine the simple lidar tutorial.

Chapter 24

WritingLidarTutorial(python)

Description: This tutorial covers how to write a lidar tutorial in Python. Tutorial Level: BEGINNER Next Tutorial: Examining the simple lidar tutorial

Table of Contents

Writing a Simple lidar tutorial (Python)

Description: This tutorial covers how to write a LiDAR data console program in Python. Tutorial Level: BEGINNER

create beginner_tutorials directories

```
1 mkdir beginner_tutorials
2 cd beginner_tutorials
```

Create the lidar_tutorial.py file within the beginner_tutorials project and paste the following inside it:

https://github.com/YDLIDAR/ydlidar_tutorials/blob/master/pyhton_tutorials/lidar_tutorial/lidar_tutorial.py

```
1 import os
2 import ydlidar
3
4 if __name__ == "__main__":
5     ydlidar.os_init();
6     laser = ydlidar.CYdLidar();
7     ports = ydlidar.lidarPortList();
8     port = "/dev/ydlidar";
9     for key, value in ports.items():
10         port = value;
11     laser.setlidaropt(ydlidar.LidarPropSerialPort, port);
12     laser.setlidaropt(ydlidar.LidarPropSerialBaudrate, 512000);
13     laser.setlidaropt(ydlidar.LidarPropLidarType, ydlidar.TYPE_TOF);
14     laser.setlidaropt(ydlidar.LidarPropDeviceType, ydlidar.YDLIDAR_TYPE_SERIAL);
15     laser.setlidaropt(ydlidar.LidarPropScanFrequency, 10.0);
16     laser.setlidaropt(ydlidar.LidarPropSampleRate, 20);
17     laser.setlidaropt(ydlidar.LidarPropSingleChannel, False);
```

```
18
19     ret = laser.initialize();
20     if ret:
21         ret = laser.turnOn();
22         scan = ydlidar.LaserScan()
23         while ret and ydlidar.os_isOk() :
24             r = laser.doProcessSimple(scan);
25             if r:
26                 print("Scan received[",scan.stamp,"]:",scan.points.size(),"ranges is
       [",1.0/scan.config.scan_time,"]Hz");
27             else :
28                 print("Failed to get Lidar Data.")
29         laser.turnOff();
30     laser.disconnecting();
```

The Code Explained

Now, let's break the code down.

```
1 import ydlidar
```

You need to import ydlidar if you are writing a YDLIDAR SDK.

```
1 ydlidar.os_init();
```

Initialize system signal. install a SIGINT handler which provides Ctrl-C handling

```
1 laser = ydlidar.CYdLidar();
```

Create a handle to this Lidar.

```
1 ports = ydlidar.lidarPortList();
2 port = "/dev/ydlidar";
3 for key, value in ports.items():
4   port = value;
```

Query avaliable Lidar ports.

```
1 laser.setlidaropt(ydlidar.LidarPropSerialPort, port);
2 laser.setlidaropt(ydlidar.LidarPropSerialBaudrate, 512000);
3 laser.setlidaropt(ydlidar.LidarPropLidarType, ydlidar.TYPE_TOF);
4 laser.setlidaropt(ydlidar.LidarPropDeviceType, ydlidar.YDLIDAR_TYPE_SERIAL);
5 laser.setlidaropt(ydlidar.LidarPropScanFrequency, 10.0);
6 laser.setlidaropt(ydlidar.LidarPropSampleRate, 20);
7 laser.setlidaropt(ydlidar.LidarPropSingleChannel, False);
```

Set Lidar property paramters.

```
1 {c++}
2   // initialize SDK and LiDAR
3   ret = laser.initialize();
```

Initialize the SDK and LiDAR.

`initialize` will return false if:

- Serial port does not correspond to the actual Lidar.

- Serial port does not have read and write permissions.
- Lidar baud rate settings error.
- Incorrect Lidar type setting.

```
1 if ret:
2   ret = laser.turnOn();
```

Start the device scanning routine which runs on a separate thread and enable motor.

`turnOn` will return false if:

- Lidar stall.
- Lidar power suppy is unstable.

```
1 // Turn On success and loop
2 while ret and ydlidar.os_isOk() :
```

By `ydlidar.os_isOk()` will install a SIGINT handler which provides Ctrl-C handling which will cause `ydlidar.os_isOk()` to return false if that happens.

`ydlidar.os_isOk()` will return false if:

- a SIGINT is received (Ctrl-C)
- ydlidar.os_shutdown() has been called by another part of the application.

Once `ydlidar.os_isOk()` returns false, Loop exit.

```
1 r = laser.doProcessSimple(scan);
2 if r:
3   print("Scan received[",scan.stamp,"]:",scan.points.size(),"ranges is [",1.0/scan.config.scan_time,"]Hz");
4 else :
5   print("Failed to get Lidar Data.")
```

Get the LiDAR Scan Data.

```
1 // Stop the device scanning thread and disable motor.
2 laser.turnOff();
```

Stop the device scanning thread and disable motor.

```
1 // Uninitialize the SDK and Disconnect the LiDAR.
2 laser.disconnecting();
```

Uninitialize the SDK and Disconnect the LiDAR.

Now that you have written a simple lidar tutorial, let's examine the simple lidar tutorial.

Chapter 25

YDLIDAR SDK Tutorials

Non-Beginners: If you're already familiar enough with YDLIDAR SDK, you can go through more in-depth SDK tutorial here. However, going over all basic Beginner Level tutorials is still recommended for all users to get exposed to new features.

If you are new to Linux/windows: You may find it helpful to first do a quick tutorial on common command line tools for linux/windows. A good one is here.

Table of Contents

Beginner Level

tutorials/writing_lidar_tutorial_c++.md "Writing a Simple Lidar Tutorial (C++)"

This tutorial covers how to write a lidar tutorial in C++.

tutorials/writing_lidar_tutorial_python.md "Writing a Simple Lidar Tutorial (Python)"

This tutorial covers how to write a lidar tutorial in Python.

tutorials/writing_lidar_tutorial_c.md "Writing a Simple Lidar Tutorial (C)"

This tutorial covers how to write a lidar tutorial in C.

tutorials/examine_the_simple_lidar_tutorial.md "Examining the simple lidar tutorial"

This tutorial examines running the lidar tutorial.

Chapter 26

YDLidar-SDK Communication Protocol

Package Format

The response content is the point cloud data scanned by the system. According to the following data format, the data is sent to the external device in hexadecimal to the serial port. No Intensity Byte Offset:

Intensity Byte Offset:

Scan data format output by LiDAR:

Content	Name	Description
PH(2B)	Packet header	2 Byte in length, Fixed at 0x55AA, low is front, high in back.
CT(1B)	Package type	Indicates the current packet type. (0x00 = CT & 0x01): Normal Point cloud packet. (0x01 = CT & 0x01): Zero packet.
LSN(1B)	Sample Data Number	Indicates the number of sampling points contained in the current packet. There is only once zero point of data in thre zero packet. the value is 1.
FSA(2B)	Starting angle	The angle data corresponding to the first sample point in the smapled data.
LSA(2B)	End angle	The angle data corresponding to the last sample point in the sampled data.
CS(2B)	Check code	The check code of the current data packet uses a two-byte exclusive OR to check the current data packet.
Si(2B/3B)	Sampling data	The system test sampling data is the distance data of the sampling point. Note: If the LiDAR has intensity, Si is 3 Byte. otherwise is 2 Byte. Si(3B)–>I(1B)(D(2B)): first Byte is Inentsity, The last two bytes are the Distance.

Zero resolution

Start data packet: (CT & 0x01) = 0x01, LSN = 1, Si = 1. scan frequency: When it was a zero packet, The Lidar Scan frequency: SF = (CT >> 1) / 10.f; The Calculated frequency is the Lidar real-time frequency of the previous frame. If SF is non-zero, the protocol has real-time frequency. For the analysis of the specific values of distance and angle, see the analysis of distance and angle.

Distance analysis:

- Distance solution formula:

  - Triangle LiDAR: “‘ Distance(i) = Si / 4; “‘
  - TOF LiDAR: “‘ Distance(i) = Si; “‘

Si is sampling data. Sampling data is set to E5 6F. Since the system is in the little-endian mode, the sampling point S = 0x6FE5, and it is substituted into the distance solution formula, which yields

- Triangle LiDAR: “‘ Distance = 7161.25mm “‘
- TOF LiDAR: “‘ Distance = 28645mm “‘

Intensity analysis:

Si(3B) split into three bytes : S(0) S(1) S(2)

- Inensity solution formula:
  - Triangle LiDAR: “‘ Intensity(i) = uint16_t((S(1) & 0x03)$<<$ 8 | S(0)); Distance(i) = uint16_t(S(2) $<<$ 8 | S(1)) $>>$ 2; “‘

Si is sampling data. Sampling data is set to 1F E5 6F. Since the system is in the little-endian mode, the

- Triangle LiDAR: “‘ Intensity = uint16_t((0xE5 & 0x03)$<<$ 8 | 0x1F) = 287; Distance = uint16_t(0x6F $<<$ 8 | 0xE5) $>>$ 2 = 7161mm; “‘

Angle analysis:

First level analysis:

Starting angle solution formula:$$Angle_{FSA}={Rshiftbit(FSA, 1)}{64}$$ End angle solution formula:$$Angle_{LSA}={Rshiftbit(LSA, 1)}{64}$$ Intermediate angle solution formula: $$Angle_{i}={diff(Angle)}{LSN - 1}∗i + Angle_{FSA} (0,1„LSN-1)$$ $Angle_{0} : Angle_{FSA}$; $Angle_{LSN-1} : Angle_{LSA}$;

`Rshiftbit(data,1)` means shifting the data to the right by one bit.diff Angle means the clockwise angle difference from the starting angle (uncorrected value) to the ending angle (uncorrected value),and LSN represents the number of packet samples in this frame.

`diff(Angle): (Angle(LSA) - Angle(FSA))` If less than zero, `diff(Angle) = (Angle(LSA)- Angle(FSA)) + 360`, otherwise `diff(Angle) = (Angle(LSA)- Angle(FSA))`

code

```
1 double Angle_FSA = (FSA >> 1) / 64;
2 double Angle_LSA = (LSA >> 1) / 64;
3 double angle_diff = Angle_FSA - Angle_LSA;
4 if(angle_diff < 0) {
5     angle_diff += 360;
6 }
7 double Angle[LSN];
8 for(int i = 0; i < LSN; i++) {
9     Angle[i] = i* angle_diff / (LSN - 1) + Angle_FSA;
10 }
```

Second-level analysis:

Triangle Lidar only has current Second-level analysis, TOF Lidar does not need.

Angle correction formula: $Angle_{i} = Angle_{i} + AngCorrect_{i}$; ($1,2„LSN$) AngCorrect is the angle correction value, and its calculation formula is as follows, $tand^{-1}$ is an inverse trigonometric function. and the return angle value is:

if($Distance_{i}$ == 0) { $AngCorrect_{i}$ = 0; } else { $AngCorrect_{i} = atan(21.8 ∗ {155.3 - Distance_{i}}{155.↩
3∗Distance_{i}})$ }

In the data packet, the 4th to 8th bytes are `28 E5 6F BD 79`, so `LSN = 0x28 = 40(dec)`, `FSA = 0x6↩
FE5`, `LSA = 0x79BD`, and bring in the first-level solution formula, and get: $Angle_{FSA} = 223.78^{°}$ $Angle↩
_{LSA} = 243.47^{°}$ $diff(Angle) = Angle_{LSA} - Angle_{FSA} = 243.47^{°} - 223.78^{°} = 19.69^{°}$ $Angle_{i}
= {19.69^{°}}{39}∗(i -1) + 223.78^{°}$ ($1,2„LSN$) Assume that in the frame data: $Distance_{1} = 1000$ $↩
Distance_{LSN} = 8000$ bring in the second-level solution formula, you get: $AngCorrect_{1} = -6.7622^{°}$ $↩
AngCorrect_{LSN} = -7.8374^{°}$ $Angle_{FSA} = Angle_{1} + AngCorrect_{1} = 217.0178^{°}$ $Angle_{LSA} =
Angle_{LSA} + AngCorrect_{LSA} = 235.6326^{°}$ Similarly, $Angle_{i}(2,3, ,LSN-1)$, can be obtained sequentially.

```
1 for(int i = 0; i < LSN; i++) {
2     if(Distance[i] > 0) {
3         double AngCorrect = atan(21.8 * (155.3 - Distance[i]) / (155.3 * Distance[i]));
4         Angle[i] += AngCorrect;
5     }
6     if(Angle[i] >= 360) {
7         Angle[i] -= 360;
8     }
9 }
```

Note:

- TOF LiDAR does not neeed second-level analysis.

Check code parsing:

The check code uses a two-byte exclusive OR to verify the current data packet. The check code itself does not participate in XOR operations, and the XOR order is not strictly in byte order. The XOR sequence is as shown in the figure. Therefore, the check code solution formula is:

$$ CS = XOR {i=1}^{n}(C^i)$$

CS Sequence

PH	**C(1)**
FSA	**C(2)**
S1	**C(3)**
S2	**C(4)**
...	**..**
Sn	**C(n-2)**
[CT | LSN]	**C(n-1)**
LSA	**C(n)**

- Note: XOR(end) indicates the XOR of the element from subscript 1 to end. However, XOR satisfies the

exchange law, and the actual solution may not need to follow the XOR sequence.

Code

No intensity Si(2B):

```
1 uint16_t checksumcal = PH;
2 checksumcal ^= FSA;
3 for(int i = 0; i < 2 * LSN; i = i +2 ) {
4     checksumcal ^= uint16_t(data[i+1] <<8 | data[i]);
5 }
6 checksumcal ^= uint16_t(LSN << 8 | CT);
7 checksumcal ^= LSA;
8
9 ## uint16_t : unsigned short
```

Intensity Si(3B):

```
1 uint16_t checksumcal = PH;
2 checksumcal ^= FSA;
3 for(int i = 0; i < 3 * LSN; i = i + 3) {
4     checksumcal ^= data[i];
5     checksumcal ^= uint16_t(data[i+2] <<8 | data[i + 1]);
6 }
7 checksumcal ^= uint16_t(LSN << 8 | CT);
8 checksumcal ^= LSA;
9
10 ## uint16_t : unsigned short
```

example

No Intensity:

Name	Size(Byte)	Value	Contant	Buffer
PH	2	0x55AA	Header	0xAA
				0x55
CT	1	0x01	Type	0x01
LSN	1	0x01	Number	0x01
FSA	2	0xAE53	Starting Angle	0x53
				0xAE
LSA	2	0xAE53	End Andgle	0x53
				0xAE
CS	2	0x54AB	Check code	0xAB
				0x54
S0	2	0x000	0 index Distance	0x00
				0x00

```
1 uint8_t Buffer[12];
2 Buffer[0] = 0xAA;
3 Buffer[1] = 0x55;
4 Buffer[2] = 0x01;
5 Buffer[3] = 0x01;
6 Buffer[4] = 0x53;
7 Buffer[5] = 0xAE;
8 Buffer[6] = 0x53;
9 Buffer[7] = 0xAE;
10 Buffer[8] = 0xAB;
11 Buffer[9] = 0x54;
12 Buffer[10] = 0x00;
13 Buffer[11] = 0x00;
14
```

```
15 uint16_t check_code = 0x55AA;
16 uint8_t CT = Buffer[2] & 0x01;
17 uin8_t LSN = Buffer[3];
18 uint16_t FSA = uint16_t(Buffer[5] << 8 | Buffer[4]);
19 check_code ^= FSA;
20 uint16_t LSA = uint16_t(Buffer[7] << 8 | Buffer[6]);
21 uint16_t CS = uint16_t(Buffer[9] << 8 | Buffer[8]);
22
23 double Distance[LSN];
24 for(int i = 0; i < 2 * LSN; i = i + 2) {
25     uint16_t data = uint16_t(Buffer[10 + i + 1] << 8 | Buffer[10 + i]);
26     check_code ^= data;
27     Distance[i / 2 ] = data / 4;
28 }
29 check_code ^= uint16_t(LSN << 8 | CT);
30 check_code ^= LSA;
31
32 double Angle[LSN];
33
34 if(check_code == CS) {
35     double Angle_FSA = (FSA >> 1) / 64;
36     double Angle_LSA = (LSA >> 1) / 64;
37     double Angle_Diff = (Angle_LSA - Angle_FSA);
38     if(Angle_Diff < 0) {
39         Angle_Diff = Angle_Diff + 360;
40     }
41     for(int i = 0; i < LSN; i++) {
42         Angle[i] = i * Angle_Diff/ (LSN- 1) + Angle_FSA;
43         if(Distance[i] > 0) {
44             double AngCorrect = atan(21.8 * (155.3 - Distance[i]) / (155.3 * Distance[i]));
45             Angle[i] = Angle[i] + AngCorrect;
46         }
47         if(Angle[i] >= 360) {
48             Angle[i] -= 360;
49         }
50     }
51 }
```

Intensity:

Name	Size(Byte)	Value	Contant	Buffer
PH	2	0x55AA	Header	0xAA
				0x55
CT	1	0x01	Type	0x01
LSN	1	0x01	Number	0x01
FSA	2	0xAE53	Starting Angle	0x53
				0xAE
LSA	2	0xAE53	End Andgle	0x53
				0xAE
CS	2	0x54AB	Check code	0xAB
				0x54
I0	1	0x00	0 index Intensity	0x00
S0	2	0x000	0 index Distance	0x00
				0x00

```
1 uint8_t Buffer[13];
2 Buffer[0] = 0xAA;
3 Buffer[1] = 0x55;
4 Buffer[2] = 0x01;
5 Buffer[3] = 0x01;
6 Buffer[4] = 0x53;
7 Buffer[5] = 0xAE;
8 Buffer[6] = 0x53;
9 Buffer[7] = 0xAE;
10 Buffer[8] = 0xAB;
11 Buffer[9] = 0x54;
12 Buffer[10] = 0x00;
13 Buffer[11] = 0x00;
14 Buffer[12] = 0x00;
15
16 uint16_t check_code = 0x55AA;
17 uint8_t CT = Buffer[2] & 0x01;
18 uin8_t LSN = Buffer[3];
```

```
19 uint16_t FSA = uint16_t(Buffer[5] << 8 | Buffer[4]);
20 check_code ^= FSA;
21 uint16_t LSA = uint16_t(Buffer[7] << 8 | Buffer[6]);
22 uint16_t CS = uint16_t(Buffer[9] << 8 | Buffer[8]);
23
24 double Distance[LSN];
25 uin16_t Itensity[LSN];
26 for(int i = 0; i < 3 * LSN; i = i + 3) {
27     check_code ^= Buffer[10 + i];
28     uint16_t data = uint16_t(Buffer[10 + i + 2] << 8 | Buffer[10 + i + 1]);
29     check_code ^= data;
30     Itensity[i / 3] = uint16_t((Buffer[10 + i + 1] & 0x03) <<8 |  Buffer[10 + i]);
31     Distance[i / 3] = data >> 2;
32 }
33 check_code ^= uint16_t(LSN << 8 | CT);
34 check_code ^= LSA;
35
36 double Angle[LSN];
37
38 if(check_code == CS) {
39     double Angle_FSA = (FSA >> 1) / 64;
40     double Angle_LSA = (LSA >> 1) / 64;
41     double Angle_Diff = (Angle_LSA - Angle_FSA);
42     if(Angle_Diff < 0) {
43         Angle_Diff = Angle_Diff + 360;
44     }
45     for(int i = 0; i < LSN; i++) {
46         Angle[i] = i * Angle_Diff/ (LSN- 1) + Angle_FSA;
47         if(Distance[i] > 0) {
48             double AngCorrect = atan(21.8 * (155.3 - Distance[i]) / (155.3 * Distance[i]));
49             Angle[i] = Angle[i] + AngCorrect;
50         }
51     }
52 }
```

For more details and usage examples, Refer to https://github.com/YDLIDAR/ydlidar_tutorials/blob/master/↩ CommunicationProtocol/README.md "Communication Protocol"

Chapter 27

YDLIDAR SDK API for Developers

set lidar properties

This document provides an extensive technical deep dive into how to create, manipulate and use YDLIDAR SDK's API.

Table of Contents

Samples

The first part of demonstrating YDLIDAR SDK API is to understand the ydlidar_test/tof_test/etlidar_Test example. Following are one optinal concepts: `ydlidar::os_init()` (basic unit) of the example.

Create A System State

In the YDLIDAR SDK, the `ydlidar::os_init()` is optinal unit, If you need to accept `Ctrl + C` or other system abnormal signals. you can use it to create a system state, and check whether the system is normal by `ydlidar::os_isOk()`. The system signal creation interface is as follows:

```
ydlidar::os_init();
```

- when `ydlidar::os_init()` has called, the system is in an initialized state, able to accept `Ctrl + C` and `ydlidar::os_shutdown()` signals.

Code Example

Triangle LiDAR (../samples/ydlidar_test.cpp)

```
#include "CYdLidar.h"
#include <iostream>
#include <string>
#include <algorithm>
#include <cctype>
using namespace std;
using namespace ydlidar;
#if defined(_MSC_VER)
#pragma comment(lib, "ydlidar_sdk.lib")
#endif

int main(int argc, char *argv[]) {
  // init system signal
  ydlidar::os_init();

  CYdLidar laser;
  std::string port = "/dev/ydlidar
  laser.setlidaropt(LidarPropSerialPort, port.c_str(), port.size());
  std::string ignore_array;
  ignore_array.clear();
  laser.setlidaropt(LidarPropIgnoreArray, ignore_array.c_str(),
                    ignore_array.size());

  int optval = 230400;
  laser.setlidaropt(LidarPropSerialBaudrate, &optval, sizeof(int));
  int optval = TYPE_TRIANGLE;
  laser.setlidaropt(LidarPropLidarType, &optval, sizeof(int));
  optval = YDLIDAR_TYPE_SERIAL;
  laser.setlidaropt(LidarPropDeviceType, &optval, sizeof(int));
  optval = 9;
  laser.setlidaropt(LidarPropSampleRate, &optval, sizeof(int));
  optval = 4;
  laser.setlidaropt(LidarPropAbnormalCheckCount, &optval, sizeof(int));

  bool b_optvalue = false;
  laser.setlidaropt(LidarPropFixedResolution, &b_optvalue, sizeof(bool));
  laser.setlidaropt(LidarPropReversion, &b_optvalue, sizeof(bool));
  laser.setlidaropt(LidarPropInverted, &b_optvalue, sizeof(bool));
  b_optvalue = true;
  laser.setlidaropt(LidarPropAutoReconnect, &b_optvalue, sizeof(bool));
  b_optvalue = false;
  laser.setlidaropt(LidarPropSingleChannel, &b_optvalue, sizeof(bool));
  b_optvalue = false;
  laser.setlidaropt(LidarPropIntenstiy, &b_optvalue, sizeof(bool));
  b_optvalue = false;
  laser.setlidaropt(LidarPropSupportMotorDtrCtrl, &b_optvalue, sizeof(bool));

  float f_optvalue = 180.0f;
  laser.setlidaropt(LidarPropMaxAngle, &f_optvalue, sizeof(float));
  f_optvalue = -180.0f;
  laser.setlidaropt(LidarPropMinAngle, &f_optvalue, sizeof(float));
  f_optvalue = 16.f;
  laser.setlidaropt(LidarPropMaxRange, &f_optvalue, sizeof(float));
  f_optvalue = 0.1f;
  laser.setlidaropt(LidarPropMinRange, &f_optvalue, sizeof(float));
  f_optvalue = 10.f;
  laser.setlidaropt(LidarPropScanFrequency, &f_optvalue, sizeof(float));

  // initialize SDK and LiDAR
```

```
  bool ret = laser.initialize();
  if (ret) {//success
    //Start the device scanning routine which runs on a separate thread and enable motor.
    ret = laser.turnOn();
  } else {
    fprintf(stderr, "%s\n", laser.DescribeError());
    fflush(stderr);
  }

  // Turn On success and loop
  while (ret && ydlidar::os_isOk()) {
    LaserScan scan;
    if (laser.doProcessSimple(scan)) {
      fprintf(stdout, "Scan received[%llu]: %u ranges is [%f]Hz\n",
              scan.stamp,
              (unsigned int)scan.points.size(), 1.0 / scan.config.scan_time);
      fflush(stdout);
    } else {
      fprintf(stderr, "Failed to get Lidar Data\n");
      fflush(stderr);
    }
  }
  // Stop the device scanning thread and disable motor.
  laser.turnOff();
  // Uninitialize the SDK and Disconnect the LiDAR.
  laser.disconnecting();
  return 0;
}
```

TOF LiDAR (../samples/tof_test.cpp)

```
#include "CYdLidar.h"
#include <iostream>
#include <string>
#include <algorithm>
#include <cctype>
using namespace std;
using namespace ydlidar;
#if defined(_MSC_VER)
#pragma comment(lib, "ydlidar_sdk.lib")
#endif

int main(int argc, char *argv[]) {
  // init system signal
  ydlidar::os_init();

  CYdLidar laser;
  std::string port = "/dev/ydlidar
  laser.setlidaropt(LidarPropSerialPort, port.c_str(), port.size());
  std::string ignore_array;
  ignore_array.clear();
  laser.setlidaropt(LidarPropIgnoreArray, ignore_array.c_str(),
                    ignore_array.size());

  int optval = 512000;
  laser.setlidaropt(LidarPropSerialBaudrate, &optval, sizeof(int));
  int optval = TYPE_TOF;
  laser.setlidaropt(LidarPropLidarType, &optval, sizeof(int));
  optval = YDLIDAR_TYPE_SERIAL;
  laser.setlidaropt(LidarPropDeviceType, &optval, sizeof(int));
  optval = 20;
  laser.setlidaropt(LidarPropSampleRate, &optval, sizeof(int));
  optval = 4;
  laser.setlidaropt(LidarPropAbnormalCheckCount, &optval, sizeof(int));

  bool b_optvalue = false;
  laser.setlidaropt(LidarPropFixedResolution, &b_optvalue, sizeof(bool));
  laser.setlidaropt(LidarPropReversion, &b_optvalue, sizeof(bool));
  laser.setlidaropt(LidarPropInverted, &b_optvalue, sizeof(bool));
  b_optvalue = true;
  laser.setlidaropt(LidarPropAutoReconnect, &b_optvalue, sizeof(bool));
  b_optvalue = false;
  laser.setlidaropt(LidarPropSingleChannel, &b_optvalue, sizeof(bool));
  b_optvalue = false;
  laser.setlidaropt(LidarPropIntenstiy, &b_optvalue, sizeof(bool));
  b_optvalue = false;
  laser.setlidaropt(LidarPropSupportMotorDtrCtrl, &b_optvalue, sizeof(bool));

  float f_optvalue = 180.0f;
  laser.setlidaropt(LidarPropMaxAngle, &f_optvalue, sizeof(float));
  f_optvalue = -180.0f;
  laser.setlidaropt(LidarPropMinAngle, &f_optvalue, sizeof(float));
  f_optvalue = 64.f;
  laser.setlidaropt(LidarPropMaxRange, &f_optvalue, sizeof(float));
```

```
  f_optvalue = 0.05f;
  laser.setlidaropt(LidarPropMinRange, &f_optvalue, sizeof(float));
  f_optvalue = 10.f;
  laser.setlidaropt(LidarPropScanFrequency, &f_optvalue, sizeof(float));

  // initialize SDK and LiDAR
  bool ret = laser.initialize();
  if (ret) {//success
    //Start the device scanning routine which runs on a separate thread and enable motor.
    ret = laser.turnOn();
  } else {
    fprintf(stderr, "%s\n", laser.DescribeError());
    fflush(stderr);
  }

  // Turn On success and loop
  while (ret && ydlidar::os_isOk()) {
    LaserScan scan;
    if (laser.doProcessSimple(scan)) {
      fprintf(stdout, "Scan received[%llu]: %u ranges is [%f]Hz\n",
              scan.stamp,
              (unsigned int)scan.points.size(), 1.0 / scan.config.scan_time);
      fflush(stdout);
    } else {
      fprintf(stderr, "Failed to get Lidar Data\n");
      fflush(stderr);
    }
  }
  // Stop the device scanning thread and disable motor.
  laser.turnOff();
  // Uninitialize the SDK and Disconnect the LiDAR.
  laser.disconnecting();
  return 0;
}
```

CMake BUILD file(../samples/CMakeLists.txt)

```
1 cmake_minimum_required(VERSION 2.8)
2 PROJECT(ydlidar_test)
3 set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++11")
4 add_definitions(-std=c++11) # Use C++11
5
6 #Include directories
7 INCLUDE_DIRECTORIES(
8      ${CMAKE_SOURCE_DIR}
9      ${CMAKE_SOURCE_DIR}/../
10      ${CMAKE_CURRENT_BINARY_DIR}
11 )
12
13 SET(EXECUTABLE_OUTPUT_PATH ${CMAKE_BINARY_DIR})
14
15 set(curdir ${CMAKE_CURRENT_SOURCE_DIR})
16 FILE(GLOB APP_LIST "${curdir}/*.cpp")
17 foreach(child ${APP_LIST})
18   string(REPLACE "${curdir}/" "" app_main ${child})
19   string(REPLACE ".cpp" "" APP_NAME ${app_main})
20   ADD_EXECUTABLE(${APP_NAME} ${app_main})
21   TARGET_LINK_LIBRARIES(${APP_NAME} ydlidar_sdk)
22 endforeach()
```

Build and Run

- Build: cd build & cmake ../ & make
- Run ydlidar_test or tof_test in terminals:
  - ./ydlidar_test
  - ./tof_test
- Examine the results: you should see message printing out in terminals.

Python

```
1 import os
2 import ydlidar
3 import time
4
5 if __name__ == "__main__":
6     ydlidar.os_init();
7     laser = ydlidar.CYdLidar();
8     laser.setlidaropt(ydlidar.LidarPropSerialPort, "/dev/ydlidar");
9     laser.setlidaropt(ydlidar.LidarPropSerialBaudrate, 230400);
10     laser.setlidaropt(ydlidar.LidarPropLidarType, ydlidar.TYPE_TRIANGLE);
11     laser.setlidaropt(ydlidar.LidarPropDeviceType, ydlidar.YDLIDAR_TYPE_SERIAL);
12     laser.setlidaropt(ydlidar.LidarPropScanFrequency, 10.0);
13     laser.setlidaropt(ydlidar.LidarPropSampleRate, 9);
14     laser.setlidaropt(ydlidar.LidarPropSingleChannel, False);
15
16     ret = laser.initialize();
17     if ret:
18         ret = laser.turnOn();
19         scan = ydlidar.LaserScan();
20         while ret and ydlidar.os_isOk() :
21             r = laser.doProcessSimple(scan);
22             if r:
23                 print("Scan received[",scan.stamp,"]:",scan.points.size(),"ranges is
      [",1.0/scan.config.scan_time,"]Hz");
24             else :
25                 print("Failed to get Lidar Data")
26             time.sleep(0.05);
27         laser.turnOff();
28     laser.disconnecting();
```

ThridParty Project Call Library OR Source

Introduction

There are two ways to integrate YDLIDAR SDK into your project.

- Calling a compliled library.
  - Download YDLIDAR SDK
  - Build and Install
  - Add YDLIDAR SDK to the CMakeLists file of your project
- Add source code to the Project.
  - Copy YDLIDAR SDK to your project
  - Add YDLIDAR SDK project to the CMakeLists file of your project

Calling a compliled library

The implementation of the demo mainly includes the following steps.

Download YDLIDAR SDK

```
1 $git clone https://github.com/YDLIDAR/YDLidar-SDK
```

Build and Install

```
1 $cd YDLidar-SDK/build
2 $cmake ../
3 $make
4 $sudo make install
```

Note:

- Default generate static library.
- If you want to generate YDLidar-SDK dynamic library, Add the following options when compiling:

```
1 $cmake -DBUILD_SHARED_LIBS=ON ../ && make
```

Add YDLIDAR SDK to the CMakeLists file of your project

```
1 ############## YDLIDAR SDK START#######################################
2 #find ydlidar_sdk package
3 find_package(ydlidar_sdk)
4 #Include directories
5 INCLUDE_DIRECTORIES(
6   ${YDLIDAR_SDK_INCLUDE_DIRS}
7 )
8 #link library directories
9 link_directories(${YDLIDAR_SDK_LIBRARY_DIRS})
10
11 #Link your project to ydlidar_sdk library.
12 target_link_libraries(${PROJECT_NAME} ${YDLIDAR_SDK_LIBRARIES})
13
14 ############## YDLIDAR SDK END#######################################
```

Detailed call Demo, see here

Add source code to the Project

The implementation of the demo mainly includes the following steps.

Copy YDLIDAR SDK to your project

```
1 $cd 'your project'
2 $git clone https://github.com/YDLIDAR/YDLidar-SDK
```

Note: The same can be downloaded and placed in your project directory.

Add YDLIDAR SDK project to the CMakeLists file of your project

```
1 ############## YDLIDAR SDK START#######################################
2 #Include directories
3 INCLUDE_DIRECTORIES(
4   ${CMAKE_SOURCE_DIR}
5   ${CMAKE_SOURCE_DIR}/YDLidar-SDK
6   ${CMAKE_SOURCE_DIR}/YDLidar-SDK/src
7   ${CMAKE_CURRENT_BINARY_DIR}/YDLidar-SDK
8 )
9 #Add YDLIDAR SDK sub project
10 add_subdirectory(YDLidar-SDK)
11
12 #Link your project to ydlidar_sdk library.
13 target_link_libraries(${PROJECT_NAME} ydlidar_sdk)
14
15 ############## YDLIDAR SDK END#######################################
```

Note:

- If you do not want to generate samples in YDLidar-SDK, Add the following options when compiling:

  ```
  1 $cmake -DBUILD_EXAMPLES=OFF ../ && make
  ```

- If you want to generate YDLidar-SDK dynamic library, Add the following options when compiling:

  ```
  1 $cmake -DBUILD_SHARED_LIBS=ON ../ && make
  ```

- If you want to meet both of the above conditions, the operation is as follows:

  ```
  1 $cmake -DBUILD_EXAMPLES=OFF -DBUILD_SHARED_LIBS=ON ../ && make
  ```

  Detailed call Demo, see here

Development Flow

FlowChart

Sequence

API Directory

CYDLidar API

For additional information and examples, refer to CYDLidar

API List

C++ API

```
1 {C++}
2 /**
3 ** @param optname        option name
4 * \xrefitem todo 1.@note set string property example
5 * @code
6 * CYdLidar laser;
7 * std::string lidar_port = "/dev/ydlidar";
8 * laser.setlidaropt(LidarPropSerialPort,lidar_port.c_str(), lidar_port.size());
9 *
```

- **Todo** int properties

  Note

  set int property example

- ```
  1 * CYdLidar laser;
  2 * int lidar_baudrate = 230400;
  3 * laser.setlidaropt(LidarPropSerialPort,&lidar_baudrate, sizeof(int));
  4 *
  ```

- **Todo** bool properties

  Note

  set bool property example

- ```
  1 * CYdLidar laser;
  2 * bool lidar_fixedresolution = true;
  3 * laser.setlidaropt(LidarPropSerialPort,&lidar_fixedresolution, sizeof(bool));
  4 *
  ```

- **Todo** float properties

  Note

  set float property example

- ```
  1 * CYdLidar laser;
  2 * float lidar_maxrange = 16.0f;
  3 * laser.setlidaropt(LidarPropSerialPort,&lidar_maxrange, sizeof(float));
  4 *
  ```

Parameters

optval	option value
  ```

-
- - std::string(or char*)
- - int
- - bool
- - float

Parameters

optlen	option length

-
- - data type size

  Returns
- true if the Property is set successfully, otherwise false.

  See also
- LidarProperty */ bool setlidaropt(int optname, const void *optval, int optlen);

  /**
- get lidar property

Parameters

optname	option name

-
- **Todo** string properties

  Note

  get string property example
- ```
  1 * CYdLidar laser;
  2 * char lidar_port[30];
  3 * laser.getlidaropt(LidarPropSerialPort,lidar_port, sizeof(lidar_port));
  4 *
  ```
- **Todo** int properties

  Note

  get int property example
- ```
  1 * CYdLidar laser;
  2 * int lidar_baudrate;
  3 * laser.getlidaropt(LidarPropSerialPort,&lidar_baudrate, sizeof(int));
  4 *
  ```
- **Todo** bool properties

  Note

  get bool property example
- ```
  1 * CYdLidar laser;
  2 * bool lidar_fixedresolution;
  3 * laser.getlidaropt(LidarPropSerialPort,&lidar_fixedresolution, sizeof(bool));
  4 *
  ```
- **Todo** float properties
  ```

Note

set float property example

```
1 * CYdLidar laser;
2 * float lidar_maxrange;
3 * laser.getlidaropt(LidarPropSerialPort,&lidar_maxrange, sizeof(float));
4 *
```

Parameters

optval	option value

-
- - std::string(or char∗)
- - int
- - bool
- - float

Parameters

optlen	option length

-
- - data type size

  Returns
- true if the Property is get successfully, otherwise false.

  See also
- LidarProperty ∗/ bool getlidaropt(int optname, void ∗optval, int optlen); /∗∗
  - Initialize the SDK and LiDAR.
  -

  Returns

  true if successfully initialized, otherwise false. ∗/ bool initialize();

  /∗∗
- Return LiDAR's version information in a numeric form.

Parameters

version	Pointer to a version structure for returning the version information. ∗/ void GetLidarVersion(LidarVersion &version);

- /∗∗
  - Start the device scanning routine which runs on a separate thread and enable motor.
    Returns
  - true if successfully started, otherwise false. ∗/ bool turnOn(); /∗∗
  - Get the LiDAR Scan Data. turnOn is successful before doProcessSimple scan data.

Parameters

out	*outscan*	LiDAR Scan Data

–

Parameters

out	*hardwareError*	hardware error status

–

Returns

– true if successfully started, otherwise false. ∗/ bool doProcessSimple(LaserScan &outscan, bool &hardwareError); /∗∗

– Stop the device scanning thread and disable motor.

Returns

– true if successfully Stoped, otherwise false. ∗/ bool turnOff(); /∗∗

– Uninitialize the SDK and Disconnect the LiDAR. ∗/ void disconnecting();

/∗∗

- Get the last error information of a (socket or serial)

  Returns

- a human-readable description of the given error information
- or the last error information of a (socket or serial) ∗/ const char ∗DescribeError() const; #### C API

```
YDLidar *lidarCreate(void);

void lidarDestroy(YDLidar **lidar);
```

- **Todo** int properties

  Note

  set int property example

```
1 * CYdLidar laser;
2 * int lidar_baudrate = 230400;
3 * laser.setlidaropt(LidarPropSerialPort,&lidar_baudrate, sizeof(int));
4 *
```

- **Todo** bool properties

  Note

  set bool property example

```
1 * CYdLidar laser;
2 * bool lidar_fixedresolution = true;
3 * laser.setlidaropt(LidarPropSerialPort,&lidar_fixedresolution, sizeof(bool));
4 *
```

- **Todo** float properties

  Note

  set float property example

```
1 * CYdLidar laser;
2 * float lidar_maxrange = 16.0f;
3 * laser.setlidaropt(LidarPropSerialPort,&lidar_maxrange, sizeof(float));
4 *
```

Parameters

optval	option value

-
- - std::string(or char∗)
- - int
- - bool
- - float

Parameters

optlen	option length

-
- - data type size

  Returns
- true if the Property is set successfully, otherwise false.

  See also
- LidarProperty ∗/ bool setlidaropt(YDLidar ∗lidar, int optname, const void ∗optval, int optlen);

  /∗∗
- get lidar property

Parameters

lidar	a lidar instance

-

Parameters

optname	option name

-
- **Todo** string properties

  Note

  get string property example
- ```
  1 * CYdLidar laser;
  2 * char lidar_port[30];
  3 * laser.getlidaropt(LidarPropSerialPort,lidar_port, sizeof(lidar_port));
  4 *
  ```
- **Todo** int properties
  ```

Note

get int property example

```
1 * CYdLidar laser;
2 * int lidar_baudrate;
3 * laser.getlidaropt(LidarPropSerialPort,&lidar_baudrate, sizeof(int));
4 *
```

- **Todo** bool properties

Note

get bool property example

```
1 * CYdLidar laser;
2 * bool lidar_fixedresolution;
3 * laser.getlidaropt(LidarPropSerialPort,&lidar_fixedresolution, sizeof(bool));
4 *
```

- **Todo** float properties

Note

set float property example

```
1 * CYdLidar laser;
2 * float lidar_maxrange;
3 * laser.getlidaropt(LidarPropSerialPort,&lidar_maxrange, sizeof(float));
4 *
```

Parameters

optval	option value

-
- - std::string(or char*)
- - int
- - bool
- - float

Parameters

optlen	option length

-
- - data type size

Returns

- true if the Property is get successfully, otherwise false.

See also

- LidarProperty */ bool getlidaropt(YDLidar *lidar, int optname, void *optval, int optlen);

/**

Return SDK's version information in a numeric form.

Parameters

version	Pointer to a version for returning the version information. */ void GetSdkVersion(char *version);

- /**
  - Initialize the SDK.
    Returns
  - true if successfully initialized, otherwise false. */ bool initialize(YDLidar *lidar);
    /**
- Return LiDAR's version information in a numeric form.

Parameters

version	Pointer to a version structure for returning the version information. */ void GetLidarVersion(YDLidar *lidar, LidarVersion *version);

- /**
  - Start the device scanning routine which runs on a separate thread.
    Returns
  - true if successfully started, otherwise false. */ bool turnOn(YDLidar *lidar);
    /**
  - Get the LiDAR Scan Data. turnOn is successful before doProcessSimple scan data.

Parameters

in	*lidar*	LiDAR instance

  -

Parameters

out	*outscan*	LiDAR Scan Data

  -
    Returns
  - true if successfully started, otherwise false. */ bool doProcessSimple(YDLidar *lidar, LaserFan *outscan); /**
  - Stop the device scanning thread and disable motor.
    Returns
  - true if successfully Stoped, otherwise false. */ bool turnOff(YDLidar *lidar); /**
  - Uninitialize the SDK and Disconnect the LiDAR. */ void disconnecting(YDLidar *lidar);

/**

- Get the last error information of a (socket or serial)

Returns

- a human-readable description of the given error information
- or the last error information of a (socket or serial) */ const char *DescribeError(YDLidar *lidar);

/**

- initialize system signals */ void os_init(); /**
- isOk

  Returns
- true if successfully initialize, otherwise false. */ bool os_isOk(); /**
- os_shutdown */ void os_shutdown();

/**

- get lidar serial port

Parameters

ports	serial port lists

-

  Returns
- valid port number */ int lidarPortList(LidarPort *ports);

YDlidarDriver API

For additional information and examples, refer to `YDlidarDriver`

API List

```
virtual result_t connect(const char *port_path, uint32_t baudrate);

virtual const char *DescribeError(bool isTCP = true);

virtual void disconnect();

virtual std::string getSDKVersion();

static std::map<std::string, std::string> lidarPortList();

virtual bool isscanning() const;

virtual bool isconnected() const;

virtual void setIntensities(const bool &isintensities);

virtual void setAutoReconnect(const bool &enable);

virtual result_t getHealth(device_health &health,
                           uint32_t timeout = DEFAULT_TIMEOUT);

virtual result_t getDeviceInfo(device_info &info,
                               uint32_t timeout = DEFAULT_TIMEOUT);
```

```
virtual result_t startScan(bool force = false,
                           uint32_t timeout = DEFAULT_TIMEOUT) ;

virtual result_t stop();


virtual result_t grabScanData(node_info *nodebuffer, size_t &count,
                              uint32_t timeout = DEFAULT_TIMEOUT) ;


result_t ascendScanData(node_info *nodebuffer, size_t count);

result_t reset(uint32_t timeout = DEFAULT_TIMEOUT);

result_t startMotor();

result_t stopMotor();

virtual  result_t getScanFrequency(scan_frequency &
      frequency,
                                   uint32_t timeout = DEFAULT_TIMEOUT);

virtual result_t setScanFrequencyAdd(scan_frequency &
      frequency,
                                     uint32_t timeout = DEFAULT_TIMEOUT);

virtual result_t setScanFrequencyDis(scan_frequency &
      frequency,
                                     uint32_t timeout = DEFAULT_TIMEOUT);

virtual result_t setScanFrequencyAddMic(scan_frequency &
      frequency,
                                        uint32_t timeout = DEFAULT_TIMEOUT);

virtual result_t setScanFrequencyDisMic(scan_frequency &
      frequency,
                                        uint32_t timeout = DEFAULT_TIMEOUT);

virtual result_t getSamplingRate(sampling_rate &rate,
                                 uint32_t timeout = DEFAULT_TIMEOUT);

virtual result_t setSamplingRate(sampling_rate &rate,
                                 uint32_t timeout = DEFAULT_TIMEOUT);

virtual result_t getZeroOffsetAngle(offset_angle &angle,
                                    uint32_t timeout = DEFAULT_TIMEOUT);
```

ETLidarDriver API

For additional information and examples, refer to `ETLidarDriver`

API List

```
virtual result_t connect(const char *port_path, uint32_t baudrate = 8000);


virtual const char *DescribeError(bool isTCP = true);

virtual void disconnect();

virtual std::string getSDKVersion();

virtual bool isscanning() const;

virtual bool isconnected() const;

virtual void setIntensities(const bool &isintensities);

virtual void setAutoReconnect(const bool &enable);

virtual result_t getHealth(device_health &health,
                           uint32_t timeout = DEFAULT_TIMEOUT);

virtual result_t getDeviceInfo(device_info &info,
                               uint32_t timeout = DEFAULT_TIMEOUT);
```

```
virtual result_t startScan(bool force = false,
                           uint32_t timeout = DEFAULT_TIMEOUT) ;

virtual result_t stop();


virtual result_t grabScanData(node_info *nodebuffer, size_t &count,
                              uint32_t timeout = DEFAULT_TIMEOUT) ;

virtual result_t getScanFrequency(scan_frequency &
      frequency,
                                  uint32_t timeout = DEFAULT_TIMEOUT);

virtual result_t setScanFrequencyAdd(scan_frequency &
      frequency,
                                     uint32_t timeout = DEFAULT_TIMEOUT);

virtual result_t setScanFrequencyDis(scan_frequency &
      frequency,
                                     uint32_t timeout = DEFAULT_TIMEOUT);

virtual result_t setScanFrequencyAddMic(scan_frequency &
      frequency,
                                        uint32_t timeout = DEFAULT_TIMEOUT);

virtual result_t setScanFrequencyDisMic(scan_frequency &
      frequency,
                                        uint32_t timeout = DEFAULT_TIMEOUT);

virtual result_t getSamplingRate(sampling_rate &rate,
                                 uint32_t timeout = DEFAULT_TIMEOUT);

virtual result_t setSamplingRate(sampling_rate &rate,
                                 uint32_t timeout = DEFAULT_TIMEOUT);
virtual result_t getZeroOffsetAngle(offset_angle &angle,
                                    uint32_t timeout = DEFAULT_TIMEOUT);


bool getScanCfg(lidarConfig &config, const std::string &ip_address = "");

lidarConfig getFinishedScanCfg();

void updateScanCfg(const lidarConfig &config);
```

Parameter Table

The Table that the user uses to perform parameter related operations:

- Set the parameter related API by table.

For additional information and examples, refer to Parameter

Table List - Models

LIDAR	Model	Baudrate	Sample↩ Rate(K)	Range(m)	Frequency(HZ)	Intenstiy(b	Single↩ Channel	voltage(↩ V)
F4	1	115200	4	0.12∼12	5∼12	false	false	4.8∼5.2
S4	4	115200	4	0.10∼8.0	5∼12 (PWM)	false	false	4.8∼5.2
S4B	4/11	153600	4	0.10∼8.0	5∼12(P↩ WM)	true(8)	false	4.8∼5.2
S2	4/12	115200	3	0.10∼8.0	4∼8(P↩ WM)	false	true	4.8∼5.2
G4	5	230400	9/8/4	0.↩ 28/0.26/0.↩ 1∼16	5∼12	false	false	4.8∼5.2

LIDAR	Model	Baudrate	Sample↩ Rate(K)	Range(m)	Frequency(HZ)	Intenstiy(b	Single↩ Channel	voltage(↩ V)
X4	6	128000	5	0.12∼10	5∼12(P↩ WM)	false	false	4.8∼5.2
X2/X2L	6	115200	3	0.10∼8.0	4∼8(P↩ WM)	false	true	4.8∼5.2
G4PRO	7	230400	9/8/4	0.↩ 28/0.26/0.↩ 1∼16	5∼12	false	false	4.8∼5.2
F4PRO	8	230400	4/6	0.12∼12	5∼12	false	false	4.8∼5.2
R2	9	230400	5	0.12∼16	5∼12	false	false	4.8∼5.2
G6	13	512000	18/16/8	0.↩ 28/0.26/0.↩ 1∼25	5∼12	false	false	4.8∼5.2
G2A	14	230400	5	0.12∼12	5∼12	false	false	4.8∼5.2
G2	15	230400	5	0.28∼16	5∼12	true(8)	false	4.8∼5.2
G2C	16	115200	4	0.1∼12	5∼12	false	false	4.8∼5.2
G4B	17	512000	10	0.12∼16	5∼12	true(10)	false	4.8∼5.2
G4C	18	115200	4	0.1∼12	5∼12	false	false	4.8∼5.2
G1	19	230400	9	0.28∼16	5∼12	false	false	4.8∼5.2
TX8	100	115200	4	0.1∼8	4∼8(P↩ WM)	false	true	4.8∼5.2
TX20	100	115200	4	0.1∼20	4∼8(P↩ WM)	false	true	4.8∼5.2
TG15	100	512000	20/18/10	0.05∼30	3∼16	false	false	4.8∼5.2
TG30	101	512000	20/18/10	0.05∼30	3∼16	false	false	4.8∼5.2
TG50	102	512000	20/18/10	0.05∼50	3∼16	false	false	4.8∼5.2
T15	200	8000	20	0.05∼15	10-35	true	false	4.8∼5.2
T30	200	8000	20	0.05∼30	10-35	true	false	4.8∼5.2

Table List - SerialBaudrate

LiDAR	SerialBaudrate
F4/S2/X2/X2L/S4/TX8/TX20/G4C	115200
X4	128000
S4B	153600
G1/G2/R2/G4/G4PRO/F4PRO	230400
G2A/G2C	230400
G6/G4B/TG15/TG30/TG50	512000
T5/T15(network)	8000

Table List - SampleRate

LiDAR	SampleRate
G4/F4	4,8,9
F4PRO	4,6
G6	8,16,18
G4B	10
G1	9
G2A/G2/R2/X4	5
S4/S4B/G4C/TX8/TX20	4

G2C	4
S2	3
TG15/TG30/TG50	10,18,20
T5/T15	20

Table List - ScanFrequency

LiDAR	ScanFrequency
S2/X2/X2L/TX8/TX20	4∼8(PWM)
F4/F4PRO/G4/G4PRO/R2	5∼12
G6/G2A/G2/G2C/G4B/G4C/G1	5∼12
S4/S4B/X4	5∼12(PWM)
TG15/TG30/TG50	3∼16
T5/T15	5∼40

Table List - SingleChannel

LiDAR	**SingleChannel**
G1/G2/G2A/G2C	false
G4/G4B/G4PRO/G6/F4/F4PRO	false
S4/S4B/X4/R2/G4C	false
S2/X2/X2L	true
TG15/TG30/TG50	false
TX8/TX20	true
T5/T15	false
	true

Table List - LidarType

LiDAR	**LidarType**
G1/G2A/G2/G2C	TYPE_TRIANGLE
G4/G4B/G4C/G4PRO	TYPE_TRIANGLE
G6/F4/F4PRO	TYPE_TRIANGLE
S4/S4B/X4/R2/S2/X2/X2L	TYPE_TRIANGLE
TG15/TG30/TG50/TX8/TX20	TYPE_TOF
T5/T15	TYPE_TOF_NET

Table List - Intensity

LiDAR	**Intensity**
S4B/G2/G4B	true
G4/G4C/G4PRO/F4/F4PRO/G6	false
G1/G2A/G2C/R2	false
S2/X2/X2L/X4	false
TG15/TG30/TG50	false
TX8/TX20	false
T5/T15	true
	false

Table List - SupportMotorDtrCtrl

S4/S4B/S2/X2/X2L/X4	SupportMotorDtrCtrl
S4/S4B/S2/X2/X2L/X4	true
TX8/TX20	true
G4/G4C/G4PRO/F4/F4PRO/G6	false
G1/G2A/G2C/R2/G2/G4B	false
TG15/TG30/TG50	false
T5/T15	false

Chapter 28

README

Table of Contents

Introduction

YDLidar SDK is the software development kit designed for all YDLIDAR products. It is developed based on C/C++ following YDLidar SDK Communication Protocol, and provides easy-to-use C/C++, Python, C# style API. With YDLidar SDK, users can quickly connect to YDLidar products and receive Laser scan data.

YDLidar SDK consists of YDLidar SDK communication protocol, YDLidar SDK core, YDLidar SDK API, Linux/windows samples, and Python demo.

Prerequisites

- Linux
- Windows 7/10, Visual Studio 2015/2017
- C++11 compiler

Supported Languages

- C / C++
- Python
- C#

YDLidar SDK Communication Protocol

YDLidar SDK communication protocol opens to all users. It is the communication protocol between user programs and YDLIDAR products. The protocol consists of control commands and data format. Please refer to the YDLidar SDK Communication Protocol for detailed information.

Architecture

YDLidar SDK provides the implementation of control commands and Laser scan data transmission, as well as the C/C++,Python API. The basic structure of YDLidar SDK is shown as below:

Serial or network is used for communication between YDLidar SDK and LiDAR sensors. Please refer to the YDLidar SDK Communication Protocol for further information. LaserScan supports Laser Scan Data transmission, while Command handler receives and sends control commands. And the C++ API is based on Command and LaserScan Hander.

The YDLidar LiDAR sensors can be connected to host directly by serial or through the YDLidar Adapter board. YDLidar SDK supports both connection methods. When LiDAR units are connected to host directly by Serial, the host will establish communication with each LiDAR unit individually. And if the LiDAR units connect to host through Adapter board, then the host only communicates with the YDLidar Adapter board while the Adapter Board communicates with each LiDAR unit.

Installation

- `Fork and then Clone YDLidar-SDK's GitHub code`
- Build and Install - This step is required

Documents

- LiDAR Dataset: All you need to know about LiDAR Models.
- SDK FlowChart: Development flowchart.
- YDLIDAR SDK API for Developers: All you need to know about YDLiDAR-SDK API
- YDLIDAR SDK Communication Protocol: All you need to know about YDLiDAR-SDK Communication Protocol.
- HowTo: Brief technical solutions to common problems that developers face during the installation and use of the YDLidar-SDk
- Tutorials: Quick Tutorials
- FAQs

Support

You can get support from YDLidar with the following methods:

- Send email to support@ydlidar.com with a clear description of your problem and your setup
- Github Issues

Contact EAI

If you have any extra questions, please feel free to contact us

Chapter 29

ETLidarDriver

ETLidarDriver API

Library	ETLidarDriver
File	ETLidarDriver.h
Author	Tony [code at ydlidar com]
Source	https://github.com/ydlidar/YDLidar-SDK
Version	1.0.0

This ETLidarDriver support TYPE_TOF_NET LiDAR

Copyright

Chapter 30

YDlidarDriver

YDlidarDriver API

Library	YDlidarDriver
File	ydlidar_driver.h
Author	Tony [code at ydlidar com]
Source	https://github.com/ydlidar/YDLidar-SDK
Version	1.0.0

This YDlidarDriver support TYPE_TRIANGLE and TYPE_TOF LiDAR

Copyright

Chapter 31

C API

YDLIDAR C API

Library	ydlidar_sdk
File	ydlidar_sdk.h
Author	Tony [code at ydlidar com]
Source	https://github.com/ydlidar/YDLidar-SDK
Version	1.0.0

Copyright

Chapter 32

Todo List

Member CYdLidar::getlidaropt (int optname, void *optval, int optlen)

string properties

- LidarPropSerialPort
- LidarPropIgnoreArray

int properties

- LidarPropSerialBaudrate
- LidarPropLidarType
- LidarPropDeviceType
- LidarPropSampleRate

bool properties

- LidarPropFixedResolution
- LidarPropReversion
- LidarPropInverted
- LidarPropAutoReconnect
- LidarPropSingleChannel
- LidarPropIntenstiy

float properties

- LidarPropMaxRange
- LidarPropMinRange
- LidarPropMaxAngle
- LidarPropMinAngle
- LidarPropScanFrequency

Member CYdLidar::setlidaropt (int optname, const void *optval, int optlen)

string properties

- LidarPropSerialPort
- LidarPropIgnoreArray

int properties

- LidarPropSerialBaudrate
- LidarPropLidarType

- LidarPropDeviceType
- LidarPropSampleRate

bool properties

- LidarPropFixedResolution
- LidarPropReversion
- LidarPropInverted
- LidarPropAutoReconnect
- LidarPropSingleChannel
- LidarPropIntenstiy

float properties

- LidarPropMaxRange
- LidarPropMinRange
- LidarPropMaxAngle
- LidarPropMinAngle
- LidarPropScanFrequency

Member getlidaropt (YDLidar *lidar, int optname, void *optval, int optlen)

string properties

- LidarPropSerialPort
- LidarPropIgnoreArray

int properties

- LidarPropSerialBaudrate
- LidarPropLidarType
- LidarPropDeviceType
- LidarPropSampleRate

bool properties

- LidarPropFixedResolution
- LidarPropReversion
- LidarPropInverted
- LidarPropAutoReconnect
- LidarPropSingleChannel
- LidarPropIntenstiy

float properties

- LidarPropMaxRange
- LidarPropMinRange
- LidarPropMaxAngle
- LidarPropMinAngle
- LidarPropScanFrequency

Member setlidaropt (YDLidar *lidar, int optname, const void *optval, int optlen)

string properties

- LidarPropSerialPort
- LidarPropIgnoreArray

int properties

- LidarPropSerialBaudrate
- LidarPropLidarType
- LidarPropDeviceType
- LidarPropSampleRate

bool properties

- LidarPropFixedResolution
- LidarPropReversion
- LidarPropInverted
- LidarPropAutoReconnect
- LidarPropSingleChannel
- LidarPropIntenstiy

float properties

- LidarPropMaxRange
- LidarPropMinRange
- LidarPropMaxAngle
- LidarPropMinAngle
- LidarPropScanFrequency

Page YDLIDAR SDK API for Developers

string properties

- - LidarPropSerialPort
- - LidarPropIgnoreArray
-

int properties

- - LidarPropSerialBaudrate
- - LidarPropLidarType
- - LidarPropDeviceType
- - LidarPropSampleRate
-

bool properties

- - LidarPropFixedResolution
- - LidarPropReversion
- - LidarPropInverted
- - LidarPropAutoReconnect
- - LidarPropSingleChannel
- - LidarPropIntenstiy
-

float properties

- - LidarPropMaxRange
- - LidarPropMinRange
- - LidarPropMaxAngle
- - LidarPropMinAngle

- - LidarPropScanFrequency
-

string properties

- - LidarPropSerialPort
- - LidarPropIgnoreArray
-

int properties

- - LidarPropSerialBaudrate
- - LidarPropLidarType
- - LidarPropDeviceType
- - LidarPropSampleRate
-

bool properties

- - LidarPropFixedResolution
- - LidarPropReversion
- - LidarPropInverted
- - LidarPropAutoReconnect
- - LidarPropSingleChannel
- - LidarPropIntenstiy
-

float properties

- - LidarPropMaxRange
- - LidarPropMinRange
- - LidarPropMaxAngle
- - LidarPropMinAngle
- - LidarPropScanFrequency
-

string properties

- - LidarPropSerialPort
- - LidarPropIgnoreArray
-

int properties

- - LidarPropSerialBaudrate
- - LidarPropLidarType
- - LidarPropDeviceType
- - LidarPropSampleRate
-

bool properties

- - LidarPropFixedResolution
- - LidarPropReversion
- - LidarPropInverted

- - LidarPropAutoReconnect
- - LidarPropSingleChannel
- - LidarPropIntenstiy
-

float properties

- - LidarPropMaxRange
- - LidarPropMinRange
- - LidarPropMaxAngle
- - LidarPropMinAngle
- - LidarPropScanFrequency
-

string properties

- - LidarPropSerialPort
- - LidarPropIgnoreArray
-

int properties

- - LidarPropSerialBaudrate
- - LidarPropLidarType
- - LidarPropDeviceType
- - LidarPropSampleRate
-

bool properties

- - LidarPropFixedResolution
- - LidarPropReversion
- - LidarPropInverted
- - LidarPropAutoReconnect
- - LidarPropSingleChannel
- - LidarPropIntenstiy
-

float properties

- - LidarPropMaxRange
- - LidarPropMinRange
- - LidarPropMaxAngle
- - LidarPropMinAngle
- - LidarPropScanFrequency
-

Chapter 33

Namespace Index

33.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 34

Hierarchical Index

34.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 35

Class Index

35.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 36

File Index

36.1 File List

Here is a list of all files with brief descriptions:

Chapter 37

Namespace Documentation

37.1 etlidar_test Namespace Reference

Variables

- laser = ydlidar.CYdLidar();
- ret = laser.initialize();
- scan = ydlidar.LaserScan();
- r = laser.doProcessSimple(scan);

37.1.1 Variable Documentation

37.1.1.1 etlidar_test.laser = ydlidar.CYdLidar();

Definition at line 7 of file etlidar_test.py.

37.1.1.2 etlidar_test.r = laser.doProcessSimple(scan);

Definition at line 22 of file etlidar_test.py.

37.1.1.3 etlidar_test.ret = laser.initialize();

Definition at line 17 of file etlidar_test.py.

37.1.1.4 etlidar_test.scan = ydlidar.LaserScan();

Definition at line 20 of file etlidar_test.py.

37.2 impl Namespace Reference

Functions

- uint32_t getHDTimer ()
- uint64_t getCurrentTime ()

37.2.1 Function Documentation

37.2.1.1 uint64_t impl::getCurrentTime ( )

Definition at line 44 of file timer.cpp.

37.2.1.2 uint32_t impl::getHDTimer ( )

Definition at line 38 of file timer.cpp.

37.3 plot_tof_test Namespace Reference

Functions

- def animate (num)

Variables

- float RMAX = 32.0
- fig = plt.figure()
- lidar_polar = plt.subplot(polar=True)
- ports = ydlidar.lidarPortList();
- string port = "/dev/ydlidar"
- laser = ydlidar.CYdLidar();
- scan = ydlidar.LaserScan()
- ret = laser.initialize();
- ani = animation.FuncAnimation(fig, animate, interval=50)

37.3.1 Function Documentation

37.3.1.1 def plot_tof_test.animate (*num*)

Definition at line 34 of file plot_tof_test.py.

37.3.2 Variable Documentation

37.3.2.1 plot_tof_test.ani = animation.FuncAnimation(fig, animate, interval=50)

Definition at line 52 of file plot_tof_test.py.

37.3.2.2 plot_tof_test.fig = plt.figure()

Definition at line 13 of file plot_tof_test.py.

37.3.2.3 plot_tof_test.laser = ydlidar.CYdLidar();

Definition at line 24 of file plot_tof_test.py.

37.3.2.4 plot_tof_test.lidar_polar = plt.subplot(polar=True)

Definition at line 15 of file plot_tof_test.py.

37.3.2.5 plot_tof_test.port = "/dev/ydlidar"

Definition at line 20 of file plot_tof_test.py.

37.3.2.6 plot_tof_test.ports = ydlidar.lidarPortList();

Definition at line 19 of file plot_tof_test.py.

37.3.2.7 plot_tof_test.ret = laser.initialize();

Definition at line 48 of file plot_tof_test.py.

37.3.2.8 float plot_tof_test.RMAX = 32.0

Definition at line 10 of file plot_tof_test.py.

37.3.2.9 plot_tof_test.scan = ydlidar.LaserScan()

Definition at line 32 of file plot_tof_test.py.

37.4 plot_ydlidar_test Namespace Reference

Functions

- def animate (num)

Variables

- float RMAX = 32.0
- fig = plt.figure()
- lidar_polar = plt.subplot(polar=True)
- ports = ydlidar.lidarPortList();
- string port = "/dev/ydlidar"
- laser = ydlidar.CYdLidar();
- scan = ydlidar.LaserScan()
- ret = laser.initialize();
- ani = animation.FuncAnimation(fig, animate, interval=50)

37.4.1 Function Documentation

37.4.1.1 def plot_ydlidar_test.animate (*num*)

Definition at line 34 of file plot_ydlidar_test.py.

37.4.2 Variable Documentation

37.4.2.1 plot_ydlidar_test.ani = animation.FuncAnimation(fig, animate, interval=50)

Definition at line 52 of file plot_ydlidar_test.py.

37.4.2.2 plot_ydlidar_test.fig = plt.figure()

Definition at line 13 of file plot_ydlidar_test.py.

37.4.2.3 plot_ydlidar_test.laser = ydlidar.CYdLidar();

Definition at line 24 of file plot_ydlidar_test.py.

37.4.2.4 plot_ydlidar_test.lidar_polar = plt.subplot(polar=True)

Definition at line 15 of file plot_ydlidar_test.py.

37.4.2.5 plot_ydlidar_test.port = "/dev/ydlidar"

Definition at line 20 of file plot_ydlidar_test.py.

37.4.2.6 plot_ydlidar_test.ports = ydlidar.lidarPortList();

Definition at line 19 of file plot_ydlidar_test.py.

37.4.2.7 plot_ydlidar_test.ret = laser.initialize();

Definition at line 48 of file plot_ydlidar_test.py.

37.4.2.8 float plot_ydlidar_test.RMAX = 32.0

Definition at line 10 of file plot_ydlidar_test.py.

37.4.2.9 plot_ydlidar_test.scan = ydlidar.LaserScan()

Definition at line 32 of file plot_ydlidar_test.py.

37.5 pytest Namespace Reference

Classes

- class PyTestTestCase

37.6 serial Namespace Reference

Namespaces

- Serial

37.7 serial::Serial Namespace Reference

Classes

- class ScopedReadLock
- class ScopedWriteLock
- class SerialImpl

37.8 setup Namespace Reference

Classes

- class CMakeBuild
- class CMakeExtension

Variables

- string YDLIDAR_SDK_REPO = "https://github.com/YDLIDAR/YDLidar-SDK"
- string YDLIDAR_SDK_BRANCH = "master"

37.8.1 Variable Documentation

37.8.1.1 string setup.YDLIDAR_SDK_BRANCH = "master"

Definition at line 13 of file setup.py.

37.8.1.2 string setup.YDLIDAR_SDK_REPO = "https://github.com/YDLIDAR/YDLidar-SDK"

Definition at line 12 of file setup.py.

37.9 test Namespace Reference

Variables

- ports = ydlidar.lidarPortList();
- string port = "/dev/ydlidar"
- laser = ydlidar.CYdLidar();
- ret = laser.initialize();
- scan = ydlidar.LaserScan();
- r = laser.doProcessSimple(scan);

37.9.1 Variable Documentation

37.9.1.1 test.laser = ydlidar.CYdLidar();

Definition at line 10 of file test.py.

37.9.1.2 test.port = "/dev/ydlidar"

Definition at line 7 of file test.py.

37.9.1.3 test.ports = ydlidar.lidarPortList();

Definition at line 6 of file test.py.

37.9.1.4 test.r = laser.doProcessSimple(scan);

Definition at line 24 of file test.py.

37.9.1.5 test.ret = laser.initialize();

Definition at line 19 of file test.py.

37.9.1.6 test.scan = ydlidar.LaserScan();

Definition at line 22 of file test.py.

37.10 tof_test Namespace Reference

Variables

- ports = ydlidar.lidarPortList();
- string port = "/dev/ydlidar"
- laser = ydlidar.CYdLidar();
- ret = laser.initialize();
- scan = ydlidar.LaserScan()
- r = laser.doProcessSimple(scan);

37.10.1 Variable Documentation

37.10.1.1 tof_test.laser = ydlidar.CYdLidar();

Definition at line 12 of file tof_test.py.

37.10.1.2 tof_test.port = "/dev/ydlidar"

Definition at line 9 of file tof_test.py.

37.10.1.3 tof_test.ports = ydlidar.lidarPortList();

Definition at line 8 of file tof_test.py.

37.10.1.4 tof_test.r = laser.doProcessSimple(scan);

Definition at line 26 of file tof_test.py.

37.10.1.5 tof_test.ret = laser.initialize();

Definition at line 21 of file tof_test.py.

37.10.1.6 tof_test.scan = ydlidar.LaserScan()

Definition at line 24 of file tof_test.py.

37.11 ydlidar Namespace Reference

ydlidar

Namespaces

- core
  ydlidar core

Classes

- class ETLidarDriver
- class YDlidarDriver

Functions

- void os_init ()
  system signal initialize
- bool os_isOk ()
  Whether system signal is initialized.
- void os_shutdown ()
  shutdown system signal
- std::map< std::string, std::string > lidarPortList ()
  lidarPortList

37.11.1 Detailed Description

ydlidar

Provides a platform independent class to create a passive socket. A passive socket is used to create a "listening" socket. This type of object would be used when an application needs to wait for inbound connections. Support for CSimpleSocket::SocketTypeTcp, CSimpleSocket::SocketTypeUdp, and CSimpleSocket::SocketTypeRaw is handled in a similar fashion. The big difference is that the method CPassiveSocket::Accept should not be called on the latter two socket types.

setup_port - Configure the port, eg. baud rate, data bits,etc.

Parameters

fd	: The serial port
speed	: The baud rate
data_bits	: The data bits
parity	: The parity bits
stop_bits	: The stop bits

Returns

Return 0 if everything is OK, otherwise -1 with some error msg.

Note

Here are termios structure members:

```
Member      Description
c_cflag     Control options
c_lflag     Line options
c_iflag     Input options
c_oflag     Output options
c_cc        Control characters
c_ispeed    Input baud (new interface)
c_ospeed    Output baud (new interface)
```

The c_cflag member controls the baud rate, number of data bits, parity, stop bits, and hardware flow control. There are constants for all of the supported configurations. Constant Description

```
CBAUD   Bit mask for baud rate
B0  0 baud (drop DTR)
B50 50 baud
B75 75 baud
B110    110 baud
B134    134.5 baud
B150    150 baud
B200    200 baud
B300    300 baud
B600    600 baud
B1200   1200 baud
B1800   1800 baud
B2400   2400 baud
B4800   4800 baud
B9600   9600 baud
B19200  19200 baud
B38400  38400 baud
B57600  57,600 baud
B76800  76,800 baud
B115200 115,200 baud
EXTA    External rate clock
EXTB    External rate clock
CSIZE   Bit mask for data bits
CS5 5   data bits
CS6 6   data bits
CS7 7   data bits
CS8 8   data bits
CSTOPB  2 stop bits (1 otherwise)
CREAD   Enable receiver
PARENB  Enable parity bit
PARODD  Use odd parity instead of even
HUPCL   Hangup (drop DTR) on last close
CLOCAL  Local line - do not change "owner" of port
LOBLK   Block job control output
CNEW_RTSCTS CRTSCTS Enable hardware flow control (not supported on all
platforms)
```

The input modes member c_iflag controls any input processing that is done to characters received on the port. Like the c_cflag field, the final value stored in c_iflag is the bitwise OR of the desired options.

```
Constant    Description
INPCK   Enable parity check
IGNPAR  Ignore parity errors
PARMRK  Mark parity errors
ISTRIP  Strip parity bits
IXON    Enable software flow control (outgoing)
IXOFF   Enable software flow control (incoming)
IXANY   Allow any character to start flow again
IGNBRK  Ignore break condition
BRKINT  Send a SIGINT when a break condition is detected
INLCR   Map NL to CR
IGNCR   Ignore CR
ICRNL   Map CR to NL
IUCLC   Map uppercase to lowercase
IMAXBEL Echo BEL on input line too long
```

Here are some examples of setting parity checking:
No parity (8N1):

```
options.c_cflag &= ~PARENB
options.c_cflag &= ~CSTOPB
options.c_cflag &= ~CSIZE;
options.c_cflag |= CS8;
```

Even parity (7E1):

```
options.c_cflag |= PARENB
options.c_cflag &= ~PARODD
options.c_cflag &= ~CSTOPB
options.c_cflag &= ~CSIZE;
options.c_cflag |= CS7;
```

Odd parity (7O1):

```
options.c_cflag |= PARENB
options.c_cflag |= PARODD
options.c_cflag &= ~CSTOPB
options.c_cflag &= ~CSIZE;
options.c_cflag |= CS7;
```

37.11.2 Function Documentation

37.11.2.1 std::map< std::string, std::string > ydlidar::lidarPortList ( )

lidarPortList

Returns

Definition at line 1390 of file CYdLidar.cpp.

37.11.2.2 void ydlidar::os_init ( )

system signal initialize

initialize system signals

Definition at line 1378 of file CYdLidar.cpp.

37.11.2.3 bool ydlidar::os_isOk ( )

Whether system signal is initialized.

isOk

Returns

Definition at line 1382 of file CYdLidar.cpp.

37.11.2.4 void ydlidar::os_shutdown ( )

shutdown system signal

os_shutdown

Definition at line 1386 of file CYdLidar.cpp.

37.12 ydlidar::core Namespace Reference

ydlidar core

Namespaces

- base
- common

  ydlidar common
- math
- network
- serial

37.12.1 Detailed Description

ydlidar core

37.13 ydlidar::core::base Namespace Reference

Classes

- class Event
- class Locker
- class ScopedLocker
- class Thread

Functions

- void init ()
  initialize system state
- bool ok ()
  ok
- void shutdown ()
  shutdown
- bool fileExists (const std::string &filename)
  fileExists

37.13.1 Function Documentation

37.13.1.1 bool ydlidar::core::base::fileExists (const std::string & *filename*) [inline]

fileExists

Parameters

filename	

Returns

Definition at line 178 of file ydlidar.h.

37.13.1.2 void ydlidar::core::base::init ( ) [inline]

initialize system state

Parameters

argc	
argv	

Definition at line 144 of file ydlidar.h.

37.13.1.3 bool ydlidar::core::base::ok ( ) [inline]

ok

Returns

Definition at line 163 of file ydlidar.h.

37.13.1.4 void ydlidar::core::base::shutdown ( ) [inline]

shutdown

Definition at line 169 of file ydlidar.h.

37.14 ydlidar::core::common Namespace Reference

ydlidar common

Classes

- class ChannelDevice
- class DriverInterface

Functions

- std::string lidarModelToString (int model)
  convert lidar model to string
- int lidarModelDefaultSampleRate (int model)
  Get LiDAR default sampling rate.
- bool isOctaveLidar (int model)
  Query whether the LiDAR is Octave LiDAR.
- bool hasSampleRate (int model)
  Supports multiple sampling rate.
- bool hasZeroAngle (int model)
  Is there a zero offset angle.
- bool hasScanFrequencyCtrl (int model)
  Whether to support adjusting the scanning frequency .
- bool isSupportLidar (int model)
  Does SDK support the LiDAR model.
- bool hasIntensity (int model)
  Whether to support intensity.
- bool isSupportMotorCtrl (int model)
  Whether to support serial DTR enable motor.
- bool isSupportScanFrequency (int model, double frequency)
  Whether the scanning frequency is supported.
- bool isTOFLidarByModel (int model)
  Whether it is a TOF Model LiDAR.
- bool isNetTOFLidarByModel (int model)
  Whether it is a Net TOF Model LiDAR.
- bool isTOFLidar (int type)
  Whether it is a TOF type LiDAR.
- bool isNetTOFLidar (int type)
  Whether it is a network hardware interface TOF type LiDAR.
- bool isTriangleLidar (int type)
  Whether it is a Triangle type LiDAR.
- bool isOldVersionTOFLidar (int model, int Major, int Minor)

Whether it is Old Version protocol TOF LiDAR.

- bool isValidSampleRate (std::map< int, int > smap)

  Whether the sampling rate is valid.
- int ConvertUserToLidarSmaple (int model, int m_SampleRate, int defaultRate)

  convert User sampling rate code to LiDAR sampling code
- int ConvertLidarToUserSmaple (int model, int rate)

  convert LiDAR sampling rate code to User sampling code
- bool isValidValue (uint8_t value)

  Whether the Value is valid.
- bool isVersionValid (const LaserDebug &info)

  Whether the Version is valid.
- bool isSerialNumbValid (const LaserDebug &info)

  Whether the serial number is valid.
- void parsePackageNode (const node_info &node, LaserDebug &info)

  convert node_info to LaserDebug
- bool ParseLaserDebugInfo (const LaserDebug &info, device_info &value)

  convert LaserDebug information to device_info
- bool printfVersionInfo (const device_info &info, const std::string &port, int baudrate)

  print LiDAR version information
- std::vector< float > split (const std::string &s, char delim)

  split string to vector by delim format
- bool isV1Protocol (uint8_t protocol)

  Whether the ET LiDAR Protocol type is V1.

37.14.1 Detailed Description

ydlidar common

37.14.2 Function Documentation

37.14.2.1 int ydlidar::core::common::ConvertLidarToUserSmaple (int *model,* int *rate*) [inline]

convert LiDAR sampling rate code to User sampling code

Parameters

model	LiDAR model
rate	LiDAR sampling rate code

Returns

user sampling code

Definition at line 651 of file ydlidar_help.h.

37.14.2.2 int ydlidar::core::common::ConvertUserToLidarSmaple (int *model,* int *m_SampleRate,* int *defaultRate*) [inline]

convert User sampling rate code to LiDAR sampling code

Parameters

model	LiDAR model
m_SampleRate	User sampling rate code
defaultRate	LiDAR Defualt sampling rate code

Returns

LiDAR sampling rate code

Definition at line 577 of file ydlidar_help.h.

37.14.2.3 bool ydlidar::core::common::hasIntensity (int *model*) [inline]

Whether to support intensity.

Parameters

model	lidar model

Returns

true if supported, otherwise false.

Definition at line 392 of file ydlidar_help.h.

37.14.2.4 bool ydlidar::core::common::hasSampleRate (int *model*) [inline]

Supports multiple sampling rate.

Parameters

model	lidar model

Returns

true if THere are multiple sampling rate, otherwise false.

Definition at line 311 of file ydlidar_help.h.

37.14.2.5 bool ydlidar::core::common::hasScanFrequencyCtrl (int *model*) [inline]

Whether to support adjusting the scanning frequency .

Parameters

model	lidar model

Returns

true if supported, otherwise false.

Definition at line 354 of file ydlidar_help.h.

37.14.2.6 bool ydlidar::core::common::hasZeroAngle (int *model*) [inline]

Is there a zero offset angle.

Parameters

model	lidar model

Returns

true if there are zero offset angle, otherwise false.

Definition at line 332 of file ydlidar_help.h.

37.14.2.7 bool ydlidar::core::common::isNetTOFLidar (int *type*) [inline]

Whether it is a network hardware interface TOF type LiDAR.

Parameters

type	LiDAR type

Returns

true if it is a network hardware interface TOF type, otherwise false.

Definition at line 502 of file ydlidar_help.h.

37.14.2.8 bool ydlidar::core::common::isNetTOFLidarByModel (int *model*) [inline]

Whether it is a Net TOF Model LiDAR.

Parameters

model	LiDAR model

Returns

tru if it is Net TOF Model, otherwise false.

Definition at line 472 of file ydlidar_help.h.

37.14.2.9 bool ydlidar::core::common::isOctaveLidar (int *model*) [inline]

Query whether the LiDAR is Octave LiDAR.

Parameters

model	lidar model

Returns

true if the current lidar sampling rate is octave, otherwise false

Definition at line 296 of file ydlidar_help.h.

37.14.2.10 bool ydlidar::core::common::isOldVersionTOFLidar (int *model,* int *Major,* int *Minor*) [inline]

Whether it is Old Version protocol TOF LiDAR.

Parameters

model	lidar model
Major	firmware Major version
Minor	firmware Minor version

Returns

true if it is old version protocol, otherwise false.

Definition at line 534 of file ydlidar_help.h.

37.14.2.11 bool ydlidar::core::common::isSerialNumbValid (const LaserDebug & *info*) [inline]

Whether the serial number is valid.

Parameters

info	LiDAR LaserDebug information

Returns

true if it is valid, otherwise false.

Definition at line 744 of file ydlidar_help.h.

37.14.2.12 bool ydlidar::core::common::isSupportLidar (int *model*) [inline]

Does SDK support the LiDAR model.

Parameters

model	lidar model

Returns

true if supported, otherwise false.

Definition at line 372 of file ydlidar_help.h.

37.14.2.13 bool ydlidar::core::common::isSupportMotorCtrl (int *model*) [inline]

Whether to support serial DTR enable motor.

Parameters

model	lidar model

Returns

true if support serial DTR enable motor, otherwise false.

Definition at line 409 of file ydlidar_help.h.

37.14.2.14 bool ydlidar::core::common::isSupportScanFrequency (int *model,* double *frequency*) [inline]

Whether the scanning frequency is supported.

Parameters

model	lidar model
frequency	scanning frequency

Returns

true if supported, otherwise false.

Definition at line 429 of file ydlidar_help.h.

37.14.2.15 bool ydlidar::core::common::isTOFLidar (int *type*) [inline]

Whether it is a TOF type LiDAR.

Parameters

type	LiDAR type

Returns

true if it is a TOF type, otherwise false.

Definition at line 487 of file ydlidar_help.h.

37.14.2.16 bool ydlidar::core::common::isTOFLidarByModel (int *model*) [inline]

Whether it is a TOF Model LiDAR.

Parameters

model	LiDAR model

Returns

tru if it is TOF Model, otherwise false.

Definition at line 456 of file ydlidar_help.h.

37.14.2.17 bool ydlidar::core::common::isTriangleLidar (int *type*) [inline]

Whether it is a Triangle type LiDAR.

Parameters

type	LiDAR type

Returns

true if it is a Triangle type, otherwise false.

Definition at line 517 of file ydlidar_help.h.

37.14.2.18 bool ydlidar::core::common::isV1Protocol (uint8_t *protocol*) [inline]

Whether the ET LiDAR Protocol type is V1.

Parameters

protocol	LiDAR Protocol Byte information

Returns

true if it is V1, otherwise false

Definition at line 941 of file ydlidar_help.h.

37.14.2.19 bool ydlidar::core::common::isValidSampleRate (std::map< int, int > *smap*) [inline]

Whether the sampling rate is valid.

Parameters

smap	sampling rate map

Returns

true if it is valid, otherwise false.

Definition at line 554 of file ydlidar_help.h.

37.14.2.20 bool ydlidar::core::common::isValidValue (uint8_t *value*) [inline]

Whether the Value is valid.

Parameters

value	LiDAR CT Byte information

Returns

true if it is valid, otherwise false

Definition at line 713 of file ydlidar_help.h.

37.14.2.21 bool ydlidar::core::common::isVersionValid (const LaserDebug & *info*) [inline]

Whether the Version is valid.

Parameters

info	the LiDAR LaserDebug information

Returns

true if it is valid, otherwise false.

Definition at line 726 of file ydlidar_help.h.

37.14.2.22 int ydlidar::core::common::lidarModelDefaultSampleRate (int *model*) [inline]

Get LiDAR default sampling rate.

Parameters

model	lidar model.

Returns

lidar sampling rate.

Definition at line 194 of file ydlidar_help.h.

37.14.2.23 std::string ydlidar::core::common::lidarModelToString (int *model*) [inline]

convert lidar model to string

Parameters

model	lidar model

Returns

lidar model name

Definition at line 66 of file ydlidar_help.h.

37.14.2.24 bool ydlidar::core::common::ParseLaserDebugInfo (const LaserDebug & *info*, device_info & *value*) [inline]

convert LaserDebug information to device_info

Parameters

info	LiDAR LaserDebug information
value	LiDAR Device information

Returns

true if converted successfully, otherwise false.

Definition at line 840 of file ydlidar_help.h.

37.14.2.25 void ydlidar::core::common::parsePackageNode (const node_info & *node*, LaserDebug & *info*) [inline]

convert node_info to LaserDebug

Parameters

node	LiDAR node_info information
info	LiDAR LaserDebug information

Definition at line 762 of file ydlidar_help.h.

37.14.2.26 bool ydlidar::core::common::printfVersionInfo (const device_info & *info*, const std::string & *port*, int *baudrate*) [inline]

print LiDAR version information

Parameters

info	LiDAR Device information
port	LiDAR serial port or IP Address
baudrate	LiDAR serial baudrate or network port

Returns

true if Device information is valid, otherwise false

Definition at line 888 of file ydlidar_help.h.

37.14.2.27 std::vector<float> ydlidar::core::common::split (const std::string & *s*, char *delim*) [inline]

split string to vector by delim format

Parameters

s	string
delim	split format

Returns

split vector

Definition at line 924 of file ydlidar_help.h.

37.15 ydlidar::core::math Namespace Reference

Functions

- static double from_degrees (double degrees)

  Convert degrees to radians.
- static double to_degrees (double radians)

  Convert radians to degrees.
- static double normalize_angle_positive (double angle)

  normalize_angle_positive
- static double normalize_angle_positive_from_degree (double angle)

  normalize_angle_positive_from_degree
- static double normalize_angle (double angle)

  normalize
- static double shortest_angular_distance (double from, double to)

  shortest_angular_distance
- static double two_pi_complement (double angle)

  returns the angle in [-2∗M_PI, 2∗M_PI] going the other way along the unit circle.
- static bool find_min_max_delta (double from, double left_limit, double right_limit, double &result_min_delta, double &result_max_delta)

  This function is only intended for internal use and not intended for external use. If you do use it, read the documentation very carefully. Returns the min and max amount (in radians) that can be moved from "from" angle to "left_limit" and "right_limit".
- static bool shortest_angular_distance_with_limits (double from, double to, double left_limit, double right_limit, double &shortest_angle)

  Returns the delta from "from_angle" to "to_angle" making sure it does not violate limits specified by left_limit and right_limit. The valid interval of angular positions is [left_limit,right_limit]. E.g., [-0.25,0.25] is a 0.5 radians wide interval that contains 0. But [0.25,-0.25] is a 2∗M_PI-0.5 wide interval that contains M_PI (but not 0). The value of shortest_angle is the angular difference between "from" and "to" that lies within the defined valid interval. E.g. shortest_angular_distance_with_limits(-0.5,0.5,0.25,-0.25,ss) evaluates ss to 2∗M_PI-1.0 and returns true while shortest_angular_distance_with_limits(-0.5,0.5,-0.25,0.25,ss) returns false since -0.5 and 0.5 do not lie in the interval [-0.25,0.25].

37.15.1 Function Documentation

37.15.1.1 static bool ydlidar::core::math::find_min_max_delta (double *from,* double *left_limit,* double *right_limit,* double & *result_min_delta,* double & *result_max_delta*) [static]

This function is only intended for internal use and not intended for external use. If you do use it, read the documentation very carefully. Returns the min and max amount (in radians) that can be moved from "from" angle to "left_limit" and "right_limit".

Returns

returns false if "from" angle does not lie in the interval [left_limit,right_limit]

Parameters

from	- "from" angle - must lie in [-M_PI, M_PI)
left_limit	- left limit of valid interval for angular position - must lie in [-M_PI, M_PI], left and right limits are specified on the unit circle w.r.t to a reference pointing inwards
right_limit	- right limit of valid interval for angular position - must lie in [-M_PI, M_PI], left and right limits are specified on the unit circle w.r.t to a reference pointing inwards
result_min_delta	- minimum (delta) angle (in radians) that can be moved from "from" position before hitting the joint stop
result_max_delta	- maximum (delta) angle (in radians) that can be movedd from "from" position before hitting the joint stop

Definition at line 162 of file angles.h.

37.15.1.2 static double ydlidar::core::math::from_degrees (double *degrees*) [inline], [static]

Convert degrees to radians.

Definition at line 51 of file angles.h.

37.15.1.3 static double ydlidar::core::math::normalize_angle (double *angle*) [inline], [static]

normalize

Normalizes the angle to be -M_PI circle to +M_PI circle It takes and returns radians.

Definition at line 100 of file angles.h.

37.15.1.4 static double ydlidar::core::math::normalize_angle_positive (double *angle*) [inline], [static]

normalize_angle_positive

Normalizes the angle to be 0 to 2∗M_PI It takes and returns radians.

Definition at line 69 of file angles.h.

37.15.1.5 static double ydlidar::core::math::normalize_angle_positive_from_degree (double *angle*) [inline], [static]

normalize_angle_positive_from_degree

Normalizes the angle to be 0 to 360 It takes and returns degree.

Definition at line 79 of file angles.h.

37.15.1.6 static double ydlidar::core::math::shortest_angular_distance (double *from,* double *to*) [inline], [static]

shortest_angular_distance

Given 2 angles, this returns the shortest angular difference. The inputs and ouputs are of course radians.

The result would always be -pi <= result <= pi. Adding the result to "from" will always get you an equivelent angle to "to".

Definition at line 123 of file angles.h.

37.15.1.7 static bool ydlidar::core::math::shortest_angular_distance_with_limits (double *from,* double *to,* double *left_limit,* double *right_limit,* double & *shortest_angle*) [inline], [static]

Returns the delta from "from_angle" to "to_angle" making sure it does not violate limits specified by left_limit and right_limit. The valid interval of angular positions is [left_limit,right_limit]. E.g., [-0.25,0.25] is a 0.5 radians wide interval that contains 0. But [0.25,-0.25] is a 2∗M_PI-0.5 wide interval that contains M_PI (but not 0). The value of shortest_angle is the angular difference between "from" and "to" that lies within the defined valid interval. E.g. shortest_angular_distance_with_limits(-0.5,0.5,0.25,-0.25,ss) evaluates ss to 2∗M_PI-1.0 and returns true while shortest_angular_distance_with_limits(-0.5,0.5,-0.25,0.25,ss) returns false since -0.5 and 0.5 do not lie in the interval [-0.25,0.25].

Returns

true if "from" and "to" positions are within the limit interval, false otherwise

Parameters

from	- "from" angle
to	- "to" angle
left_limit	- left limit of valid interval for angular position, left and right limits are specified on the unit circle w.r.t to a reference pointing inwards
right_limit	- right limit of valid interval for angular position, left and right limits are specified on the unit circle w.r.t to a reference pointing inwards
shortest_angle	- result of the shortest angle calculation

Definition at line 237 of file angles.h.

37.15.1.8 static double ydlidar::core::math::to_degrees (double *radians*) [inline], [static]

Convert radians to degrees.

Definition at line 58 of file angles.h.

37.15.1.9 static double ydlidar::core::math::two_pi_complement (double *angle*) [inline], [static]

returns the angle in [-2∗M_PI, 2∗M_PI] going the other way along the unit circle.

Parameters

angle	The angle to which you want to turn in the range [-2∗M_PI, 2∗M_PI] E.g. two_pi_complement(-M_PI/4) returns 7_M_PI/4 two_pi_complement(M_PI/4) returns -7∗M_PI/4

Definition at line 136 of file angles.h.

37.16 ydlidar::core::network Namespace Reference

Classes

- class CActiveSocket
- class CPassiveSocket
- class CSimpleSocket

37.17 ydlidar::core::serial Namespace Reference

Classes

- class MillisecondTimer
- struct PortInfo
- class Serial
- struct termios2
- struct Timeout

Enumerations

- enum bytesize_t { fivebits = 5, sixbits = 6, sevenbits = 7, eightbits = 8 }
- enum parity_t {
  parity_none = 0, parity_odd = 1, parity_even = 2, parity_mark = 3,
  parity_space = 4 }
- enum stopbits_t { stopbits_one = 1, stopbits_two = 2, stopbits_one_point_five }
- enum flowcontrol_t { flowcontrol_none = 0, flowcontrol_software, flowcontrol_hardware }

Functions

- timespec timespec_from_ms (const uint32_t millis)
- static void set_common_props (termios ∗tio)
- static void set_databits (termios ∗tio, serial::bytesize_t databits)
- static void set_parity (termios ∗tio, serial::parity_t parity)
- static void set_stopbits (termios ∗tio, serial::stopbits_t stopbits)
- static void set_flowcontrol (termios ∗tio, serial::flowcontrol_t flowcontrol)
- static bool is_standardbaudrate (unsigned long baudrate, speed_t &baud)
- std::vector< PortInfo > list_ports ()

37.17.1 Enumeration Type Documentation

37.17.1.1 enum ydlidar::core::serial::bytesize_t

Enumeration defines the possible bytesizes for the serial port.

Enumerator

fivebits
sixbits
sevenbits
eightbits

Definition at line 21 of file serial.h.

37.17.1.2 enum ydlidar::core::serial::flowcontrol_t

Enumeration defines the possible flowcontrol types for the serial port.

Enumerator

flowcontrol_none
flowcontrol_software
flowcontrol_hardware

Definition at line 51 of file serial.h.

37.17.1.3 enum ydlidar::core::serial::parity_t

Enumeration defines the possible parity types for the serial port.

Enumerator

parity_none
parity_odd
parity_even
parity_mark
parity_space

Definition at line 31 of file serial.h.

37.17.1.4 enum ydlidar::core::serial::stopbits_t

Enumeration defines the possible stopbit types for the serial port.

Enumerator

stopbits_one
stopbits_two
stopbits_one_point_five

Definition at line 42 of file serial.h.

37.17.2 Function Documentation

37.17.2.1 static bool ydlidar::core::serial::is_standardbaudrate (unsigned long *baudrate,* speed_t & *baud*) [inline], [static]

Definition at line 402 of file unix_serial.cpp.

37.17.2.2 std::vector<PortInfo> ydlidar::core::serial::list_ports ( )

37.17.2.3 static void ydlidar::core::serial::set_common_props (termios * *tio*) [inline], [static]

Definition at line 275 of file unix_serial.cpp.

37.17.2.4 static void ydlidar::core::serial::set_databits (termios * *tio,* serial::bytesize_t *databits*) [inline], [static]

Definition at line 292 of file unix_serial.cpp.

37.17.2.5 static void ydlidar::core::serial::set_flowcontrol (termios * *tio,* serial::flowcontrol_t *flowcontrol*) [inline], [static]

Definition at line 376 of file unix_serial.cpp.

37.17.2.6 static void ydlidar::core::serial::set_parity (termios * *tio,* serial::parity_t *parity*) [inline], [static]

Definition at line 319 of file unix_serial.cpp.

37.17.2.7 static void ydlidar::core::serial::set_stopbits (termios * *tio,* serial::stopbits_t *stopbits*) [inline], [static]

Definition at line 360 of file unix_serial.cpp.

37.17.2.8 timespec ydlidar::core::serial::timespec_from_ms (const uint32_t *millis*)

Definition at line 267 of file unix_serial.cpp.

37.18 ydlidar_test Namespace Reference

Variables

- ports = ydlidar.lidarPortList();
- string port = "/dev/ydlidar"
- laser = ydlidar.CYdLidar();
- ret = laser.initialize();
- scan = ydlidar.LaserScan();
- r = laser.doProcessSimple(scan);

37.18.1 Variable Documentation

37.18.1.1 ydlidar_test.laser = ydlidar.CYdLidar();

Definition at line 11 of file ydlidar_test.py.

37.18.1.2 ydlidar_test.port = "/dev/ydlidar"

Definition at line 8 of file ydlidar_test.py.

37.18.1.3 ydlidar_test.ports = ydlidar.lidarPortList();

Definition at line 7 of file ydlidar_test.py.

37.18.1.4 ydlidar_test.r = laser.doProcessSimple(scan);

Definition at line 25 of file ydlidar_test.py.

37.18.1.5 ydlidar_test.ret = laser.initialize();

Definition at line 20 of file ydlidar_test.py.

37.18.1.6 ydlidar_test.scan = ydlidar.LaserScan();

Definition at line 23 of file ydlidar_test.py.

Chapter 38

Class Documentation

38.1 _dataFrame Struct Reference

UDP Data format.

`#include <ydlidar_protocol.h>`

Public Attributes

- uint16_t frameHead
- uint8_t deviceType
- uint8_t frameType
- uint8_t dataIndex
- uint8_t frameIndex
- uint32_t timestamp
- uint8_t headFrameFlag
- uint8_t dataFormat
- uint8_t disScale
- uint32_t startAngle
- uint32_t dataNum
- uint32_t frameCrc
- uint8_t frameBuf [2048]

38.1.1 Detailed Description

UDP Data format.

Definition at line 268 of file ydlidar_protocol.h.

38.1.2 Member Data Documentation

38.1.2.1 uint8_t _dataFrame::dataFormat

Definition at line 276 of file ydlidar_protocol.h.

38.1.2.2 uint8_t _dataFrame::dataIndex

Definition at line 272 of file ydlidar_protocol.h.

38.1.2.3 uint32_t _dataFrame::dataNum

Definition at line 279 of file ydlidar_protocol.h.

38.1.2.4 uint8_t _dataFrame::deviceType

Definition at line 270 of file ydlidar_protocol.h.

38.1.2.5 uint8_t _dataFrame::disScale

Definition at line 277 of file ydlidar_protocol.h.

38.1.2.6 uint8_t _dataFrame::frameBuf[2048]

Definition at line 281 of file ydlidar_protocol.h.

38.1.2.7 uint32_t _dataFrame::frameCrc

Definition at line 280 of file ydlidar_protocol.h.

38.1.2.8 uint16_t _dataFrame::frameHead

Definition at line 269 of file ydlidar_protocol.h.

38.1.2.9 uint8_t _dataFrame::frameIndex

Definition at line 273 of file ydlidar_protocol.h.

38.1.2.10 uint8_t _dataFrame::frameType

Definition at line 271 of file ydlidar_protocol.h.

38.1.2.11 uint8_t _dataFrame::headFrameFlag

Definition at line 275 of file ydlidar_protocol.h.

38.1.2.12 uint32_t _dataFrame::startAngle

Definition at line 278 of file ydlidar_protocol.h.

38.1.2.13 uint32_t _dataFrame::timestamp

Definition at line 274 of file ydlidar_protocol.h.

The documentation for this struct was generated from the following file:

- core/common/ydlidar_protocol.h

38.2 _lidarConfig Struct Reference

`#include <ydlidar_protocol.h>`

Public Attributes

- int laser_en
  Scanning enable.
- int motor_en
  rotate enable.
- int motor_rpm
  motor RPM.
- int fov_start
  start FOV angle.
- int fov_end
  end FOV angle.
- int trans_sel
  data receive interface, USB or Ethernet.
- char dataRecvIp [16]
  data receive IP.
- int dataRecvPort
  data receive PORT.
- int dhcp_en
  device network config, HDCP or Manual.
- char deviceIp [16]
  device IP.
- char deviceNetmask [16]
  device netmask.
- char deviceGatewayIp [16]
  device gateway ip.
- int laserScanFrequency
- int correction_angle
  correction_angle
- int correction_distance
  correction_distance

38.2.1 Detailed Description

Definition at line 290 of file ydlidar_protocol.h.

38.2.2 Member Data Documentation

38.2.2.1 int _lidarConfig::correction_angle

correction_angle

Definition at line 357 of file ydlidar_protocol.h.

38.2.2.2 int _lidarConfig::correction_distance

correction_distance

Definition at line 362 of file ydlidar_protocol.h.

38.2.2.3 char _lidarConfig::dataRecvIp[16]

data receive IP.

Definition at line 324 of file ydlidar_protocol.h.

38.2.2.4 int _lidarConfig::dataRecvPort

data receive PORT.

Definition at line 329 of file ydlidar_protocol.h.

38.2.2.5 char _lidarConfig::deviceGatewayIp[16]

device gateway ip.

Definition at line 349 of file ydlidar_protocol.h.

38.2.2.6 char _lidarConfig::deviceIp[16]

device IP.

Definition at line 339 of file ydlidar_protocol.h.

38.2.2.7 char _lidarConfig::deviceNetmask[16]

device netmask.

Definition at line 344 of file ydlidar_protocol.h.

38.2.2.8 int _lidarConfig::dhcp_en

device network config, HDCP or Manual.

Definition at line 334 of file ydlidar_protocol.h.

38.2.2.9 int _lidarConfig::fov_end

end FOV angle.

Definition at line 314 of file ydlidar_protocol.h.

38.2.2.10 int _lidarConfig::fov_start

start FOV angle.

Definition at line 309 of file ydlidar_protocol.h.

38.2.2.11 int _lidarConfig::laser_en

Scanning enable.

Definition at line 294 of file ydlidar_protocol.h.

38.2.2.12 int _lidarConfig::laserScanFrequency

Definition at line 351 of file ydlidar_protocol.h.

38.2.2.13 int _lidarConfig::motor_en

rotate enable.

Definition at line 299 of file ydlidar_protocol.h.

38.2.2.14 int _lidarConfig::motor_rpm

motor RPM.

Definition at line 304 of file ydlidar_protocol.h.

38.2.2.15 int _lidarConfig::trans_sel

data receive interface, USB or Ethernet.

Definition at line 319 of file ydlidar_protocol.h.

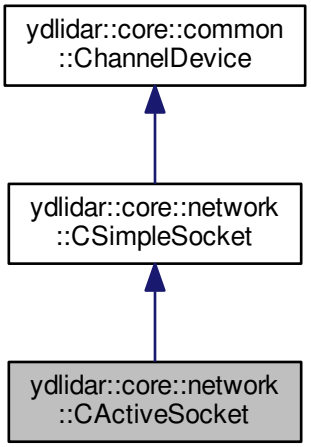
The documentation for this struct was generated from the following file:

- core/common/ydlidar_protocol.h

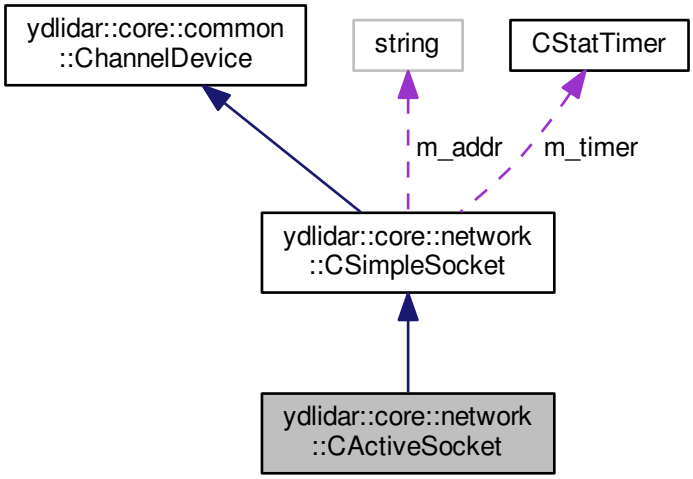
38.3 ydlidar::core::network::CActiveSocket Class Reference

`#include <ActiveSocket.h>`

Inheritance diagram for ydlidar::core::network::CActiveSocket:



Collaboration diagram for ydlidar::core::network::CActiveSocket:



Public Member Functions

- CActiveSocket (CSocketType type=SocketTypeTcp)
- virtual ~CActiveSocket ()
- virtual bool Open (const char *pAddr, uint16_t nPort)

Friends

- class CPassiveSocket

Additional Inherited Members

38.3.1 Detailed Description

Provides a platform independent class to create an active socket. An active socket is used to create a socket which connects to a server. This type of object would be used when an application needs to send/receive data from a server.

Definition at line 58 of file ActiveSocket.h.

38.3.2 Constructor & Destructor Documentation

38.3.2.1 CActiveSocket::CActiveSocket (CSocketType *type* = SocketTypeTcp) [explicit]

Definition at line 51 of file ActiveSocket.cpp.

38.3.2.2 virtual ydlidar::core::network::CActiveSocket::~CActiveSocket ( ) [inline], [virtual]

Definition at line 63 of file ActiveSocket.h.

38.3.3 Member Function Documentation

38.3.3.1 bool CActiveSocket::Open (const char * *pAddr*, uint16_t *nPort*) [virtual]

Established a connection to the address specified by pAddr. Connection-based protocol sockets (CSocket::SocketTypeTcp) may successfully call open() only once, however; connectionless protocol sockets (CSocket::SocketTypeUdp) may use Open() multiple times to change their association.

Parameters

pAddr	specifies the destination address to connect.
nPort	specifies the destination port.

Returns

true if successful connection made, otherwise false.

Reimplemented from ydlidar::core::network::CSimpleSocket.

Definition at line 249 of file ActiveSocket.cpp.

38.3.4 Friends And Related Function Documentation

38.3.4.1 friend class CPassiveSocket [friend]

Definition at line 60 of file ActiveSocket.h.

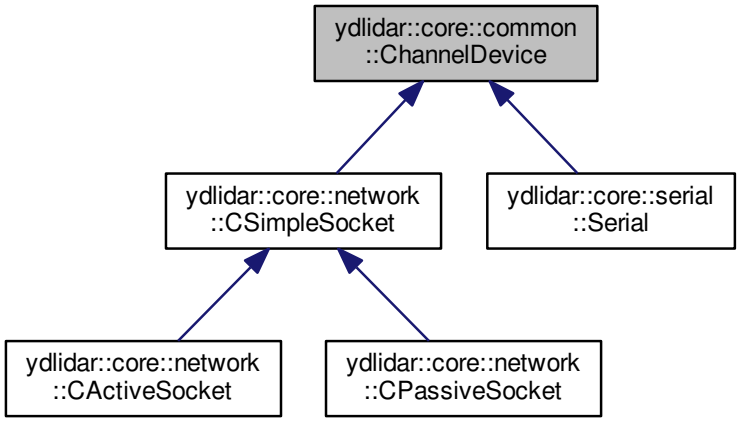
The documentation for this class was generated from the following files:

- core/network/ActiveSocket.h
- core/network/ActiveSocket.cpp

38.4 ydlidar::core::common::ChannelDevice Class Reference

`#include <ChannelDevice.h>`

Inheritance diagram for ydlidar::core::common::ChannelDevice:



Public Member Functions

- ChannelDevice ()
- virtual ~ChannelDevice ()
- virtual bool bindport (const char *, uint32_t)

  bind device port
- virtual bool open ()=0

  open device
- virtual bool isOpen ()=0

  Whether is open.
- virtual void closePort ()=0

  close serial port or network
- virtual size_t available ()=0

  Return the number of characters in the buffer.
- virtual void flush ()=0

  Flush the input and output buffers.
- virtual int waitfordata (size_t data_count, uint32_t timeout=-1, size_t *returned_size=NULL)=0

  Block until there is serial or network data to read or read_timeout_constant number of milliseconds have elapsed. The return value is greater than zero when the function exits with the serial port or network buffer is greater than or equal to data_count, false otherwise(due to timeout or select interruption).
- virtual std::string readSize (size_t size=1)=0
- virtual size_t writeData (const uint8_t *data, size_t size)=0
- virtual size_t readData (uint8_t *data, size_t size)=0
- virtual bool setDTR (bool level=true)

  Set the DTR handshaking line to the given level.
- virtual int getByteTime ()

  Returns the singal byte time.
- virtual const char * DescribeError ()

  Returns a human-readable description of the given error code or the last error code of a socket or serial port.

38.4.1 Detailed Description

Definition at line 7 of file ChannelDevice.h.

38.4.2 Constructor & Destructor Documentation

38.4.2.1 ydlidar::core::common::ChannelDevice::ChannelDevice ( ) [inline]

Definition at line 9 of file ChannelDevice.h.

38.4.2.2 virtual ydlidar::core::common::ChannelDevice::~ChannelDevice ( ) [inline], [virtual]

Definition at line 10 of file ChannelDevice.h.

38.4.3 Member Function Documentation

38.4.3.1 virtual size_t ydlidar::core::common::ChannelDevice::available ( ) [pure virtual]

Return the number of characters in the buffer.

Returns

Implemented in ydlidar::core::network::CSimpleSocket, and ydlidar::core::serial::Serial.

38.4.3.2 virtual bool ydlidar::core::common::ChannelDevice::bindport (const char *, uint32_t) [inline], [virtual]

bind device port

Returns

true if successfully bind, otherwise false

Reimplemented in ydlidar::core::network::CSimpleSocket.

Definition at line 15 of file ChannelDevice.h.

38.4.3.3 virtual void ydlidar::core::common::ChannelDevice::closePort ( ) [pure virtual]

close serial port or network

Implemented in ydlidar::core::network::CSimpleSocket, and ydlidar::core::serial::Serial.

38.4.3.4 virtual const char* ydlidar::core::common::ChannelDevice::DescribeError ( ) [inline], [virtual]

Returns a human-readable description of the given error code or the last error code of a socket or serial port.

Returns

error information

Reimplemented in ydlidar::core::serial::Serial, and ydlidar::core::network::CSimpleSocket.

Definition at line 127 of file ChannelDevice.h.

38.4.3.5 virtual void ydlidar::core::common::ChannelDevice::flush ( ) [pure virtual]

Flush the input and output buffers.

Implemented in ydlidar::core::serial::Serial, and ydlidar::core::network::CSimpleSocket.

38.4.3.6 virtual int ydlidar::core::common::ChannelDevice::getByteTime ( ) [inline], [virtual]

Returns the singal byte time.

Returns

one byte transfer time

Reimplemented in ydlidar::core::serial::Serial.

Definition at line 118 of file ChannelDevice.h.

38.4.3.7 virtual bool ydlidar::core::common::ChannelDevice::isOpen ( ) [pure virtual]

Whether is open.

Returns

true if already open, otherwise false

Implemented in ydlidar::core::network::CSimpleSocket, and ydlidar::core::serial::Serial.

38.4.3.8 virtual bool ydlidar::core::common::ChannelDevice::open ( ) [pure virtual]

open device

Returns

true if successfully open, otherwise false

Implemented in ydlidar::core::network::CSimpleSocket, and ydlidar::core::serial::Serial.

38.4.3.9 virtual size_t ydlidar::core::common::ChannelDevice::readData (uint8_t * *data*, size_t *size*) [pure virtual]

Read a given amount of bytes from the serial port or network into a given buffer.

The read function will return in one of three cases:

- The number of requested bytes was read.
  - In this case the number of bytes requested will match the size_t returned by read.
- A timeout occurred, in this case the number of bytes read will not match the amount requested, but no exception will be thrown. One of two possible timeouts occurred:
  - The inter byte timeout expired, this means that number of milliseconds elapsed between receiving bytes from the serial port exceeded the inter byte timeout.
  - The total timeout expired, which is calculated by multiplying the read timeout multiplier by the number of requested bytes and then added to the read timeout constant. If that total number of milliseconds elapses after the initial call to read a timeout will occur.
- An exception occurred, in this case an actual exception will be thrown.

Parameters

buffer	An uint8_t array of at least the requested size.
size	A size_t defining how many bytes to be read.

Returns

A size_t representing the number of bytes read as a result of the call to read.

Implemented in ydlidar::core::network::CSimpleSocket, and ydlidar::core::serial::Serial.

38.4.3.10 virtual std::string ydlidar::core::common::ChannelDevice::readSize (size_t *size* = 1) [pure virtual]

Read a given amount of bytes from the serial port or network and return a string containing the data.

Parameters

size	A size_t defining how many bytes to be read.

Returns

A std::string containing the data read from the port.

Implemented in ydlidar::core::network::CSimpleSocket, and ydlidar::core::serial::Serial.

38.4.3.11 virtual bool ydlidar::core::common::ChannelDevice::setDTR (bool *level* = true) [inline], [virtual]

Set the DTR handshaking line to the given level.

Parameters

level	Defaults to true.

Reimplemented in ydlidar::core::serial::Serial.

Definition at line 111 of file ChannelDevice.h.

38.4.3.12 virtual int ydlidar::core::common::ChannelDevice::waitfordata (size_t *data_count*, uint32_t *timeout* = −1, size_t * *returned_size* = NULL) [pure virtual]

Block until there is serial or network data to read or read_timeout_constant number of milliseconds have elapsed. The return value is greater than zero when the function exits with the serial port or network buffer is greater than or equal to data_count, false otherwise(due to timeout or select interruption).

Parameters

data_count	A size_t that indicates how many bytes should be wait from the given serial port or network buffer.
timeout	waiting timeout time
returned_size	if it is not NULL, the actual number of bytes will be returned.

Returns

A size_t representing the number of bytes wait as a result of the call to wait.

Implemented in ydlidar::core::network::CSimpleSocket, and ydlidar::core::serial::Serial.

38.4.3.13 virtual size_t ydlidar::core::common::ChannelDevice::writeData (const uint8_t * *data*, size_t *size*) [pure virtual]

Write a string to the serial port or network.

Parameters

data	A const reference containing the data to be written to the serial port.
size	A size_t that indicates how many bytes should be written from the given data buffer.

Returns

A size_t representing the number of bytes actually written to the serial port.

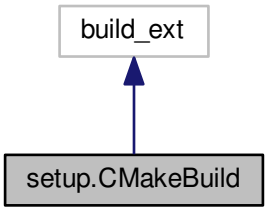
Implemented in ydlidar::core::network::CSimpleSocket, and ydlidar::core::serial::Serial.

The documentation for this class was generated from the following file:

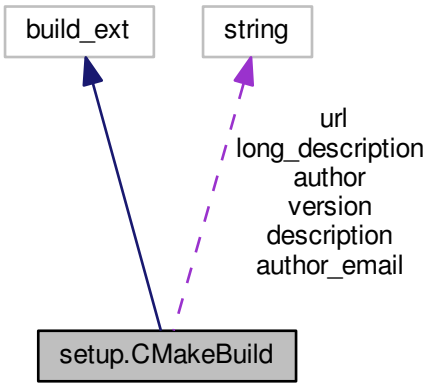
- core/common/ChannelDevice.h

38.5 setup.CMakeBuild Class Reference

Inheritance diagram for setup.CMakeBuild:



Collaboration diagram for setup.CMakeBuild:



Public Member Functions

- def run (self)
- def clone (self)
- def build_extension (self, ext)

Static Public Attributes

- name
- string version = '1.0.0'
- string author = 'Tony'
- string author_email = 'chushuifurong618@eaibot.com'

- string url = 'https://github.com/YDLIDAR/YDLidar-SDK'
- string description = 'YDLIDAR python SDK'
- string long_description = ''
- list ext_modules = [CMakeExtension('ydlidar')],
- cmdclass = dict(build_ext=CMakeBuild),
- bool zip_safe = False

38.5.1 Detailed Description

Definition at line 22 of file setup.py.

38.5.2 Member Function Documentation

38.5.2.1 def setup.CMakeBuild.build_extension (*self, ext*)

Definition at line 43 of file setup.py.

38.5.2.2 def setup.CMakeBuild.clone (*self*)

Definition at line 38 of file setup.py.

38.5.2.3 def setup.CMakeBuild.run (*self*)

Definition at line 23 of file setup.py.

38.5.3 Member Data Documentation

38.5.3.1 string setup.CMakeBuild.author = 'Tony' [static]

Definition at line 74 of file setup.py.

38.5.3.2 string setup.CMakeBuild.author_email = 'chushuifurong618@eaibot.com' [static]

Definition at line 75 of file setup.py.

38.5.3.3 setup.CMakeBuild.cmdclass = dict(build_ext=CMakeBuild), [static]

Definition at line 80 of file setup.py.

38.5.3.4 string setup.CMakeBuild.description = 'YDLIDAR python SDK' [static]

Definition at line 77 of file setup.py.

38.5.3.5 list setup.CMakeBuild.ext_modules = [CMakeExtension('ydlidar')], [static]

Definition at line 79 of file setup.py.

38.5.3.6 string setup.CMakeBuild.long_description = '' [static]

Definition at line 78 of file setup.py.

38.5.3.7 setup.CMakeBuild.name [static]

Definition at line 72 of file setup.py.

38.5.3.8 string setup.CMakeBuild.url = 'https://github.com/YDLIDAR/YDLidar-SDK' [static]

Definition at line 76 of file setup.py.

38.5.3.9 string setup.CMakeBuild.version = '1.0.0' [static]

Definition at line 73 of file setup.py.

38.5.3.10 bool setup.CMakeBuild.zip_safe = False [static]

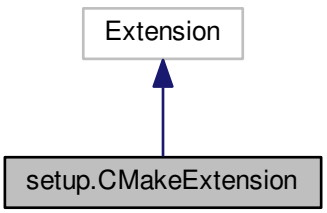
Definition at line 81 of file setup.py.

The documentation for this class was generated from the following file:

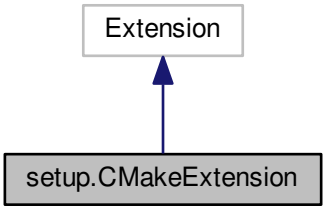
- setup.py

38.6 setup.CMakeExtension Class Reference

Inheritance diagram for setup.CMakeExtension:



Collaboration diagram for setup.CMakeExtension:



Public Member Functions

- def __init__ (self, name, sourcedir='')

Public Attributes

- sourcedir

38.6.1 Detailed Description

Definition at line 16 of file setup.py.

38.6.2 Constructor & Destructor Documentation

38.6.2.1 def setup.CMakeExtension.__init__ (*self, name, sourcedir = ''*)

Definition at line 17 of file setup.py.

38.6.3 Member Data Documentation

38.6.3.1 setup.CMakeExtension.sourcedir

Definition at line 19 of file setup.py.

The documentation for this class was generated from the following file:

- setup.py

38.7 cmd_packet Struct Reference

LiDAR request command packet.

`#include <ydlidar_protocol.h>`

Public Attributes

- uint8_t syncByte
- uint8_t cmd_flag
- uint8_t size
- uint8_t data

38.7.1 Detailed Description

LiDAR request command packet.

Definition at line 245 of file ydlidar_protocol.h.

38.7.2 Member Data Documentation

38.7.2.1 uint8_t cmd_packet::cmd_flag

Definition at line 247 of file ydlidar_protocol.h.

38.7.2.2 uint8_t cmd_packet::data

Definition at line 249 of file ydlidar_protocol.h.

38.7.2.3 uint8_t cmd_packet::size

Definition at line 248 of file ydlidar_protocol.h.

38.7.2.4 uint8_t cmd_packet::syncByte

Definition at line 246 of file ydlidar_protocol.h.

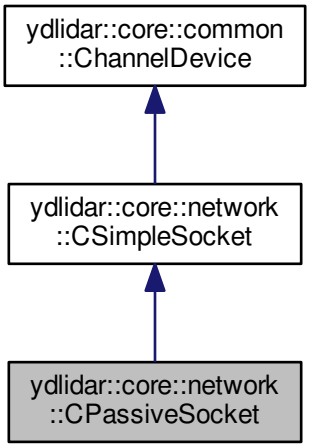
The documentation for this struct was generated from the following file:

- core/common/ydlidar_protocol.h

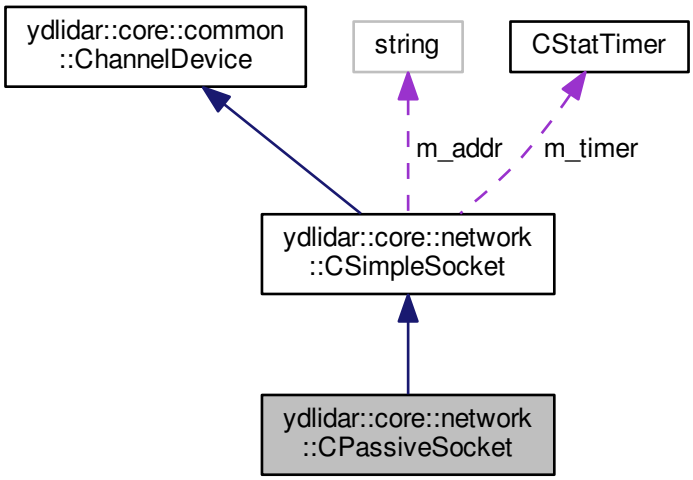
38.8 ydlidar::core::network::CPassiveSocket Class Reference

`#include <PassiveSocket.h>`

Inheritance diagram for ydlidar::core::network::CPassiveSocket:



Collaboration diagram for ydlidar::core::network::CPassiveSocket:



Public Member Functions

- CPassiveSocket (CSocketType type=SocketTypeTcp)
- virtual ~CPassiveSocket ()
- virtual CActiveSocket * Accept (void)
- bool BindMulticast (const char *pInterface, const char *pGroup, uint16_t nPort)
- virtual bool Listen (const char *pAddr, uint16_t nPort, int32_t nConnectionBacklog=30000)
- virtual int32_t Send (const uint8_t *pBuf, size_t bytesToSend)

Additional Inherited Members

38.8.1 Detailed Description

Definition at line 60 of file PassiveSocket.h.

38.8.2 Constructor & Destructor Documentation

38.8.2.1 CPassiveSocket::CPassiveSocket (CSocketType *type* = SocketTypeTcp) [explicit]

Definition at line 51 of file PassiveSocket.cpp.

38.8.2.2 virtual ydlidar::core::network::CPassiveSocket::~CPassiveSocket ( ) [inline], [virtual]

Definition at line 63 of file PassiveSocket.h.

38.8.3 Member Function Documentation

38.8.3.1 CActiveSocket * CPassiveSocket::Accept (void) [virtual]

Extracts the first connection request on the queue of pending connections and creates a newly connected socket. Used with CSocketType CSimpleSocket::SocketTypeTcp. It is the responsibility of the caller to delete the returned object when finished.

Returns

if successful a pointer to a newly created CActiveSocket object will be returned and the internal error condition of the CPassiveSocket object will be CPassiveSocket::SocketSuccess. If an error condition was encountered the NULL will be returned and one of the following error conditions will be set: CPassiveSocket::SocketEwouldblock, CPassiveSocket::SocketInvalidSocket, CPassiveSocket::SocketConnectionAborted, CPassiveSocket::SocketInterrupted CPassiveSocket::SocketProtocolError, CPassiveSocket::SocketFirewallError

Definition at line 209 of file PassiveSocket.cpp.

38.8.3.2 bool CPassiveSocket::BindMulticast (const char * *pInterface*, const char * *pGroup*, uint16_t *nPort*)

Bind to a multicast group on a specified interface, multicast group, and port

Parameters

pInterface	- interface on which to bind.
pGroup	- multicast group address to bind.
nPort	- port on which multicast

Returns

true if able to bind to interface and multicast group. If not successful, the false is returned and one of the following error condiitions will be set: CPassiveSocket::SocketAddressInUse, CPassiveSocket::SocketProtocolError, CPassiveSocket::SocketInvalidSocket. The following socket errors are for Linux/Unix derived systems only: CPassiveSocket::SocketInvalidSocketBuffer

Definition at line 54 of file PassiveSocket.cpp.

38.8.3.3 bool CPassiveSocket::Listen (const char * *pAddr*, uint16_t *nPort*, int32_t *nConnectionBacklog* = 30000) [virtual]

Create a listening socket at local ip address 'x.x.x.x' or 'localhost' if pAddr is NULL on port nPort.

Parameters

pAddr	specifies the IP address on which to listen.
nPort	specifies the port on which to listen.
nConnectionBacklog	specifies connection queue backlog (default 30,000)

Returns

true if a listening socket was created. If not successful, the false is returned and one of the following error conditions will be set: CPassiveSocket::SocketAddressInUse, CPassiveSocket::SocketProtocolError, CPassiveSocket::SocketInvalidSocket. The following socket errors are for Linux/Unix derived systems only: CPassiveSocket::SocketInvalidSocketBuffer

Definition at line 130 of file PassiveSocket.cpp.

38.8.3.4 int32_t CPassiveSocket::Send (const uint8_t * *pBuf*, size_t *bytesToSend*) [virtual]

Attempts to send a block of data on an established connection.

Parameters

pBuf	block of data to be sent.
bytesToSend	size of data block to be sent.

Returns

number of bytes actually sent, return of zero means the connection has been shutdown on the other side, and a return of -1 means that an error has occurred. If an error was signaled then one of the following error codes

will be set: CPassiveSocket::SocketInvalidSocket, CPassiveSocket::SocketEwouldblock, SimpleSocket↩ ::SocketConnectionReset, CPassiveSocket::SocketInvalidSocketBuffer, CPassiveSocket::SocketInterrupted, CPassiveSocket::SocketProtocolError, CPassiveSocket::SocketNotconnected
Note: This function is used only for a socket of type CSimpleSocket::SocketTypeUdp

Reimplemented from ydlidar::core::network::CSimpleSocket.

Definition at line 279 of file PassiveSocket.cpp.

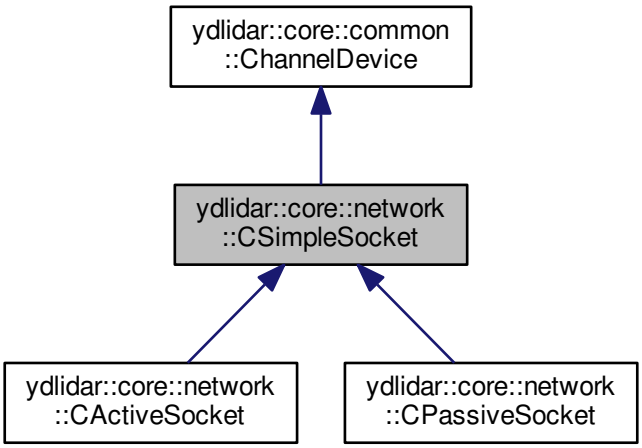
The documentation for this class was generated from the following files:

- core/network/PassiveSocket.h
- core/network/PassiveSocket.cpp

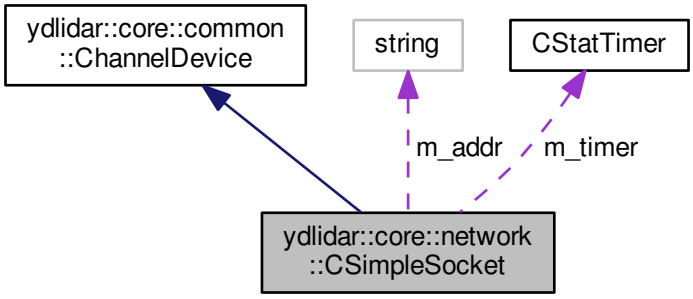
38.9 ydlidar::core::network::CSimpleSocket Class Reference

```
#include <SimpleSocket.h>
```

Inheritance diagram for ydlidar::core::network::CSimpleSocket:



Collaboration diagram for ydlidar::core::network::CSimpleSocket:



Public Types

- enum CShutdownMode { Receives = SHUT_RD, Sends = SHUT_WR, Both = SHUT_RDWR }

  Defines the three possible states for shuting down a socket.
- enum CSocketType {
  SocketTypeInvalid = 0, SocketTypeTcp, SocketTypeUdp, SocketTypeTcp6,
  SocketTypeUdp6, SocketTypeRaw }

  Defines the socket types defined by CSimpleSocket class.
- enum CSocketError {
  SocketError = -1, SocketSuccess = 0, SocketInvalidSocket, SocketInvalidAddress,
  SocketInvalidPort, SocketConnectionRefused, SocketTimedout, SocketEwouldblock,
  SocketNotconnected, SocketEinprogress, SocketInterrupted, SocketConnectionAborted,
  SocketProtocolError, SocketFirewallError, SocketInvalidSocketBuffer, SocketConnectionReset,
  SocketAddressInUse, SocketInvalidPointer, SocketEunknown }

  Defines all error codes handled by the CSimpleSocket class.

Public Member Functions

- CSimpleSocket (CSocketType type=SocketTypeTcp)
- CSimpleSocket (CSimpleSocket &socket)
- virtual ~CSimpleSocket ()
- virtual bool Initialize (void)
- virtual bool Close (void)
- virtual bool Shutdown (CShutdownMode nShutdown)
- virtual bool Select (void)
- virtual bool Select (int32_t nTimeoutSec, int32_t nTimeoutUSec)
- virtual int WaitForData (size_t data_count, uint32_t timeout, size_t *returned_size)
- virtual bool IsSocketValid (void)
- void TranslateSocketError (void)
- virtual const char * DescribeError ()

  DescribeError.
- virtual int32_t Receive (int32_t nMaxBytes=1, uint8_t *pBuffer=0)
- virtual int32_t Send (const uint8_t *pBuf, size_t bytesToSend)
- virtual int32_t Send (const struct iovec *sendVector, int32_t nNumItems)

- virtual int32_t SendFile (int32_t nOutFd, int32_t nInFd, off_t *pOffset, int32_t nCount)
- bool IsNonblocking (void)
- bool SetBlocking (void)
- bool SetNonblocking (void)
- uint8_t * GetData (void)
- int32_t GetBytesReceived (void)
- int32_t GetBytesSent (void)
- bool SetOptionLinger (bool bEnable, uint16_t nTime)
- bool SetOptionReuseAddr ()
- int32_t GetConnectTimeoutSec (void)
- int32_t GetConnectTimeoutUSec (void)
- void SetConnectTimeout (int32_t nConnectTimeoutSec, int32_t nConnectTimeoutUsec=0)
- int32_t GetReceiveTimeoutSec (void)
- int32_t GetReceiveTimeoutUSec (void)
- bool SetReceiveTimeout (int32_t nRecvTimeoutSec, int32_t nRecvTimeoutUsec=0)
- bool SetMulticast (bool bEnable, uint8_t multicastTTL=1)
- bool GetMulticast ()
- bool BindInterface (const char *pInterface)
- int32_t GetSendTimeoutSec (void)
- int32_t GetSendTimeoutUSec (void)
- bool SetSendTimeout (int32_t nSendTimeoutSec, int32_t nSendTimeoutUsec=0)
- CSocketError GetSocketError (void)
- uint32_t GetTotalTimeMs ()
- uint64_t GetTotalTimeUsec ()
- int GetSocketDscp (void)
- bool SetSocketDscp (int nDscp)
- SOCKET GetSocketDescriptor ()
- CSocketType GetSocketType ()
- void SetSocketType (const CSocketType &type)

  set socket descriptor
- const char * GetClientAddr ()
- uint16_t GetClientPort ()
- const char * GetServerAddr ()
- uint16_t GetServerPort ()
- uint32_t GetReceiveWindowSize ()
- uint32_t GetSendWindowSize ()
- uint32_t SetReceiveWindowSize (uint32_t nWindowSize)
- uint32_t SetSendWindowSize (uint32_t nWindowSize)
- bool DisableNagleAlgoritm ()
- bool EnableNagleAlgoritm ()
- virtual bool Open (const char *pAddr, uint16_t nPort)
- virtual bool bindport (const char *, uint32_t)

  bindport
- virtual bool open ()

  open
- virtual bool isOpen ()

  isOpen
- virtual void closePort ()

  closePort
- virtual void flush ()

  flush
- virtual size_t available ()

  available

- virtual std::string readSize (size_t size=1)

  readSize
- virtual int waitfordata (size_t data_count, uint32_t timeout=-1, size_t *returned_size=NULL)

  waitfordata
- virtual size_t writeData (const uint8_t *data, size_t size)

  writeData
- virtual size_t readData (uint8_t *data, size_t size)

  readData

Static Public Member Functions

- static void WSACleanUp ()
- static const char * DescribeError (CSocketError err)

Protected Member Functions

- void SetSocketError (CSimpleSocket::CSocketError error)
- void SetSocketHandle (SOCKET socket)
- bool Flush ()

Protected Attributes

- SOCKET m_socket
- CSocketError m_socketErrno

  socket handle
- uint8_t * m_pBuffer

  number of last error
- int32_t m_nBufferSize

  internal send/receive buffer
- int32_t m_nSocketDomain

  size of internal send/receive buffer
- CSocketType m_nSocketType

  socket type PF_INET, PF_INET6
- int32_t m_nBytesReceived

  socket type - UDP, TCP or RAW
- int32_t m_nBytesSent

  number of bytes received
- uint32_t m_nFlags

  number of bytes sent
- bool m_bIsBlocking

  socket flags
- bool m_bIsMulticast

  is socket blocking
- struct timeval m_stConnectTimeout

  is the UDP socket multicast;
- struct timeval m_stRecvTimeout

  connection timeout
- struct timeval m_stSendTimeout

  receive timeout

- struct sockaddr_in m_stServerSockaddr

  send timeout
- struct sockaddr_in m_stClientSockaddr

  server address
- struct sockaddr_in m_stMulticastGroup

  client address
- struct linger m_stLinger

  multicast group to bind to
- CStatTimer m_timer

  linger flag
- fd_set m_writeFds

  internal statistics.
- fd_set m_readFds

  write file descriptor set
- fd_set m_errorFds

  read file descriptor set
- std::string m_addr

  error file descriptor set
- uint32_t m_port
- bool m_open

38.9.1 Detailed Description

Provides a platform independent class to for socket development. This class is designed to abstract socket communication development in a platform independent manner.

- Socket types
  1. CActiveSocket Class
  2. CPassiveSocket Class

Definition at line 116 of file SimpleSocket.h.

38.9.2 Member Enumeration Documentation

38.9.2.1 enum ydlidar::core::network::CSimpleSocket::CShutdownMode

Defines the three possible states for shuting down a socket.

Enumerator

Receives Shutdown passive socket.

Sends Shutdown active socket.

Both Shutdown both active and passive sockets.

Definition at line 119 of file SimpleSocket.h.

38.9.2.2 enum ydlidar::core::network::CSimpleSocket::CSocketError

Defines all error codes handled by the CSimpleSocket class.

Enumerator

SocketError Generic socket error translates to error below.

SocketSuccess No socket error.

SocketInvalidSocket Invalid socket handle.

SocketInvalidAddress Invalid destination address specified.

SocketInvalidPort Invalid destination port specified.

SocketConnectionRefused No server is listening at remote address.

SocketTimedout Timed out while attempting operation.

SocketEwouldblock Operation would block if socket were blocking.

SocketNotconnected Currently not connected.

SocketEinprogress Socket is non-blocking and the connection cannot be completed immediately.

SocketInterrupted Call was interrupted by a signal that was caught before a valid connection arrived.

SocketConnectionAborted The connection has been aborted.

SocketProtocolError Invalid protocol for operation.

SocketFirewallError Firewall rules forbid connection.

SocketInvalidSocketBuffer The receive buffer point outside the process's address space.

SocketConnectionReset Connection was forcibly closed by the remote host.

SocketAddressInUse Address already in use.

SocketInvalidPointer Pointer type supplied as argument is invalid.

SocketEunknown Unknown error please report to mark@carrierlabs.com.

Definition at line 136 of file SimpleSocket.h.

38.9.2.3 enum ydlidar::core::network::CSimpleSocket::CSocketType

Defines the socket types defined by CSimpleSocket class.

Enumerator

SocketTypeInvalid Invalid socket type.

SocketTypeTcp Defines socket as TCP socket.

SocketTypeUdp Defines socket as UDP socket.

SocketTypeTcp6 Defines socket as IPv6 TCP socket.

SocketTypeUdp6 Defines socket as IPv6 UDP socket.

SocketTypeRaw Provides raw network protocol access.

Definition at line 126 of file SimpleSocket.h.

38.9.3 Constructor & Destructor Documentation

38.9.3.1 CSimpleSocket::CSimpleSocket (CSocketType *type* = SocketTypeTcp) [explicit]

Definition at line 51 of file SimpleSocket.cpp.

38.9.3.2 CSimpleSocket::CSimpleSocket (CSimpleSocket & *socket*) [explicit]

Definition at line 115 of file SimpleSocket.cpp.

38.9.3.3 virtual ydlidar::core::network::CSimpleSocket::~CSimpleSocket ( ) [inline], [virtual]

Definition at line 162 of file SimpleSocket.h.

38.9.4 Member Function Documentation

38.9.4.1 size_t CSimpleSocket::available ( ) [virtual]

available

Returns

Implements ydlidar::core::common::ChannelDevice.

Definition at line 209 of file SimpleSocket.cpp.

38.9.4.2 bool CSimpleSocket::BindInterface (const char * *pInterface*)

Bind socket to a specific interface when using multicast.

Returns

true if successfully bound to interface

Definition at line 306 of file SimpleSocket.cpp.

38.9.4.3 bool CSimpleSocket::bindport (const char * *addr*, uint32_t *port*) [virtual]

bindport

Returns

Reimplemented from ydlidar::core::common::ChannelDevice.

Definition at line 144 of file SimpleSocket.cpp.

38.9.4.4 bool CSimpleSocket::Close (void) [virtual]

Close socket

Returns

true if successfully closed otherwise returns false.

Definition at line 597 of file SimpleSocket.cpp.

38.9.4.5 void CSimpleSocket::closePort ( ) [virtual]

closePort

Implements ydlidar::core::common::ChannelDevice.

Definition at line 190 of file SimpleSocket.cpp.

38.9.4.6 const char * CSimpleSocket::DescribeError (CSocketError *err*) [static]

Returns a human-readable description of the given error code or the last error code of a socket

Definition at line 1270 of file SimpleSocket.cpp.

38.9.4.7 virtual const char* ydlidar::core::network::CSimpleSocket::DescribeError ( ) [inline], [virtual]

DescribeError.

Returns

Reimplemented from ydlidar::core::common::ChannelDevice.

Definition at line 232 of file SimpleSocket.h.

38.9.4.8 bool CSimpleSocket::DisableNagleAlgoritm ( )

Disable the Nagle algorithm (Set TCP_NODELAY to true)

Returns

false if failed to set socket option otherwise return true;

Definition at line 453 of file SimpleSocket.cpp.

38.9.4.9 bool CSimpleSocket::EnableNagleAlgoritm ( )

Enable the Nagle algorithm (Set TCP_NODELAY to false)

Returns

false if failed to set socket option otherwise return true;

Definition at line 475 of file SimpleSocket.cpp.

38.9.4.10 void CSimpleSocket::flush ( ) [virtual]

flush

Implements ydlidar::core::common::ChannelDevice.

Definition at line 196 of file SimpleSocket.cpp.

38.9.4.11 bool CSimpleSocket::Flush ( ) [protected]

Flush the socket descriptor owned by the object.

Returns

true data was successfully sent, else return false;

Definition at line 641 of file SimpleSocket.cpp.

38.9.4.12 int32_t ydlidar::core::network::CSimpleSocket::GetBytesReceived (void) [inline]

Returns the number of bytes received on the last call to CSocket::Receive().

Returns

number of bytes received.

Definition at line 304 of file SimpleSocket.h.

38.9.4.13 int32_t ydlidar::core::network::CSimpleSocket::GetBytesSent (void) [inline]

Returns the number of bytes sent on the last call to CSocket::Send().

Returns

number of bytes sent.

Definition at line 311 of file SimpleSocket.h.

38.9.4.14 const char* ydlidar::core::network::CSimpleSocket::GetClientAddr ( ) [inline]

Returns clients Internet host address as a string in standard numbers-and-dots notation.

Returns

NULL if invalid

Definition at line 480 of file SimpleSocket.h.

38.9.4.15 uint16_t ydlidar::core::network::CSimpleSocket::GetClientPort ( ) [inline]

Returns the port number on which the client is connected.

Returns

client port number.

Definition at line 486 of file SimpleSocket.h.

38.9.4.16 int32_t ydlidar::core::network::CSimpleSocket::GetConnectTimeoutSec (void) [inline]

Gets the timeout value that specifies the maximum number of seconds a call to CSimpleSocket::Open waits until it completes.

Returns

the length of time in seconds

Definition at line 341 of file SimpleSocket.h.

38.9.4.17 int32_t ydlidar::core::network::CSimpleSocket::GetConnectTimeoutUSec (void) [inline]

Gets the timeout value that specifies the maximum number of microseconds a call to CSimpleSocket::Open waits until it completes.

Returns

the length of time in microseconds

Definition at line 348 of file SimpleSocket.h.

38.9.4.18 uint8_t* ydlidar::core::network::CSimpleSocket::GetData (void) [inline]

Get a pointer to internal receive buffer. The user MUST not free this pointer when finished. This memory is managed internally by the CSocket class.

Returns

pointer to data if valid, else returns NULL.

Definition at line 297 of file SimpleSocket.h.

38.9.4.19 bool ydlidar::core::network::CSimpleSocket::GetMulticast ( ) [inline]

Return true if socket is multicast or false is socket is unicast

Returns

true if multicast is enabled

Definition at line 403 of file SimpleSocket.h.

38.9.4.20 int32_t ydlidar::core::network::CSimpleSocket::GetReceiveTimeoutSec (void) [inline]

Gets the timeout value that specifies the maximum number of seconds a a call to CSimpleSocket::Receive waits until it completes.

Returns

the length of time in seconds

Definition at line 371 of file SimpleSocket.h.

38.9.4.21 int32_t ydlidar::core::network::CSimpleSocket::GetReceiveTimeoutUSec (void) [inline]

Gets the timeout value that specifies the maximum number of microseconds a call to CSimpleSocket::Receive waits until it completes.

Returns

the length of time in microseconds

Definition at line 378 of file SimpleSocket.h.

38.9.4.22 uint32_t ydlidar::core::network::CSimpleSocket::GetReceiveWindowSize ( ) [inline]

Get the TCP receive buffer window size for the current socket object.

NOTE: Linux will set the receive buffer to twice the value passed.

Returns

zero on failure else the number of bytes of the TCP receive buffer window size if successful.

Definition at line 505 of file SimpleSocket.h.

38.9.4.23 int32_t ydlidar::core::network::CSimpleSocket::GetSendTimeoutSec (void) [inline]

Gets the timeout value that specifies the maximum number of seconds a a call to CSimpleSocket::Send waits until it completes.

Returns

the length of time in seconds

Definition at line 414 of file SimpleSocket.h.

38.9.4.24 int32_t ydlidar::core::network::CSimpleSocket::GetSendTimeoutUSec (void) [inline]

Gets the timeout value that specifies the maximum number of microseconds a call to CSimpleSocket::Send waits until it completes.

Returns

the length of time in microseconds

Definition at line 421 of file SimpleSocket.h.

38.9.4.25 uint32_t ydlidar::core::network::CSimpleSocket::GetSendWindowSize ( ) [inline]

Get the TCP send buffer window size for the current socket object.

NOTE: Linux will set the send buffer to twice the value passed.

Returns

zero on failure else the number of bytes of the TCP receive buffer window size if successful.

Definition at line 512 of file SimpleSocket.h.

38.9.4.26 const char* ydlidar::core::network::CSimpleSocket::GetServerAddr ( ) [inline]

Returns server Internet host address as a string in standard numbers-and-dots notation.

Returns

NULL if invalid

Definition at line 492 of file SimpleSocket.h.

38.9.4.27 uint16_t ydlidar::core::network::CSimpleSocket::GetServerPort ( ) [inline]

Returns the port number on which the server is connected.

Returns

server port number.

Definition at line 498 of file SimpleSocket.h.

38.9.4.28 SOCKET ydlidar::core::network::CSimpleSocket::GetSocketDescriptor ( ) [inline]

Return socket descriptor

Returns

socket descriptor which is a signed 32 bit integer.

Definition at line 463 of file SimpleSocket.h.

38.9.4.29 int32_t CSimpleSocket::GetSocketDscp (void)

Return Differentiated Services Code Point (DSCP) value currently set on the socket object.

Returns

DSCP for current socket object.

NOTE: Windows special notes http://support.microsoft.com/kb/248611.

Definition at line 381 of file SimpleSocket.cpp.

38.9.4.30 CSocketError ydlidar::core::network::CSimpleSocket::GetSocketError (void) [inline]

Returns the last error that occured for the instace of the CSimpleSocket instance. This method should be called immediately to retrieve the error code for the failing mehtod call.

Returns

last error that occured.

Definition at line 434 of file SimpleSocket.h.

38.9.4.31 CSocketType ydlidar::core::network::CSimpleSocket::GetSocketType ( ) [inline]

Return socket descriptor

Returns

socket descriptor which is a signed 32 bit integer.

Definition at line 469 of file SimpleSocket.h.

38.9.4.32 uint32_t ydlidar::core::network::CSimpleSocket::GetTotalTimeMs ( ) [inline]

Get the total time the of the last operation in milliseconds.

Returns

number of milliseconds of last operation.

Definition at line 440 of file SimpleSocket.h.

38.9.4.33 uint64_t ydlidar::core::network::CSimpleSocket::GetTotalTimeUsec ( ) [inline]

Get the total time the of the last operation in microseconds.

Returns

number of microseconds or last operation.

Definition at line 446 of file SimpleSocket.h.

38.9.4.34 bool CSimpleSocket::Initialize (void) [virtual]

Initialize instance of CSocket. This method MUST be called before an object can be used. Errors : CSocket::↩ SocketProtocolError, CSocket::SocketInvalidSocket,

Returns

true if properly initialized.

Definition at line 266 of file SimpleSocket.cpp.

38.9.4.35 bool ydlidar::core::network::CSimpleSocket::IsNonblocking (void) [inline]

Returns blocking/non-blocking state of socket.

Returns

true if the socket is non-blocking, else return false.

Definition at line 281 of file SimpleSocket.h.

38.9.4.36 bool CSimpleSocket::isOpen ( ) [virtual]

isOpen

Returns

Implements ydlidar::core::common::ChannelDevice.

Definition at line 186 of file SimpleSocket.cpp.

38.9.4.37 virtual bool ydlidar::core::network::CSimpleSocket::IsSocketValid (void) [inline], [virtual]

Does the current instance of the socket object contain a valid socket descriptor.

Returns

true if the socket object contains a valid socket descriptor.

Definition at line 216 of file SimpleSocket.h.

38.9.4.38 virtual bool ydlidar::core::network::CSimpleSocket::Open (const char * *pAddr*, uint16_t *nPort*) [inline], [virtual]

Reimplemented in ydlidar::core::network::CActiveSocket.

Definition at line 538 of file SimpleSocket.h.

38.9.4.39 bool CSimpleSocket::open ( ) [virtual]

open

Returns

Implements ydlidar::core::common::ChannelDevice.

Definition at line 161 of file SimpleSocket.cpp.

38.9.4.40 size_t CSimpleSocket::readData (uint8_t * *data*, size_t *size*) [virtual]

readData

Parameters

data	
size	

Returns

Implements ydlidar::core::common::ChannelDevice.

Definition at line 248 of file SimpleSocket.cpp.

38.9.4.41 std::string CSimpleSocket::readSize (size_t *size* = 1) [virtual]

readSize

Parameters

size	

Returns

Implements ydlidar::core::common::ChannelDevice.

Definition at line 225 of file SimpleSocket.cpp.

38.9.4.42 int32_t CSimpleSocket::Receive (int32_t *nMaxBytes* = 1, uint8_t * *pBuffer* = 0) [virtual]

Attempts to receive a block of data on an established connection.

Parameters

nMaxBytes	maximum number of bytes to receive.
pBuffer,memory	where to receive the data, NULL receives to internal buffer returned with GetData() Non-NULL receives directly there, but GetData() will return WRONG ptr!

Returns

number of bytes actually received.
of zero means the connection has been shutdown on the other side.
of -1 means that an error has occurred.

Definition at line 854 of file SimpleSocket.cpp.

38.9.4.43 virtual bool ydlidar::core::network::CSimpleSocket::Select (void) [inline], [virtual]

Examine the socket descriptor sets currently owned by the instance of the socket class (the readfds, writefds, and errorfds parameters) to see whether some of their descriptors are ready for reading, are ready for writing, or have an exceptional condition pending, respectively. Block until an event happens on the specified file descriptors.

Returns

true if socket has data ready, or false if not ready or timed out.

Definition at line 196 of file SimpleSocket.h.

38.9.4.44 bool CSimpleSocket::Select (int32_t *nTimeoutSec*, int32_t *nTimeoutUSec*) [virtual]

Examine the socket descriptor sets currently owned by the instance of the socket class (the readfds, writefds, and errorfds parameters) to see whether some of their descriptors are ready for reading, are ready for writing, or have an exceptional condition pending, respectively.

Parameters

nTimeoutSec	timeout in seconds for select.
nTimeoutUSec	timeout in micro seconds for select.

Returns

true if socket has data ready, or false if not ready or timed out.

Definition at line 1339 of file SimpleSocket.cpp.

38.9.4.45 int32_t CSimpleSocket::Send (const uint8_t * *pBuf*, size_t *bytesToSend*) [virtual]

Attempts to send a block of data on an established connection.

Parameters

pBuf	block of data to be sent.
bytesToSend	size of data block to be sent.

Returns

number of bytes actually sent.
of zero means the connection has been shutdown on the other side.
of -1 means that an error has occurred.

Reimplemented in ydlidar::core::network::CPassiveSocket.

Definition at line 497 of file SimpleSocket.cpp.

38.9.4.46 int32_t CSimpleSocket::Send (const struct iovec * *sendVector,* int32_t *nNumItems*) [virtual]

Attempts to send at most nNumItem blocks described by sendVector to the socket descriptor associated with the socket object.

Parameters

sendVector	pointer to an array of iovec structures
nNumItems	number of items in the vector to process **NOTE:** Buffers are processed in the order specified.

Returns

number of bytes actually sent, return of zero means the connection has been shutdown on the other side, and a return of -1 means that an error has occurred.

Definition at line 717 of file SimpleSocket.cpp.

38.9.4.47 int32_t CSimpleSocket::SendFile (int32_t *nOutFd,* int32_t *nInFd,* off_t * *pOffset,* int32_t *nCount*) [virtual]

Copies data between one file descriptor and another. On some systems this copying is done within the kernel, and thus is more efficient than the combination of CSimpleSocket::Send and CSimpleSocket::Receive, which would require transferring data to and from user space.
Note: This is available on all implementations, but the kernel implementation is only available on Unix type systems.

Parameters

nOutFd	descriptor opened for writing.
nInFd	descriptor opened for reading.
pOffset	from which to start reading data from input file.
nCount	number of bytes to copy between file descriptors.

Returns

number of bytes written to the out socket descriptor.

Definition at line 1079 of file SimpleSocket.cpp.

38.9.4.48 bool CSimpleSocket::SetBlocking (void)

Set the socket to blocking.

Returns

true if successful set to blocking, else return false;

Definition at line 1045 of file SimpleSocket.cpp.

38.9.4.49 void ydlidar::core::network::CSimpleSocket::SetConnectTimeout (int32_t *nConnectTimeoutSec,* int32_t *nConnectTimeoutUsec* = `0`) [inline]

Sets the timeout value that specifies the maximum amount of time a call to CSimpleSocket::Receive waits until it completes. Use the method CSimpleSocket::SetReceiveTimeout to specify the number of seconds to wait. If a call to CSimpleSocket::Receive has blocked for the specified length of time without receiving additional data, it returns with a partial count or CSimpleSocket::GetSocketError set to CSimpleSocket::SocketEwouldblock if no data were received.

Parameters

nConnectTimeoutSec	of timeout in seconds.
nConnectTimeoutUsec	of timeout in microseconds.

Returns

true if socket connection timeout was successfully set.

Definition at line 362 of file SimpleSocket.h.

38.9.4.50 bool CSimpleSocket::SetMulticast (bool *bEnable,* uint8_t *multicastTTL* = `1`)

Enable/disable multicast for a socket. This options is only valid for socket descriptors of type CSimpleSocket::SocketTypeUdp.

Returns

true if multicast was enabled or false if socket type is not CSimpleSocket::SocketTypeUdp and the error will be set to CSimpleSocket::SocketProtocolError

Definition at line 332 of file SimpleSocket.cpp.

38.9.4.51 bool CSimpleSocket::SetNonblocking (void)

Set the socket as non-blocking.

Returns

true if successful set to non-blocking, else return false;

Definition at line 1010 of file SimpleSocket.cpp.

38.9.4.52 bool CSimpleSocket::SetOptionLinger (bool *bEnable,* uint16_t *nTime*)

Controls the actions taken when CSimpleSocket::Close is executed on a socket object that has unsent data. The default value for this option is **off**.

- Following are the three possible scenarios.
  1. **bEnable** is false, CSimpleSocket::Close returns immediately, but any unset data is transmitted (after CSimpleSocket::Close returns)
  2. **bEnable** is true and **nTime** is zero, CSimpleSocket::Close return immediately and any unsent data is discarded.
  3. **bEnable** is true and **nTime** is nonzero, CSimpleSocket::Close does not return until all unsent data is transmitted (or the connection is Closed by the remote system).

Parameters

bEnable	true to enable option false to disable option.
nTime	time in seconds to linger.

Returns

true if option successfully set

Definition at line 830 of file SimpleSocket.cpp.

38.9.4.53 bool CSimpleSocket::SetOptionReuseAddr ( )

Tells the kernel that even if this port is busy (in the TIME_WAIT state), go ahead and reuse it anyway. If it is busy, but with another state, you will still get an address already in use error.

Returns

true if option successfully set

Definition at line 811 of file SimpleSocket.cpp.

38.9.4.54 bool CSimpleSocket::SetReceiveTimeout (int32_t *nRecvTimeoutSec,* int32_t *nRecvTimeoutUsec* = `0`)

Sets the timeout value that specifies the maximum amount of time a call to CSimpleSocket::Receive waits until it completes. Use the method CSimpleSocket::SetReceiveTimeout to specify the number of seconds to wait. If a call to CSimpleSocket::Receive has blocked for the specified length of time without receiving additional data, it returns with a partial count or CSimpleSocket::GetSocketError set to CSimpleSocket::SocketEwouldblock if no data were received.

Parameters

nRecvTimeoutSec	of timeout in seconds.
nRecvTimeoutUsec	of timeout in microseconds.

Returns

true if socket timeout was successfully set.

Definition at line 735 of file SimpleSocket.cpp.

38.9.4.55 uint32_t ydlidar::core::network::CSimpleSocket::SetReceiveWindowSize (uint32_t *nWindowSize*) `[inline]`

Set the TCP receive buffer window size for the current socket object.

NOTE: Linux will set the receive buffer to twice the value passed.

Returns

zero on failure else the number of bytes of the TCP send buffer window size if successful.

Definition at line 519 of file SimpleSocket.h.

38.9.4.56 bool CSimpleSocket::SetSendTimeout (int32_t *nSendTimeoutSec,* int32_t *nSendTimeoutUsec* = 0)

Gets the timeout value that specifies the maximum amount of time a call to CSimpleSocket::Send waits until it completes.

Returns

the length of time in seconds

Definition at line 773 of file SimpleSocket.cpp.

38.9.4.57 uint32_t ydlidar::core::network::CSimpleSocket::SetSendWindowSize (uint32_t *nWindowSize*) [inline]

Set the TCP send buffer window size for the current socket object.

NOTE: Linux will set the send buffer to twice the value passed.

Returns

zero on failure else the number of bytes of the TCP send buffer window size if successful.

Definition at line 526 of file SimpleSocket.h.

38.9.4.58 bool CSimpleSocket::SetSocketDscp (int *nDscp*)

Set Differentiated Services Code Point (DSCP) for socket object.

Parameters

nDscp	value of TOS setting which will be converted to DSCP

Returns

true if DSCP value was properly set

NOTE: Windows special notes http://support.microsoft.com/kb/248611.

Definition at line 358 of file SimpleSocket.cpp.

38.9.4.59 void ydlidar::core::network::CSimpleSocket::SetSocketError (CSimpleSocket::CSocketError *error*) [inline], [protected]

Set internal socket error to that specified error

Parameters

error	type of error

Definition at line 613 of file SimpleSocket.h.

38.9.4.60 void ydlidar::core::network::CSimpleSocket::SetSocketHandle (SOCKET *socket*) [inline],[protected]

Set object socket handle to that specified as parameter

Parameters

socket	value of socket descriptor

Definition at line 619 of file SimpleSocket.h.

38.9.4.61 void ydlidar::core::network::CSimpleSocket::SetSocketType (const CSocketType & *type*) [inline]

set socket descriptor

Definition at line 474 of file SimpleSocket.h.

38.9.4.62 bool CSimpleSocket::Shutdown (CShutdownMode *nShutdown*) [virtual]

Shutdown shut down socket send and receive operations CShutdownMode::Receives - Disables further receive operations. CShutdownMode::Sends - Disables further send operations. CShutdownBoth:: - Disables further send and receive operations.

Parameters

nShutdown	specifies the type of shutdown.

Returns

true if successfully shutdown otherwise returns false.

Definition at line 628 of file SimpleSocket.cpp.

38.9.4.63 void CSimpleSocket::TranslateSocketError (void)

Provides a standard error code for cross platform development by mapping the operating system error to an error defined by the CSocket class.

Definition at line 1116 of file SimpleSocket.cpp.

38.9.4.64 int CSimpleSocket::WaitForData (size_t *data_count*, uint32_t *timeout*, size_t * *returned_size*) [virtual]

Definition at line 1402 of file SimpleSocket.cpp.

38.9.4.65 int CSimpleSocket::waitfordata (size_t *data_count*, uint32_t *timeout* = -1, size_t * *returned_size* = NULL) [virtual]

waitfordata

Parameters

data_count	
timeout	
returned_size	

Returns

Implements ydlidar::core::common::ChannelDevice.

Definition at line 233 of file SimpleSocket.cpp.

38.9.4.66 size_t CSimpleSocket::writeData (const uint8_t * *data*, size_t *size*) [virtual]

writeData

Parameters

data	
size	

Returns

Implements ydlidar::core::common::ChannelDevice.

Definition at line 238 of file SimpleSocket.cpp.

38.9.4.67 void CSimpleSocket::WSACleanUp ( ) [static]

Definition at line 133 of file SimpleSocket.cpp.

38.9.5 Member Data Documentation

38.9.5.1 std::string ydlidar::core::network::CSimpleSocket::m_addr [protected]

error file descriptor set

Definition at line 680 of file SimpleSocket.h.

38.9.5.2 bool ydlidar::core::network::CSimpleSocket::m_blsBlocking [protected]

socket flags

Definition at line 663 of file SimpleSocket.h.

38.9.5.3 bool ydlidar::core::network::CSimpleSocket::m_blsMulticast [protected]

is socket blocking

Definition at line 664 of file SimpleSocket.h.

38.9.5.4 fd_set ydlidar::core::network::CSimpleSocket::m_errorFds [protected]

read file descriptor set

Definition at line 678 of file SimpleSocket.h.

38.9.5.5 int32_t ydlidar::core::network::CSimpleSocket::m_nBufferSize [protected]

internal send/receive buffer

Definition at line 657 of file SimpleSocket.h.

38.9.5.6 int32_t ydlidar::core::network::CSimpleSocket::m_nBytesReceived [protected]

socket type - UDP, TCP or RAW

Definition at line 660 of file SimpleSocket.h.

38.9.5.7 int32_t ydlidar::core::network::CSimpleSocket::m_nBytesSent [protected]

number of bytes received

Definition at line 661 of file SimpleSocket.h.

38.9.5.8 uint32_t ydlidar::core::network::CSimpleSocket::m_nFlags [protected]

number of bytes sent

Definition at line 662 of file SimpleSocket.h.

38.9.5.9 int32_t ydlidar::core::network::CSimpleSocket::m_nSocketDomain [protected]

size of internal send/receive buffer

Definition at line 658 of file SimpleSocket.h.

38.9.5.10 CSocketType ydlidar::core::network::CSimpleSocket::m_nSocketType [protected]

socket type PF_INET, PF_INET6

Definition at line 659 of file SimpleSocket.h.

38.9.5.11 bool ydlidar::core::network::CSimpleSocket::m_open [protected]

Definition at line 682 of file SimpleSocket.h.

38.9.5.12 uint8_t* ydlidar::core::network::CSimpleSocket::m_pBuffer [protected]

number of last error

Definition at line 655 of file SimpleSocket.h.

38.9.5.13 uint32_t ydlidar::core::network::CSimpleSocket::m_port [protected]

Definition at line 681 of file SimpleSocket.h.

38.9.5.14 fd_set ydlidar::core::network::CSimpleSocket::m_readFds [protected]

write file descriptor set

Definition at line 677 of file SimpleSocket.h.

38.9.5.15 SOCKET ydlidar::core::network::CSimpleSocket::m_socket [protected]

Definition at line 653 of file SimpleSocket.h.

38.9.5.16 CSocketError ydlidar::core::network::CSimpleSocket::m_socketErrno [protected]

socket handle

Definition at line 654 of file SimpleSocket.h.

38.9.5.17 struct sockaddr_in ydlidar::core::network::CSimpleSocket::m_stClientSockaddr [protected]

server address

Definition at line 669 of file SimpleSocket.h.

38.9.5.18 struct timeval ydlidar::core::network::CSimpleSocket::m_stConnectTimeout [protected]

is the UDP socket multicast;

Definition at line 665 of file SimpleSocket.h.

38.9.5.19 struct linger ydlidar::core::network::CSimpleSocket::m_stLinger [protected]

multicast group to bind to

Definition at line 671 of file SimpleSocket.h.

38.9.5.20 struct sockaddr_in ydlidar::core::network::CSimpleSocket::m_stMulticastGroup [protected]

client address

Definition at line 670 of file SimpleSocket.h.

38.9.5.21 struct timeval ydlidar::core::network::CSimpleSocket::m_stRecvTimeout [protected]

connection timeout

Definition at line 666 of file SimpleSocket.h.

38.9.5.22 struct timeval ydlidar::core::network::CSimpleSocket::m_stSendTimeout [protected]

receive timeout

Definition at line 667 of file SimpleSocket.h.

38.9.5.23 struct sockaddr_in ydlidar::core::network::CSimpleSocket::m_stServerSockaddr [protected]

send timeout

Definition at line 668 of file SimpleSocket.h.

38.9.5.24 CStatTimer ydlidar::core::network::CSimpleSocket::m_timer [protected]

linger flag

Definition at line 672 of file SimpleSocket.h.

38.9.5.25 fd_set ydlidar::core::network::CSimpleSocket::m_writeFds [protected]

internal statistics.

Definition at line 676 of file SimpleSocket.h.

The documentation for this class was generated from the following files:

- core/network/SimpleSocket.h
- core/network/SimpleSocket.cpp

38.10 CStatTimer Class Reference

`#include <StatTimer.h>`

Public Member Functions

- CStatTimer ()
- ~CStatTimer ()
- void Initialize ()
- struct timeval GetStartTime ()
- void SetStartTime ()
- struct timeval GetEndTime ()
- void SetEndTime ()
- uint32_t GetMilliSeconds ()
- uint64_t GetMicroSeconds ()
- uint32_t GetSeconds ()

Static Public Member Functions

- static uint64_t GetCurrentTime ()

38.10.1 Detailed Description

Class to abstract socket communications in a cross platform manner. This class is designed

Definition at line 87 of file StatTimer.h.

38.10.2 Constructor & Destructor Documentation

38.10.2.1 CStatTimer::CStatTimer ( ) [inline]

Definition at line 89 of file StatTimer.h.

38.10.2.2 CStatTimer::~CStatTimer ( ) [inline]

Definition at line 92 of file StatTimer.h.

38.10.3 Member Function Documentation

38.10.3.1 static uint64_t CStatTimer::GetCurrentTime ( ) [inline], [static]

Definition at line 124 of file StatTimer.h.

38.10.3.2 struct timeval CStatTimer::GetEndTime ( ) [inline]

Definition at line 107 of file StatTimer.h.

38.10.3.3 uint64_t CStatTimer::GetMicroSeconds ( ) [inline]

Definition at line 117 of file StatTimer.h.

38.10.3.4 uint32_t CStatTimer::GetMilliSeconds ( ) [inline]

Definition at line 114 of file StatTimer.h.

38.10.3.5 uint32_t CStatTimer::GetSeconds ( ) [inline]

Definition at line 120 of file StatTimer.h.

38.10.3.6 struct timeval CStatTimer::GetStartTime ( ) [inline]

Definition at line 100 of file StatTimer.h.

38.10.3.7 void CStatTimer::Initialize (void) [inline]

Definition at line 95 of file StatTimer.h.

38.10.3.8 void CStatTimer::SetEndTime ( ) [inline]

Definition at line 110 of file StatTimer.h.

38.10.3.9 void CStatTimer::SetStartTime ( ) [inline]

Definition at line 103 of file StatTimer.h.

The documentation for this class was generated from the following file:

- core/network/StatTimer.h

38.11 CYdLidar Class Reference

Set and Get LiDAR Maximum effective range.

`#include <CYdLidar.h>`

Public Member Functions

- CYdLidar ()

  create object
- virtual ~CYdLidar ()

  destroy object
- bool setlidaropt (int optname, const void *optval, int optlen)

  set lidar properties
- bool getlidaropt (int optname, void *optval, int optlen)

  get lidar property
- bool initialize ()

  Initialize the SDK and LiDAR.
- void GetLidarVersion (LidarVersion &version)

  Return LiDAR's version information in a numeric form.
- bool turnOn ()

  Start the device scanning routine which runs on a separate thread and enable motor.
- bool doProcessSimple (LaserScan &outscan)

  Get the LiDAR Scan Data. turnOn is successful before doProcessSimple scan data.
- bool turnOff ()

  Stop the device scanning thread and disable motor.
- void disconnecting ()

  Uninitialize the SDK and Disconnect the LiDAR.
- const char * DescribeError () const

  Get the last error information of a (socket or serial)

38.11.1 Detailed Description

Set and Get LiDAR Maximum effective range.

"Dataset"

LIDAR	Model	Baudrate	Sample↩ Rate(K)	Range(m)	Frequency HZ)	Intenstiy(I	Single↩ Channel	voltage(↩ V)
F4	1	115200	4	0.12∼12	5∼12	false	false	4.8∼5.2
S4	4	115200	4	0.↩ 10∼8.0	5∼12 (PWM)	false	false	4.8∼5.2
S4B	4/11	153600	4	0.↩ 10∼8.0	5∼12(P↩ WM)	true(8)	false	4.8∼5.2
S2	4/12	115200	3	0.↩ 10∼8.0	4∼8(P↩ WM)	false	true	4.8∼5.2
G4	5	230400	9/8/4	0.↩ 28/0.26/0.↩ 1∼16	5∼12	false	false	4.8∼5.2
X4	6	128000	5	0.12∼10	5∼12(P↩ WM)	false	false	4.8∼5.2
X2/X2L	6	115200	3	0.↩ 10∼8.0	4∼8(P↩ WM)	false	true	4.8∼5.2
G4PRO	7	230400	9/8/4	0.↩ 28/0.26/0.↩ 1∼16	5∼12	false	false	4.8∼5.2

LIDAR	Model	Baudrate	Sample↩ Rate(K)	Range(m)	Frequency HZ)	Intenstiy(I	Single↩ Channel	voltage(↩ V)
F4PRO	8	230400	4/6	0.12∼12	5∼12	false	false	4.8∼5.2
R2	9	230400	5	0.12∼16	5∼12	false	false	4.8∼5.2
G6	13	512000	18/16/8	0.↩ 28/0.26/0.↩ 1∼25	5∼12	false	false	4.8∼5.2
G2A	14	230400	5	0.12∼12	5∼12	false	false	4.8∼5.2
G2	15	230400	5	0.28∼16	5∼12	true(8)	false	4.8∼5.2
G2C	16	115200	4	0.1∼12	5∼12	false	false	4.8∼5.2
G4B	17	512000	10	0.12∼16	5∼12	true(10)	false	4.8∼5.2
G4C	18	115200	4	0.1∼12	5∼12	false	false	4.8∼5.2
G1	19	230400	9	0.28∼16	5∼12	false	false	4.8∼5.2
TX8	100	115200	4	0.05∼8	4∼8(P↩ WM)	false	true	4.8∼5.2
TX20	100	115200	4	0.05∼20	4∼8(P↩ WM)	false	true	4.8∼5.2
TG15	100	512000	20/18/10	0.05∼30	3∼16	false	false	4.8∼5.2
TG30	101	512000	20/18/10	0.05∼30	3∼16	false	false	4.8∼5.2
TG50	102	512000	20/18/10	0.05∼50	3∼16	false	false	4.8∼5.2
T15	200	8000	20	0.05∼30	5∼35	true	false	4.8∼5.2

Dataset

example: G4 LiDAR

```
CYdLidar laser;
std::string port = "/dev/ydlidar";
laser.setlidaropt(LidarPropSerialPort, port.c_str(), port.size());
std::string ignore_array;
ignore_array.clear();
laser.setlidaropt(LidarPropIgnoreArray, ignore_array.c_str(),
                  ignore_array.size());

int optval = 230400;
laser.setlidaropt(LidarPropSerialBaudrate, &optval, sizeof(int));
optval = TYPE_TRIANGLE;
laser.setlidaropt(LidarPropLidarType, &optval, sizeof(int));
optval = YDLIDAR_TYPE_SERIAL;
laser.setlidaropt(LidarPropDeviceType, &optval, sizeof(int));
optval = 9;
laser.setlidaropt(LidarPropSampleRate, &optval, sizeof(int));
optval = 4;
laser.setlidaropt(LidarPropAbnormalCheckCount, &optval, sizeof(int));

bool b_optvalue = false;
laser.setlidaropt(LidarPropFixedResolution, &b_optvalue, sizeof(bool));
laser.setlidaropt(LidarPropReversion, &b_optvalue, sizeof(bool));
laser.setlidaropt(LidarPropInverted, &b_optvalue, sizeof(bool));
b_optvalue = true;
laser.setlidaropt(LidarPropAutoReconnect, &b_optvalue, sizeof(bool));
b_optvalue = false;
laser.setlidaropt(LidarPropSingleChannel, &b_optvalue, sizeof(bool));
b_optvalue = false;
laser.setlidaropt(LidarPropIntenstiy, &b_optvalue, sizeof(bool));
laser.setlidaropt(LidarPropSupportMotorDtrCtrl, &b_optvalue, sizeof(
      bool));

float f_optvalue = 180.0f;
laser.setlidaropt(LidarPropMaxAngle, &f_optvalue, sizeof(float));
f_optvalue = -180.0f;
laser.setlidaropt(LidarPropMinAngle, &f_optvalue, sizeof(float));

f_optvalue = 16.f;
laser.setlidaropt(LidarPropMaxRange, &f_optvalue, sizeof(float));
f_optvalue = 0.1f;
laser.setlidaropt(LidarPropMinRange, &f_optvalue, sizeof(float));
f_optvalue = 10.f
laser.setlidaropt(LidarPropScanFrequency, &f_optvalue, sizeof(float));
```

example: S2 LiDAR

```
CYdLidar laser;
std::string port = "/dev/ydlidar";
laser.setlidaropt(LidarPropSerialPort, port.c_str(), port.size());
std::string ignore_array;
ignore_array.clear();
laser.setlidaropt(LidarPropIgnoreArray, ignore_array.c_str(),
                  ignore_array.size());

int optval = 115200;
laser.setlidaropt(LidarPropSerialBaudrate, &optval, sizeof(int));
optval = TYPE_TRIANGLE;
laser.setlidaropt(LidarPropLidarType, &optval, sizeof(int));
optval = YDLIDAR_TYPE_SERIAL;
laser.setlidaropt(LidarPropDeviceType, &optval, sizeof(int));
optval = 3;
laser.setlidaropt(LidarPropSampleRate, &optval, sizeof(int));
optval = 4;
laser.setlidaropt(LidarPropAbnormalCheckCount, &optval, sizeof(int));

bool b_optvalue = false;
laser.setlidaropt(LidarPropFixedResolution, &b_optvalue, sizeof(bool));
laser.setlidaropt(LidarPropReversion, &b_optvalue, sizeof(bool));
laser.setlidaropt(LidarPropInverted, &b_optvalue, sizeof(bool));
b_optvalue = true;
laser.setlidaropt(LidarPropAutoReconnect, &b_optvalue, sizeof(bool));
b_optvalue = true;
laser.setlidaropt(LidarPropSingleChannel, &b_optvalue, sizeof(bool));
b_optvalue = false;
laser.setlidaropt(LidarPropIntenstiy, &b_optvalue, sizeof(bool));
b_optvalue = true;
laser.setlidaropt(LidarPropSupportMotorDtrCtrl, &b_optvalue, sizeof(
      bool));

float f_optvalue = 180.0f;
laser.setlidaropt(LidarPropMaxAngle, &f_optvalue, sizeof(float));
f_optvalue = -180.0f;
laser.setlidaropt(LidarPropMinAngle, &f_optvalue, sizeof(float));

f_optvalue = 10.f;
laser.setlidaropt(LidarPropMaxRange, &f_optvalue, sizeof(float));
f_optvalue = 0.1f;
laser.setlidaropt(LidarPropMinRange, &f_optvalue, sizeof(float));
f_optvalue = 6.f
laser.setlidaropt(LidarPropScanFrequency, &f_optvalue, sizeof(float));
```

example: TG30 LiDAR

```
CYdLidar laser;
std::string port = "/dev/ydlidar";
laser.setlidaropt(LidarPropSerialPort, port.c_str(), port.size());
std::string ignore_array;
ignore_array.clear();
laser.setlidaropt(LidarPropIgnoreArray, ignore_array.c_str(),
                  ignore_array.size());

int optval = 512000;
laser.setlidaropt(LidarPropSerialBaudrate, &optval, sizeof(int));
optval = TYPE_TOF;
laser.setlidaropt(LidarPropLidarType, &optval, sizeof(int));
optval = YDLIDAR_TYPE_SERIAL;
laser.setlidaropt(LidarPropDeviceType, &optval, sizeof(int));
optval = 20;
laser.setlidaropt(LidarPropSampleRate, &optval, sizeof(int));
optval = 4;
laser.setlidaropt(LidarPropAbnormalCheckCount, &optval, sizeof(int));

bool b_optvalue = false;
laser.setlidaropt(LidarPropFixedResolution, &b_optvalue, sizeof(bool));
laser.setlidaropt(LidarPropReversion, &b_optvalue, sizeof(bool));
laser.setlidaropt(LidarPropInverted, &b_optvalue, sizeof(bool));
b_optvalue = true;
laser.setlidaropt(LidarPropAutoReconnect, &b_optvalue, sizeof(bool));
b_optvalue = false;
laser.setlidaropt(LidarPropSingleChannel, &b_optvalue, sizeof(bool));
b_optvalue = false;
laser.setlidaropt(LidarPropIntenstiy, &b_optvalue, sizeof(bool));
b_optvalue = false;
laser.setlidaropt(LidarPropSupportMotorDtrCtrl, &b_optvalue, sizeof(
      bool));

float f_optvalue = 180.0f;
laser.setlidaropt(LidarPropMaxAngle, &f_optvalue, sizeof(float));
f_optvalue = -180.0f;
```

```
laser.setlidaropt(LidarPropMinAngle, &f_optvalue, sizeof(float));

f_optvalue = 64.f;
laser.setlidaropt(LidarPropMaxRange, &f_optvalue, sizeof(float));
f_optvalue = 0.05f;
laser.setlidaropt(LidarPropMinRange, &f_optvalue, sizeof(float));
f_optvalue = 10.f
laser.setlidaropt(LidarPropScanFrequency, &f_optvalue, sizeof(float));
```

example: TX8 LiDAR

```
CYdLidar laser;
std::string port = "/dev/ydlidar";
laser.setlidaropt(LidarPropSerialPort, port.c_str(), port.size());
std::string ignore_array;
ignore_array.clear();
laser.setlidaropt(LidarPropIgnoreArray, ignore_array.c_str(),
                  ignore_array.size());

int optval = 115200;
laser.setlidaropt(LidarPropSerialBaudrate, &optval, sizeof(int));
optval = TYPE_TOF;
laser.setlidaropt(LidarPropLidarType, &optval, sizeof(int));
optval = YDLIDAR_TYPE_SERIAL;
laser.setlidaropt(LidarPropDeviceType, &optval, sizeof(int));
optval = 4;
laser.setlidaropt(LidarPropSampleRate, &optval, sizeof(int));
optval = 4;
laser.setlidaropt(LidarPropAbnormalCheckCount, &optval, sizeof(int));

bool b_optvalue = false;
laser.setlidaropt(LidarPropFixedResolution, &b_optvalue, sizeof(bool));
laser.setlidaropt(LidarPropReversion, &b_optvalue, sizeof(bool));
laser.setlidaropt(LidarPropInverted, &b_optvalue, sizeof(bool));
b_optvalue = true;
laser.setlidaropt(LidarPropAutoReconnect, &b_optvalue, sizeof(bool));
b_optvalue = true;
laser.setlidaropt(LidarPropSingleChannel, &b_optvalue, sizeof(bool));
b_optvalue = false;
laser.setlidaropt(LidarPropIntenstiy, &b_optvalue, sizeof(bool));
b_optvalue = true;
laser.setlidaropt(LidarPropSupportMotorDtrCtrl, &b_optvalue, sizeof(
      bool));

float f_optvalue = 180.0f;
laser.setlidaropt(LidarPropMaxAngle, &f_optvalue, sizeof(float));
f_optvalue = -180.0f;
laser.setlidaropt(LidarPropMinAngle, &f_optvalue, sizeof(float));

f_optvalue = 12.f;
laser.setlidaropt(LidarPropMaxRange, &f_optvalue, sizeof(float));
f_optvalue = 0.05f;
laser.setlidaropt(LidarPropMinRange, &f_optvalue, sizeof(float));
f_optvalue = 6.f
laser.setlidaropt(LidarPropScanFrequency, &f_optvalue, sizeof(float));
```

example: T15 LiDAR

```
CYdLidar laser;
std::string ipaddress = "192.168.1.11";
laser.setlidaropt(LidarPropSerialPort, ipaddress.c_str(), ipaddress.size());
std::string ignore_array;
ignore_array.clear();
laser.setlidaropt(LidarPropIgnoreArray, ignore_array.c_str(),
                  ignore_array.size());

int optval = 8000;
laser.setlidaropt(LidarPropSerialBaudrate, &optval, sizeof(int));
optval = TYPE_TOF_NET;
laser.setlidaropt(LidarPropLidarType, &optval, sizeof(int));
optval = YDLIDAR_TYPE_TCP;
laser.setlidaropt(LidarPropDeviceType, &optval, sizeof(int));
optval = 20;
laser.setlidaropt(LidarPropSampleRate, &optval, sizeof(int));
optval = 4;
laser.setlidaropt(LidarPropAbnormalCheckCount, &optval, sizeof(int));

bool b_optvalue = false;
laser.setlidaropt(LidarPropFixedResolution, &b_optvalue, sizeof(bool));
laser.setlidaropt(LidarPropReversion, &b_optvalue, sizeof(bool));
laser.setlidaropt(LidarPropInverted, &b_optvalue, sizeof(bool));
b_optvalue = true;
laser.setlidaropt(LidarPropAutoReconnect, &b_optvalue, sizeof(bool));
```

```
b_optvalue = false;
laser.setlidaropt(LidarPropSingleChannel, &b_optvalue, sizeof(bool));
b_optvalue = true;
laser.setlidaropt(LidarPropIntenstiy, &b_optvalue, sizeof(bool));
b_optvalue = false;
laser.setlidaropt(LidarPropSupportMotorDtrCtrl, &b_optvalue, sizeof(
      bool));

float f_optvalue = 180.0f;
laser.setlidaropt(LidarPropMaxAngle, &f_optvalue, sizeof(float));
f_optvalue = -180.0f;
laser.setlidaropt(LidarPropMinAngle, &f_optvalue, sizeof(float));

f_optvalue = 64.f;
laser.setlidaropt(LidarPropMaxRange, &f_optvalue, sizeof(float));
f_optvalue = 0.05f;
laser.setlidaropt(LidarPropMinRange, &f_optvalue, sizeof(float));
f_optvalue = 20.f
laser.setlidaropt(LidarPropScanFrequency, &f_optvalue, sizeof(float));
```

LidarPropMaxRange

Note

The effective range beyond the maxmum is set to zero.
the MaxRange should be greater than the MinRange.

Remarks

unit: m

See also

LidarPropMaxRange
DataSet
CYdLidar::lidarSetProp and CYdLidar::lidarGetProp

LidarPropMinRange

Set and Get LiDAR Minimum effective range.

Note

The effective range less than the minmum is set to zero.
the MinRange should be less than the MaxRange.

Remarks

unit: m

See also

LidarPropMinRange
Dataset
CYdLidar::lidarSetProp and CYdLidar::lidarGetProp

LidarPropMaxAngle

Set and Get LiDAR Maximum effective angle.

Note

The effective angle beyond the maxmum will be ignored.
the MaxAngle should be greater than the MinAngle

Remarks

unit: degree, Range:-180∼180

See also

Dataset
CYdLidar::lidarSetProp and CYdLidar::lidarGetProp

LidarPropMinAngle

Set and Get LiDAR Minimum effective angle.

Note

The effective angle less than the minmum will be ignored.
the MinAngle should be less than the MaxAngle

Remarks

unit: degree, Range:-180∼180

See also

Dataset
CYdLidar::lidarSetProp and CYdLidar::lidarGetProp

LidarPropSampleRate

Set and Get LiDAR Sampling rate.

Note

If the set sampling rate does no exist. the actual sampling rate is the LiDAR's default sampling rate.
Set the sampling rate to match the LiDAR.

Remarks

unit: kHz/s, Ranges: 2,3,4,5,6,8,9,10,16,18,20

G4/F4	4,8,9
F4PRO	4,6
G6	8,16,18
G4B	10
G1	9
G2A/G2/R2/X4	5
S4/S4B/G4C/TX8/TX20	4
G2C	4
S2	3
TG15/TG30/TG50	10,18,20
T5/T15	20

See also

CYdLidar::lidarSetProp and CYdLidar::lidarGetProp

LidarPropScanFrequency

Set and Get LiDAR Scan frequency.

Note

If the LiDAR is a single channel, the scanning frequency nneds to be adjusted by external PWM.
Set the scan frequency to match the LiDAR.

Remarks

unit: Hz

S2/X2/X2L/TX8/TX20	4∼8(PWM)
F4/F4PRO/G4/G4PRO/R2	5∼12
G6/G2A/G2/G2C/G4B/G4C/G1	5∼12
S4/S4B/X4	5∼12(PWM)
TG15/TG30/TG50	3∼16
T5/T15	5∼40

See also

CYdLidar::lidarSetProp and CYdLidar::lidarGetProp

LidarPropFixedResolution

Set and Get LiDAR Fixed angluar resolution.

Note

The Lidar scanning frequency will change slightly due to various reasons. so the number of points per circle will also change slightly.
if a fixed angluar resolution is required. a fixed number of points is required.

If set to true, the angle_increment of the fixed angle resolution in LaserConfig will be a fixed value.

See also

CYdLidar::lidarSetProp and CYdLidar::lidarGetProp

LidarPropReversion

Set and Get LiDAR Reversion.
true: LiDAR data rotated 180 degrees.
false: Keep raw Data.
default: false

Note

Refer to the table below for the LiDAR Reversion.
This is currently related to your coordinate system and install direction. Whether to reverse it depends on your actual scene.

LiDAR	reversion
G1/G2/G2A/G2C/F4/F4PRO/R2	true
G4/G4PRO/G4B/G4C/G6	true
TG15/TG30/TG50	true
T5/T15	true
S2/X2/X2L/X4/S4/S4B	false
TX8/TX20	false

Reversion Table

See also

CYdLidar::lidarSetProp and CYdLidar::lidarGetProp

LidarPropInverted

Set and Get LiDAR inverted.
true: Data is counterclockwise
false: Data is clockwise
Default: clockwise

Note

If set to true, LiDAR data direction is positive counterclockwise. otherwise it is positive clockwise.

See also

CYdLidar::lidarSetProp and CYdLidar::lidarGetProp

LidarPropAutoReconnect

Set and Get LiDAR Automatically reconnect flag.
Whether to support hot plug.

See also

CYdLidar::lidarSetProp and CYdLidar::lidarGetProp

LidarPropSerialBaudrate

Set and Get LiDAR baudrate or network port.

Note

Refer to the table below for the LiDAR Baud Rate.
Set the baudrate or network port to match the LiDAR.

F4/S2/X2/X2L/S4/TX8/TX20/G4C	115200
X4	128000
S4B	153600
G1/G2/R2/G4/G4PRO/F4PRO	230400
G2A/G2C	230400
G6/G4B/TG15/TG30/TG50	512000
T5/T15(network)	8000

Remarks

See also

CYdLidar::lidarSetProp and CYdLidar::lidarGetProp

LidarPropAbnormalCheckCount

Set and Get LiDAR Maximum number of abnormal checks.

Note

When the LiDAR Turn On, if the number of times of abnormal data acquisition is greater than the current AbnormalCheckCount, the LiDAR Fails to Turn On.

The Minimum abnormal value is Two, if it is less than the Minimum Value, it will be set to the Mimimum Value.

See also

CYdLidar::lidarSetProp and CYdLidar::lidarGetProp

LidarPropSerialPort

Set and Get LiDAR Serial port or network IP address.

Note

If it is serial port, your need to ensure that the serial port had read and write permissions.
If it is a network, make sure the network can ping.

See also

CYdLidar::lidarSetProp and CYdLidar::lidarGetProp

LidarPropIgnoreArray

Set and Get LiDAR filtering angle area.

Note

If the LiDAR angle is in the IgnoreArray, the current range will be set to zero.
Filtering angles need to appear in pairs.

The purpose of the current paramter is to filter out the angular area set by user

example: Filters 10 degrees to 30 degrees and 80 degrees to 90 degrees.

```
CYdLidar laser;//Defining an CYdLidar instance.
std::string ignore_array= "10.0, 30.0, 80.0, 90.0";
laser.lidarSetProp(LidarPropIgnoreArray, ignore_array);
```

See also

CYdLidar::lidarSetProp and CYdLidar::lidarGetProp

LidarPropSingleChannel

Set and Get LiDAR single channel. Whether LiDAR communication channel is a single-channel

Note

For a single-channel LiDAR, if the settings are reversed.
an error will occur in obtaining device information and the LiDAR will Faied to Start.
For dual-channel LiDAR, if th setttings are reversed.
the device information cannot be obtained.
Set the single channel to match the LiDAR.

G1/G2/G2A/G2C	false
G4/G4B/G4PRO/G6/F4/F4PRO	false
S4/S4B/X4/R2/G4C	false
S2/X2/X2L	true
TG15/TG30/TG50	false
TX8/TX20	true
T5/T15	false
	true

Remarks

See also

CYdLidar::lidarSetProp and CYdLidar::lidarGetProp

LidarPropLidarType

Set and Get LiDAR Type.

Note

Refer to the table below for the LiDAR Type.
Set the LiDAR Type to match the LiDAR.

G1/G2A/G2/G2C	TYPE_TRIANGLE
G4/G4B/G4C/G4PRO	TYPE_TRIANGLE
G6/F4/F4PRO	TYPE_TRIANGLE
S4/S4B/X4/R2/S2/X2/X2L	TYPE_TRIANGLE
TG15/TG30/TG50/TX8/TX20	TYPE_TOF
T5/T15	TYPE_TOF_NET

Remarks

See also

LidarTypeID
CYdLidar::lidarSetProp and CYdLidar::lidarGetProp

LidarPropIntensity

Set and Get LiDAR Intensity.

Note

If the settings are reversed. the LiDAR cannot parse the data correctly.
Set the Intensity to match the LiDAR.

S4B/G2/G4B	true
G4/G4C/G4PRO/F4/F4PRO/G6	false
G1/G2A/G2C/R2	false
S2/X2/X2L/X4	false
TG15/TG30/TG50	false
TX8/TX20	false
T5/T15	true
	false

Remarks

See also

CYdLidar::lidarSetProp and CYdLidar::lidarGetProp

LidarPropDeviceType

Set and Get LiDAR connection Type.

Note

If you connect the LiDAR through the network to serial port adapter board.
you need to set the current connection type to YDLIDAR_TYPE_TCP.
otherwise set connection type to YDLIDAR_TYPE_SERIAL.
Set the LiDAR connection Type to match the LiDAR.

See also

CYdLidar::lidarSetProp and CYdLidar::lidarGetProp

LidarPropSupportMotorDtrCtrl

Set and Get LiDAR Support Motor DTR.

Note

The current paramter settings are only valid if the LiDAR is connected to the serial port adapter via USB.
If the LiDAR does not have external motor enable line, the current paramters do not need to be set.
Set the LiDAR Motro DTR to match the LiDAR.

S4/S4B/S2/X2/X2L/X4	true
TX8/TX20	true
G4/G4C/G4PRO/F4/F4PRO/G6	false
G1/G2A/G2C/R2/G2/G4B	false
TG15/TG30/TG50	false
T5/T15	false

Remarks

See also

CYdLidar::lidarSetProp and CYdLidar::lidarGetPropProvides a platform independent class to for LiDAR development. This class is designed to serial or socket communication development in a platform independent manner.

- LiDAR types
  1. ydlidar::YDlidarDriver Class
  2. ydlidar::ETLidarDriver Class

Definition at line 727 of file CYdLidar.h.

38.11.2 Constructor & Destructor Documentation

38.11.2.1 CYdLidar::CYdLidar ( )

create object

Definition at line 46 of file CYdLidar.cpp.

38.11.2.2 CYdLidar::~CYdLidar ( ) [virtual]

destroy object

Definition at line 90 of file CYdLidar.cpp.

38.11.3 Member Function Documentation

38.11.3.1 const char * CYdLidar::DescribeError ( ) const

Get the last error information of a (socket or serial)

Returns

a human-readable description of the given error information or the last error information of a (socket or serial)

Definition at line 693 of file CYdLidar.cpp.

38.11.3.2 void CYdLidar::disconnecting ( )

Uninitialize the SDK and Disconnect the LiDAR.

Definition at line 666 of file CYdLidar.cpp.

38.11.3.3 bool CYdLidar::doProcessSimple (LaserScan & *outscan*)

Get the LiDAR Scan Data. turnOn is successful before doProcessSimple scan data.

Parameters

out	*outscan*	LiDAR Scan Data
out	*hardwareError*	hardware error status

Returns

true if successfully started, otherwise false.

Definition at line 463 of file CYdLidar.cpp.

38.11.3.4 bool CYdLidar::getlidaropt (int *optname*, void * *optval*, int *optlen*)

get lidar property

Parameters

optname	option name

Todo string properties

- LidarPropSerialPort
- LidarPropIgnoreArray

Note

get string property example

```
CYdLidar laser;
char lidar_port[30];
laser.getlidaropt(LidarPropSerialPort,lidar_port, sizeof(lidar_port));
```

Todo int properties

- LidarPropSerialBaudrate
- LidarPropLidarType
- LidarPropDeviceType
- LidarPropSampleRate

Note

get int property example

```
CYdLidar laser;
int lidar_baudrate;
laser.getlidaropt(LidarPropSerialPort,&lidar_baudrate, sizeof(int));
```

Todo bool properties

- LidarPropFixedResolution
- LidarPropReversion
- LidarPropInverted
- LidarPropAutoReconnect
- LidarPropSingleChannel
- LidarPropIntenstiy

Note

get bool property example

```
CYdLidar laser;
bool lidar_fixedresolution;
laser.getlidaropt(LidarPropSerialPort,&lidar_fixedresolution, sizeof(bool));
```

Todo float properties

- LidarPropMaxRange
- LidarPropMinRange
- LidarPropMaxAngle
- LidarPropMinAngle
- LidarPropScanFrequency

Note

set float property example

```
CYdLidar laser;
float lidar_maxrange;
laser.getlidaropt(LidarPropSerialPort,&lidar_maxrange, sizeof(float));
```

Parameters

optval	option value • std::string(or char∗) • int • bool • float
optlen	option length • data type size

Returns

true if the Property is get successfully, otherwise false.

See also

LidarProperty

Definition at line 236 of file CYdLidar.cpp.

38.11.3.5 void CYdLidar::GetLidarVersion (LidarVersion & *version*)

Return LiDAR's version information in a numeric form.

Parameters

version	Pointer to a version structure for returning the version information.

Definition at line 392 of file CYdLidar.cpp.

38.11.3.6 bool CYdLidar::initialize ( )

Initialize the SDK and LiDAR.

Returns

true if successfully initialized, otherwise false.

Definition at line 369 of file CYdLidar.cpp.

38.11.3.7 bool CYdLidar::setlidaropt (int *optname*, const void * *optval*, int *optlen*)

set lidar properties

Parameters

optname	option name

Todo string properties

- LidarPropSerialPort
- LidarPropIgnoreArray

Note

set string property example

```
CYdLidar laser;
std::string lidar_port = "/dev/ydlidar";
laser.setlidaropt(LidarPropSerialPort,lidar_port.c_str(), lidar_port.size());
```

Todo int properties

- LidarPropSerialBaudrate
- LidarPropLidarType
- LidarPropDeviceType
- LidarPropSampleRate

Note

set int property example

```
CYdLidar laser;
int lidar_baudrate = 230400;
laser.setlidaropt(LidarPropSerialPort,&lidar_baudrate, sizeof(int));
```

Todo bool properties

- LidarPropFixedResolution
- LidarPropReversion
- LidarPropInverted
- LidarPropAutoReconnect
- LidarPropSingleChannel
- LidarPropIntenstiy

Note

set bool property example

```
CYdLidar laser;
bool lidar_fixedresolution = true;
laser.setlidaropt(LidarPropSerialPort,&lidar_fixedresolution, sizeof(bool));
```

Todo float properties

- LidarPropMaxRange
- LidarPropMinRange
- LidarPropMaxAngle
- LidarPropMinAngle
- LidarPropScanFrequency

Note

set float property example, Must be float type, not double type.

```
CYdLidar laser;
float lidar_maxrange = 16.0f;
laser.setlidaropt(LidarPropSerialPort,&lidar_maxrange, sizeof(float));
```

Parameters

optval	option value • std::string(or char*) • int • bool • float
optlen	option length • data type size

Returns

true if the Property is set successfully, otherwise false.

See also

LidarProperty

Definition at line 99 of file CYdLidar.cpp.

38.11.3.8 bool CYdLidar::turnOff ( )

Stop the device scanning thread and disable motor.

Returns

true if successfully Stoped, otherwise false.

Definition at line 649 of file CYdLidar.cpp.

38.11.3.9 bool CYdLidar::turnOn ( )

Start the device scanning routine which runs on a separate thread and enable motor.

Returns

true if successfully started, otherwise false.

Definition at line 399 of file CYdLidar.cpp.

The documentation for this class was generated from the following files:

- src/CYdLidar.h
- src/CYdLidar.cpp

38.12 dataFrame Class Reference

data frame Structure.

`#include <datatype.h>`

38.12.1 Detailed Description

data frame Structure.

Author

jzhang

The documentation for this class was generated from the following file:

- core/base/datatype.h

38.13 device_health Struct Reference

LiDAR Health Information.

`#include <ydlidar_protocol.h>`

Public Attributes

- uint8_t status
  health state
- uint16_t error_code
  error code

38.13.1 Detailed Description

LiDAR Health Information.

Definition at line 202 of file ydlidar_protocol.h.

38.13.2 Member Data Documentation

38.13.2.1 uint16_t device_health::error_code

error code

Definition at line 204 of file ydlidar_protocol.h.

38.13.2.2 uint8_t device_health::status

health state

Definition at line 203 of file ydlidar_protocol.h.

The documentation for this struct was generated from the following file:

- core/common/ydlidar_protocol.h

38.14 device_info Struct Reference

LiDAR Device Information.

`#include <ydlidar_protocol.h>`

Public Attributes

- uint8_t model
  LiDAR model.
- uint16_t firmware_version
  firmware version
- uint8_t hardware_version
  hardare version
- uint8_t serialnum [16]
  serial number

38.14.1 Detailed Description

LiDAR Device Information.

Definition at line 194 of file ydlidar_protocol.h.

38.14.2 Member Data Documentation

38.14.2.1 uint16_t device_info::firmware_version

firmware version

Definition at line 196 of file ydlidar_protocol.h.

38.14.2.2 uint8_t device_info::hardware_version

hardare version

Definition at line 197 of file ydlidar_protocol.h.

38.14.2.3 uint8_t device_info::model

LiDAR model.

Definition at line 195 of file ydlidar_protocol.h.

38.14.2.4 uint8_t device_info::serialnum[16]

serial number

Definition at line 198 of file ydlidar_protocol.h.

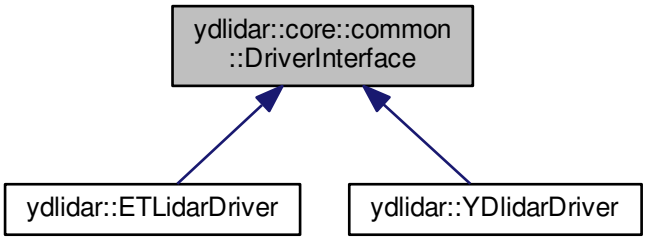
The documentation for this struct was generated from the following file:

- core/common/ydlidar_protocol.h

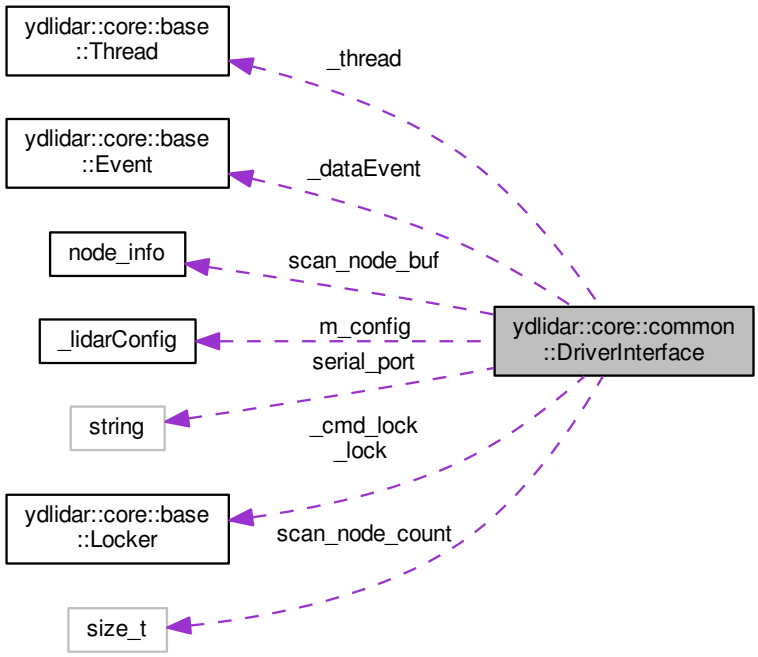
38.15 ydlidar::core::common::DriverInterface Class Reference

`#include <DriverInterface.h>`

Inheritance diagram for ydlidar::core::common::DriverInterface:



Collaboration diagram for ydlidar::core::common::DriverInterface:



Public Types

- enum {
  YDLIDAR_F4 = 1, YDLIDAR_T1 = 2, YDLIDAR_F2 = 3, YDLIDAR_S4 = 4,
  YDLIDAR_G4 = 5, YDLIDAR_X4 = 6, YDLIDAR_G4PRO = 7, YDLIDAR_F4PRO = 8,
  YDLIDAR_R2 = 9, YDLIDAR_G10 = 10, YDLIDAR_S4B = 11, YDLIDAR_S2 = 12,
  YDLIDAR_G6 = 13, YDLIDAR_G2A = 14, YDLIDAR_G2B = 15, YDLIDAR_G2C = 16,
  YDLIDAR_G4B = 17, YDLIDAR_G4C = 18, YDLIDAR_G1 = 19, YDLIDAR_TG15 = 100,
  YDLIDAR_TG30 = 101, YDLIDAR_TG50 = 102, YDLIDAR_T15 = 200, YDLIDAR_Tail }
- enum { YDLIDAR_RATE_4K = 0, YDLIDAR_RATE_8K = 1, YDLIDAR_RATE_9K = 2, YDLIDAR_RATE_10K = 3 }
- enum { DEFAULT_TIMEOUT = 2000, DEFAULT_HEART_BEAT = 1000, MAX_SCAN_NODES = 7200, DEFAULT_TIMEOUT_COUNT = 1 }

Public Member Functions

- void setSingleChannel (bool v)
- bool getSingleChannel () const
- void setLidarType (int v)
- int getLidarType () const
- void setPointTime (uint32_t v)
- uint32_t getPointTime () const
- void setSupportMotorDtrCtrl (bool v)
- bool getSupportMotorDtrCtrl () const
- DriverInterface ()

- virtual ~DriverInterface ()
- virtual result_t connect (const char *port_path, uint32_t baudrate=8000)=0

  Connecting Lidar
  After the connection if successful, you must use ::disconnect to close.
- virtual const char * DescribeError (bool isTCP=true)=0

  Returns a human-readable description of the given error code or the last error code of a socket or serial port.
- virtual void disconnect ()=0

  Disconnect the LiDAR.
- virtual std::string getSDKVersion ()=0

  Get SDK Version
  static function.
- virtual bool isscanning () const =0

  Is the Lidar in the scan
  .
- virtual bool isconnected () const =0

  Is it connected to the lidar
  .
- virtual void setIntensities (const bool &isintensities)=0

  Is there intensity
  .
- virtual void setAutoReconnect (const bool &enable)=0

  whether to support hot plug
- virtual lidarConfig getFinishedScanCfg () const

  Get current scan update configuration.
- virtual result_t getHealth (device_health &health, uint32_t timeout=DEFAULT_TIMEOUT)=0

  get Health status
- virtual result_t getDeviceInfo (device_info &info, uint32_t timeout=DEFAULT_TIMEOUT)=0

  get Device information
- virtual result_t startScan (bool force=false, uint32_t timeout=DEFAULT_TIMEOUT)=0

  Turn on scanning
  .
- virtual result_t stop ()=0

  turn off scanning
- virtual result_t grabScanData (node_info *nodebuffer, size_t &count, uint32_t timeout=DEFAULT_TIMEOUT)=0

  Get a circle of laser data
  .
- virtual result_t getScanFrequency (scan_frequency &frequency, uint32_t timeout=DEFAULT_TIMEOUT)=0

  Get lidar scan frequency
  .
- virtual result_t setScanFrequencyAdd (scan_frequency &frequency, uint32_t timeout=DEFAULT_TIMEOUT)=0

  Increase the scanning frequency by 1.0 HZ
  .
- virtual result_t setScanFrequencyDis (scan_frequency &frequency, uint32_t timeout=DEFAULT_TIMEOUT)=0

  Reduce the scanning frequency by 1.0 HZ
  .
- virtual result_t setScanFrequencyAddMic (scan_frequency &frequency, uint32_t timeout=DEFAULT_TIMEOUT)=0

Increase the scanning frequency by 0.1 HZ

.

- virtual result_t setScanFrequencyDisMic (scan_frequency &frequency, uint32_t timeout=DEFAULT_TIMEOUT)=0

  Reduce the scanning frequency by 0.1 HZ

  .
- virtual result_t getSamplingRate (sampling_rate &rate, uint32_t timeout=DEFAULT_TIMEOUT)=0

  Get lidar sampling frequency

  .
- virtual result_t setSamplingRate (sampling_rate &rate, uint32_t timeout=DEFAULT_TIMEOUT)=0

  Set the lidar sampling frequency

  .
- virtual result_t getZeroOffsetAngle (offset_angle &angle, uint32_t timeout=DEFAULT_TIMEOUT)=0

  get lidar zero offset angle

Protected Attributes

- bool m_SingleChannel

  Set and Get LiDAR single channel. Whether LiDAR communication channel is a single-channel.
- int m_LidarType

  Set and Get LiDAR Type.
- uint32_t m_PointTime

  Set and Get Sampling interval.
- bool m_SupportMotorDtrCtrl

  Set and Get LiDAR Support Motor DTR.
- bool m_isScanning

  LiDAR Scanning state.
- bool m_isConnected

  LiDAR connected state.
- Event _dataEvent

  Scan Data Event.
- Locker _lock

  Data Locker.
- Thread _thread

  Parse Data thread.
- Locker _cmd_lock

  command locker
- std::string serial_port

  LiDAR com port or IP Address.
- uint32_t m_baudrate

  baudrate or IP port
- bool m_intensities

  LiDAR intensity.
- node_info * scan_node_buf

  LiDAR Point pointer.
- size_t scan_node_count

  LiDAR scan count.
- uint16_t package_Sample_Index

  package sample index
- int retryCount

- bool isAutoReconnect

  auto reconnect
- bool isAutoconnting

  auto connecting state
- lidarConfig m_config

38.15.1 Detailed Description

Definition at line 14 of file DriverInterface.h.

38.15.2 Member Enumeration Documentation

38.15.2.1 anonymous enum

Enumerator

YDLIDAR_F4 F4 LiDAR Model.

YDLIDAR_T1 T1 LiDAR Model.

YDLIDAR_F2 F2 LiDAR Model.

YDLIDAR_S4 S4 LiDAR Model.

YDLIDAR_G4 G4 LiDAR Model.

YDLIDAR_X4 X4 LiDAR Model.

YDLIDAR_G4PRO G4PRO LiDAR Model.

YDLIDAR_F4PRO F4PRO LiDAR Model.

YDLIDAR_R2 R2 LiDAR Model.

YDLIDAR_G10 G10 LiDAR Model.

YDLIDAR_S4B S4B LiDAR Model.

YDLIDAR_S2 S2 LiDAR Model.

YDLIDAR_G6 G6 LiDAR Model.

YDLIDAR_G2A G2A LiDAR Model.

YDLIDAR_G2B G2 LiDAR Model.

YDLIDAR_G2C G2C LiDAR Model.

YDLIDAR_G4B G4B LiDAR Model.

YDLIDAR_G4C G4C LiDAR Model.

YDLIDAR_G1 G1 LiDAR Model.

YDLIDAR_TG15 TG15 LiDAR Model.

YDLIDAR_TG30 T30 LiDAR Model.

YDLIDAR_TG50 TG50 LiDAR Model.

YDLIDAR_T15 T15 LiDAR Model.

YDLIDAR_Tail

Definition at line 337 of file DriverInterface.h.

38.15.2.2 anonymous enum

Enumerator

YDLIDAR_RATE_4K 4K sample rate code

YDLIDAR_RATE_8K 8K sample rate code

YDLIDAR_RATE_9K 9K sample rate code

YDLIDAR_RATE_10K 10K sample rate code

Definition at line 366 of file DriverInterface.h.

38.15.2.3 anonymous enum

Enumerator

DEFAULT_TIMEOUT Default timeout.

DEFAULT_HEART_BEAT Default heartbeat timeout.

MAX_SCAN_NODES Default Max Scan Count.

DEFAULT_TIMEOUT_COUNT Default Timeout Count.

Definition at line 374 of file DriverInterface.h.

38.15.3 Constructor & Destructor Documentation

38.15.3.1 ydlidar::core::common::DriverInterface::DriverInterface ( ) [inline]

Constructor

Definition at line 87 of file DriverInterface.h.

38.15.3.2 virtual ydlidar::core::common::DriverInterface::~DriverInterface ( ) [inline], [virtual]

Definition at line 108 of file DriverInterface.h.

38.15.4 Member Function Documentation

38.15.4.1 virtual result_t ydlidar::core::common::DriverInterface::connect (const char * *port_path,* uint32_t *baudrate =* 8000) [pure virtual]

Connecting Lidar
After the connection if successful, you must use ::disconnect to close.

Parameters

in	*port_path*	serial port
in	*baudrate*	serial baudrate, YDLIDAR-SS: 230400 G2-SS-1 R2-SS-1

Returns

connection status

Return values

0	success
<	0 failed

Note

After the connection if successful, you must use ::disconnect to close

See also

function ::YDlidarDriver::disconnect ()

Implemented in ydlidar::YDlidarDriver, and ydlidar::ETLidarDriver.

38.15.4.2 virtual const char* ydlidar::core::common::DriverInterface::DescribeError (bool *isTCP* = true) [pure virtual]

Returns a human-readable description of the given error code or the last error code of a socket or serial port.

Parameters

isTCP	TCP or UDP

Returns

error information

Implemented in ydlidar::YDlidarDriver, and ydlidar::ETLidarDriver.

38.15.4.3 virtual void ydlidar::core::common::DriverInterface::disconnect ( ) [pure virtual]

Disconnect the LiDAR.

Implemented in ydlidar::YDlidarDriver, and ydlidar::ETLidarDriver.

38.15.4.4 virtual result_t ydlidar::core::common::DriverInterface::getDeviceInfo (device_info & *info*, uint32_t *timeout* = DEFAULT_TIMEOUT) [pure virtual]

get Device information

Parameters

in	*info*	Device information
in	*timeout*	timeout

Returns

result status

Return values

RESULT_OK	success
RESULT_FAILE	or RESULT_TIMEOUT failed

Implemented in ydlidar::YDlidarDriver, and ydlidar::ETLidarDriver.

38.15.4.5 virtual lidarConfig ydlidar::core::common::DriverInterface::getFinishedScanCfg ( ) const [inline], [virtual]

Get current scan update configuration.

Returns

scanCfg structure.

Definition at line 181 of file DriverInterface.h.

38.15.4.6 virtual result_t ydlidar::core::common::DriverInterface::getHealth (device_health & *health*, uint32_t *timeout* = DEFAULT_TIMEOUT) [pure virtual]

get Health status

Returns

result status

Return values

RESULT_OK	success
RESULT_FAILE	or RESULT_TIMEOUT failed

Implemented in ydlidar::YDlidarDriver, and ydlidar::ETLidarDriver.

38.15.4.7 int ydlidar::core::common::DriverInterface::getLidarType ( ) const [inline]

Definition at line 54 of file DriverInterface.h.

38.15.4.8 uint32_t ydlidar::core::common::DriverInterface::getPointTime ( ) const [inline]

Definition at line 62 of file DriverInterface.h.

38.15.4.9 virtual result_t ydlidar::core::common::DriverInterface::getSamplingRate (sampling_rate & *rate*, uint32_t *timeout* = DEFAULT_TIMEOUT) [pure virtual]

Get lidar sampling frequency
.

Parameters

in	*frequency*	sampling frequency
in	*timeout*	timeout

Returns

return status

Return values

RESULT_OK	success
RESULT_FAILE	failed

Note

Non-scan state, perform currect operation.

Implemented in ydlidar::YDlidarDriver, and ydlidar::ETLidarDriver.

38.15.4.10 virtual result_t ydlidar::core::common::DriverInterface::getScanFrequency (scan_frequency & *frequency*, uint32_t *timeout* = DEFAULT_TIMEOUT) [pure virtual]

Get lidar scan frequency
.

Parameters

in	*frequency*	scanning frequency
in	*timeout*	timeout

Returns

return status

Return values

RESULT_OK	success
RESULT_FAILE	failed

Note

Non-scan state, perform currect operation.

Implemented in ydlidar::YDlidarDriver, and ydlidar::ETLidarDriver.

38.15.4.11 virtual std::string ydlidar::core::common::DriverInterface::getSDKVersion ( ) [pure virtual]

Get SDK Version
static function.

Returns

Version

Implemented in ydlidar::YDlidarDriver, and ydlidar::ETLidarDriver.

38.15.4.12 bool ydlidar::core::common::DriverInterface::getSingleChannel ( ) const [inline]

Definition at line 37 of file DriverInterface.h.

38.15.4.13 bool ydlidar::core::common::DriverInterface::getSupportMotorDtrCtrl ( ) const [inline]

Definition at line 81 of file DriverInterface.h.

38.15.4.14 virtual result_t ydlidar::core::common::DriverInterface::getZeroOffsetAngle (offset_angle & *angle*, uint32_t *timeout* = DEFAULT_TIMEOUT) [pure virtual]

get lidar zero offset angle

Parameters

in	*angle*	zero offset angle
in	*timeout*	timeout

Returns

return status

Return values

RESULT_OK	success
RESULT_FAILE	failed

Note

Non-scan state, perform currect operation.

Implemented in ydlidar::YDlidarDriver, and ydlidar::ETLidarDriver.

38.15.4.15 virtual result_t ydlidar::core::common::DriverInterface::grabScanData (node_info * *nodebuffer*, size_t & *count*, uint32_t *timeout* = DEFAULT_TIMEOUT) [pure virtual]

Get a circle of laser data
.

Parameters

in	*nodebuffer*	Laser data
in	*count*	one circle of laser points
in	*timeout*	timeout

Returns

return status

Return values

RESULT_OK	success
RESULT_FAILE	failed

Note

Before starting, you must start the start the scan successfully with the ::startScan function

Implemented in ydlidar::YDlidarDriver, and ydlidar::ETLidarDriver.

38.15.4.16 virtual bool ydlidar::core::common::DriverInterface::isconnected ( ) const [pure virtual]

Is it connected to the lidar
.

Returns

connection status

Return values

true	connected
false	Non-connected

Implemented in ydlidar::YDlidarDriver, and ydlidar::ETLidarDriver.

38.15.4.17 virtual bool ydlidar::core::common::DriverInterface::isscanning ( ) const [pure virtual]

Is the Lidar in the scan
.

Returns

scanning status

Return values

true	scanning
false	non-scanning

Implemented in ydlidar::YDlidarDriver, and ydlidar::ETLidarDriver.

38.15.4.18 virtual void ydlidar::core::common::DriverInterface::setAutoReconnect (const bool & *enable*) [pure virtual]

whether to support hot plug

Parameters

in	*enable*	hot plug : true support false no support

Implemented in ydlidar::YDlidarDriver, and ydlidar::ETLidarDriver.

38.15.4.19 virtual void ydlidar::core::common::DriverInterface::setIntensities (const bool & *isintensities*) [pure virtual]

Is there intensity
.

Parameters

in	*isintensities*	intentsity true intensity false no intensity

Implemented in ydlidar::YDlidarDriver, and ydlidar::ETLidarDriver.

38.15.4.20 void ydlidar::core::common::DriverInterface::setLidarType (int *v*) [inline]

Definition at line 54 of file DriverInterface.h.

38.15.4.21 void ydlidar::core::common::DriverInterface::setPointTime (uint32_t *v*) [inline]

Definition at line 62 of file DriverInterface.h.

38.15.4.22 virtual result_t ydlidar::core::common::DriverInterface::setSamplingRate (sampling_rate & *rate*, uint32_t *timeout* = DEFAULT_TIMEOUT) [pure virtual]

Set the lidar sampling frequency
.

Parameters

in	*rate*	sampling frequency
in	*timeout*	timeout

Returns

return status

Return values

RESULT_OK	success
RESULT_FAILE	failed

Note

Non-scan state, perform currect operation.

Implemented in ydlidar::YDlidarDriver, and ydlidar::ETLidarDriver.

38.15.4.23 virtual result_t ydlidar::core::common::DriverInterface::setScanFrequencyAdd (scan_frequency & *frequency*, uint32_t *timeout* = DEFAULT_TIMEOUT) [pure virtual]

Increase the scanning frequency by 1.0 HZ
.

Parameters

in	*frequency*	scanning frequency
in	*timeout*	timeout

Returns

return status

Return values

RESULT_OK	success
RESULT_FAILE	failed

Note

Non-scan state, perform currect operation.

Implemented in ydlidar::YDlidarDriver, and ydlidar::ETLidarDriver.

38.15.4.24 virtual result_t ydlidar::core::common::DriverInterface::setScanFrequencyAddMic (scan_frequency & *frequency*, uint32_t *timeout* = DEFAULT_TIMEOUT) [pure virtual]

Increase the scanning frequency by 0.1 HZ
.

Parameters

in	*frequency*	scanning frequency
in	*timeout*	timeout

Returns

return status

Return values

RESULT_OK	success
RESULT_FAILE	failed

Note

Non-scan state, perform currect operation.

Implemented in ydlidar::YDlidarDriver, and ydlidar::ETLidarDriver.

38.15.4.25 virtual result_t ydlidar::core::common::DriverInterface::setScanFrequencyDis (scan_frequency & *frequency*, uint32_t *timeout* = DEFAULT_TIMEOUT) [pure virtual]

Reduce the scanning frequency by 1.0 HZ
.

Parameters

in	*frequency*	scanning frequency
in	*timeout*	timeout

Returns

return status

Return values

RESULT_OK	success
RESULT_FAILE	failed

Note

Non-scan state, perform currect operation.

Implemented in ydlidar::YDlidarDriver, and ydlidar::ETLidarDriver.

38.15.4.26 virtual result_t ydlidar::core::common::DriverInterface::setScanFrequencyDisMic (scan_frequency & *frequency*, uint32_t *timeout* = DEFAULT_TIMEOUT) [pure virtual]

Reduce the scanning frequency by 0.1 HZ
.

Parameters

in	*frequency*	scanning frequency
in	*timeout*	timeout

Returns

return status

Return values

RESULT_OK	success
RESULT_FAILE	failed

Note

Non-scan state, perform currect operation.

Implemented in ydlidar::YDlidarDriver, and ydlidar::ETLidarDriver.

38.15.4.27 void ydlidar::core::common::DriverInterface::setSingleChannel (bool *v*) [inline]

Definition at line 37 of file DriverInterface.h.

38.15.4.28 void ydlidar::core::common::DriverInterface::setSupportMotorDtrCtrl (bool *v*) [inline]

Definition at line 81 of file DriverInterface.h.

38.15.4.29 virtual result_t ydlidar::core::common::DriverInterface::startScan (bool *force* = false, uint32_t *timeout* = DEFAULT_TIMEOUT) [pure virtual]

Turn on scanning
.

Parameters

in	*force*	Scan mode
in	*timeout*	timeout

Returns

result status

Return values

RESULT_OK	success
RESULT_FAILE	failed

Note

Just turn it on once

Implemented in ydlidar::YDlidarDriver, and ydlidar::ETLidarDriver.

38.15.4.30 virtual result_t ydlidar::core::common::DriverInterface::stop ( ) [pure virtual]

turn off scanning

Returns

result status

Return values

RESULT_OK	success
RESULT_FAILE	failed

Implemented in ydlidar::YDlidarDriver, and ydlidar::ETLidarDriver.

38.15.5 Member Data Documentation

38.15.5.1 Locker ydlidar::core::common::DriverInterface::_cmd_lock [protected]

command locker

Definition at line 394 of file DriverInterface.h.

38.15.5.2 Event ydlidar::core::common::DriverInterface::_dataEvent [protected]

Scan Data Event.

Definition at line 388 of file DriverInterface.h.

38.15.5.3 Locker ydlidar::core::common::DriverInterface::_lock [protected]

Data Locker.

Definition at line 390 of file DriverInterface.h.

38.15.5.4 Thread ydlidar::core::common::DriverInterface::_thread [protected]

Parse Data thread.

Definition at line 392 of file DriverInterface.h.

38.15.5.5 bool ydlidar::core::common::DriverInterface::isAutoconnting [protected]

auto connecting state

Definition at line 414 of file DriverInterface.h.

38.15.5.6 bool ydlidar::core::common::DriverInterface::isAutoReconnect [protected]

auto reconnect

Definition at line 412 of file DriverInterface.h.

38.15.5.7 uint32_t ydlidar::core::common::DriverInterface::m_baudrate [protected]

baudrate or IP port

Definition at line 399 of file DriverInterface.h.

38.15.5.8 lidarConfig ydlidar::core::common::DriverInterface::m_config [protected]

Definition at line 415 of file DriverInterface.h.

38.15.5.9 bool ydlidar::core::common::DriverInterface::m_intensities [protected]

LiDAR intensity.

Definition at line 401 of file DriverInterface.h.

38.15.5.10 bool ydlidar::core::common::DriverInterface::m_isConnected [protected]

LiDAR connected state.

Definition at line 386 of file DriverInterface.h.

38.15.5.11 bool ydlidar::core::common::DriverInterface::m_isScanning [protected]

LiDAR Scanning state.

Definition at line 384 of file DriverInterface.h.

38.15.5.12 int ydlidar::core::common::DriverInterface::m_LidarType [protected]

Set and Get LiDAR Type.

Note

Refer to the table below for the LiDAR Type.
Set the LiDAR Type to match the LiDAR.

G1/G2A/G2/G2C	TYPE_TRIANGLE
G4/G4B/G4C/G4PRO	TYPE_TRIANGLE
G6/F4/F4PRO	TYPE_TRIANGLE
S4/S4B/X4/R2/S2/X2/X2L	TYPE_TRIANGLE
TG15/TG30/TG50/TX8/TX20	TYPE_TOF
T5/T15	TYPE_TOF_NET

Remarks

See also

LidarTypeID
DriverInterface::setLidarType and DriverInterface::getLidarType

Definition at line 37 of file DriverInterface.h.

38.15.5.13 uint32_t ydlidar::core::common::DriverInterface::m_PointTime [protected]

Set and Get Sampling interval.

Note

Negative correlation between sampling interval and lidar sampling rate.
sampling interval = 1e9 / sampling rate(/s)
Set the LiDAR sampling interval to match the LiDAR.

See also

DriverInterface::setPointTime and DriverInterface::getPointTime

Definition at line 54 of file DriverInterface.h.

38.15.5.14 bool ydlidar::core::common::DriverInterface::m_SingleChannel [protected]

Set and Get LiDAR single channel. Whether LiDAR communication channel is a single-channel.

Note

For a single-channel LiDAR, if the settings are reversed.
an error will occur in obtaining device information and the LiDAR will Faied to Start.
For dual-channel LiDAR, if th setttings are reversed.
the device information cannot be obtained.
Set the single channel to match the LiDAR.

G1/G2/G2A/G2C	false
G4/G4B/G4PRO/G6/F4/F4PRO	false
S4/S4B/X4/R2/G4C	false
S2/X2/X2L	true
TG15/TG30/TG50	false
TX8/TX20	true
T5/T15	false
	true

Remarks

See also

DriverInterface::setSingleChannel and DriverInterface::getSingleChannel

Definition at line 37 of file DriverInterface.h.

38.15.5.15 bool ydlidar::core::common::DriverInterface::m_SupportMotorDtrCtrl [protected]

Set and Get LiDAR Support Motor DTR.

Note

The current paramter settings are only valid if the LiDAR is connected to the serial port adapter via USB.
If the LiDAR does not have external motor enable line, the current paramters do not need to be set.
Set the LiDAR Motro DTR to match the LiDAR.

S4/S4B/S2/X2/X2L/X4	true
TX8/TX20	true
G4/G4C/G4PRO/F4/F4PRO/G6	false
G1/G2A/G2C/R2/G2/G4B	false
TG15/TG30/TG50	false
T5/T15	false

Remarks

See also

DriverInterface::setSupportMotorDtrCtrl and DriverInterface::getSupportMotorDtrCtrl

Definition at line 62 of file DriverInterface.h.

38.15.5.16 uint16_t ydlidar::core::common::DriverInterface::package_Sample_Index [protected]

package sample index

Definition at line 408 of file DriverInterface.h.

38.15.5.17 int ydlidar::core::common::DriverInterface::retryCount [protected]

Definition at line 410 of file DriverInterface.h.

38.15.5.18 node_info* ydlidar::core::common::DriverInterface::scan_node_buf [protected]

LiDAR Point pointer.

Definition at line 404 of file DriverInterface.h.

38.15.5.19 size_t ydlidar::core::common::DriverInterface::scan_node_count [protected]

LiDAR scan count.

Definition at line 406 of file DriverInterface.h.

38.15.5.20 std::string ydlidar::core::common::DriverInterface::serial_port [protected]

LiDAR com port or IP Address.

Definition at line 397 of file DriverInterface.h.

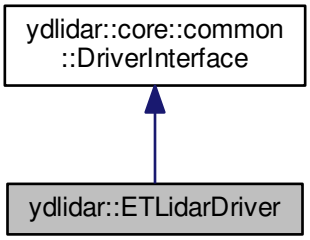
The documentation for this class was generated from the following file:

- core/common/DriverInterface.h

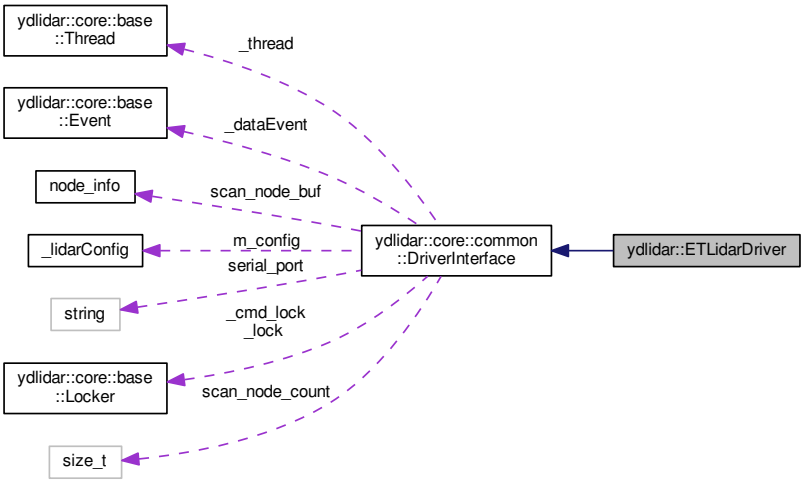
38.16 ydlidar::ETLidarDriver Class Reference

`#include <ETLidarDriver.h>`

Inheritance diagram for ydlidar::ETLidarDriver:



Collaboration diagram for ydlidar::ETLidarDriver:



Public Member Functions

- ETLidarDriver ()

  ETLidarDriver.
- ~ETLidarDriver ()
- virtual result_t connect (const char *port_path, uint32_t baudrate=8000)

  Connecting Lidar
  After the connection if successful, you must use ::disconnect to close.
- virtual const char * DescribeError (bool isTCP=true)

Returns a human-readable description of the given error code or the last error code of a socket.

- virtual void disconnect ()

  Disconnect from ETLidar device.

- virtual std::string getSDKVersion ()

  Get SDK Version.

- virtual bool isscanning () const

  Is the Lidar in the scan

  .

- virtual bool isconnected () const

  Is it connected to the lidar

  .

- virtual void setIntensities (const bool &isintensities)

  Is there intensity

  .

- virtual void setAutoReconnect (const bool &enable)

  whether to support hot plug

- virtual result_t getHealth (device_health &health, uint32_t timeout=DEFAULT_TIMEOUT)

  get Health status

- virtual result_t getDeviceInfo (device_info &info, uint32_t timeout=DEFAULT_TIMEOUT)

  get Device information

- virtual result_t startScan (bool force=false, uint32_t timeout=DEFAULT_TIMEOUT)

  Turn on scanning

  .

- virtual result_t stop ()

  turn off scanning

- virtual result_t grabScanData (node_info *nodebuffer, size_t &count, uint32_t timeout=DEFAULT_TIMEOUT)

  Get a circle of laser data

  .

- virtual result_t getScanFrequency (scan_frequency &frequency, uint32_t timeout=DEFAULT_TIMEOUT)

  Get lidar scan frequency

  .

- virtual result_t setScanFrequencyAdd (scan_frequency &frequency, uint32_t timeout=DEFAULT_TIMEOUT)

  Increase the scanning frequency by 1.0 HZ

  .

- virtual result_t setScanFrequencyDis (scan_frequency &frequency, uint32_t timeout=DEFAULT_TIMEOUT)

  Reduce the scanning frequency by 1.0 HZ

  .

- virtual result_t setScanFrequencyAddMic (scan_frequency &frequency, uint32_t timeout=DEFAULT_TIME↩OUT)

  Increase the scanning frequency by 0.1 HZ

  .

- virtual result_t setScanFrequencyDisMic (scan_frequency &frequency, uint32_t timeout=DEFAULT_TIME↩OUT)

  Reduce the scanning frequency by 0.1 HZ

  .

- virtual result_t getSamplingRate (sampling_rate &rate, uint32_t timeout=DEFAULT_TIMEOUT)

  Get lidar sampling frequency

  .

- virtual result_t setSamplingRate (sampling_rate &rate, uint32_t timeout=DEFAULT_TIMEOUT)

Set the lidar sampling frequency

.

- virtual result_t getZeroOffsetAngle (offset_angle &angle, uint32_t timeout=DEFAULT_TIMEOUT)

  get lidar zero offset angle

- bool getScanCfg (lidarConfig &config, const std::string &ip_address="")

  Get current scan configuration.

- lidarConfig getFinishedScanCfg ()

  Get current scan update configuration.

- void updateScanCfg (const lidarConfig &config)

  updateScanCfg

Additional Inherited Members

38.16.1 Detailed Description

Definition at line 67 of file ETLidarDriver.h.

38.16.2 Constructor & Destructor Documentation

38.16.2.1 ETLidarDriver::ETLidarDriver ( ) [explicit]

ETLidarDriver.

Parameters

lidar↩ IP	
port	

Definition at line 51 of file ETLidarDriver.cpp.

38.16.2.2 ETLidarDriver::~ETLidarDriver ( )

Definition at line 88 of file ETLidarDriver.cpp.

38.16.3 Member Function Documentation

38.16.3.1 result_t ETLidarDriver::connect (const char * *port_path*, uint32_t *baudrate* = 8000) [virtual]

Connecting Lidar
After the connection if successful, you must use ::disconnect to close.

Parameters

in	*port_path*	Ip Address
in	*baudrate*	network port

Returns

connection status

Return values

0	success
<	0 failed

Note

After the connection if successful, you must use ::disconnect to close

See also

function ::YDlidarDriver::disconnect ()

Implements ydlidar::core::common::DriverInterface.

Definition at line 121 of file ETLidarDriver.cpp.

38.16.3.2 const char * ETLidarDriver::DescribeError (bool *isTCP* = true) [virtual]

Returns a human-readable description of the given error code or the last error code of a socket.

Parameters

isTCP	TCP or UDP

Returns

error information

Implements ydlidar::core::common::DriverInterface.

Definition at line 178 of file ETLidarDriver.cpp.

38.16.3.3 void ETLidarDriver::disconnect ( ) [virtual]

Disconnect from ETLidar device.

Implements ydlidar::core::common::DriverInterface.

Definition at line 230 of file ETLidarDriver.cpp.

38.16.3.4 result_t ETLidarDriver::getDeviceInfo (device_info & *info*, uint32_t *timeout* = DEFAULT_TIMEOUT) [virtual]

get Device information

Parameters

in	*info*	Device information
in	*timeout*	timeout

Returns

result status

Return values

RESULT_OK	success
RESULT_FAILE	or RESULT_TIMEOUT failed

Implements ydlidar::core::common::DriverInterface.

Definition at line 256 of file ETLidarDriver.cpp.

38.16.3.5 lidarConfig ETLidarDriver::getFinishedScanCfg ( )

Get current scan update configuration.

Returns

scanCfg structure.

Definition at line 388 of file ETLidarDriver.cpp.

38.16.3.6 result_t ETLidarDriver::getHealth (device_health & *health*, uint32_t *timeout* = DEFAULT_TIMEOUT) [virtual]

get Health status

Returns

result status

Return values

RESULT_OK	success
RESULT_FAILE	or RESULT_TIMEOUT failed

Implements ydlidar::core::common::DriverInterface.

Definition at line 246 of file ETLidarDriver.cpp.

38.16.3.7 result_t ETLidarDriver::getSamplingRate (sampling_rate & *rate,* uint32_t *timeout* = DEFAULT_TIMEOUT) [virtual]

Get lidar sampling frequency
.

Parameters

in	*frequency*	sampling frequency
in	*timeout*	timeout

Returns

return status

Return values

RESULT_OK	success
RESULT_FAILE	failed

Note

Non-scan state, perform currect operation.

Implements ydlidar::core::common::DriverInterface.

Definition at line 832 of file ETLidarDriver.cpp.

38.16.3.8 bool ETLidarDriver::getScanCfg (lidarConfig & *config,* const std::string & *ip_address* = " ")

Get current scan configuration.

Returns

scanCfg structure.

Definition at line 392 of file ETLidarDriver.cpp.

38.16.3.9 result_t ETLidarDriver::getScanFrequency (scan_frequency & *frequency,* uint32_t *timeout* = DEFAULT_TIMEOUT) [virtual]

Get lidar scan frequency
.

Parameters

in	*frequency*	scanning frequency
in	*timeout*	timeout

Returns

return status

Return values

RESULT_OK	success
RESULT_FAILE	failed

Note

Non-scan state, perform currect operation.

Implements ydlidar::core::common::DriverInterface.

Definition at line 668 of file ETLidarDriver.cpp.

38.16.3.10 std::string ETLidarDriver::getSDKVersion ( ) [virtual]

Get SDK Version.

Returns

version

Implements ydlidar::core::common::DriverInterface.

Definition at line 242 of file ETLidarDriver.cpp.

38.16.3.11 result_t ETLidarDriver::getZeroOffsetAngle (offset_angle & *angle*, uint32_t *timeout* = DEFAULT_TIMEOUT) [virtual]

get lidar zero offset angle

Parameters

in	*angle*	zero offset angle
in	*timeout*	timeout

Returns

return status

Return values

RESULT_OK	success
RESULT_FAILE	failed

Note

Non-scan state, perform currect operation.

Implements ydlidar::core::common::DriverInterface.

Definition at line 862 of file ETLidarDriver.cpp.

38.16.3.12 result_t ETLidarDriver::grabScanData (node_info * *nodebuffer*, size_t & *count*, uint32_t *timeout* = DEFAULT_TIMEOUT) [virtual]

Get a circle of laser data
.

Parameters

in	*nodebuffer*	Laser data
in	*count*	one circle of laser points
in	*timeout*	timeout

Returns

return status

Return values

RESULT_OK	success
RESULT_FAILE	failed

Note

Before starting, you must start the start the scan successfully with the ::startScan function

Implements ydlidar::core::common::DriverInterface.

Definition at line 640 of file ETLidarDriver.cpp.

38.16.3.13 bool ETLidarDriver::isconnected ( ) const [virtual]

Is it connected to the lidar
.

Returns

connection status

Return values

true	connected
false	Non-connected

Implements ydlidar::core::common::DriverInterface.

Definition at line 162 of file ETLidarDriver.cpp.

38.16.3.14 bool ETLidarDriver::isscanning ( ) const [virtual]

Is the Lidar in the scan
.

Returns

scanning status

Return values

true	scanning
false	non-scanning

Implements ydlidar::core::common::DriverInterface.

Definition at line 166 of file ETLidarDriver.cpp.

38.16.3.15 void ETLidarDriver::setAutoReconnect (const bool & *enable*) [virtual]

whether to support hot plug

Parameters

in	*enable*	hot plug : true support false no support

Implements ydlidar::core::common::DriverInterface.

Definition at line 174 of file ETLidarDriver.cpp.

38.16.3.16 void ETLidarDriver::setIntensities (const bool & *isintensities*) [virtual]

Is there intensity
.

Parameters

in	*isintensities*	intentsity true intensity false no intensity

Implements ydlidar::core::common::DriverInterface.

Definition at line 170 of file ETLidarDriver.cpp.

38.16.3.17 result_t ETLidarDriver::setSamplingRate (sampling_rate & *rate,* uint32_t *timeout* = DEFAULT_TIMEOUT) [virtual]

Set the lidar sampling frequency
.

Parameters

in	*rate*	sampling frequency
in	*timeout*	timeout

Returns

return status

Return values

RESULT_OK	success
RESULT_FAILE	failed

Note

Non-scan state, perform currect operation.

Implements ydlidar::core::common::DriverInterface.

Definition at line 858 of file ETLidarDriver.cpp.

38.16.3.18 result_t ETLidarDriver::setScanFrequencyAdd (scan_frequency & *frequency,* uint32_t *timeout* = DEFAULT_TIMEOUT) [virtual]

Increase the scanning frequency by 1.0 HZ
.

Parameters

in	*frequency*	scanning frequency
in	*timeout*	timeout

Returns

return status

Return values

RESULT_OK	success
RESULT_FAILE	failed

Note

Non-scan state, perform currect operation.

Implements ydlidar::core::common::DriverInterface.

Definition at line 689 of file ETLidarDriver.cpp.

38.16.3.19 result_t ETLidarDriver::setScanFrequencyAddMic (scan_frequency & *frequency*, uint32_t *timeout* = DEFAULT_TIMEOUT) [virtual]

Increase the scanning frequency by 0.1 HZ
.

Parameters

in	*frequency*	scanning frequency
in	*timeout*	timeout

Returns

return status

Return values

RESULT_OK	success
RESULT_FAILE	failed

Note

Non-scan state, perform currect operation.

Implements ydlidar::core::common::DriverInterface.

Definition at line 760 of file ETLidarDriver.cpp.

38.16.3.20 result_t ETLidarDriver::setScanFrequencyDis (scan_frequency & *frequency*, uint32_t *timeout* = DEFAULT_TIMEOUT) [virtual]

Reduce the scanning frequency by 1.0 HZ
.

Parameters

in	*frequency*	scanning frequency
in	*timeout*	timeout

Returns

return status

Return values

RESULT_OK	success
RESULT_FAILE	failed

Note

Non-scan state, perform currect operation.

Implements ydlidar::core::common::DriverInterface.

Definition at line 725 of file ETLidarDriver.cpp.

38.16.3.21 result_t ETLidarDriver::setScanFrequencyDisMic (scan_frequency & *frequency*, uint32_t *timeout* = DEFAULT_TIMEOUT) [virtual]

Reduce the scanning frequency by 0.1 HZ
.

Parameters

in	*frequency*	scanning frequency
in	*timeout*	timeout

Returns

return status

Return values

RESULT_OK	success
RESULT_FAILE	failed

Note

Non-scan state, perform currect operation.

Implements ydlidar::core::common::DriverInterface.

Definition at line 797 of file ETLidarDriver.cpp.

38.16.3.22 result_t ETLidarDriver::startScan (bool *force* = false, uint32_t *timeout* = DEFAULT_TIMEOUT) [virtual]

Turn on scanning
.

Parameters

in	*force*	Scan mode
in	*timeout*	timeout

Returns

result status

Return values

RESULT_OK	success
RESULT_FAILE	failed

Note

Just turn it on once

Implements ydlidar::core::common::DriverInterface.

Definition at line 266 of file ETLidarDriver.cpp.

38.16.3.23 result_t ETLidarDriver::stop ( ) [virtual]

turn off scanning

Returns

result status

Return values

RESULT_OK	success
RESULT_FAILE	failed

Implements ydlidar::core::common::DriverInterface.

Definition at line 292 of file ETLidarDriver.cpp.

38.16.3.24 void ETLidarDriver::updateScanCfg (const lidarConfig & *config*)

updateScanCfg

Parameters

config	

Definition at line 111 of file ETLidarDriver.cpp.

The documentation for this class was generated from the following files:

- src/ETLidarDriver.h
- src/ETLidarDriver.cpp

38.17 ydlidar::core::base::Event Class Reference

`#include <locker.h>`

Public Types

- enum { EVENT_OK = 1, EVENT_TIMEOUT = 2, EVENT_FAILED = 0 }

Public Member Functions

- Event (bool isAutoReset=true, bool isSignal=false)
- ~Event ()
- void set (bool isSignal=true)
- unsigned long wait (unsigned long timeout=0xFFFFFFFF)

Protected Member Functions

- void release ()

Protected Attributes

- pthread_condattr_t _cond_cattr
- pthread_cond_t _cond_var
- pthread_mutex_t _cond_locker
- bool _is_signalled
- bool _isAutoReset

38.17.1 Detailed Description

Definition at line 187 of file locker.h.

38.17.2 Member Enumeration Documentation

38.17.2.1 anonymous enum

Enumerator

EVENT_OK

EVENT_TIMEOUT

EVENT_FAILED

Definition at line 190 of file locker.h.

38.17.3 Constructor & Destructor Documentation

38.17.3.1 ydlidar::core::base::Event::Event (bool *isAutoReset* = `true`, bool *isSignal* = `false`) [inline], [explicit]

Definition at line 196 of file locker.h.

38.17.3.2 ydlidar::core::base::Event::~Event ( ) [inline]

Definition at line 221 of file locker.h.

38.17.4 Member Function Documentation

38.17.4.1 void ydlidar::core::base::Event::release ( ) [inline], [protected]

Definition at line 320 of file locker.h.

38.17.4.2 void ydlidar::core::base::Event::set (bool *isSignal* = `true`) [inline]

Definition at line 225 of file locker.h.

38.17.4.3 unsigned long ydlidar::core::base::Event::wait (unsigned long *timeout* = `0xFFFFFFFF`) [inline]

Definition at line 250 of file locker.h.

38.17.5 Member Data Documentation

38.17.5.1 pthread_condattr_t ydlidar::core::base::Event::_cond_cattr [protected]

Definition at line 333 of file locker.h.

38.17.5.2 pthread_mutex_t ydlidar::core::base::Event::_cond_locker [protected]

Definition at line 335 of file locker.h.

38.17.5.3 pthread_cond_t ydlidar::core::base::Event::_cond_var [protected]

Definition at line 334 of file locker.h.

38.17.5.4 bool ydlidar::core::base::Event::_is_signalled [protected]

Definition at line 336 of file locker.h.

38.17.5.5 bool ydlidar::core::base::Event::_isAutoReset [protected]

Definition at line 337 of file locker.h.

The documentation for this class was generated from the following file:

- core/base/locker.h

38.18 function_state Struct Reference

`#include <ydlidar_protocol.h>`

Public Attributes

- uint8_t state

38.18.1 Detailed Description

Definition at line 235 of file ydlidar_protocol.h.

38.18.2 Member Data Documentation

38.18.2.1 uint8_t function_state::state

Definition at line 236 of file ydlidar_protocol.h.

The documentation for this struct was generated from the following file:

- core/common/ydlidar_protocol.h

38.19 LaserConfig Struct Reference

A struct for returning configuration from the YDLIDAR.

`#include <ydlidar_def.h>`

Public Attributes

- float min_angle

  Start angle for the laser scan [rad]. 0 is forward and angles are measured clockwise when viewing YDLIDAR from the top.
- float max_angle

  Stop angle for the laser scan [rad]. 0 is forward and angles are measured clockwise when viewing YDLIDAR from the top.
- float angle_increment

  angle resoltuion [rad]
- float time_increment

  Scan resoltuion [s].
- float scan_time

  Time between scans.
- float min_range

  Minimum range [m].
- float max_range

  Maximum range [m].

38.19.1 Detailed Description

A struct for returning configuration from the YDLIDAR.

Note

angle unit: rad.
time unit: second.
range unit: meter.

Definition at line 107 of file ydlidar_def.h.

38.19.2 Member Data Documentation

38.19.2.1 float LaserConfig::angle_increment

angle resoltuion [rad]

Definition at line 113 of file ydlidar_def.h.

38.19.2.2 float LaserConfig::max_angle

Stop angle for the laser scan [rad]. 0 is forward and angles are measured clockwise when viewing YDLIDAR from the top.

Definition at line 111 of file ydlidar_def.h.

38.19.2.3 float LaserConfig::max_range

Maximum range [m].

Definition at line 121 of file ydlidar_def.h.

38.19.2.4 float LaserConfig::min_angle

Start angle for the laser scan [rad]. 0 is forward and angles are measured clockwise when viewing YDLIDAR from the top.

Definition at line 109 of file ydlidar_def.h.

38.19.2.5 float LaserConfig::min_range

Minimum range [m].

Definition at line 119 of file ydlidar_def.h.

38.19.2.6 float LaserConfig::scan_time

Time between scans.

Definition at line 117 of file ydlidar_def.h.

38.19.2.7 float LaserConfig::time_increment

Scan resoltuion [s].

Definition at line 115 of file ydlidar_def.h.

The documentation for this struct was generated from the following file:

- core/common/ydlidar_def.h

38.20 LaserDebug Struct Reference

The Laser Debug struct.

`#include <ydlidar_datatype.h>`

Public Attributes

- uint8_t W3F4CusMajor_W4F0CusMinor
- uint8_t W4F3Model_W3F0DebugInfTranVer
- uint8_t W3F4HardwareVer_W4F0FirewareMajor
- uint8_t W7F0FirewareMinor
- uint8_t W3F4BoradHardVer_W4F0Moth
- uint8_t W2F5Output2K4K5K_W5F0Date
- uint8_t W1F6GNoise_W1F5SNoise_W1F4MotorCtl_W4F0SnYear
- uint8_t W7F0SnNumH
- uint8_t W7F0SnNumL
- uint8_t W7F0Health
- uint8_t W3F4CusHardVer_W4F0CusSoftVer
- uint8_t W7F0LaserCurrent
- uint8_t MaxDebugIndex

38.20.1 Detailed Description

The Laser Debug struct.

Definition at line 42 of file ydlidar_datatype.h.

38.20.2 Member Data Documentation

38.20.2.1 uint8_t LaserDebug::MaxDebugIndex

Definition at line 55 of file ydlidar_datatype.h.

38.20.2.2 uint8_t LaserDebug::W1F6GNoise_W1F5SNoise_W1F4MotorCtl_W4F0SnYear

Definition at line 49 of file ydlidar_datatype.h.

38.20.2.3 uint8_t LaserDebug::W2F5Output2K4K5K_W5F0Date

Definition at line 48 of file ydlidar_datatype.h.

38.20.2.4 uint8_t LaserDebug::W3F4BoradHardVer_W4F0Moth

Definition at line 47 of file ydlidar_datatype.h.

38.20.2.5 uint8_t LaserDebug::W3F4CusHardVer_W4F0CusSoftVer

Definition at line 53 of file ydlidar_datatype.h.

38.20.2.6 uint8_t LaserDebug::W3F4CusMajor_W4F0CusMinor

Definition at line 43 of file ydlidar_datatype.h.

38.20.2.7 uint8_t LaserDebug::W3F4HardwareVer_W4F0FirewareMajor

Definition at line 45 of file ydlidar_datatype.h.

38.20.2.8 uint8_t LaserDebug::W4F3Model_W3F0DebugInfTranVer

Definition at line 44 of file ydlidar_datatype.h.

38.20.2.9 uint8_t LaserDebug::W7F0FirewareMinor

Definition at line 46 of file ydlidar_datatype.h.

38.20.2.10 uint8_t LaserDebug::W7F0Health

Definition at line 52 of file ydlidar_datatype.h.

38.20.2.11 uint8_t LaserDebug::W7F0LaserCurrent

Definition at line 54 of file ydlidar_datatype.h.

38.20.2.12 uint8_t LaserDebug::W7F0SnNumH

Definition at line 50 of file ydlidar_datatype.h.

38.20.2.13 uint8_t LaserDebug::W7F0SnNumL

Definition at line 51 of file ydlidar_datatype.h.

The documentation for this struct was generated from the following file:

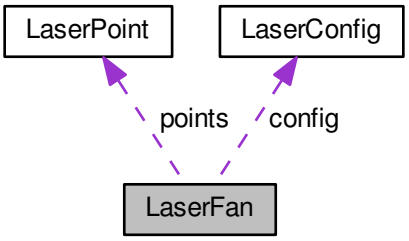
- core/common/ydlidar_datatype.h

38.21 LaserFan Struct Reference

The Laser Scan Data struct.

`#include <ydlidar_def.h>`

Collaboration diagram for LaserFan:



Public Attributes

- uint64_t stamp

  System time when first range was measured in nanoseconds.
- uint32_t npoints

  Array of lidar points.
- LaserPoint * points
- LaserConfig config

  Configuration of scan.

38.21.1 Detailed Description

The Laser Scan Data struct.

usage

```
LaserScan data;
for(int i = 0; i < data.npoints; i++) {
 //current LiDAR angle
 float angle = data.points[i].angle;
 //current LiDAR range
 float range = data.points[i].range;
 //current LiDAR intensity
 float intensity = data.points[i].intensity;
 //current LiDAR point stamp
 uint64_t timestamp = data.stamp + i * data.config.time_increment * 1e9;
}
LaserScanDestroy(&data);
```

convert to ROS sensor_msgs::LaserScan

```
LaserScan scan;
sensor_msgs::LaserScan scan_msg;
std::string frame_id = "laser_frame";
ros::Time start_scan_time;
start_scan_time.sec = scan.stamp/1000000000ul;
start_scan_time.nsec = scan.stamp%1000000000ul;
scan_msg.header.stamp = start_scan_time;
scan_msg.header.frame_id = frame_id;
scan_msg.angle_min =(scan.config.min_angle);
scan_msg.angle_max = (scan.config.max_angle);
scan_msg.angle_increment = (scan.config.angle_increment);
scan_msg.scan_time = scan.config.scan_time;
scan_msg.time_increment = scan.config.time_increment;
scan_msg.range_min = (scan.config.min_range);
scan_msg.range_max = (scan.config.max_range);
int size = (scan.config.max_angle - scan.config.min_angle)/ scan.
      config.angle_increment + 1;
scan_msg.ranges.resize(size);
scan_msg.intensities.resize(size);
for(int i=0; i < scan.npoints; i++) {
 int index = std::ceil((scan.points[i].angle - scan.config.min_angle)/scan.
       config.angle_increment);
 if(index >=0 && index < size) {
     scan_msg.ranges[index] = scan.points[i].range;
     scan_msg.intensities[index] = scan.points[i].intensity;
 }
}
LaserScanDestroy(&scan);
```

Definition at line 173 of file ydlidar_def.h.

38.21.2 Member Data Documentation

38.21.2.1 LaserConfig LaserFan::config

Configuration of scan.

Definition at line 180 of file ydlidar_def.h.

38.21.2.2 uint32_t LaserFan::npoints

Array of lidar points.

Definition at line 177 of file ydlidar_def.h.

38.21.2.3 LaserPoint* LaserFan::points

Definition at line 178 of file ydlidar_def.h.

38.21.2.4 uint64_t LaserFan::stamp

System time when first range was measured in nanoseconds.

ns

Definition at line 175 of file ydlidar_def.h.

The documentation for this struct was generated from the following file:

- core/common/ydlidar_def.h

38.22 LaserPoint Struct Reference

The Laser Point struct.

`#include <ydlidar_def.h>`

Public Attributes

- float angle
  lidar angle. unit(rad)
- float range
  lidar range. unit(m)
- float intensity
  lidar intensity

38.22.1 Detailed Description

The Laser Point struct.

Note

angle unit: rad.
range unit: meter.

Definition at line 92 of file ydlidar_def.h.

38.22.2 Member Data Documentation

38.22.2.1 float LaserPoint::angle

lidar angle. unit(rad)

Definition at line 94 of file ydlidar_def.h.

38.22.2.2 float LaserPoint::intensity

lidar intensity

Definition at line 98 of file ydlidar_def.h.

38.22.2.3 float LaserPoint::range

lidar range. unit(m)

Definition at line 96 of file ydlidar_def.h.

The documentation for this struct was generated from the following file:

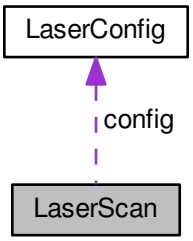
- core/common/ydlidar_def.h

38.23 LaserScan Struct Reference

The Laser Scan Data struct.

`#include <ydlidar_datatype.h>`

Collaboration diagram for LaserScan:



Public Attributes

- uint64_t stamp

  System time when first range was measured in nanoseconds.
- std::vector< LaserPoint > points

  Array of lidar points.
- LaserConfig config

  Configuration of scan.

38.23.1 Detailed Description

The Laser Scan Data struct.

usage

```
LaserScan data;
for(int i = 0; i < data.points.size(); i++) {
 //current LiDAR angle
 float angle = data.points[i].angle;
 //current LiDAR range
 float range = data.points[i].range;
 //current LiDAR intensity
 float intensity = data.points[i].intensity;
 //current LiDAR point stamp
 uint64_t timestamp = data.stamp + i * data.config.time_increment * 1e9;
}
LaserScanDestroy(&data);
```

convert to ROS sensor_msgs::LaserScan

```
LaserScan scan;
sensor_msgs::LaserScan scan_msg;
std::string frame_id = "laser_frame";
ros::Time start_scan_time;
start_scan_time.sec = scan.stamp/1000000000ul;
start_scan_time.nsec = scan.stamp%1000000000ul;
scan_msg.header.stamp = start_scan_time;
scan_msg.header.frame_id = frame_id;
scan_msg.angle_min =(scan.config.min_angle);
scan_msg.angle_max = (scan.config.max_angle);
scan_msg.angle_increment = (scan.config.angle_increment);
scan_msg.scan_time = scan.config.scan_time;
scan_msg.time_increment = scan.config.time_increment;
scan_msg.range_min = (scan.config.min_range);
scan_msg.range_max = (scan.config.max_range);
int size = (scan.config.max_angle - scan.config.min_angle)/ scan.
      config.angle_increment + 1;
scan_msg.ranges.resize(size);
scan_msg.intensities.resize(size);
for(int i=0; i < scan.points.size(); i++) {
 int index = std::ceil((scan.points[i].angle - scan.config.min_angle)/scan.
      config.angle_increment);
 if(index >=0 && index < size) {
     scan_msg.ranges[index] = scan.points[i].range;
     scan_msg.intensities[index] = scan.points[i].intensity;
 }
}
```

Definition at line 106 of file ydlidar_datatype.h.

38.23.2 Member Data Documentation

38.23.2.1 LaserConfig LaserScan::config

Configuration of scan.

Definition at line 112 of file ydlidar_datatype.h.

38.23.2.2 std::vector<LaserPoint> LaserScan::points

Array of lidar points.

Definition at line 110 of file ydlidar_datatype.h.

38.23.2.3 uint64_t LaserScan::stamp

System time when first range was measured in nanoseconds.

Definition at line 108 of file ydlidar_datatype.h.

The documentation for this struct was generated from the following file:

- core/common/ydlidar_datatype.h

38.24 lidar_ans_header Struct Reference

LiDAR response Header.

`#include <ydlidar_protocol.h>`

Public Attributes

- uint8_t syncByte1
- uint8_t syncByte2
- uint32_t size: 30
- uint32_t subType: 2
- uint8_t type

38.24.1 Detailed Description

LiDAR response Header.

Definition at line 253 of file ydlidar_protocol.h.

38.24.2 Member Data Documentation

38.24.2.1 uint32_t lidar_ans_header::size

Definition at line 256 of file ydlidar_protocol.h.

38.24.2.2 uint32_t lidar_ans_header::subType

Definition at line 257 of file ydlidar_protocol.h.

38.24.2.3 uint8_t lidar_ans_header::syncByte1

Definition at line 254 of file ydlidar_protocol.h.

38.24.2.4 uint8_t lidar_ans_header::syncByte2

Definition at line 255 of file ydlidar_protocol.h.

38.24.2.5 uint8_t lidar_ans_header::type

Definition at line 258 of file ydlidar_protocol.h.

The documentation for this struct was generated from the following file:

- core/common/ydlidar_protocol.h

38.25 lidarConfig Class Reference

Structure containing scan configuration.

`#include <ydlidar_protocol.h>`

38.25.1 Detailed Description

Structure containing scan configuration.

Author

jzhang

The documentation for this class was generated from the following file:

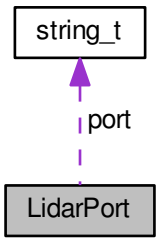
- core/common/ydlidar_protocol.h

38.26 LidarPort Struct Reference

lidar ports

`#include <ydlidar_def.h>`

Collaboration diagram for LidarPort:



Public Attributes

- string_t port [8]

38.26.1 Detailed Description

lidar ports

Definition at line 194 of file ydlidar_def.h.

38.26.2 Member Data Documentation

38.26.2.1 string_t LidarPort::port[8]

Definition at line 195 of file ydlidar_def.h.

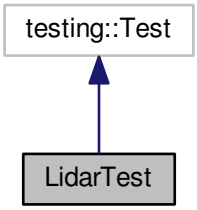
The documentation for this struct was generated from the following file:

- core/common/ydlidar_def.h

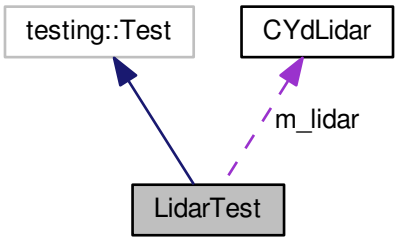
38.27 LidarTest Class Reference

`#include <lidar_test.h>`

Inheritance diagram for LidarTest:



Collaboration diagram for LidarTest:



Protected Member Functions

- LidarTest ()
- virtual ~LidarTest ()
- virtual void SetUp ()
- virtual void TearDown ()

Protected Attributes

- CYdLidar m_lidar

38.27.1 Detailed Description

Definition at line 8 of file lidar_test.h.

38.27.2 Constructor & Destructor Documentation

38.27.2.1 LidarTest::LidarTest ( ) [protected]

Definition at line 3 of file lidar_test.cpp.

38.27.2.2 LidarTest::~LidarTest ( ) [protected], [virtual]

Definition at line 17 of file lidar_test.cpp.

38.27.3 Member Function Documentation

38.27.3.1 void LidarTest::SetUp ( ) [protected], [virtual]

Definition at line 19 of file lidar_test.cpp.

38.27.3.2 void LidarTest::TearDown ( ) [protected], [virtual]

Definition at line 21 of file lidar_test.cpp.

38.27.4 Member Data Documentation

38.27.4.1 CYdLidar LidarTest::m_lidar [protected]

Definition at line 30 of file lidar_test.h.

The documentation for this class was generated from the following files:

- test/lidar_test.h
- test/lidar_test.cpp

38.28 LidarVersion Struct Reference

```
#include <ydlidar_def.h>
```

Public Attributes

- uint8_t hardware
- uint8_t soft_major
- uint8_t soft_minor
- uint8_t soft_patch
- uint8_t sn [16]

38.28.1 Detailed Description

The numeric version information struct.

Definition at line 199 of file ydlidar_def.h.

38.28.2 Member Data Documentation

38.28.2.1 uint8_t LidarVersion::hardware

Hardware version

Definition at line 200 of file ydlidar_def.h.

38.28.2.2 uint8_t LidarVersion::sn[16]

serial number

Definition at line 204 of file ydlidar_def.h.

38.28.2.3 uint8_t LidarVersion::soft_major

major number

Definition at line 201 of file ydlidar_def.h.

38.28.2.4 uint8_t LidarVersion::soft_minor

minor number

Definition at line 202 of file ydlidar_def.h.

38.28.2.5 uint8_t LidarVersion::soft_patch

patch number

Definition at line 203 of file ydlidar_def.h.

The documentation for this struct was generated from the following file:

- core/common/ydlidar_def.h

38.29 ydlidar::core::base::Locker Class Reference

`#include <locker.h>`

Public Types

- enum LOCK_STATUS { LOCK_OK = 0, LOCK_TIMEOUT = -1, LOCK_FAILED = -2 }

Public Member Functions

- Locker ()
- ~Locker ()
- Locker::LOCK_STATUS lock (unsigned long timeout=0xFFFFFFFF)
- void unlock ()
- pthread_mutex_t * getLockHandle ()

Protected Member Functions

- void init ()
- void release ()

Protected Attributes

- pthread_mutex_t _lock

38.29.1 Detailed Description

Definition at line 23 of file locker.h.

38.29.2 Member Enumeration Documentation

38.29.2.1 enum ydlidar::core::base::Locker::LOCK_STATUS

Enumerator

LOCK_OK

LOCK_TIMEOUT

LOCK_FAILED

Definition at line 25 of file locker.h.

38.29.3 Constructor & Destructor Documentation

38.29.3.1 ydlidar::core::base::Locker::Locker ( ) [inline]

Definition at line 31 of file locker.h.

38.29.3.2 ydlidar::core::base::Locker::~Locker ( ) [inline]

Definition at line 38 of file locker.h.

38.29.4 Member Function Documentation

38.29.4.1 pthread_mutex_t* ydlidar::core::base::Locker::getLockHandle ( ) [inline]

Definition at line 149 of file locker.h.

38.29.4.2 void ydlidar::core::base::Locker::init ( ) [inline], [protected]

Definition at line 157 of file locker.h.

38.29.4.3 Locker::LOCK_STATUS ydlidar::core::base::Locker::lock (unsigned long *timeout* = 0xFFFFFFFF) [inline]

Definition at line 42 of file locker.h.

38.29.4.4 void ydlidar::core::base::Locker::release ( ) [inline], [protected]

Definition at line 165 of file locker.h.

38.29.4.5 void ydlidar::core::base::Locker::unlock ( ) [inline]

Definition at line 136 of file locker.h.

38.29.5 Member Data Documentation

38.29.5.1 pthread_mutex_t ydlidar::core::base::Locker::_lock [protected]

Definition at line 182 of file locker.h.

The documentation for this class was generated from the following file:

- core/base/locker.h

38.30 ydlidar::core::serial::MillisecondTimer Class Reference

`#include <unix_serial.h>`

Public Member Functions

- MillisecondTimer (const uint32_t millis)
- int64_t remaining ()

38.30.1 Detailed Description

Definition at line 19 of file unix_serial.h.

38.30.2 Constructor & Destructor Documentation

38.30.2.1 ydlidar::core::serial::MillisecondTimer::MillisecondTimer (const uint32_t *millis*) [explicit]

Definition at line 231 of file unix_serial.cpp.

38.30.3 Member Function Documentation

38.30.3.1 int64_t ydlidar::core::serial::MillisecondTimer::remaining ( )

Definition at line 244 of file unix_serial.cpp.

The documentation for this class was generated from the following files:

- core/serial/impl/unix/unix_serial.h
- core/serial/impl/unix/unix_serial.cpp

38.31 node_info Struct Reference

LiDAR Node info.

`#include <ydlidar_protocol.h>`

Public Attributes

- uint8_t sync_flag
  sync flag
- uint16_t sync_quality
  intensity
- uint16_t angle_q6_checkbit
  angle
- uint16_t distance_q2
  range
- uint64_t stamp
  time stamp
- uint8_t scan_frequence
  scan frequency. invalid: 0
- uint8_t debugInfo
  debug information
- uint8_t index
  package index
- uint8_t error_package
  error package state

38.31.1 Detailed Description

LiDAR Node info.

Definition at line 153 of file ydlidar_protocol.h.

38.31.2 Member Data Documentation

38.31.2.1 uint16_t node_info::angle_q6_checkbit

angle

Definition at line 156 of file ydlidar_protocol.h.

38.31.2.2 uint8_t node_info::debugInfo

debug information

Definition at line 160 of file ydlidar_protocol.h.

38.31.2.3 uint16_t node_info::distance_q2

range

Definition at line 157 of file ydlidar_protocol.h.

38.31.2.4 uint8_t node_info::error_package

error package state

Definition at line 162 of file ydlidar_protocol.h.

38.31.2.5 uint8_t node_info::index

package index

Definition at line 161 of file ydlidar_protocol.h.

38.31.2.6 uint8_t node_info::scan_frequence

scan frequency. invalid: 0

Definition at line 159 of file ydlidar_protocol.h.

38.31.2.7 uint64_t node_info::stamp

time stamp

Definition at line 158 of file ydlidar_protocol.h.

38.31.2.8 uint8_t node_info::sync_flag

sync flag

Definition at line 154 of file ydlidar_protocol.h.

38.31.2.9 uint16_t node_info::sync_quality

intensity

Definition at line 155 of file ydlidar_protocol.h.

The documentation for this struct was generated from the following file:

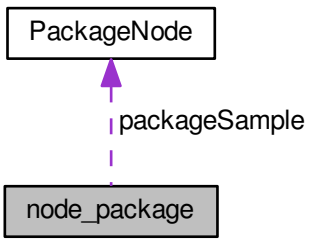
- core/common/ydlidar_protocol.h

38.32 node_package Struct Reference

LiDAR Intensity Nodes Package.

`#include <ydlidar_protocol.h>`

Collaboration diagram for node_package:



Public Attributes

- uint16_t package_Head
  package header
- uint8_t package_CT
  package ct
- uint8_t nowPackageNum
  package number
- uint16_t packageFirstSampleAngle
  first sample angle
- uint16_t packageLastSampleAngle
  last sample angle
- uint16_t checkSum
  checksum
- PackageNode packageSample [0x100]

38.32.1 Detailed Description

LiDAR Intensity Nodes Package.

Definition at line 172 of file ydlidar_protocol.h.

38.32.2 Member Data Documentation

38.32.2.1 uint16_t node_package::checkSum

checksum

Definition at line 178 of file ydlidar_protocol.h.

38.32.2.2 uint8_t node_package::nowPackageNum

package number

Definition at line 175 of file ydlidar_protocol.h.

38.32.2.3 uint8_t node_package::package_CT

package ct

Definition at line 174 of file ydlidar_protocol.h.

38.32.2.4 uint16_t node_package::package_Head

package header

Definition at line 173 of file ydlidar_protocol.h.

38.32.2.5 uint16_t node_package::packageFirstSampleAngle

first sample angle

Definition at line 176 of file ydlidar_protocol.h.

38.32.2.6 uint16_t node_package::packageLastSampleAngle

last sample angle

Definition at line 177 of file ydlidar_protocol.h.

38.32.2.7 PackageNode node_package::packageSample[0x100]

Definition at line 179 of file ydlidar_protocol.h.

The documentation for this struct was generated from the following file:

- core/common/ydlidar_protocol.h

38.33 node_packages Struct Reference

LiDAR Normal Nodes package.

`#include <ydlidar_protocol.h>`

Public Attributes

- uint16_t package_Head
  package header
- uint8_t package_CT
  package ct
- uint8_t nowPackageNum
  package number
- uint16_t packageFirstSampleAngle
  first sample angle
- uint16_t packageLastSampleAngle
  last sample angle
- uint16_t checkSum
  checksum
- uint16_t packageSampleDistance [0x100]

38.33.1 Detailed Description

LiDAR Normal Nodes package.

Definition at line 183 of file ydlidar_protocol.h.

38.33.2 Member Data Documentation

38.33.2.1 uint16_t node_packages::checkSum

checksum

Definition at line 189 of file ydlidar_protocol.h.

38.33.2.2 uint8_t node_packages::nowPackageNum

package number

Definition at line 186 of file ydlidar_protocol.h.

38.33.2.3 uint8_t node_packages::package_CT

package ct

Definition at line 185 of file ydlidar_protocol.h.

38.33.2.4 uint16_t node_packages::package_Head

package header

Definition at line 184 of file ydlidar_protocol.h.

38.33.2.5 uint16_t node_packages::packageFirstSampleAngle

first sample angle

Definition at line 187 of file ydlidar_protocol.h.

38.33.2.6 uint16_t node_packages::packageLastSampleAngle

last sample angle

Definition at line 188 of file ydlidar_protocol.h.

38.33.2.7 uint16_t node_packages::packageSampleDistance[0x100]

Definition at line 190 of file ydlidar_protocol.h.

The documentation for this struct was generated from the following file:

- core/common/ydlidar_protocol.h

38.34 offset_angle Struct Reference

LiDAR Zero Offset Angle.

`#include <ydlidar_protocol.h>`

Public Attributes

- int32_t angle

38.34.1 Detailed Description

LiDAR Zero Offset Angle.

Definition at line 240 of file ydlidar_protocol.h.

38.34.2 Member Data Documentation

38.34.2.1 int32_t offset_angle::angle

Definition at line 241 of file ydlidar_protocol.h.

The documentation for this struct was generated from the following file:

- core/common/ydlidar_protocol.h

38.35 PackageNode Struct Reference

package node info

`#include <ydlidar_protocol.h>`

Public Attributes

- uint8_t PakageSampleQuality
  intensity
- uint16_t PakageSampleDistance
  range

38.35.1 Detailed Description

package node info

Definition at line 166 of file ydlidar_protocol.h.

38.35.2 Member Data Documentation

38.35.2.1 uint16_t PackageNode::PakageSampleDistance

range

Definition at line 168 of file ydlidar_protocol.h.

38.35.2.2 uint8_t PackageNode::PakageSampleQuality

intensity

Definition at line 167 of file ydlidar_protocol.h.

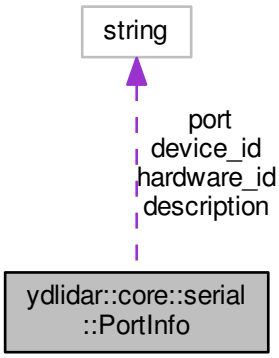
The documentation for this struct was generated from the following file:

- core/common/ydlidar_protocol.h

38.36 ydlidar::core::serial::PortInfo Struct Reference

```
#include <serial.h>
```

Collaboration diagram for ydlidar::core::serial::PortInfo:



Public Attributes

- std::string port
- std::string description
- std::string hardware_id
- std::string device_id

38.36.1 Detailed Description

Structure that describes a serial device.

Definition at line 681 of file serial.h.

38.36.2 Member Data Documentation

38.36.2.1 std::string ydlidar::core::serial::PortInfo::description

Human readable description of serial device if available.

Definition at line 687 of file serial.h.

38.36.2.2 std::string ydlidar::core::serial::PortInfo::device_id

Hardware Device ID or "" if not available.

Definition at line 693 of file serial.h.

38.36.2.3 std::string ydlidar::core::serial::PortInfo::hardware_id

Hardware ID (e.g. VID:PID of USB serial devices) or "n/a" if not available.

Definition at line 690 of file serial.h.

38.36.2.4 std::string ydlidar::core::serial::PortInfo::port

Address of the serial port (this can be passed to the constructor of Serial).

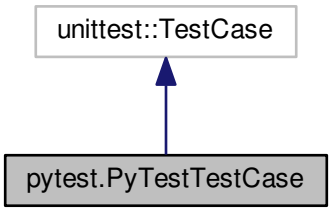
Definition at line 684 of file serial.h.

The documentation for this struct was generated from the following file:

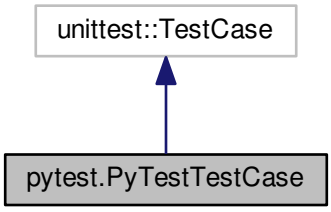
- core/serial/serial.h

38.37 pytest.PyTestTestCase Class Reference

Inheritance diagram for pytest.PyTestTestCase:



Collaboration diagram for pytest.PyTestTestCase:



Public Member Functions

- def testOSInitIsWrappedCorrectly (self)
- def testParamtersIsWrappedCorrectly (self)

38.37.1 Detailed Description

Definition at line 5 of file pytest.py.

38.37.2 Member Function Documentation

38.37.2.1 def pytest.PyTestTestCase.testOSInitIsWrappedCorrectly (*self*)

Definition at line 7 of file pytest.py.

38.37.2.2 def pytest.PyTestTestCase.testParamtersIsWrappedCorrectly (*self*)

Definition at line 14 of file pytest.py.

The documentation for this class was generated from the following file:

- python/test/pytest.py

38.38 sampling_rate Struct Reference

LiDAR sampling Rate struct.

`#include <ydlidar_protocol.h>`

Public Attributes

- uint8_t rate
  sample rate

38.38.1 Detailed Description

LiDAR sampling Rate struct.

Definition at line 208 of file ydlidar_protocol.h.

38.38.2 Member Data Documentation

38.38.2.1 uint8_t sampling_rate::rate

sample rate

Definition at line 209 of file ydlidar_protocol.h.

The documentation for this struct was generated from the following file:

- core/common/ydlidar_protocol.h

38.39 scan_exposure Struct Reference

LiDAR Exposure struct.

`#include <ydlidar_protocol.h>`

Public Attributes

- uint8_t exposure
  low exposure

38.39.1 Detailed Description

LiDAR Exposure struct.

Definition at line 222 of file ydlidar_protocol.h.

38.39.2 Member Data Documentation

38.39.2.1 uint8_t scan_exposure::exposure

low exposure

Definition at line 223 of file ydlidar_protocol.h.

The documentation for this struct was generated from the following file:

- core/common/ydlidar_protocol.h

38.40 scan_frequency Struct Reference

LiDAR scan frequency struct.

`#include <ydlidar_protocol.h>`

Public Attributes

- uint32_t frequency
  scan frequency

38.40.1 Detailed Description

LiDAR scan frequency struct.

Definition at line 213 of file ydlidar_protocol.h.

38.40.2 Member Data Documentation

38.40.2.1 uint32_t scan_frequency::frequency

scan frequency

Definition at line 214 of file ydlidar_protocol.h.

The documentation for this struct was generated from the following file:

- core/common/ydlidar_protocol.h

38.41 scan_heart_beat Struct Reference

LiDAR Heart beat struct.

`#include <ydlidar_protocol.h>`

Public Attributes

- uint8_t enable
  heart beat

38.41.1 Detailed Description

LiDAR Heart beat struct.

Definition at line 227 of file ydlidar_protocol.h.

38.41.2 Member Data Documentation

38.41.2.1 uint8_t scan_heart_beat::enable

heart beat

Definition at line 228 of file ydlidar_protocol.h.

The documentation for this struct was generated from the following file:

- core/common/ydlidar_protocol.h

38.42 scan_points Struct Reference

`#include <ydlidar_protocol.h>`

Public Attributes

- uint8_t flag

38.42.1 Detailed Description

Definition at line 231 of file ydlidar_protocol.h.

38.42.2 Member Data Documentation

38.42.2.1 uint8_t scan_points::flag

Definition at line 232 of file ydlidar_protocol.h.

The documentation for this struct was generated from the following file:

- core/common/ydlidar_protocol.h

38.43 scan_rotation Struct Reference

`#include <ydlidar_protocol.h>`

Public Attributes

- uint8_t rotation

38.43.1 Detailed Description

Definition at line 217 of file ydlidar_protocol.h.

38.43.2 Member Data Documentation

38.43.2.1 uint8_t scan_rotation::rotation

Definition at line 218 of file ydlidar_protocol.h.

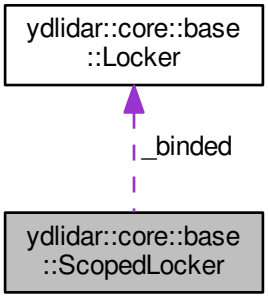
The documentation for this struct was generated from the following file:

- core/common/ydlidar_protocol.h

38.44 ydlidar::core::base::ScopedLocker Class Reference

`#include <locker.h>`

Collaboration diagram for ydlidar::core::base::ScopedLocker:



Public Member Functions

- ScopedLocker (Locker &l)
- void forceUnlock ()
- ~ScopedLocker ()

Public Attributes

- Locker & _binded

38.44.1 Detailed Description

Definition at line 341 of file locker.h.

38.44.2 Constructor & Destructor Documentation

38.44.2.1 ydlidar::core::base::ScopedLocker::ScopedLocker (Locker & l) [inline], [explicit]

Definition at line 343 of file locker.h.

38.44.2.2 ydlidar::core::base::ScopedLocker::~ScopedLocker ( ) [inline]

Definition at line 350 of file locker.h.

38.44.3 Member Function Documentation

38.44.3.1 void ydlidar::core::base::ScopedLocker::forceUnlock ( ) [inline]

Definition at line 347 of file locker.h.

38.44.4 Member Data Documentation

38.44.4.1 Locker& ydlidar::core::base::ScopedLocker::_binded

Definition at line 353 of file locker.h.

The documentation for this class was generated from the following file:

- core/base/locker.h

38.45 serial::Serial::ScopedReadLock Class Reference

Public Member Functions

- ScopedReadLock (Serial::SerialImpl *pimpl)
- ~ScopedReadLock ()

38.45.1 Detailed Description

Definition at line 30 of file serial.cpp.

38.45.2 Constructor & Destructor Documentation

38.45.2.1 serial::Serial::ScopedReadLock::ScopedReadLock (Serial::SerialImpl * *pimpl*) [inline], [explicit]

Definition at line 32 of file serial.cpp.

38.45.2.2 serial::Serial::ScopedReadLock::~ScopedReadLock ( ) [inline]

Definition at line 35 of file serial.cpp.

The documentation for this class was generated from the following file:

- core/serial/serial.cpp

38.46 serial::Serial::ScopedWriteLock Class Reference

Public Member Functions

- ScopedWriteLock (Serial::SerialImpl *pimpl)
- ~ScopedWriteLock ()

38.46.1 Detailed Description

Definition at line 46 of file serial.cpp.

38.46.2 Constructor & Destructor Documentation

38.46.2.1 serial::Serial::ScopedWriteLock::ScopedWriteLock (Serial::SerialImpl * *pimpl*) [inline], [explicit]

Definition at line 48 of file serial.cpp.

38.46.2.2 serial::Serial::ScopedWriteLock::~ScopedWriteLock ( ) [inline]

Definition at line 51 of file serial.cpp.

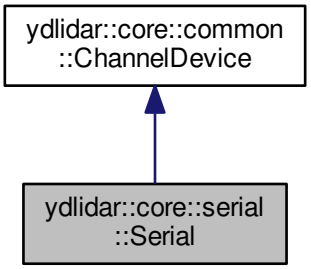
The documentation for this class was generated from the following file:

- core/serial/serial.cpp

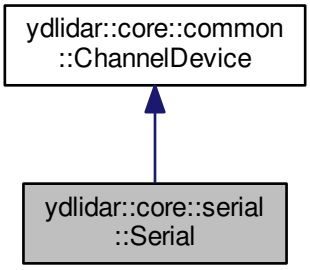
38.47 ydlidar::core::serial::Serial Class Reference

`#include <serial.h>`

Inheritance diagram for ydlidar::core::serial::Serial:



Collaboration diagram for ydlidar::core::serial::Serial:



Public Types

- enum SerialPortError {
  NoError, DeviceNotFoundError, PermissionError, OpenError,
  ParityError, FramingError, BreakConditionError, WriteError,
  ReadError, ResourceError, UnsupportedOperationError, UnknownError,
  TimeoutError, NotOpenError }

  Defines all error codes handled by the CSimpleSocket class.

Public Member Functions

- Serial (const std::string &port="", uint32_t baudrate=9600, Timeout timeout=Timeout(), bytesize_t bytesize=eightbits, parity_t parity=parity_none, stopbits_t stopbits=stopbits_one, flowcontrol_t flowcontrol=flowcontrol_none)
- virtual ~Serial ()
- virtual bool open ()
- virtual bool isOpen ()
- virtual void closePort ()
- virtual size_t available ()
- bool waitReadable ()
- void waitByteTimes (size_t count)
- Serial::SerialPortError getSystemError (int systemErrorCode=-1) const

  getSystemError
- virtual int waitfordata (size_t data_count, uint32_t timeout, size_t *returned_size)

  Block until there is serial data to read or read_timeout_constant number of milliseconds have elapsed. The return value is greater than zero when the function exits with the serial port buffer is greater than or equal to data_count, false otherwise(due to timeout or select interruption).
- virtual size_t writeData (const uint8_t *data, size_t size)
- virtual size_t readData (uint8_t *data, size_t size)
- virtual const char * DescribeError ()

  DescribeError.
- size_t read (uint8_t *buffer, size_t size)
- size_t read (std::vector< uint8_t > &buffer, size_t size=1)
- size_t read (std::string &buffer, size_t size=1)
- virtual std::string readSize (size_t size=1)

- size_t readline (std::string &buffer, size_t size=65536, std::string eol="\n")
- std::string readline (size_t size=65536, std::string eol="\n")
- std::vector< std::string > readlines (size_t size=65536, std::string eol="\n")
- size_t write (const uint8_t *data, size_t size)
- size_t write (const std::vector< uint8_t > &data)
- size_t write (const std::string &data)
- void setPort (const std::string &port)
- std::string getPort () const
- void setTimeout (Timeout &timeout)
- void setTimeout (uint32_t inter_byte_timeout, uint32_t read_timeout_constant, uint32_t read_timeout_multiplier, uint32_t write_timeout_constant, uint32_t write_timeout_multiplier)
- Timeout getTimeout () const
- bool setBaudrate (uint32_t baudrate)
- uint32_t getBaudrate () const
- bool setBytesize (bytesize_t bytesize)
- bytesize_t getBytesize () const
- bool setParity (parity_t parity)
- parity_t getParity () const
- bool setStopbits (stopbits_t stopbits)
- stopbits_t getStopbits () const
- bool setFlowcontrol (flowcontrol_t flowcontrol)
- flowcontrol_t getFlowcontrol () const
- void flush ()
- void flushInput ()
- void flushOutput ()
- void sendBreak (int duration)
- bool setBreak (bool level=true)
- bool setRTS (bool level=true)
- virtual bool setDTR (bool level=true)
- bool waitForChange ()
- bool getCTS ()
- bool getDSR ()
- bool getRI ()
- bool getCD ()
- virtual int getByteTime ()

Static Public Member Functions

- static const char * DescribeError (SerialPortError err)

38.47.1 Detailed Description

Class that provides a portable serial port interface.

Definition at line 114 of file serial.h.

38.47.2 Member Enumeration Documentation

38.47.2.1 enum ydlidar::core::serial::Serial::SerialPortError

Defines all error codes handled by the CSimpleSocket class.

Enumerator

NoError

DeviceNotFoundError

PermissionError

OpenError

ParityError

FramingError

BreakConditionError

WriteError

ReadError

ResourceError

UnsupportedOperationError

UnknownError

TimeoutError

NotOpenError

Definition at line 118 of file serial.h.

38.47.3 Constructor & Destructor Documentation

38.47.3.1 ydlidar::core::serial::Serial::Serial (const std::string & *port* = " ", uint32_t *baudrate* = 9600, Timeout *timeout* = Timeout (), bytesize_t *bytesize* = eightbits, parity_t *parity* = parity_none, stopbits_t *stopbits* = stopbits_one, flowcontrol_t *flowcontrol* = flowcontrol_none) [explicit]

Creates a Serial object and opens the port if a port is specified, otherwise it remains closed until serial::Serial::open is called.

Parameters

port	A std::string containing the address of the serial port, which would be something like 'COM1' on Windows and '/dev/ttyS0' on Linux.
baudrate	An unsigned 32-bit integer that represents the baudrate
timeout	A serial::Timeout struct that defines the timeout conditions for the serial port.

See also

serial::Timeout

Parameters

bytesize	Size of each byte in the serial transmission of data, default is eightbits, possible values are: fivebits, sixbits, sevenbits, eightbits
parity	Method of parity, default is parity_none, possible values are: parity_none, parity_odd, parity_even
stopbits	Number of stop bits used, default is stopbits_one, possible values are: stopbits_one, stopbits_one_point_five, stopbits_two
flowcontrol	Type of flowcontrol used, default is flowcontrol_none, possible values are: flowcontrol_none, flowcontrol_software, flowcontrol_hardware

Exceptions

serial::PortNotOpenedException	
serial::IOException	
std::invalid_argument	

38.47.3.2 ydlidar::core::serial::Serial::~Serial ( ) [virtual]

Destructor

Definition at line 69 of file serial.cpp.

38.47.4 Member Function Documentation

38.47.4.1 size_t ydlidar::core::serial::Serial::available ( ) [virtual]

Return the number of characters in the buffer.

Implements ydlidar::core::common::ChannelDevice.

Definition at line 85 of file serial.cpp.

38.47.4.2 void ydlidar::core::serial::Serial::closePort ( ) [virtual]

Closes the serial port.

Implements ydlidar::core::common::ChannelDevice.

Definition at line 77 of file serial.cpp.

38.47.4.3 const char * ydlidar::core::serial::Serial::DescribeError (SerialPortError *err*) [static]

Returns a human-readable description of the given error code or the last error code of a socket

Definition at line 116 of file serial.cpp.

38.47.4.4 virtual const char* ydlidar::core::serial::Serial::DescribeError ( ) [inline], [virtual]

DescribeError.

Returns

Reimplemented from ydlidar::core::common::ChannelDevice.

Definition at line 287 of file serial.h.

38.47.4.5 void ydlidar::core::serial::Serial::flush ( ) [virtual]

Flush the input and output buffers

Implements ydlidar::core::common::ChannelDevice.

Definition at line 352 of file serial.cpp.

38.47.4.6 void ydlidar::core::serial::Serial::flushInput ( )

Flush only the input buffer

Definition at line 358 of file serial.cpp.

38.47.4.7 void ydlidar::core::serial::Serial::flushOutput ( )

Flush only the output buffer

Definition at line 363 of file serial.cpp.

38.47.4.8 uint32_t ydlidar::core::serial::Serial::getBaudrate ( ) const

Gets the baudrate for the serial port.

Returns

An integer that sets the baud rate for the serial port.

See also

Serial::setBaudrate

\

Definition at line 316 of file serial.cpp.

38.47.4.9 bytesize_t ydlidar::core::serial::Serial::getBytesize ( ) const

Gets the bytesize for the serial port.

See also

Serial::setBytesize

\

Definition at line 324 of file serial.cpp.

38.47.4.10 int ydlidar::core::serial::Serial::getByteTime ( ) [virtual]

Returns the singal byte time.

Reimplemented from ydlidar::core::common::ChannelDevice.

Definition at line 404 of file serial.cpp.

38.47.4.11 bool ydlidar::core::serial::Serial::getCD ( )

Returns the current status of the CD line.

Definition at line 400 of file serial.cpp.

38.47.4.12 bool ydlidar::core::serial::Serial::getCTS ( )

Returns the current status of the CTS line.

Definition at line 388 of file serial.cpp.

38.47.4.13 bool ydlidar::core::serial::Serial::getDSR ( )

Returns the current status of the DSR line.

Definition at line 392 of file serial.cpp.

38.47.4.14 flowcontrol_t ydlidar::core::serial::Serial::getFlowcontrol ( ) const

Gets the flow control for the serial port.

See also

Serial::setFlowcontrol

\

Definition at line 348 of file serial.cpp.

38.47.4.15 parity_t ydlidar::core::serial::Serial::getParity ( ) const

Gets the parity for the serial port.

See also

Serial::setParity

\

Definition at line 332 of file serial.cpp.

38.47.4.16 string ydlidar::core::serial::Serial::getPort ( ) const

Gets the serial port identifier.

See also

Serial::setPort

Exceptions

std::invalid_argument	

Definition at line 300 of file serial.cpp.

38.47.4.17 bool ydlidar::core::serial::Serial::getRI ( )

Returns the current status of the RI line.

Definition at line 396 of file serial.cpp.

38.47.4.18 stopbits_t ydlidar::core::serial::Serial::getStopbits ( ) const

Gets the stopbits for the serial port.

See also

Serial::setStopbits

\

Definition at line 340 of file serial.cpp.

38.47.4.19 Serial::SerialPortError ydlidar::core::serial::Serial::getSystemError (int *systemErrorCode* = `-1`) const

getSystemError

Parameters

systemErrorCode	

Returns

Definition at line 98 of file serial.cpp.

38.47.4.20 serial::Timeout ydlidar::core::serial::Serial::getTimeout ( ) const

Gets the timeout for reads in seconds.

Returns

A Timeout struct containing the inter_byte_timeout, and read and write timeout constants and multipliers.

See also

Serial::setTimeout

Definition at line 308 of file serial.cpp.

38.47.4.21 bool ydlidar::core::serial::Serial::isOpen ( ) [virtual]

Gets the open status of the serial port.

Returns

Returns true if the port is open, false otherwise.

Implements ydlidar::core::common::ChannelDevice.

Definition at line 81 of file serial.cpp.

38.47.4.22 bool ydlidar::core::serial::Serial::open ( ) [virtual]

Opens the serial port as long as the port is set and the port isn't already open.

If the port is provided to the constructor then an explicit call to open is not needed.

See also

Serial::Serial

Returns

Returns true if the port is open, false otherwise.

Implements ydlidar::core::common::ChannelDevice.

Definition at line 73 of file serial.cpp.

38.47.4.23 size_t ydlidar::core::serial::Serial::read (uint8_t * *buffer*, size_t *size*)

Read a given amount of bytes from the serial port into a given buffer.

The read function will return in one of three cases:

- The number of requested bytes was read.
  - In this case the number of bytes requested will match the size_t returned by read.
- A timeout occurred, in this case the number of bytes read will not match the amount requested, but no exception will be thrown. One of two possible timeouts occurred:
  - The inter byte timeout expired, this means that number of milliseconds elapsed between receiving bytes from the serial port exceeded the inter byte timeout.
  - The total timeout expired, which is calculated by multiplying the read timeout multiplier by the number of requested bytes and then added to the read timeout constant. If that total number of milliseconds elapses after the initial call to read a timeout will occur.
- An exception occurred, in this case an actual exception will be thrown.

Parameters

buffer	An uint8_t array of at least the requested size.
size	A size_t defining how many bytes to be read.

Returns

A size_t representing the number of bytes read as a result of the call to read.

Definition at line 161 of file serial.cpp.

38.47.4.24 size_t ydlidar::core::serial::Serial::read (std::vector< uint8_t > & *buffer*, size_t *size* = 1)

Read a given amount of bytes from the serial port into a give buffer.

Parameters

buffer	A reference to a std::vector of uint8_t.
size	A size_t defining how many bytes to be read.

Returns

A size_t representing the number of bytes read as a result of the call to read.

Definition at line 166 of file serial.cpp.

38.47.4.25 size_t ydlidar::core::serial::Serial::read (std::string & *buffer*, size_t *size* = 1)

Read a given amount of bytes from the serial port into a give buffer.

Parameters

buffer	A reference to a std::string.
size	A size_t defining how many bytes to be read.

Returns

A size_t representing the number of bytes read as a result of the call to read.

Definition at line 174 of file serial.cpp.

38.47.4.26 size_t ydlidar::core::serial::Serial::readData (uint8_t * *data*, size_t *size*) [virtual]

Read a given amount of bytes from the serial port into a given buffer.

The read function will return in one of three cases:

- The number of requested bytes was read.
  - In this case the number of bytes requested will match the size_t returned by read.
- A timeout occurred, in this case the number of bytes read will not match the amount requested, but no exception will be thrown. One of two possible timeouts occurred:
  - The inter byte timeout expired, this means that number of milliseconds elapsed between receiving bytes from the serial port exceeded the inter byte timeout.
  - The total timeout expired, which is calculated by multiplying the read timeout multiplier by the number of requested bytes and then added to the read timeout constant. If that total number of milliseconds elapses after the initial call to read a timeout will occur.
- An exception occurred, in this case an actual exception will be thrown.

Parameters

buffer	An uint8_t array of at least the requested size.
size	A size_t defining how many bytes to be read.

Returns

A size_t representing the number of bytes read as a result of the call to read.

Implements ydlidar::core::common::ChannelDevice.

Definition at line 112 of file serial.cpp.

38.47.4.27 size_t ydlidar::core::serial::Serial::readline (std::string & *buffer*, size_t *size* = 65536, std::string *eol* = "\n")

Reads in a line or until a given delimiter has been processed.

Reads from the serial port until a single line has been read.

Parameters

buffer	A std::string reference used to store the data.
size	A maximum length of a line, defaults to 65536 (2^16)
eol	A string to match against for the EOL.

Returns

A size_t representing the number of bytes read.

38.47.4.28 std::string ydlidar::core::serial::Serial::readline (size_t *size* = 65536, std::string *eol* = "\n")

Reads in a line or until a given delimiter has been processed.

Reads from the serial port until a single line has been read.

Parameters

size	A maximum length of a line, defaults to 65536 (2^16)
eol	A string to match against for the EOL.

Returns

A std::string containing the line.

38.47.4.29 vector< string > ydlidar::core::serial::Serial::readlines (size_t *size* = 65536, std::string *eol* = "\n")

Reads in multiple lines until the serial port times out.

This requires a timeout > 0 before it can be run. It will read until a timeout occurs and return a list of strings.

Parameters

size	A maximum length of combined lines, defaults to 65536 (2^16)
eol	A string to match against for the EOL.

Returns

A vector<string> containing the lines.

Definition at line 222 of file serial.cpp.

38.47.4.30 string ydlidar::core::serial::Serial::readSize (size_t *size* = 1) [virtual]

Read a given amount of bytes from the serial port and return a string containing the data.

Parameters

size	A size_t defining how many bytes to be read.

Returns

A std::string containing the data read from the port.

Implements ydlidar::core::common::ChannelDevice.

Definition at line 182 of file serial.cpp.

38.47.4.31 void ydlidar::core::serial::Serial::sendBreak (int *duration*)

Sends the RS-232 break signal. See tcsendbreak(3).

Definition at line 368 of file serial.cpp.

38.47.4.32 bool ydlidar::core::serial::Serial::setBaudrate (uint32_t *baudrate*)

Sets the baudrate for the serial port.

Possible baudrates depends on the system but some safe baudrates include: 110, 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 56000, 57600, 115200 Some other baudrates that are supported by some comports: 128000, 153600, 230400, 256000, 460800, 921600

Parameters

baudrate	An integer that sets the baud rate for the serial port.

Definition at line 312 of file serial.cpp.

38.47.4.33 bool ydlidar::core::serial::Serial::setBreak (bool *level* = `true`)

Set the break condition to a given level. Defaults to true.

Definition at line 372 of file serial.cpp.

38.47.4.34 bool ydlidar::core::serial::Serial::setBytesize (bytesize_t *bytesize*)

Sets the bytesize for the serial port.

Parameters

bytesize	Size of each byte in the serial transmission of data, default is eightbits, possible values are: fivebits, sixbits, sevenbits, eightbits

\

Definition at line 320 of file serial.cpp.

38.47.4.35 bool ydlidar::core::serial::Serial::setDTR (bool *level* = `true`) [virtual]

Set the DTR handshaking line to the given level. Defaults to true.

Reimplemented from ydlidar::core::common::ChannelDevice.

Definition at line 380 of file serial.cpp.

38.47.4.36 bool ydlidar::core::serial::Serial::setFlowcontrol (flowcontrol_t *flowcontrol*)

Sets the flow control for the serial port.

Parameters

flowcontrol	Type of flowcontrol used, default is flowcontrol_none, possible values are: flowcontrol_none, flowcontrol_software, flowcontrol_hardware

\

Definition at line 344 of file serial.cpp.

38.47.4.37 bool ydlidar::core::serial::Serial::setParity (parity_t *parity*)

Sets the parity for the serial port.

Parameters

parity	Method of parity, default is parity_none, possible values are: parity_none, parity_odd, parity_even

\

Definition at line 328 of file serial.cpp.

38.47.4.38 void ydlidar::core::serial::Serial::setPort (const std::string & *port*)

Sets the serial port identifier.

Parameters

port	A const std::string reference containing the address of the serial port, which would be something like 'COM1' on Windows and '/dev/ttyS0' on Linux.

Exceptions

std::invalid_argument	

Definition at line 284 of file serial.cpp.

38.47.4.39 bool ydlidar::core::serial::Serial::setRTS (bool *level* = `true`)

Set the RTS handshaking line to the given level. Defaults to true.

Definition at line 376 of file serial.cpp.

38.47.4.40 bool ydlidar::core::serial::Serial::setStopbits (stopbits_t *stopbits*)

Sets the stopbits for the serial port.

Parameters

stopbits	Number of stop bits used, default is stopbits_one, possible values are: stopbits_one, stopbits_one_point_five, stopbits_two

\

Definition at line 336 of file serial.cpp.

38.47.4.41 void ydlidar::core::serial::Serial::setTimeout (serial::Timeout & *timeout*)

Sets the timeout for reads and writes using the Timeout struct.

There are two timeout conditions described here:

- The inter byte timeout:
  - The inter_byte_timeout component of serial::Timeout defines the maximum amount of time, in milliseconds, between receiving bytes on the serial port that can pass before a timeout occurs. Setting this to zero will prevent inter byte timeouts from occurring.
- Total time timeout:
  - The constant and multiplier component of this timeout condition, for both read and write, are defined in serial::Timeout. This timeout occurs if the total time since the read or write call was made exceeds the specified time in milliseconds.
  - The limit is defined by multiplying the multiplier component by the number of requested bytes and adding that product to the constant component. In this way if you want a read call, for example, to timeout after exactly one second regardless of the number of bytes you asked for then set the read_timeout_constant component of serial::Timeout to 1000 and the read_timeout_multiplier to zero. This timeout condition can be used in conjunction with the inter byte timeout condition with out any problems, timeout will simply occur when one of the two timeout conditions is met. This allows users to have maximum control over the trade-off between responsiveness and efficiency.

Read and write functions will return in one of three cases. When the reading or writing is complete, when a timeout occurs, or when an exception occurs.

A timeout of 0 enables non-blocking mode.

Parameters

timeout	A serial::Timeout struct containing the inter byte timeout, and the read and write timeout constants and multipliers.

See also

serial::Timeout

Definition at line 304 of file serial.cpp.

38.47.4.42 void ydlidar::core::serial::Serial::setTimeout (uint32_t *inter_byte_timeout,* uint32_t *read_timeout_constant,* uint32_t *read_timeout_multiplier,* uint32_t *write_timeout_constant,* uint32_t *write_timeout_multiplier*) [inline]

Sets the timeout for reads and writes.

Definition at line 487 of file serial.h.

38.47.4.43 void ydlidar::core::serial::Serial::waitByteTimes (size_t *count*)

Block for a period of time corresponding to the transmission time of count characters at present serial settings. This may be used in con- junction with waitReadable to read larger blocks of data from the port.

Definition at line 94 of file serial.cpp.

38.47.4.44 bool ydlidar::core::serial::Serial::waitForChange ( )

Blocks until CTS, DSR, RI, CD changes or something interrupts it.

Can throw an exception if an error occurs while waiting. You can check the status of CTS, DSR, RI, and CD once this returns. Uses TIOCMIWAIT via ioctl if available (mostly only on Linux) with a resolution of less than +-1ms and as good as +-0.2ms. Otherwise a polling method is used which can give +-2ms.

Returns

Returns true if one of the lines changed, false if something else occurred.

Definition at line 384 of file serial.cpp.

38.47.4.45 int ydlidar::core::serial::Serial::waitfordata (size_t *data_count,* uint32_t *timeout,* size_t * *returned_size*) [virtual]

Block until there is serial data to read or read_timeout_constant number of milliseconds have elapsed. The return value is greater than zero when the function exits with the serial port buffer is greater than or equal to data_count, false otherwise(due to timeout or select interruption).

Parameters

data_count	A size_t that indicates how many bytes should be wait from the given serial port or network buffer.
timeout	waiting timeout time
returned_size	if it is not NULL, the actual number of bytes will be returned.

Returns

A size_t representing the number of bytes wait as a result of the call to wait.

Implements ydlidar::core::common::ChannelDevice.

Definition at line 103 of file serial.cpp.

38.47.4.46 bool ydlidar::core::serial::Serial::waitReadable ( )

Block until there is serial data to read or read_timeout_constant number of milliseconds have elapsed. The return value is true when the function exits with the port in a readable state, false otherwise (due to timeout or select interruption).

Definition at line 89 of file serial.cpp.

38.47.4.47 size_t ydlidar::core::serial::Serial::write (const uint8_t * *data*, size_t *size*)

Write a string to the serial port.

Parameters

data	A const reference containing the data to be written to the serial port.
size	A size_t that indicates how many bytes should be written from the given data buffer.

Returns

A size_t representing the number of bytes actually written to the serial port.

Exceptions

serial::PortNotOpenedException	
serial::SerialException	
serial::IOException	

Definition at line 275 of file serial.cpp.

38.47.4.48 size_t ydlidar::core::serial::Serial::write (const std::vector< uint8_t > & *data*)

Write a string to the serial port.

Parameters

data	A const reference containing the data to be written to the serial port.

Returns

A size_t representing the number of bytes actually written to the serial port.

Definition at line 270 of file serial.cpp.

38.47.4.49 size_t ydlidar::core::serial::Serial::write (const std::string & *data*)

Write a string to the serial port.

Parameters

data	A const reference containing the data to be written to the serial port.

Returns

A size_t representing the number of bytes actually written to the serial port.

38.47.4.50 size_t ydlidar::core::serial::Serial::writeData (const uint8_t * *data*, size_t *size*) [virtual]

Write a string to the serial port.

Parameters

data	A const reference containing the data to be written to the serial port.
size	A size_t that indicates how many bytes should be written from the given data buffer.

Returns

A size_t representing the number of bytes actually written to the serial port.

Implements ydlidar::core::common::ChannelDevice.

Definition at line 108 of file serial.cpp.

The documentation for this class was generated from the following files:

- core/serial/serial.h
- core/serial/serial.cpp

38.48 serial::Serial::SerialImpl Class Reference

`#include <unix_serial.h>`

Public Member Functions

- SerialImpl (const string &port, unsigned long baudrate, bytesize_t bytesize, parity_t parity, stopbits_t stopbits, flowcontrol_t flowcontrol)
- virtual ~SerialImpl ()
- bool open ()
- Serial::SerialPortError getSystemError (int systemErrorCode) const
- void close ()
- bool isOpen () const
- size_t available ()
- bool waitReadable (uint32_t timeout)
- void waitByteTimes (size_t count)
- int waitfordata (size_t data_count, uint32_t timeout, size_t *returned_size)
- size_t read (uint8_t *buf, size_t size=1)
- size_t write (const uint8_t *data, size_t length)
- void flush ()
- void flushInput ()
- void flushOutput ()
- void sendBreak (int duration)
- bool setBreak (bool level)
- bool setRTS (bool level)
- bool setDTR (bool level)
- bool waitForChange ()
- bool getCTS ()
- bool getDSR ()
- bool getRI ()
- bool getCD ()
- uint32_t getByteTime ()
- void setPort (const string &port)
- string getPort () const
- void setTimeout (Timeout &timeout)
- Timeout getTimeout () const
- bool setBaudrate (unsigned long baudrate)
- bool setStandardBaudRate (speed_t baudrate)
- bool setCustomBaudRate (unsigned long baudrate)
- unsigned long getBaudrate () const
- bool setBytesize (bytesize_t bytesize)
- bytesize_t getBytesize () const
- bool setParity (parity_t parity)
- parity_t getParity () const
- bool setStopbits (stopbits_t stopbits)
- stopbits_t getStopbits () const
- bool setFlowcontrol (flowcontrol_t flowcontrol)
- flowcontrol_t getFlowcontrol () const
- bool setTermios (const termios *tio)
- bool getTermios (termios *tio)
- int readLock ()
- int readUnlock ()
- int writeLock ()
- int writeUnlock ()

38.48.1 Detailed Description

Definition at line 29 of file unix_serial.h.

38.48.2 Constructor & Destructor Documentation

38.48.2.1 serial::Serial::SerialImpl::SerialImpl (const string & *port,* unsigned long *baudrate,* bytesize_t *bytesize,* parity_t *parity,* stopbits_t *stopbits,* flowcontrol_t *flowcontrol*) [explicit]

Definition at line 640 of file unix_serial.cpp.

38.48.2.2 serial::Serial::SerialImpl::~SerialImpl ( ) [virtual]

Definition at line 652 of file unix_serial.cpp.

38.48.3 Member Function Documentation

38.48.3.1 size_t serial::Serial::SerialImpl::available ( )

Definition at line 848 of file unix_serial.cpp.

38.48.3.2 void serial::Serial::SerialImpl::close ( )

Definition at line 829 of file unix_serial.cpp.

38.48.3.3 void serial::Serial::SerialImpl::flush ( )

Definition at line 1428 of file unix_serial.cpp.

38.48.3.4 void serial::Serial::SerialImpl::flushInput ( )

Definition at line 1438 of file unix_serial.cpp.

38.48.3.5 void serial::Serial::SerialImpl::flushOutput ( )

Definition at line 1446 of file unix_serial.cpp.

38.48.3.6 unsigned long serial::Serial::SerialImpl::getBaudrate ( ) const

Definition at line 1216 of file unix_serial.cpp.

38.48.3.7 serial::bytesize_t serial::Serial::SerialImpl::getBytesize ( ) const

Definition at line 1347 of file unix_serial.cpp.

38.48.3.8 uint32_t serial::Serial::SerialImpl::getByteTime ( )

Definition at line 1609 of file unix_serial.cpp.

38.48.3.9 bool serial::Serial::SerialImpl::getCD ( )

Definition at line 1595 of file unix_serial.cpp.

38.48.3.10 bool serial::Serial::SerialImpl::getCTS ( )

Definition at line 1553 of file unix_serial.cpp.

38.48.3.11 bool serial::Serial::SerialImpl::getDSR ( )

Definition at line 1567 of file unix_serial.cpp.

38.48.3.12 serial::flowcontrol_t serial::Serial::SerialImpl::getFlowcontrol ( ) const

Definition at line 1398 of file unix_serial.cpp.

38.48.3.13 serial::parity_t serial::Serial::SerialImpl::getParity ( ) const

Definition at line 1364 of file unix_serial.cpp.

38.48.3.14 string serial::Serial::SerialImpl::getPort ( ) const

Definition at line 1168 of file unix_serial.cpp.

38.48.3.15 bool serial::Serial::SerialImpl::getRI ( )

Definition at line 1581 of file unix_serial.cpp.

38.48.3.16 serial::stopbits_t serial::Serial::SerialImpl::getStopbits ( ) const

Definition at line 1381 of file unix_serial.cpp.

38.48.3.17 Serial::SerialPortError serial::Serial::SerialImpl::getSystemError (int *systemErrorCode*) const

Definition at line 752 of file unix_serial.cpp.

38.48.3.18 bool serial::Serial::SerialImpl::getTermios (termios * *tio*)

Definition at line 1418 of file unix_serial.cpp.

38.48.3.19 serial::Timeout serial::Serial::SerialImpl::getTimeout ( ) const

Definition at line 1176 of file unix_serial.cpp.

38.48.3.20 bool serial::Serial::SerialImpl::isOpen ( ) const

Definition at line 844 of file unix_serial.cpp.

38.48.3.21 bool serial::Serial::SerialImpl::open ( )

Definition at line 658 of file unix_serial.cpp.

38.48.3.22 size_t serial::Serial::SerialImpl::read (uint8_t * *buf,* size_t *size* = `1`)

Definition at line 982 of file unix_serial.cpp.

38.48.3.23 int serial::Serial::SerialImpl::readLock ( )

Definition at line 1613 of file unix_serial.cpp.

38.48.3.24 int serial::Serial::SerialImpl::readUnlock ( )

Definition at line 1618 of file unix_serial.cpp.

38.48.3.25 void serial::Serial::SerialImpl::sendBreak (int *duration*)

Definition at line 1454 of file unix_serial.cpp.

38.48.3.26 bool serial::Serial::SerialImpl::setBaudrate (unsigned long *baudrate*)

Definition at line 1180 of file unix_serial.cpp.

38.48.3.27 bool serial::Serial::SerialImpl::setBreak (bool *level*)

Definition at line 1462 of file unix_serial.cpp.

38.48.3.28 bool serial::Serial::SerialImpl::setBytesize (bytesize_t *bytesize*)

Definition at line 1334 of file unix_serial.cpp.

38.48.3.29 bool serial::Serial::SerialImpl::setCustomBaudRate (unsigned long *baudrate*)

Definition at line 1276 of file unix_serial.cpp.

38.48.3.30 bool serial::Serial::SerialImpl::setDTR (bool *level*)

Definition at line 1500 of file unix_serial.cpp.

38.48.3.31 bool serial::Serial::SerialImpl::setFlowcontrol (flowcontrol_t *flowcontrol*)

Definition at line 1385 of file unix_serial.cpp.

38.48.3.32 bool serial::Serial::SerialImpl::setParity (parity_t *parity*)

Definition at line 1351 of file unix_serial.cpp.

38.48.3.33 void serial::Serial::SerialImpl::setPort (const string & *port*)

Definition at line 1164 of file unix_serial.cpp.

38.48.3.34 bool serial::Serial::SerialImpl::setRTS (bool *level*)

Definition at line 1480 of file unix_serial.cpp.

38.48.3.35 bool serial::Serial::SerialImpl::setStandardBaudRate (speed_t *baudrate*)

Definition at line 1221 of file unix_serial.cpp.

38.48.3.36 bool serial::Serial::SerialImpl::setStopbits (stopbits_t *stopbits*)

Definition at line 1368 of file unix_serial.cpp.

38.48.3.37 bool serial::Serial::SerialImpl::setTermios (const termios * *tio*)

Definition at line 1403 of file unix_serial.cpp.

38.48.3.38 void serial::Serial::SerialImpl::setTimeout (Timeout & *timeout*)

Definition at line 1172 of file unix_serial.cpp.

38.48.3.39 void serial::Serial::SerialImpl::waitByteTimes (size_t *count*)

Definition at line 977 of file unix_serial.cpp.

38.48.3.40 bool serial::Serial::SerialImpl::waitForChange ( )

Definition at line 1520 of file unix_serial.cpp.

38.48.3.41 int serial::Serial::SerialImpl::waitfordata (size_t *data_count,* uint32_t *timeout,* size_t * *returned_size*)

Definition at line 895 of file unix_serial.cpp.

38.48.3.42 bool serial::Serial::SerialImpl::waitReadable (uint32_t *timeout*)

Definition at line 862 of file unix_serial.cpp.

38.48.3.43 size_t serial::Serial::SerialImpl::write (const uint8_t * *data,* size_t *length*)

Error

Timeout

Port ready to write

Definition at line 1066 of file unix_serial.cpp.

38.48.3.44 int serial::Serial::SerialImpl::writeLock ( )

Definition at line 1623 of file unix_serial.cpp.

38.48.3.45 int serial::Serial::SerialImpl::writeUnlock ( )

Definition at line 1628 of file unix_serial.cpp.

The documentation for this class was generated from the following files:

- core/serial/impl/unix/unix_serial.h
- core/serial/impl/unix/unix_serial.cpp

38.49 string_t Struct Reference

c string

`#include <ydlidar_def.h>`

Public Attributes

- char data [50]
  data

38.49.1 Detailed Description

c string

Definition at line 186 of file ydlidar_def.h.

38.49.2 Member Data Documentation

38.49.2.1 char string_t::data[50]

data

Definition at line 188 of file ydlidar_def.h.

The documentation for this struct was generated from the following file:

- core/common/ydlidar_def.h

38.50 ydlidar::core::serial::termios2 Struct Reference

Public Attributes

- tcflag_t c_iflag
- tcflag_t c_oflag
- tcflag_t c_cflag
- tcflag_t c_lflag
- cc_t c_line
- cc_t c_cc [19]
- speed_t c_ispeed
- speed_t c_ospeed

38.50.1 Detailed Description

Definition at line 179 of file unix_serial.cpp.

38.50.2 Member Data Documentation

38.50.2.1 cc_t ydlidar::core::serial::termios2::c_cc[19]

Definition at line 185 of file unix_serial.cpp.

38.50.2.2 tcflag_t ydlidar::core::serial::termios2::c_cflag

Definition at line 182 of file unix_serial.cpp.

38.50.2.3 tcflag_t ydlidar::core::serial::termios2::c_iflag

Definition at line 180 of file unix_serial.cpp.

38.50.2.4 speed_t ydlidar::core::serial::termios2::c_ispeed

Definition at line 186 of file unix_serial.cpp.

38.50.2.5 tcflag_t ydlidar::core::serial::termios2::c_lflag

Definition at line 183 of file unix_serial.cpp.

38.50.2.6 cc_t ydlidar::core::serial::termios2::c_line

Definition at line 184 of file unix_serial.cpp.

38.50.2.7 tcflag_t ydlidar::core::serial::termios2::c_oflag

Definition at line 181 of file unix_serial.cpp.

38.50.2.8 speed_t ydlidar::core::serial::termios2::c_ospeed

Definition at line 187 of file unix_serial.cpp.

The documentation for this struct was generated from the following file:

- core/serial/impl/unix/unix_serial.cpp

38.51 ydlidar::core::base::Thread Class Reference

`#include <thread.h>`

Public Member Functions

- Thread ()
- virtual ~Thread ()
- _size_t getHandle ()
- int terminate ()
- void * getParam ()
- int join (unsigned long timeout=-1)
- bool operator== (const Thread &right)

Static Public Member Functions

- template<class CLASS , int(CLASS::*)(void) PROC>
  static Thread ThreadCreateObjectFunctor (CLASS *pthis)
- template<class CLASS , int(CLASS::*)(void) PROC>
  static _size_t THREAD_PROC createThreadAux (void *param)
- static Thread createThread (thread_proc_t proc, void *param=NULL)

Protected Member Functions

- Thread (thread_proc_t proc, void *param)

Protected Attributes

- void * _param
- thread_proc_t _func
- _size_t _handle

38.51.1 Detailed Description

Definition at line 25 of file thread.h.

38.51.2 Constructor & Destructor Documentation

38.51.2.1 ydlidar::core::base::Thread::Thread ( ) [inline], [explicit]

Definition at line 53 of file thread.h.

38.51.2.2 virtual ydlidar::core::base::Thread::~Thread ( ) [inline], [virtual]

Definition at line 54 of file thread.h.

38.51.2.3 ydlidar::core::base::Thread::Thread (thread_proc_t *proc*, void * *param*) [inline], [explicit], [protected]

Definition at line 132 of file thread.h.

38.51.3 Member Function Documentation

38.51.3.1 static Thread ydlidar::core::base::Thread::createThread (thread_proc_t *proc*, void * *param* = NULL) [inline], [static]

Definition at line 38 of file thread.h.

38.51.3.2 template<class CLASS , int(CLASS::*)(void) PROC> static _size_t THREAD_PROC ydlidar::core::base::Thread::createThreadAux (void * *param*) [inline], [static]

Definition at line 34 of file thread.h.

38.51.3.3 _size_t ydlidar::core::base::Thread::getHandle ( ) [inline]

Definition at line 55 of file thread.h.

38.51.3.4 void* ydlidar::core::base::Thread::getParam ( ) [inline]

Definition at line 82 of file thread.h.

38.51.3.5 int ydlidar::core::base::Thread::join (unsigned long *timeout* = -1) [inline]

Definition at line 85 of file thread.h.

38.51.3.6 bool ydlidar::core::base::Thread::operator== (const Thread & *right*) [inline]

Definition at line 128 of file thread.h.

38.51.3.7 int ydlidar::core::base::Thread::terminate ( ) [inline]

Definition at line 58 of file thread.h.

38.51.3.8 template<class CLASS , int(CLASS::*)(void) PROC> static Thread ydlidar::core::base::Thread::ThreadCreateObjectFunctor (CLASS * *pthis*) [inline], [static]

Definition at line 29 of file thread.h.

38.51.4 Member Data Documentation

38.51.4.1 thread_proc_t ydlidar::core::base::Thread::_func [protected]

Definition at line 135 of file thread.h.

38.51.4.2 _size_t ydlidar::core::base::Thread::_handle [protected]

Definition at line 136 of file thread.h.

38.51.4.3 void* ydlidar::core::base::Thread::_param [protected]

Definition at line 134 of file thread.h.

The documentation for this class was generated from the following file:

- core/base/thread.h

38.52 ydlidar::core::serial::Timeout Struct Reference

`#include <serial.h>`

Public Member Functions

- Timeout (uint32_t inter_byte_timeout_=0, uint32_t read_timeout_constant_=0, uint32_t read_timeout_multiplier_=0, uint32_t write_timeout_constant_=0, uint32_t write_timeout_multiplier_=0)

Static Public Member Functions

- static uint32_t max ()
- static Timeout simpleTimeout (uint32_t timeout)

Public Attributes

- uint32_t inter_byte_timeout
- uint32_t read_timeout_constant
- uint32_t read_timeout_multiplier
- uint32_t write_timeout_constant
- uint32_t write_timeout_multiplier

38.52.1 Detailed Description

Structure for setting the timeout of the serial port, times are in milliseconds.

In order to disable the interbyte timeout, set it to Timeout::max().

Definition at line 63 of file serial.h.

38.52.2 Constructor & Destructor Documentation

38.52.2.1 ydlidar::core::serial::Timeout::Timeout (uint32_t *inter_byte_timeout_* = 0, uint32_t *read_timeout_constant_* = 0, uint32_t *read_timeout_multiplier_* = 0, uint32_t *write_timeout_constant_* = 0, uint32_t *write_timeout_multiplier_* = 0) [inline],[explicit]

Definition at line 98 of file serial.h.

38.52.3 Member Function Documentation

38.52.3.1 static uint32_t ydlidar::core::serial::Timeout::max ( ) [inline],[static]

Definition at line 67 of file serial.h.

38.52.3.2 static Timeout ydlidar::core::serial::Timeout::simpleTimeout (uint32_t *timeout*) [inline],[static]

Convenience function to generate Timeout structs using a single absolute timeout.

Parameters

timeout	A long that defines the time in milliseconds until a timeout occurs after a call to read or write is made.

Returns

Timeout struct that represents this simple timeout provided.

Definition at line 79 of file serial.h.

38.52.4 Member Data Documentation

38.52.4.1 uint32_t ydlidar::core::serial::Timeout::inter_byte_timeout

Number of milliseconds between bytes received to timeout on.

Definition at line 84 of file serial.h.

38.52.4.2 uint32_t ydlidar::core::serial::Timeout::read_timeout_constant

A constant number of milliseconds to wait after calling read.

Definition at line 86 of file serial.h.

38.52.4.3 uint32_t ydlidar::core::serial::Timeout::read_timeout_multiplier

A multiplier against the number of requested bytes to wait after calling read.

Definition at line 90 of file serial.h.

38.52.4.4 uint32_t ydlidar::core::serial::Timeout::write_timeout_constant

A constant number of milliseconds to wait after calling write.

Definition at line 92 of file serial.h.

38.52.4.5 uint32_t ydlidar::core::serial::Timeout::write_timeout_multiplier

A multiplier against the number of requested bytes to wait after calling write.

Definition at line 96 of file serial.h.

The documentation for this struct was generated from the following file:

- core/serial/serial.h

38.53 YDLidar Struct Reference

lidar instance

`#include <ydlidar_def.h>`

Public Attributes

- void * lidar
  CYdLidar instance.

38.53.1 Detailed Description

lidar instance

Definition at line 81 of file ydlidar_def.h.

38.53.2 Member Data Documentation

38.53.2.1 void* YDLidar::lidar

CYdLidar instance.

Definition at line 82 of file ydlidar_def.h.

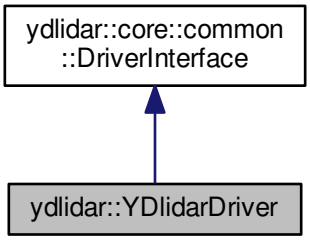
The documentation for this struct was generated from the following file:

- core/common/ydlidar_def.h

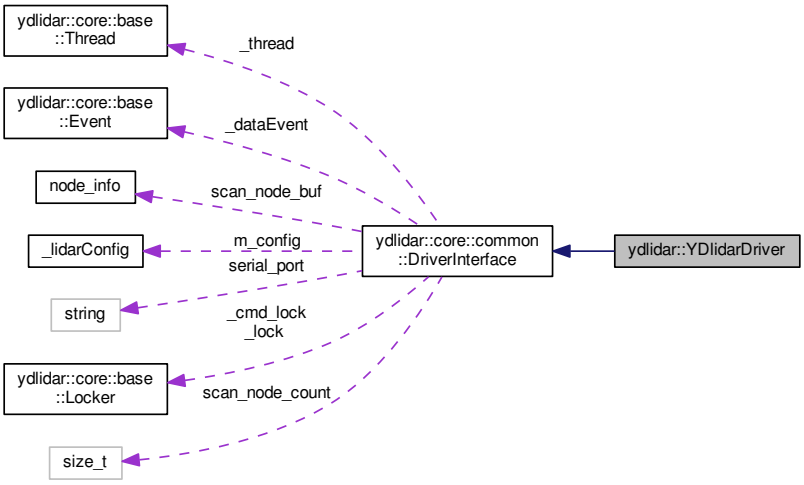
38.54 ydlidar::YDlidarDriver Class Reference

`#include <ydlidar_driver.h>`

Inheritance diagram for ydlidar::YDlidarDriver:



Collaboration diagram for ydlidar::YDlidarDriver:



Public Member Functions

- YDlidarDriver (uint8_t type=YDLIDAR_TYPE_SERIAL)
- virtual ~YDlidarDriver ()
- virtual result_t connect (const char *port_path, uint32_t baudrate)

  Connecting Lidar
  After the connection if successful, you must use ::disconnect to close.
- virtual const char * DescribeError (bool isTCP=true)

  Returns a human-readable description of the given error code or the last error code of a socket or serial port.

- virtual void disconnect ()

  Disconnect the LiDAR.
- virtual std::string getSDKVersion ()

  Get SDK Version
  static function.
- virtual bool isscanning () const

  Is the Lidar in the scan
  .
- virtual bool isconnected () const

  Is it connected to the lidar
  .
- virtual void setIntensities (const bool &isintensities)

  Is there intensity
  .
- virtual void setAutoReconnect (const bool &enable)

  whether to support hot plug
- virtual result_t getHealth (device_health &health, uint32_t timeout=DEFAULT_TIMEOUT)

  get Health status
- virtual result_t getDeviceInfo (device_info &info, uint32_t timeout=DEFAULT_TIMEOUT)

  get Device information
- virtual result_t startScan (bool force=false, uint32_t timeout=DEFAULT_TIMEOUT)

  Turn on scanning
  .
- virtual result_t stop ()

  turn off scanning
- virtual result_t grabScanData (node_info *nodebuffer, size_t &count, uint32_t timeout=DEFAULT_TIMEOUT)

  Get a circle of laser data
  .
- result_t ascendScanData (node_info *nodebuffer, size_t count)

  Normalized angle
  Normalize the angel between 0 and 360.
- result_t reset (uint32_t timeout=DEFAULT_TIMEOUT)

  reset lidar
- result_t startMotor ()

  start motor
- result_t stopMotor ()

  stop motor
- virtual result_t getScanFrequency (scan_frequency &frequency, uint32_t timeout=DEFAULT_TIMEOUT)

  Get lidar scan frequency
  .
- virtual result_t setScanFrequencyAdd (scan_frequency &frequency, uint32_t timeout=DEFAULT_TIMEOUT)

  Increase the scanning frequency by 1.0 HZ
  .
- virtual result_t setScanFrequencyDis (scan_frequency &frequency, uint32_t timeout=DEFAULT_TIMEOUT)

  Reduce the scanning frequency by 1.0 HZ
  .

- virtual result_t setScanFrequencyAddMic (scan_frequency &frequency, uint32_t timeout=DEFAULT_TIME↩OUT)

  Increase the scanning frequency by 0.1 HZ

  .
- virtual result_t setScanFrequencyDisMic (scan_frequency &frequency, uint32_t timeout=DEFAULT_TIME↩OUT)

  Reduce the scanning frequency by 0.1 HZ

  .
- virtual result_t getSamplingRate (sampling_rate &rate, uint32_t timeout=DEFAULT_TIMEOUT)

  Get lidar sampling frequency

  .
- virtual result_t setSamplingRate (sampling_rate &rate, uint32_t timeout=DEFAULT_TIMEOUT)

  Set the lidar sampling frequency

  .
- virtual result_t getZeroOffsetAngle (offset_angle &angle, uint32_t timeout=DEFAULT_TIMEOUT)

  get lidar zero offset angle

Static Public Member Functions

- static std::map< std::string, std::string > lidarPortList ()

  lidarPortList Get Lidar Port lists

Protected Member Functions

- result_t getAutoZeroOffsetAngle (offset_angle &angle, uint32_t timeout=DEFAULT_TIMEOUT)

  get lidar zero offset angle

- result_t createThread ()

  Data parsing thread

  .
- result_t startAutoScan (bool force=false, uint32_t timeout=DEFAULT_TIMEOUT)

  Automatically reconnect the lidar

  .
- result_t stopScan (uint32_t timeout=DEFAULT_TIMEOUT)

  stop Scanning state
- result_t checkDeviceInfo (uint8_t *recvBuffer, uint8_t byte, int recvPos, int recvSize, int pos)

  check single-channel lidar device information
- result_t waitDevicePackage (uint32_t timeout=DEFAULT_TIMEOUT)

  waiting device information
- result_t waitPackage (node_info *node, uint32_t timeout=DEFAULT_TIMEOUT)

  Unpacking

  .
- result_t waitScanData (node_info *nodebuffer, size_t &count, uint32_t timeout=DEFAULT_TIMEOUT)

  get unpacked data

- int cacheScanData ()

  data parsing thread

- result_t sendCommand (uint8_t cmd, const void *payload=NULL, size_t payloadsize=0)

  send data to lidar

- result_t waitResponseHeader (lidar_ans_header *header, uint32_t timeout=DEFAULT_TIMEOUT)
  waiting for package header
- result_t waitForData (size_t data_count, uint32_t timeout=DEFAULT_TIMEOUT, size_t *returned_size=NULL)
  Waiting for the specified size data from the lidar
  .
- result_t getData (uint8_t *data, size_t size)
  get data from serial
- result_t sendData (const uint8_t *data, size_t size)
  send data to serial
- void checkTransDelay ()
  checkTransDelay
- void disableDataGrabbing ()
  disable Data scan channel
- void setDTR ()
  set DTR
- void clearDTR ()
  clear DTR
- void flushSerial ()
  flushSerial
- result_t checkAutoConnecting ()
  checkAutoConnecting

Additional Inherited Members

38.54.1 Detailed Description

Class that provides a lidar interface.

Definition at line 67 of file ydlidar_driver.h.

38.54.2 Constructor & Destructor Documentation

38.54.2.1 ydlidar::YDlidarDriver::YDlidarDriver (uint8_t *type* = YDLIDAR_TYPE_SERIAL) [explicit]

A constructor. A more elaborate description of the constructor.

Definition at line 37 of file ydlidar_driver.cpp.

38.54.2.2 ydlidar::YDlidarDriver::~YDlidarDriver ( ) [virtual]

A destructor. A more elaborate description of the destructor.

Definition at line 89 of file ydlidar_driver.cpp.

38.54.3 Member Function Documentation

38.54.3.1 result_t ydlidar::YDlidarDriver::ascendScanData (node_info * *nodebuffer,* size_t *count*)

Normalized angle
Normalize the angel between 0 and 360.

Parameters

in	*nodebuffer*	Laser data
in	*count*	one circle of laser points

Returns

return status

Return values

RESULT_OK	success
RESULT_FAILE	failed

Note

Before the normalization, you must use the ::grabScanData function to get the laser data successfully.

Definition at line 1167 of file ydlidar_driver.cpp.

38.54.3.2 int ydlidar::YDlidarDriver::cacheScanData ( ) [protected]

data parsing thread

Definition at line 471 of file ydlidar_driver.cpp.

38.54.3.3 result_t ydlidar::YDlidarDriver::checkAutoConnecting ( ) [protected]

checkAutoConnecting

Definition at line 405 of file ydlidar_driver.cpp.

38.54.3.4 result_t ydlidar::YDlidarDriver::checkDeviceInfo (uint8_t * *recvBuffer*, uint8_t *byte*, int *recvPos*, int *recvSize*, int *pos*) [protected]

check single-channel lidar device information

Parameters

recvBuffer	LiDAR Data buffer
byte	current byte
recvPos	current recived pos
recvSize	Buffer size
pos	Device Buffer pos

Returns

status

Return values

RESULT_OK	success
RESULT_FAILE	failed

Definition at line 553 of file ydlidar_driver.cpp.

38.54.3.5 void ydlidar::YDlidarDriver::checkTransDelay ( ) [protected]

checkTransDelay

Definition at line 1415 of file ydlidar_driver.cpp.

38.54.3.6 void ydlidar::YDlidarDriver::clearDTR ( ) [protected]

clear DTR

Definition at line 179 of file ydlidar_driver.cpp.

38.54.3.7 result_t ydlidar::YDlidarDriver::connect (const char * *port_path,* uint32_t *baudrate*) [virtual]

Connecting Lidar
After the connection if successful, you must use ::disconnect to close.

Parameters

in	*port_path*	serial port
in	*baudrate*	serial baudrate, YDLIDAR-SS: 230400 G2-SS-1 R2-SS-1

Returns

connection status

Return values

0	success
<	0 failed

Note

After the connection if successful, you must use ::disconnect to close

See also

function ::YDlidarDriver::disconnect ()

Implements ydlidar::core::common::DriverInterface.

Definition at line 123 of file ydlidar_driver.cpp.

38.54.3.8 result_t ydlidar::YDlidarDriver::createThread ( ) [protected]

Data parsing thread
.

Note

Before you create a dta parsing thread, you must use the ::startScan function to start the lidar scan successfully.

Definition at line 1527 of file ydlidar_driver.cpp.

38.54.3.9 const char * ydlidar::YDlidarDriver::DescribeError (bool *isTCP* = true) [virtual]

Returns a human-readable description of the given error code or the last error code of a socket or serial port.

Parameters

isTCP	TCP or UDP

Returns

error information

Implements ydlidar::core::common::DriverInterface.

Definition at line 157 of file ydlidar_driver.cpp.

38.54.3.10 void ydlidar::YDlidarDriver::disableDataGrabbing ( ) [protected]

disable Data scan channel

Definition at line 225 of file ydlidar_driver.cpp.

38.54.3.11 void ydlidar::YDlidarDriver::disconnect ( ) [virtual]

Disconnect the LiDAR.

Implements ydlidar::core::common::DriverInterface.

Definition at line 203 of file ydlidar_driver.cpp.

38.54.3.12 void ydlidar::YDlidarDriver::flushSerial ( ) [protected]

flushSerial

Definition at line 188 of file ydlidar_driver.cpp.

38.54.3.13 result_t ydlidar::YDlidarDriver::getAutoZeroOffsetAngle (offset_angle & *angle,* uint32_t *timeout =* DEFAULT_TIMEOUT) [protected]

get lidar zero offset angle

Parameters

in	*angle*	zero offset angle
in	*timeout*	timeout

Returns

return status

Return values

RESULT_OK	success
RESULT_FAILE	failed

Note

scanning state,perform currect operation.

Definition at line 2002 of file ydlidar_driver.cpp.

38.54.3.14 result_t ydlidar::YDlidarDriver::getData (uint8_t * *data,* size_t *size*) [protected]

get data from serial

Parameters

in	*data*	data
in	*size*	date size

Returns

return status

Return values

RESULT_OK	success
RESULT_FAILE	failed

Definition at line 306 of file ydlidar_driver.cpp.

38.54.3.15 result_t ydlidar::YDlidarDriver::getDeviceInfo (device_info & *info*, uint32_t *timeout* = DEFAULT_TIMEOUT) [virtual]

get Device information

Parameters

in	*info*	Device information
in	*timeout*	timeout

Returns

result status

Return values

RESULT_OK	success
RESULT_FAILE	or RESULT_TIMEOUT failed

Implements ydlidar::core::common::DriverInterface.

Definition at line 1329 of file ydlidar_driver.cpp.

38.54.3.16 result_t ydlidar::YDlidarDriver::getHealth (device_health & *health*, uint32_t *timeout* = DEFAULT_TIMEOUT) [virtual]

get Health status

Returns

result status

Return values

RESULT_OK	success
RESULT_FAILE	or RESULT_TIMEOUT failed

Implements ydlidar::core::common::DriverInterface.

Definition at line 1276 of file ydlidar_driver.cpp.

38.54.3.17 result_t ydlidar::YDlidarDriver::getSamplingRate (sampling_rate & *rate,* uint32_t *timeout* = DEFAULT_TIMEOUT) [virtual]

Get lidar sampling frequency
.

Parameters

in	*frequency*	sampling frequency
in	*timeout*	timeout

Returns

return status

Return values

RESULT_OK	success
RESULT_FAILE	failed

Note

Non-scan state, perform currect operation.

Implements ydlidar::core::common::DriverInterface.

Definition at line 1874 of file ydlidar_driver.cpp.

38.54.3.18 result_t ydlidar::YDlidarDriver::getScanFrequency (scan_frequency & *frequency,* uint32_t *timeout* = DEFAULT_TIMEOUT) [virtual]

Get lidar scan frequency
.

Parameters

in	*frequency*	scanning frequency
in	*timeout*	timeout

Returns

return status

Return values

RESULT_OK	success
RESULT_FAILE	failed

Note

Non-scan state, perform currect operation.

Implements ydlidar::core::common::DriverInterface.

Definition at line 1659 of file ydlidar_driver.cpp.

38.54.3.19 std::string ydlidar::YDlidarDriver::getSDKVersion ( ) [virtual]

Get SDK Version
static function.

Returns

Version

Implements ydlidar::core::common::DriverInterface.

Definition at line 2043 of file ydlidar_driver.cpp.

38.54.3.20 result_t ydlidar::YDlidarDriver::getZeroOffsetAngle (offset_angle & *angle*, uint32_t *timeout* = DEFAULT_TIMEOUT) [virtual]

get lidar zero offset angle

Parameters

in	*angle*	zero offset angle
in	*timeout*	timeout

Returns

return status

Return values

RESULT_OK	success
RESULT_FAILE	failed

Note

Non-scan state, perform currect operation.

Implements ydlidar::core::common::DriverInterface.

Definition at line 1959 of file ydlidar_driver.cpp.

38.54.3.21 result_t ydlidar::YDlidarDriver::grabScanData (node_info * *nodebuffer*, size_t & *count*, uint32_t *timeout* = DEFAULT_TIMEOUT) [virtual]

Get a circle of laser data
.

Parameters

in	*nodebuffer*	Laser data
in	*count*	one circle of laser points
in	*timeout*	timeout

Returns

return status

Return values

RESULT_OK	success
RESULT_FAILE	failed

Note

Before starting, you must start the start the scan successfully with the ::startScan function

Implements ydlidar::core::common::DriverInterface.

Definition at line 1138 of file ydlidar_driver.cpp.

38.54.3.22 bool ydlidar::YDlidarDriver::isconnected ( ) const [virtual]

Is it connected to the lidar
.

Returns

connection status

Return values

true	connected
false	Non-connected

Implements ydlidar::core::common::DriverInterface.

Definition at line 239 of file ydlidar_driver.cpp.

38.54.3.23 bool ydlidar::YDlidarDriver::isscanning ( ) const [virtual]

Is the Lidar in the scan
.

Returns

scanning status

Return values

true	scanning
false	non-scanning

Implements ydlidar::core::common::DriverInterface.

Definition at line 235 of file ydlidar_driver.cpp.

38.54.3.24 std::map< std::string, std::string > ydlidar::YDlidarDriver::lidarPortList ( ) [static]

lidarPortList Get Lidar Port lists

Returns

online lidars

Definition at line 2046 of file ydlidar_driver.cpp.

38.54.3.25 result_t ydlidar::YDlidarDriver::reset (uint32_t *timeout* = DEFAULT_TIMEOUT)

reset lidar

Parameters

in	*timeout*	timeout

Returns

return status

Return values

RESULT_OK	success
RESULT_FAILE	failed

Note

Non-scan state, perform currect operation.

Definition at line 1605 of file ydlidar_driver.cpp.

38.54.3.26 result_t ydlidar::YDlidarDriver::sendCommand (uint8_t *cmd*, const void * *payload* = NULL, size_t *payloadsize* = 0) [protected]

send data to lidar

Parameters

in	*cmd*	command code
in	*payload*	payload
in	*payloadsize*	payloadsize

Returns

result status

Return values

RESULT_OK	success
RESULT_FAILE	failed

Definition at line 243 of file ydlidar_driver.cpp.

38.54.3.27 result_t ydlidar::YDlidarDriver::sendData (const uint8_t * *data*, size_t *size*) [protected]

send data to serial

Parameters

in	*data*	data
in	*size*	data size

Returns

return status

Return values

RESULT_OK	success
RESULT_FAILE	failed

Definition at line 281 of file ydlidar_driver.cpp.

38.54.3.28 void ydlidar::YDlidarDriver::setAutoReconnect (const bool & *enable*) [virtual]

whether to support hot plug

设置雷达异常自动重新连接

Parameters

in	*enable*	hot plug : true support false no support
in	*enable*	是否开启自动重连: true 开启 false 关闭

Implements ydlidar::core::common::DriverInterface.

Definition at line 1410 of file ydlidar_driver.cpp.

38.54.3.29 void ydlidar::YDlidarDriver::setDTR ( ) [protected]

set DTR

Definition at line 168 of file ydlidar_driver.cpp.

38.54.3.30 void ydlidar::YDlidarDriver::setIntensities (const bool & *isintensities*) [virtual]

Is there intensity
.

Parameters

in	*isintensities*	intentsity true intensity false no intensity

Implements ydlidar::core::common::DriverInterface.

Definition at line 1385 of file ydlidar_driver.cpp.

38.54.3.31 result_t ydlidar::YDlidarDriver::setSamplingRate (sampling_rate & *rate*, uint32_t *timeout* = DEFAULT_TIMEOUT) [virtual]

Set the lidar sampling frequency
.

Parameters

in	*rate*	sampling frequency
in	*timeout*	timeout

Returns

return status

Return values

RESULT_OK	success
RESULT_FAILE	failed

Note

Non-scan state, perform currect operation.

Implements ydlidar::core::common::DriverInterface.

Definition at line 1917 of file ydlidar_driver.cpp.

38.54.3.32 result_t ydlidar::YDlidarDriver::setScanFrequencyAdd (scan_frequency & *frequency*, uint32_t *timeout* = DEFAULT_TIMEOUT) [virtual]

Increase the scanning frequency by 1.0 HZ
.

Parameters

in	*frequency*	scanning frequency
in	*timeout*	timeout

Returns

return status

Return values

RESULT_OK	success
RESULT_FAILE	failed

Note

Non-scan state, perform currect operation.

Implements ydlidar::core::common::DriverInterface.

Definition at line 1702 of file ydlidar_driver.cpp.

38.54.3.33 result_t ydlidar::YDlidarDriver::setScanFrequencyAddMic (scan_frequency & *frequency*, uint32_t *timeout* = DEFAULT_TIMEOUT) [virtual]

Increase the scanning frequency by 0.1 HZ
.

Parameters

in	*frequency*	scanning frequency
in	*timeout*	timeout

Returns

return status

Return values

RESULT_OK	success
RESULT_FAILE	failed

Note

Non-scan state, perform currect operation.

Implements ydlidar::core::common::DriverInterface.

Definition at line 1788 of file ydlidar_driver.cpp.

38.54.3.34 result_t ydlidar::YDlidarDriver::setScanFrequencyDis (scan_frequency & *frequency*, uint32_t *timeout* = DEFAULT_TIMEOUT) [virtual]

Reduce the scanning frequency by 1.0 HZ
.

Parameters

in	*frequency*	scanning frequency
in	*timeout*	timeout

Returns

return status

Return values

RESULT_OK	success
RESULT_FAILE	failed

Note

Non-scan state, perform currect operation.

Implements ydlidar::core::common::DriverInterface.

Definition at line 1745 of file ydlidar_driver.cpp.

38.54.3.35 result_t ydlidar::YDlidarDriver::setScanFrequencyDisMic (scan_frequency & *frequency*, uint32_t *timeout* = DEFAULT_TIMEOUT) [virtual]

Reduce the scanning frequency by 0.1 HZ
.

Parameters

in	*frequency*	scanning frequency
in	*timeout*	timeout

Returns

return status

Return values

RESULT_OK	success
RESULT_FAILE	failed

Note

Non-scan state, perform currect operation.

Implements ydlidar::core::common::DriverInterface.

Definition at line 1831 of file ydlidar_driver.cpp.

38.54.3.36 result_t ydlidar::YDlidarDriver::startAutoScan (bool *force* = false, uint32_t *timeout* = DEFAULT_TIMEOUT) [protected]

Automatically reconnect the lidar
.

Parameters

in	*force*	scan model
in	*timeout*	timeout

Returns

return status

Return values

RESULT_OK	success
RESULT_FAILE	failed

Note

Lidar abnormality automatically reconnects.

Definition at line 1540 of file ydlidar_driver.cpp.

38.54.3.37 result_t ydlidar::YDlidarDriver::startMotor ( )

start motor

Returns

return status

Return values

RESULT_OK	success
RESULT_FAILE	failed

Definition at line 1625 of file ydlidar_driver.cpp.

38.54.3.38 result_t ydlidar::YDlidarDriver::startScan (bool *force* = `false`, uint32_t *timeout* = DEFAULT_TIMEOUT) [virtual]

Turn on scanning
.

Parameters

`in`	*force*	Scan mode
`in`	*timeout*	timeout

Returns

result status

Return values

RESULT_OK	success
RESULT_FAILE	failed

Note

Just turn it on once

Implements ydlidar::core::common::DriverInterface.

Definition at line 1461 of file ydlidar_driver.cpp.

38.54.3.39 result_t ydlidar::YDlidarDriver::stop ( ) [virtual]

turn off scanning

Returns

result status

Return values

RESULT_OK	success
RESULT_FAILE	failed

Implements ydlidar::core::common::DriverInterface.

Definition at line 1586 of file ydlidar_driver.cpp.

38.54.3.40 result_t ydlidar::YDlidarDriver::stopMotor ( )

stop motor

Returns

return status

Return values

RESULT_OK	success
RESULT_FAILE	failed

Definition at line 1642 of file ydlidar_driver.cpp.

38.54.3.41 result_t ydlidar::YDlidarDriver::stopScan (uint32_t *timeout* = DEFAULT_TIMEOUT) [protected]

stop Scanning state

Parameters

timeout	timeout

Returns

status

Return values

RESULT_OK	success
RESULT_FAILE	failed

Definition at line 1512 of file ydlidar_driver.cpp.

38.54.3.42 result_t ydlidar::YDlidarDriver::waitDevicePackage (uint32_t *timeout* = DEFAULT_TIMEOUT) [protected]

waiting device information

Parameters

timeout	timeout

Returns

status

Return values

RESULT_OK	success
RESULT_FAILE	failed

Definition at line 697 of file ydlidar_driver.cpp.

38.54.3.43 result_t ydlidar::YDlidarDriver::waitForData (size_t *data_count*, uint32_t *timeout* = DEFAULT_TIMEOUT, size_t * *returned_size* = NULL) [protected]

Waiting for the specified size data from the lidar
.

Parameters

in	*data_count*	wait max data size
in	*timeout*	timeout
in	*returned_size*	really data size

Returns

return status

Return values

RESULT_OK	success
RESULT_TIMEOUT	wait timeout
RESULT_FAILE	failed

Note

when timeout = -1, it will block...

Definition at line 394 of file ydlidar_driver.cpp.

38.54.3.44 result_t ydlidar::YDlidarDriver::waitPackage (node_info * *node*, uint32_t *timeout* = DEFAULT_TIMEOUT) [protected]

Unpacking
.

Parameters

in	*node*	lidar point information
in	*timeout*	timeout

Definition at line 741 of file ydlidar_driver.cpp.

38.54.3.45 result_t ydlidar::YDlidarDriver::waitResponseHeader (lidar_ans_header * *header*, uint32_t *timeout* = DEFAULT_TIMEOUT) [protected]

waiting for package header

Parameters

in	*header*	package header
in	*timeout*	timeout

Returns

return status

Return values

RESULT_OK	success
RESULT_TIMEOUT	timeout
RESULT_FAILE	failed

Note

when timeout = -1, it will block...

Definition at line 327 of file ydlidar_driver.cpp.

38.54.3.46 result_t ydlidar::YDlidarDriver::waitScanData (node_info * *nodebuffer*, size_t & *count*, uint32_t *timeout* = DEFAULT_TIMEOUT) [protected]

get unpacked data

Parameters

in	*nodebuffer*	laser node
in	*count*	lidar points size
in	*timeout*	timeout

Returns

result status

Return values

RESULT_OK	success
RESULT_TIMEOUT	timeout
RESULT_FAILE	failed

Definition at line 1085 of file ydlidar_driver.cpp.

The documentation for this class was generated from the following files:

- src/ydlidar_driver.h
- src/ydlidar_driver.cpp

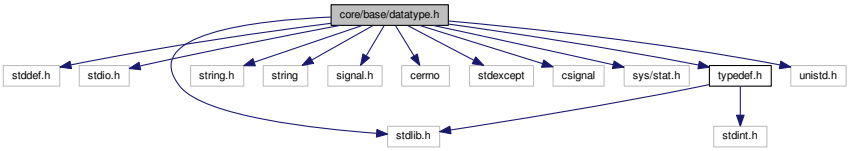
Chapter 39

File Documentation

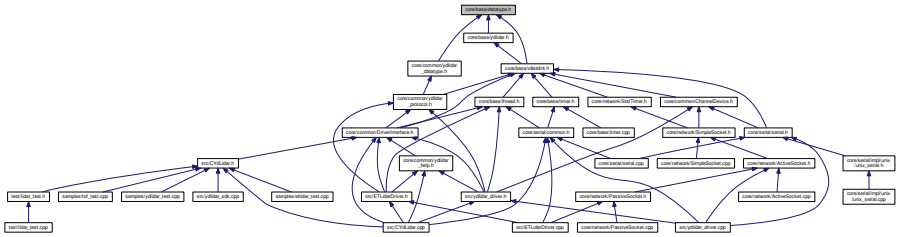
39.1 core/base/datatype.h File Reference

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <string>
#include <signal.h>
#include <cerrno>
#include <stdexcept>
#include <csignal>
#include <sys/stat.h>
#include "typedef.h"
#include <unistd.h>
```

Include dependency graph for datatype.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define _itoa(value, str, radix) {sprintf(str, "%d", value);}
- #define UNUSED(x) (void)x
- #define _access access
- #define valName(val) (#val)
- #define valLastName(val)
- #define FRAME_PREAMBLE 0xFFEE
- #define LIDAR_2D 0x2
- #define DATA_FRAME 0x1
- #define DEFAULT_INTENSITY 10
- #define DSL(c, i) ((c << i) & (0xFF << i))
- #define __WORDSIZE 32
- #define __small_endian
- #define __attribute__(x)
- #define RESULT_OK 0
- #define RESULT_TIMEOUT -1
- #define RESULT_FAIL -2
- #define INVALID_TIMESTAMP (0)
- #define IS_OK(x) ((x) == RESULT_OK)
- #define IS_TIMEOUT(x) ((x) == RESULT_TIMEOUT)
- #define IS_FAIL(x) ((x) == RESULT_FAIL)

Typedefs

- typedef int32_t result_t

39.1.1 Macro Definition Documentation

39.1.1.1 #define __attribute__(*x*)

Definition at line 111 of file datatype.h.

39.1.1.2 #define __small_endian

Definition at line 108 of file datatype.h.

39.1.1.3 #define __WORDSIZE 32

Definition at line 64 of file datatype.h.

39.1.1.4 #define _access access

Definition at line 27 of file datatype.h.

39.1.1.5 #define _itoa(value, str, radix) {sprintf(str, "%d", value);}

Definition at line 21 of file datatype.h.

39.1.1.6 #define DATA_FRAME 0x1

Definition at line 52 of file datatype.h.

39.1.1.7 #define DEFAULT_INTENSITY 10

Definition at line 53 of file datatype.h.

39.1.1.8 #define DSL(c, i) ((c << i) & (0xFF << i))

Definition at line 54 of file datatype.h.

39.1.1.9 #define FRAME_PREAMBLE 0xFFEE

Definition at line 50 of file datatype.h.

39.1.1.10 #define INVALID_TIMESTAMP (0)

Definition at line 141 of file datatype.h.

39.1.1.11 #define IS_FAIL(x) ((x) == RESULT_FAIL)

Definition at line 146 of file datatype.h.

39.1.1.12 #define IS_OK(x) ((x) == RESULT_OK)

Definition at line 144 of file datatype.h.

39.1.1.13 #define IS_TIMEOUT(x) ((x) == RESULT_TIMEOUT)

Definition at line 145 of file datatype.h.

39.1.1.14 #define LIDAR_2D 0x2

Definition at line 51 of file datatype.h.

39.1.1.15 #define RESULT_FAIL -2

Definition at line 139 of file datatype.h.

39.1.1.16 #define RESULT_OK 0

Definition at line 137 of file datatype.h.

39.1.1.17 #define RESULT_TIMEOUT -1

Definition at line 138 of file datatype.h.

39.1.1.18 #define UNUSED(*x*) (void)x

Definition at line 24 of file datatype.h.

39.1.1.19 #define valLastName(*val*)

Value:

```
{ \
    char* strToken; \
    char str[64]; \
    strncpy(str, (const char*)val, sizeof(str)); \
    strToken = strtok(str, "."); \
    while (strToken != NULL) { \
        strcpy(val, (const char*)strToken); \
        strToken = strtok(NULL, "."); \
    } \
}
```

Definition at line 31 of file datatype.h.

39.1.1.20 #define valName(*val*) (#val)

Definition at line 30 of file datatype.h.

39.1.2 Typedef Documentation

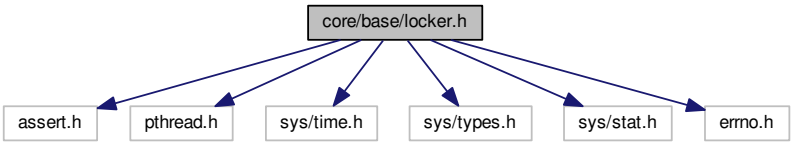
39.1.2.1 typedef int32_t result_t

Definition at line 135 of file datatype.h.

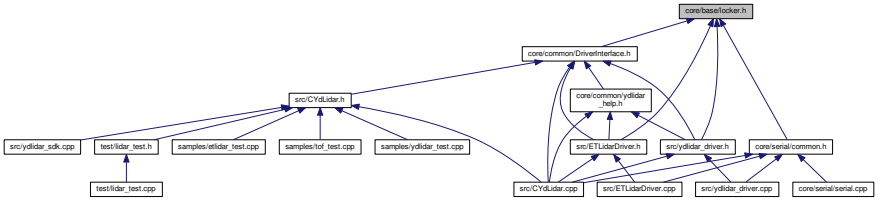
39.2 core/base/locker.h File Reference

```
#include <assert.h>
#include <pthread.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <errno.h>
```

Include dependency graph for locker.h:



This graph shows which files directly or indirectly include this file:



Classes

- class ydlidar::core::base::Locker
- class ydlidar::core::base::Event
- class ydlidar::core::base::ScopedLocker

Namespaces

- ydlidar

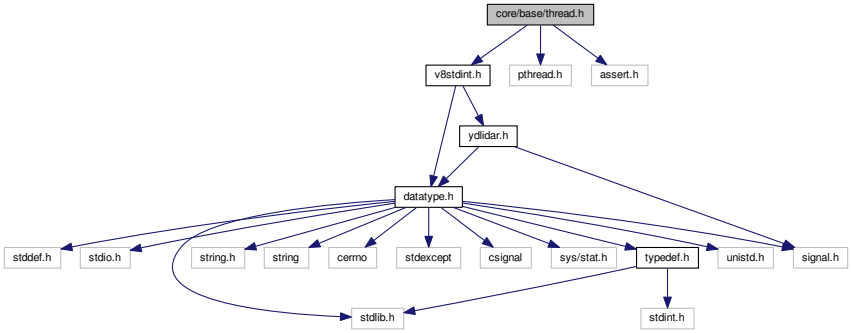
  ydlidar
- ydlidar::core

  ydlidar core
- ydlidar::core::base

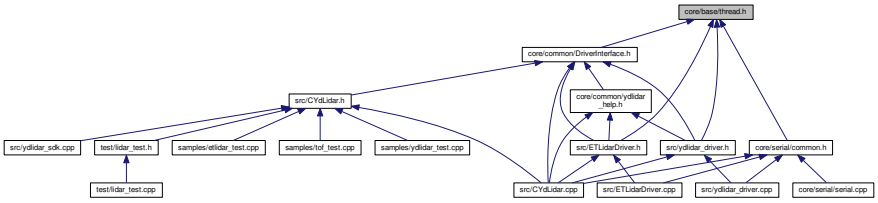
39.3 core/base/thread.h File Reference

```
#include "v8stdint.h"
#include <pthread.h>
#include <assert.h>
```

Include dependency graph for thread.h:



This graph shows which files directly or indirectly include this file:



Classes

- class ydlidar::core::base::Thread

Namespaces

- ydlidar

  ydlidar
- ydlidar::core

  ydlidar core
- ydlidar::core::base

Macros

- #define CLASS_THREAD(c, x) Thread::ThreadCreateObjectFunctor<c, &c::x>(this)

39.3.1 Macro Definition Documentation

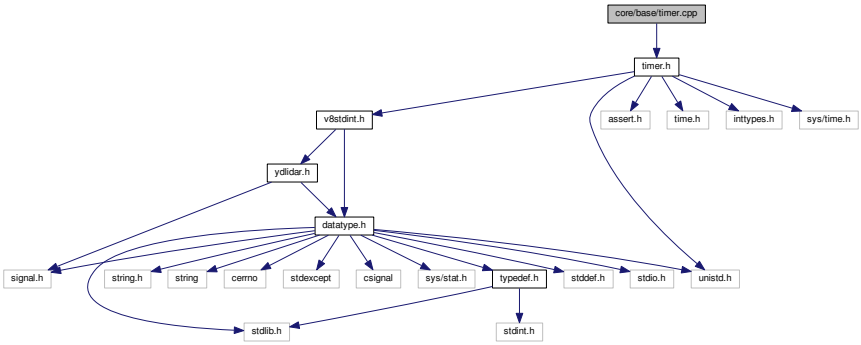
39.3.1.1 #define CLASS_THREAD(c, x) Thread::ThreadCreateObjectFunctor<c, &c::x>(this)

Definition at line 19 of file thread.h.

39.4 core/base/timer.cpp File Reference

```
#include "timer.h"
```

Include dependency graph for timer.cpp:



Namespaces

- impl

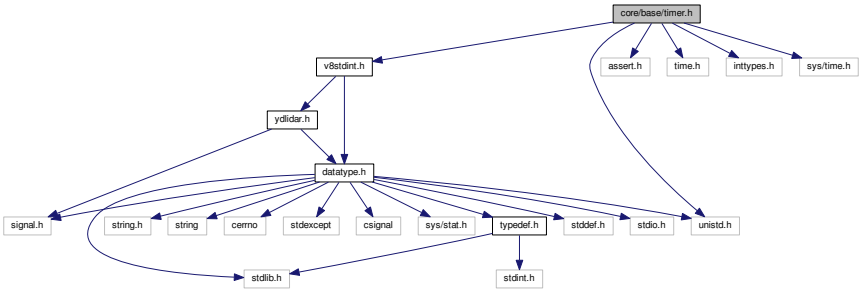
Functions

- uint32_t impl::getHDTimer ()
- uint64_t impl::getCurrentTime ()

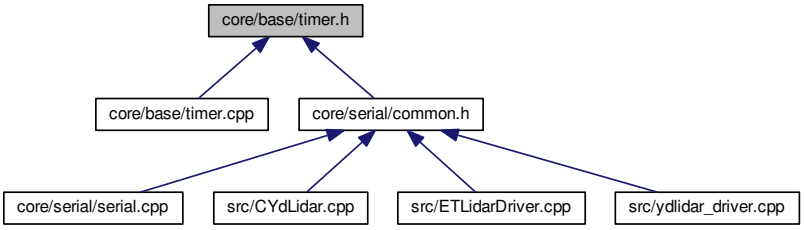
39.5 core/base/timer.h File Reference

```
#include "v8stdint.h"
#include <assert.h>
#include <time.h>
#include <inttypes.h>
#include <sys/time.h>
#include <unistd.h>
```

Include dependency graph for timer.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- impl

Macros

- #define BEGIN_STATIC_CODE(_blockname_)
- #define END_STATIC_CODE(_blockname_) } _instance_##_blockname_;
- #define getms() impl::getHDTimer()
- #define getTime() impl::getCurrentTime()

Functions

- static void delay (uint32_t ms)
- uint32_t impl::getHDTimer ()
- uint64_t impl::getCurrentTime ()

39.5.1 Macro Definition Documentation

39.5.1.1 #define BEGIN_STATIC_CODE(*_blockname_*)

Value:

```
static class _static_code_##_blockname_ {    \
    public:     \
    _static_code_##_blockname_ ()
```

Definition at line 8 of file timer.h.

39.5.1.2 #define END_STATIC_CODE(*_blockname_*) } _instance_##_blockname_;

Definition at line 14 of file timer.h.

39.5.1.3 #define getms( ) impl::getHDTimer()

Definition at line 50 of file timer.h.

39.5.1.4 #define getTime( ) impl::getCurrentTime()

Definition at line 51 of file timer.h.

39.5.2 Function Documentation

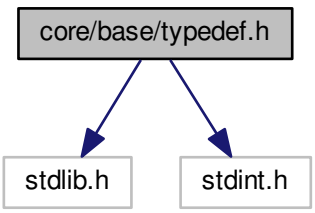
39.5.2.1 static void delay (uint32_t *ms*) [inline], [static]

Definition at line 25 of file timer.h.

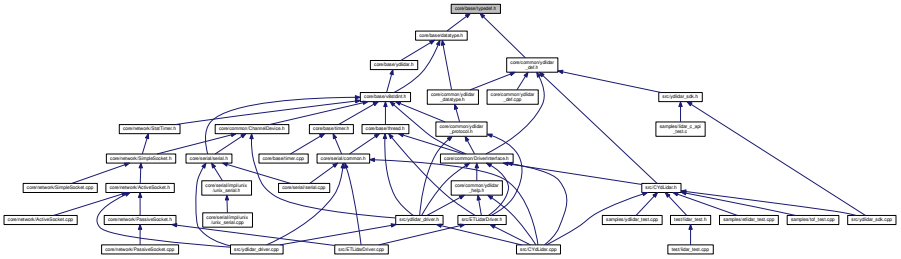
39.6 core/base/typedef.h File Reference

```
#include <stdlib.h>
#include <stdint.h>
```

Include dependency graph for typedef.h:

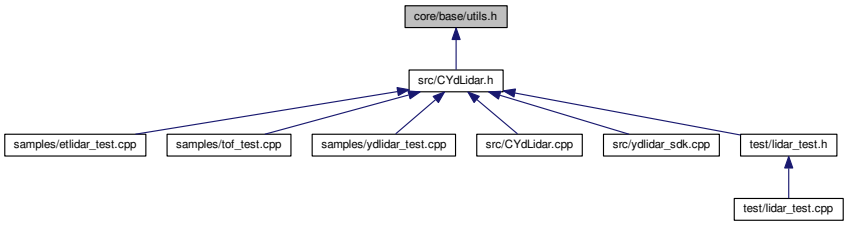


This graph shows which files directly or indirectly include this file:



39.7 core/base/utils.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define YDLIDAR_API

39.7.1 Macro Definition Documentation

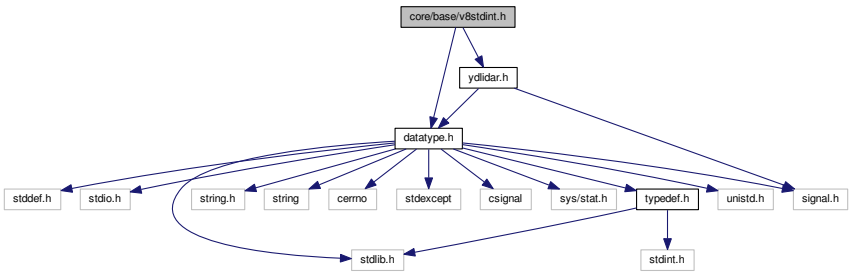
39.7.1.1 #define YDLIDAR_API

Definition at line 17 of file utils.h.

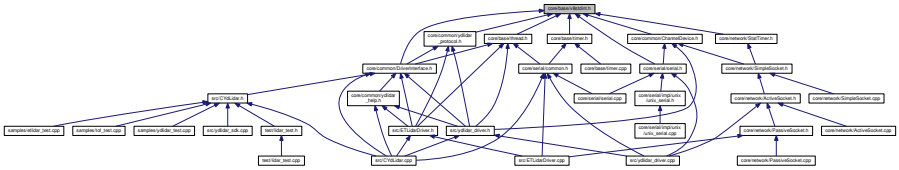
39.8 core/base/v8stdint.h File Reference

```
#include "datatype.h"
#include "ydlidar.h"
```

Include dependency graph for v8stdint.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define TRUE 1
- #define FALSE 0
- #define htonll(x) ((((uint64)htonl(x)) << 32) + htonl(x >> 32))
- #define ntohll(x) ((((uint64)ntohl(x)) << 32) + ntohl(x >> 32))
- #define STRUCT_STAT struct stat
- #define LSTAT(x, y) lstat(x,y)
- #define FILE_HANDLE FILE *
- #define CLEARERR(x) clearerr(x)
- #define FCLOSE(x) fclose(x)
- #define FEOF(x) feof(x)
- #define FERROR(x) ferror(x)
- #define FFLUSH(x) fflush(x)
- #define FILENO(s) fileno(s)
- #define FOPEN(x, y) fopen(x, y)
- #define FSTAT(s, st) fstat(FILENO(s), st)
- #define STAT_BLK_SIZE(x) ((x).st_blksize)
- #define DEFAULT_CONNECTION_TIMEOUT_SEC 2
- #define DEFAULT_CONNECTION_TIMEOUT_USEC 800000
- #define DEFAULT_REV_TIMEOUT_SEC 2
- #define DEFAULT_REV_TIMEOUT_USEC 800000
- #define STRTOULL(x) strtoull(x, NULL, 10)
- #define SNPRINTF snprintf
- #define PRINTF printf
- #define VPRINTF vprintf
- #define FPRINTF fprintf
- #define MILLISECONDS_CONVERSION 1000
- #define MICROSECONDS_CONVERSION 1000000
- #define NANOECONDS_CONVERSION 1000000000

Typedefs

- typedef _size_t(THREAD_PROC * thread_proc_t) (void *)

39.8.1 Macro Definition Documentation

39.8.1.1 #define CLEARERR(x) clearerr(x)

Definition at line 91 of file v8stdint.h.

39.8.1.2 #define DEFAULT_CONNECTION_TIMEOUT_SEC 2

Definition at line 103 of file v8stdint.h.

39.8.1.3 #define DEFAULT_CONNECTION_TIMEOUT_USEC 800000

Definition at line 104 of file v8stdint.h.

39.8.1.4 #define DEFAULT_REV_TIMEOUT_SEC 2

Definition at line 106 of file v8stdint.h.

39.8.1.5 #define DEFAULT_REV_TIMEOUT_USEC 800000

Definition at line 107 of file v8stdint.h.

39.8.1.6 #define FALSE 0

Definition at line 13 of file v8stdint.h.

39.8.1.7 #define FCLOSE(x) fclose(x)

Definition at line 92 of file v8stdint.h.

39.8.1.8 #define FEOF(x) feof(x)

Definition at line 93 of file v8stdint.h.

39.8.1.9 #define FERROR(x) ferror(x)

Definition at line 94 of file v8stdint.h.

39.8.1.10 #define FFLUSH(x) fflush(x)

Definition at line 95 of file v8stdint.h.

39.8.1.11 #define FILE_HANDLE FILE *

Definition at line 90 of file v8stdint.h.

39.8.1.12 #define FILENO(s) fileno(s)

Definition at line 96 of file v8stdint.h.

39.8.1.13 #define FOPEN(x, y) fopen(x, y)

Definition at line 97 of file v8stdint.h.

39.8.1.14 #define FPRINTF fprintf

Definition at line 133 of file v8stdint.h.

39.8.1.15 #define FSTAT(s, st) fstat(FILENO(s), st)

Definition at line 99 of file v8stdint.h.

39.8.1.16 #define htonll(x) ((((uint64)htonl(x)) << 32) + htonl(x >> 32))

Definition at line 21 of file v8stdint.h.

39.8.1.17 #define LSTAT(x, y) lstat(x,y)

Definition at line 89 of file v8stdint.h.

39.8.1.18 #define MICROSECONDS_CONVERSION 1000000

Definition at line 139 of file v8stdint.h.

39.8.1.19 #define MILLISECONDS_CONVERSION 1000

Definition at line 138 of file v8stdint.h.

39.8.1.20 #define NANOECONDS_CONVERSION 1000000000

Definition at line 140 of file v8stdint.h.

39.8.1.21 #define ntohll(x) ((((uint64)ntohl(x)) << 32) + ntohl(x >> 32))

Definition at line 22 of file v8stdint.h.

39.8.1.22 #define PRINTF printf

Definition at line 131 of file v8stdint.h.

39.8.1.23 #define SNPRINTF snprintf

Definition at line 130 of file v8stdint.h.

39.8.1.24 #define STAT_BLK_SIZE(x) ((x).st_blksize)

Definition at line 101 of file v8stdint.h.

39.8.1.25 #define STRTOULL(x) strtoull(x, NULL, 10)

Definition at line 121 of file v8stdint.h.

39.8.1.26 #define STRUCT_STAT struct stat

Definition at line 88 of file v8stdint.h.

39.8.1.27 #define TRUE 1

Definition at line 9 of file v8stdint.h.

39.8.1.28 #define VPRINTF vprintf

Definition at line 132 of file v8stdint.h.

39.8.2 Typedef Documentation

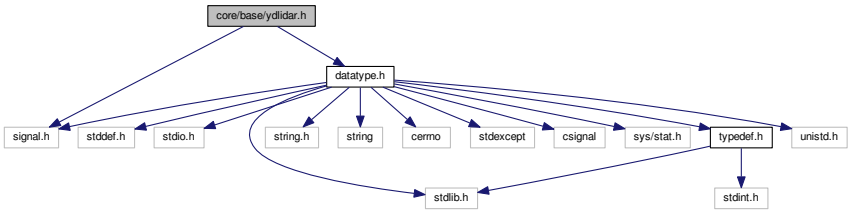
39.8.2.1 typedef _size_t(THREAD_PROC * thread_proc_t) (void *)

Definition at line 4 of file v8stdint.h.

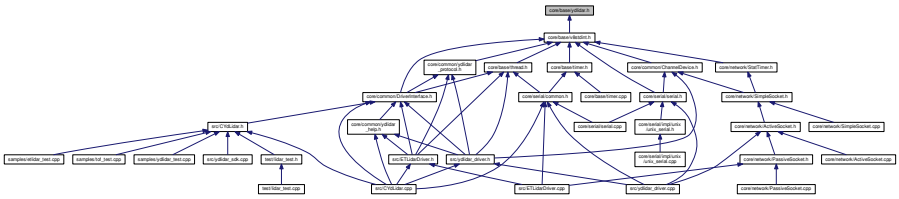
39.9 core/base/ydlidar.h File Reference

```
#include "datatype.h"
#include <signal.h>
```

Include dependency graph for ydlidar.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- ydlidar

  ydlidar
- ydlidar::core

  ydlidar core
- ydlidar::core::base

Macros

- #define PropertyBuilderByName(type, name, access_permission)

Typedefs

- typedef void(∗ signal_handler_t) (int)

Functions

- signal_handler_t set_signal_handler (int signal_value, signal_handler_t signal_handler)
- void trigger_interrupt_guard_condition (int signal_value)
- void signal_handler (int signal_value)
- void ydlidar::core::base::init ()

  initialize system state
- bool ydlidar::core::base::ok ()

  ok
- void ydlidar::core::base::shutdown ()

  shutdown
- bool ydlidar::core::base::fileExists (const std::string &filename)

  fileExists

Variables

- static volatile sig_atomic_t g_signal_status = 0
- static signal_handler_t old_signal_handler = 0

39.9.1 Macro Definition Documentation

39.9.1.1 #define PropertyBuilderByName(type, name, access_permission)

Value:

```
access_permission:\
        type m_##name;\
    public:\
    inline void set##name(type v) {\
        m_##name = v;\
    }\
    inline type get##name() const {\
        return m_##name;\
    }\
```

PropertyBuilderByName Used to generate class member variables and generate set and get methods type Variable type access_permission Variable access(public, priavte, protected)

Definition at line 10 of file ydlidar.h.

39.9.2 Typedef Documentation

39.9.2.1 typedef void(* signal_handler_t) (int)

Definition at line 32 of file ydlidar.h.

39.9.3 Function Documentation

39.9.3.1 signal_handler_t set_signal_handler (int *signal_value*, signal_handler_t *signal_handler*) [inline]

Definition at line 41 of file ydlidar.h.

39.9.3.2 void signal_handler (int *signal_value*) [inline]

Definition at line 103 of file ydlidar.h.

39.9.3.3 void trigger_interrupt_guard_condition (int *signal_value*) [inline]

Definition at line 94 of file ydlidar.h.

39.9.4 Variable Documentation

39.9.4.1 volatile sig_atomic_t g_signal_status = 0 [static]

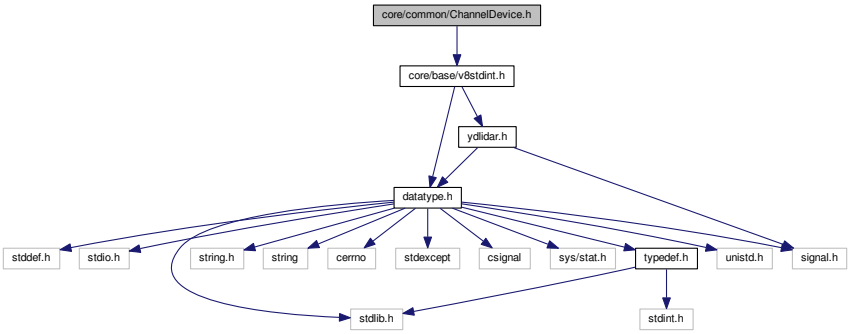
Definition at line 27 of file ydlidar.h.

39.9.4.2 signal_handler_t old_signal_handler = 0 [static]

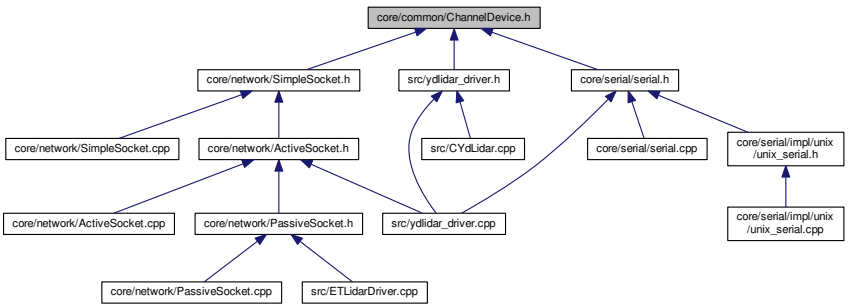
Definition at line 33 of file ydlidar.h.

39.10 core/common/ChannelDevice.h File Reference

`#include <core/base/v8stdint.h>`
Include dependency graph for ChannelDevice.h:



This graph shows which files directly or indirectly include this file:



Classes

- class ydlidar::core::common::ChannelDevice

Namespaces

- ydlidar

  ydlidar
- ydlidar::core

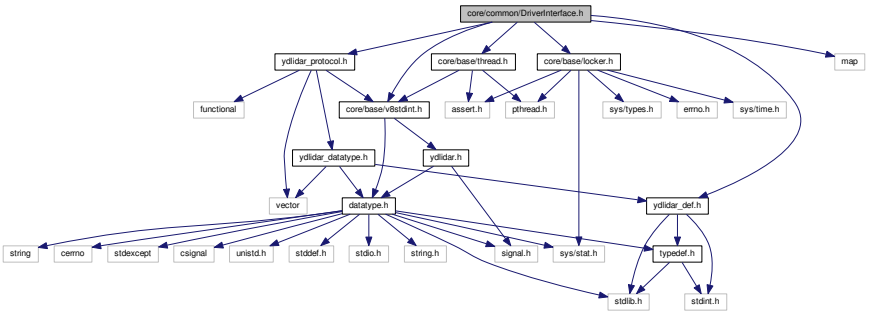
  ydlidar core
- ydlidar::core::common

  ydlidar common

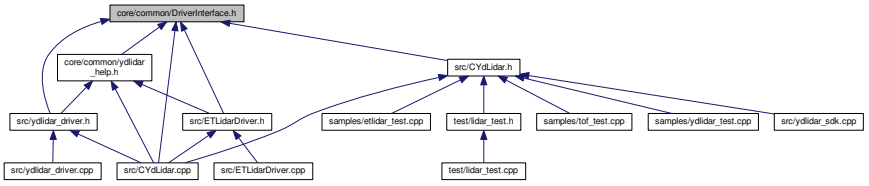
39.11 core/common/DriverInterface.h File Reference

```
#include <core/base/v8stdint.h>
#include <core/base/thread.h>
#include <core/base/locker.h>
#include <map>
#include "ydlidar_protocol.h"
#include "ydlidar_def.h"
```

Include dependency graph for DriverInterface.h:



This graph shows which files directly or indirectly include this file:



Classes

- class ydlidar::core::common::DriverInterface

Namespaces

- ydlidar

  ydlidar
- ydlidar::core

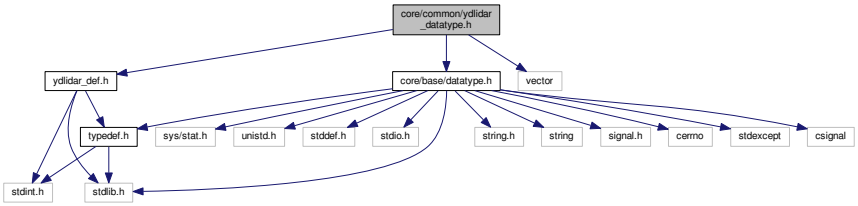
  ydlidar core
- ydlidar::core::common

  ydlidar common

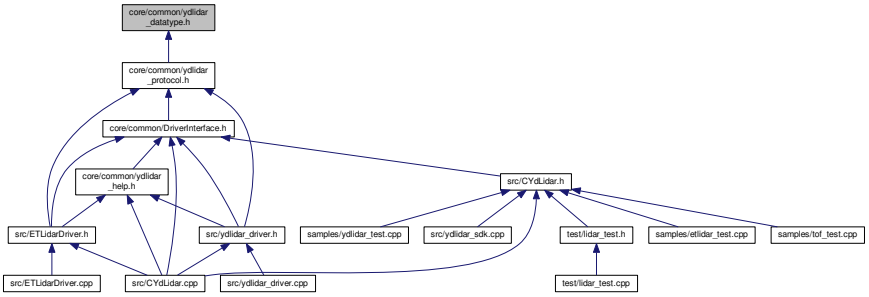
39.12 core/common/ydlidar_datatype.h File Reference

```
#include <core/base/datatype.h>
#include <vector>
#include "ydlidar_def.h"
```

Include dependency graph for ydlidar_datatype.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct LaserDebug

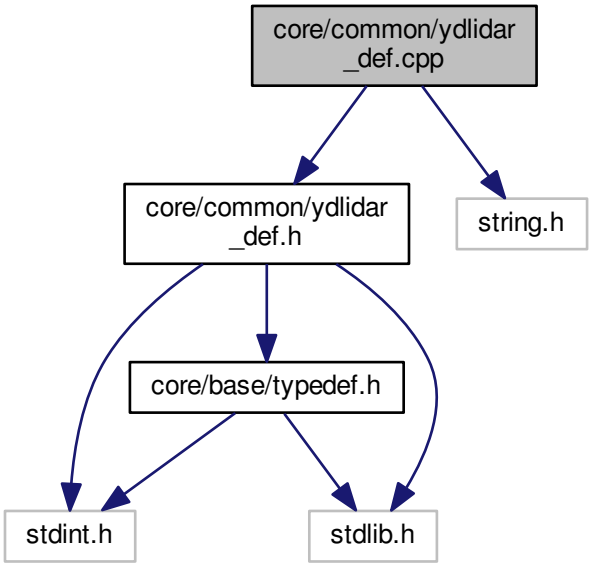
  The Laser Debug struct.
- struct LaserScan

  The Laser Scan Data struct.

39.13 core/common/ydlidar_def.cpp File Reference

```
#include <core/common/ydlidar_def.h>
#include <string.h>
```

Include dependency graph for ydlidar_def.cpp:



Functions

- void LaserFanInit (LaserFan *to_init)

  initialize LaserFan
- void LaserFanDestroy (LaserFan *to_destroy)

39.13.1 Function Documentation

39.13.1.1 void LaserFanDestroy (LaserFan * *to_destroy*)

Destroy an instance of LaserFan points

Definition at line 34 of file ydlidar_def.cpp.

39.13.1.2 void LaserFanInit (LaserFan * *to_init*)

initialize LaserFan

Parameters

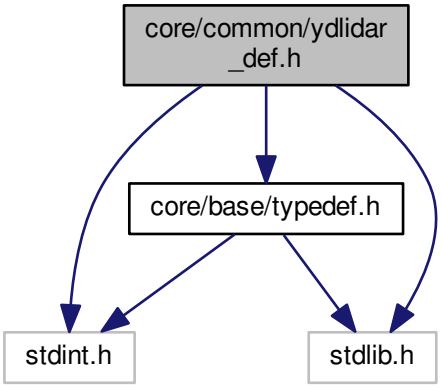
to_init	

Definition at line 28 of file ydlidar_def.cpp.

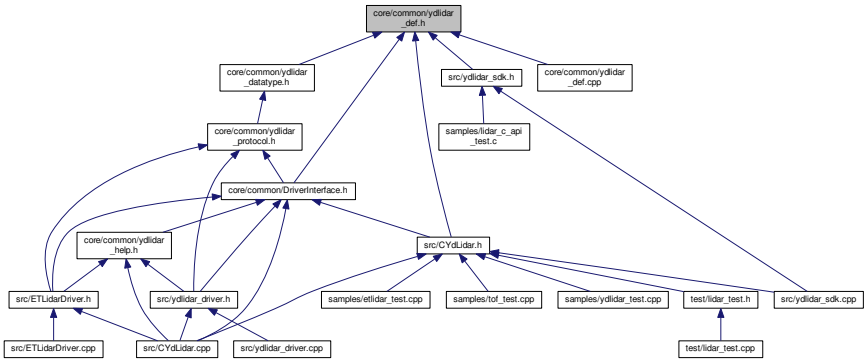
39.14 core/common/ydlidar_def.h File Reference

```
#include <stdint.h>
#include <stdlib.h>
#include <core/base/typedef.h>
```

Include dependency graph for ydlidar_def.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct YDLidar

  lidar instance
- struct LaserPoint

  The Laser Point struct.
- struct LaserConfig

  A struct for returning configuration from the YDLIDAR.
- struct LaserFan

  The Laser Scan Data struct.
- struct string_t

  c string
- struct LidarPort

  lidar ports
- struct LidarVersion

Enumerations

- enum DeviceTypeID { YDLIDAR_TYPE_SERIAL = 0x0, YDLIDAR_TYPE_TCP = 0x1, YDLIDAR_TYPC_UDP = 0x2 }
- enum LidarTypeID { TYPE_TOF = 0, TYPE_TRIANGLE = 1, TYPE_TOF_NET = 2, TYPE_Tail }
- enum LidarProperty {
  LidarPropSerialPort = 0, LidarPropIgnoreArray, LidarPropSerialBaudrate = 10, LidarPropLidarType, LidarPropDeviceType, LidarPropSampleRate, LidarPropAbnormalCheckCount, LidarPropMaxRange = 20, LidarPropMinRange, LidarPropMaxAngle, LidarPropMinAngle, LidarPropScanFrequency, LidarPropFixedResolution = 30, LidarPropReversion, LidarPropInverted, LidarPropAutoReconnect, LidarPropSingleChannel, LidarPropIntenstiy, LidarPropSupportMotorDtrCtrl }

Functions

- void LaserFanInit (LaserFan *to_init)

  initialize LaserFan
- void LaserFanDestroy (LaserFan *to_destroy)

39.14.1 Enumeration Type Documentation

39.14.1.1 enum DeviceTypeID

Device Type ID

Enumerator

YDLIDAR_TYPE_SERIAL serial type.

YDLIDAR_TYPE_TCP socket tcp type.

YDLIDAR_TYPC_UDP socket udp type.

Definition at line 36 of file ydlidar_def.h.

39.14.1.2 enum LidarProperty

Lidar Properties,Lidar Can set and get parameter property index.
float properties must be float type, not double type.

Enumerator

LidarPropSerialPort Lidar serial port or network ipaddress
LidarPropIgnoreArray Lidar ignore angle array
LidarPropSerialBaudrate lidar serial baudrate or network port
LidarPropLidarType lidar type code
LidarPropDeviceType lidar connection type code
LidarPropSampleRate lidar sample rate
LidarPropAbnormalCheckCount abnormal maximum check times
LidarPropMaxRange lidar maximum range
LidarPropMinRange lidar minimum range
LidarPropMaxAngle lidar maximum angle
LidarPropMinAngle lidar minimum angle
LidarPropScanFrequency lidar scanning frequency
LidarPropFixedResolution fixed angle resolution flag
LidarPropReversion lidar reversion flag
LidarPropInverted lidar inverted flag
LidarPropAutoReconnect lidar hot plug flag
LidarPropSingleChannel lidar single-channel flag
LidarPropIntenstiy lidar intensity flag
LidarPropSupportMotorDtrCtrl lidar support motor Dtr ctrl flag

Definition at line 54 of file ydlidar_def.h.

39.14.1.3 enum LidarTypeID

Lidar Type ID

Enumerator

TYPE_TOF TG TX LiDAR.
TYPE_TRIANGLE G4. G6. G2 LiDAR.
TYPE_TOF_NET T15 LiDAR.
TYPE_Tail

Definition at line 43 of file ydlidar_def.h.

39.14.2 Function Documentation

39.14.2.1 void LaserFanDestroy (LaserFan * *to_destroy*)

Destroy an instance of LaserFan points

Definition at line 34 of file ydlidar_def.cpp.

39.14.2.2 void LaserFanInit (LaserFan * *to_init*)

initialize LaserFan

Parameters

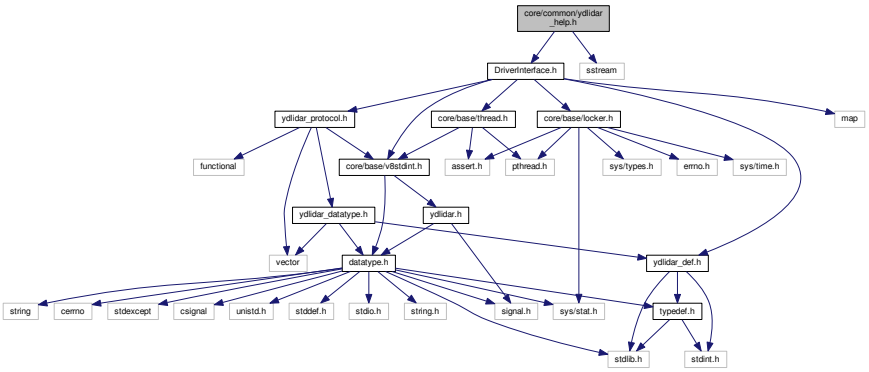
to_init	

Definition at line 28 of file ydlidar_def.cpp.

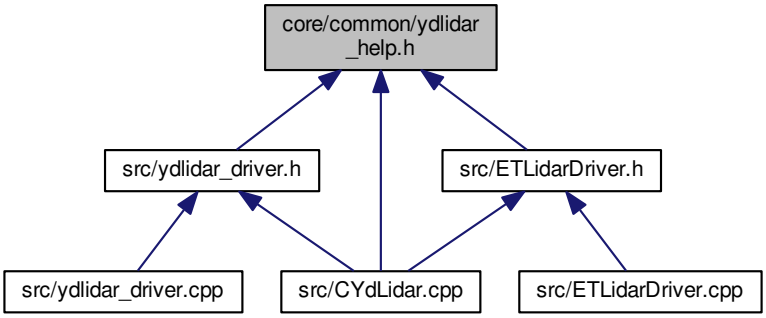
39.15 core/common/ydlidar_help.h File Reference

```
#include "DriverInterface.h"
#include <sstream>
```

Include dependency graph for ydlidar_help.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- ydlidar
  ydlidar
- ydlidar::core
  ydlidar core
- ydlidar::core::common
  ydlidar common

Functions

- std::string ydlidar::core::common::lidarModelToString (int model)

  convert lidar model to string
- int ydlidar::core::common::lidarModelDefaultSampleRate (int model)

  Get LiDAR default sampling rate.
- bool ydlidar::core::common::isOctaveLidar (int model)

  Query whether the LiDAR is Octave LiDAR.
- bool ydlidar::core::common::hasSampleRate (int model)

  Supports multiple sampling rate.
- bool ydlidar::core::common::hasZeroAngle (int model)

  Is there a zero offset angle.
- bool ydlidar::core::common::hasScanFrequencyCtrl (int model)

  Whether to support adjusting the scanning frequency .
- bool ydlidar::core::common::isSupportLidar (int model)

  Does SDK support the LiDAR model.
- bool ydlidar::core::common::hasIntensity (int model)

  Whether to support intensity.
- bool ydlidar::core::common::isSupportMotorCtrl (int model)

  Whether to support serial DTR enable motor.
- bool ydlidar::core::common::isSupportScanFrequency (int model, double frequency)

  Whether the scanning frequency is supported.
- bool ydlidar::core::common::isTOFLidarByModel (int model)

  Whether it is a TOF Model LiDAR.
- bool ydlidar::core::common::isNetTOFLidarByModel (int model)

  Whether it is a Net TOF Model LiDAR.
- bool ydlidar::core::common::isTOFLidar (int type)

  Whether it is a TOF type LiDAR.
- bool ydlidar::core::common::isNetTOFLidar (int type)

  Whether it is a network hardware interface TOF type LiDAR.
- bool ydlidar::core::common::isTriangleLidar (int type)

  Whether it is a Triangle type LiDAR.
- bool ydlidar::core::common::isOldVersionTOFLidar (int model, int Major, int Minor)

  Whether it is Old Version protocol TOF LiDAR.
- bool ydlidar::core::common::isValidSampleRate (std::map< int, int > smap)

  Whether the sampling rate is valid.
- int ydlidar::core::common::ConvertUserToLidarSmaple (int model, int m_SampleRate, int defaultRate)

  convert User sampling rate code to LiDAR sampling code
- int ydlidar::core::common::ConvertLidarToUserSmaple (int model, int rate)

  convert LiDAR sampling rate code to User sampling code
- bool ydlidar::core::common::isValidValue (uint8_t value)

  Whether the Value is valid.
- bool ydlidar::core::common::isVersionValid (const LaserDebug &info)

  Whether the Version is valid.
- bool ydlidar::core::common::isSerialNumbValid (const LaserDebug &info)

  Whether the serial number is valid.
- void ydlidar::core::common::parsePackageNode (const node_info &node, LaserDebug &info)

  convert node_info to LaserDebug
- bool ydlidar::core::common::ParseLaserDebugInfo (const LaserDebug &info, device_info &value)

  convert LaserDebug information to device_info
- bool ydlidar::core::common::printfVersionInfo (const device_info &info, const std::string &port, int baudrate)

print LiDAR version information

- std::vector< float > ydlidar::core::common::split (const std::string &s, char delim)

  split string to vector by delim format

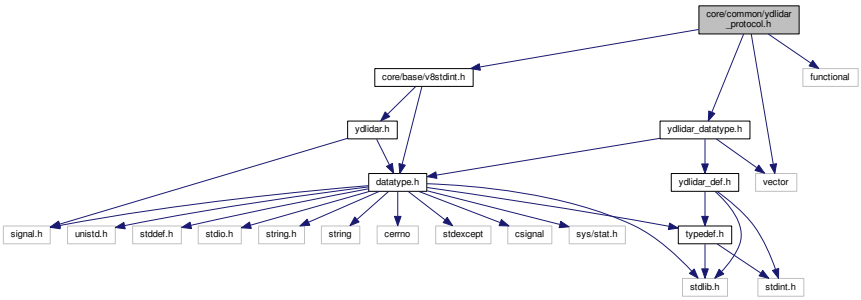
- bool ydlidar::core::common::isV1Protocol (uint8_t protocol)

  Whether the ET LiDAR Protocol type is V1.

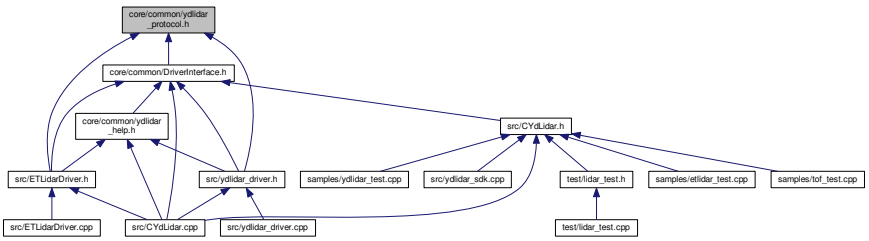
39.16 core/common/ydlidar_protocol.h File Reference

```
#include <core/base/v8stdint.h>
#include <vector>
#include <functional>
#include "ydlidar_datatype.h"
```

Include dependency graph for ydlidar_protocol.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct node_info

  LiDAR Node info.

- struct PackageNode

  package node info

- struct node_package

  LiDAR Intensity Nodes Package.

- struct node_packages

  LiDAR Normal Nodes package.

- struct device_info

LiDAR Device Information.

- struct device_health

  LiDAR Health Information.
- struct sampling_rate

  LiDAR sampling Rate struct.
- struct scan_frequency

  LiDAR scan frequency struct.
- struct scan_rotation
- struct scan_exposure

  LiDAR Exposure struct.
- struct scan_heart_beat

  LiDAR Heart beat struct.
- struct scan_points
- struct function_state
- struct offset_angle

  LiDAR Zero Offset Angle.
- struct cmd_packet

  LiDAR request command packet.
- struct lidar_ans_header

  LiDAR response Header.
- struct _dataFrame

  UDP Data format.
- struct _lidarConfig

Macros

- #define _countof(_Array) (int)(sizeof(_Array) / sizeof(_Array[0]))

  Count the number of elements in a statically allocated array.
- #define PackageSampleMaxLngth 0x100

  Maximuum number of samples in a packet.
- #define Node_Default_Quality (10)

  Default Node Quality.
- #define Node_Sync 1

  Starting Node.
- #define Node_NotSync 2

  Normal Node.
- #define PackagePaidBytes 10

  Package Header Size.
- #define PH 0x55AA

  Package Header.

PI constant

- #define M_PI 3.1415926

sun noise flag constant

- #define SUNNOISEINTENSITY 0xff

glass noise flag constant

- #define GLASSNOISEINTENSITY 0xfe

LIDAR CMD Protocol

LiDAR request and response CMD

- #define LIDAR_CMD_STOP 0x65
- #define LIDAR_CMD_SCAN 0x60
- #define LIDAR_CMD_FORCE_SCAN 0x61
- #define LIDAR_CMD_RESET 0x80
- #define LIDAR_CMD_FORCE_STOP 0x00
- #define LIDAR_CMD_GET_EAI 0x55
- #define LIDAR_CMD_GET_DEVICE_INFO 0x90
- #define LIDAR_CMD_GET_DEVICE_HEALTH 0x92
- #define LIDAR_ANS_TYPE_DEVINFO 0x4
- #define LIDAR_ANS_TYPE_DEVHEALTH 0x6
- #define LIDAR_CMD_SYNC_BYTE 0xA5
- #define LIDAR_CMDFLAG_HAS_PAYLOAD 0x80
- #define LIDAR_ANS_SYNC_BYTE1 0xA5
- #define LIDAR_ANS_SYNC_BYTE2 0x5A
- #define LIDAR_ANS_TYPE_MEASUREMENT 0x81
- #define LIDAR_RESP_MEASUREMENT_SYNCBIT (0x1<<0)
- #define LIDAR_RESP_MEASUREMENT_QUALITY_SHIFT 2
- #define LIDAR_RESP_MEASUREMENT_CHECKBIT (0x1<<0)
- #define LIDAR_RESP_MEASUREMENT_ANGLE_SHIFT 1
- #define LIDAR_RESP_MEASUREMENT_DISTANCE_SHIFT 2
- #define LIDAR_RESP_MEASUREMENT_ANGLE_SAMPLE_SHIFT 8
- #define LIDAR_CMD_RUN_POSITIVE 0x06
- #define LIDAR_CMD_RUN_INVERSION 0x07
- #define LIDAR_CMD_SET_AIMSPEED_ADDMIC 0x09
- #define LIDAR_CMD_SET_AIMSPEED_DISMIC 0x0A
- #define LIDAR_CMD_SET_AIMSPEED_ADD 0x0B
- #define LIDAR_CMD_SET_AIMSPEED_DIS 0x0C
- #define LIDAR_CMD_GET_AIMSPEED 0x0D
- #define LIDAR_CMD_SET_SAMPLING_RATE 0xD0
- #define LIDAR_CMD_GET_SAMPLING_RATE 0xD1
- #define LIDAR_STATUS_OK 0x0
- #define LIDAR_STATUS_WARNING 0x1
- #define LIDAR_STATUS_ERROR 0x2
- #define LIDAR_CMD_ENABLE_LOW_POWER 0x01
- #define LIDAR_CMD_DISABLE_LOW_POWER 0x02
- #define LIDAR_CMD_STATE_MODEL_MOTOR 0x05
- #define LIDAR_CMD_ENABLE_CONST_FREQ 0x0E
- #define LIDAR_CMD_DISABLE_CONST_FREQ 0x0F
- #define LIDAR_CMD_GET_OFFSET_ANGLE 0x93
- #define LIDAR_CMD_SAVE_SET_EXPOSURE 0x94
- #define LIDAR_CMD_SET_LOW_EXPOSURE 0x95
- #define LIDAR_CMD_ADD_EXPOSURE 0x96
- #define LIDAR_CMD_DIS_EXPOSURE 0x97

Typedefs

- typedef struct _dataFrame dataFrame

  UDP Data format.
- typedef struct _lidarConfig lidarConfig

Enumerations

- enum CT { CT_Normal = 0, CT_RingStart = 1, CT_Tail }

  CT Package Type.
- enum ProtocolVer { Protocol_V1 = 0, Protocol_V2 = 1 }

  ET LiDAR Protocol Type.

Functions

- struct node_info __attribute__ ((packed))

Variables

- uint8_t sync_flag
  sync flag
- uint16_t sync_quality
  intensity
- uint16_t angle_q6_checkbit
  angle
- uint16_t distance_q2
  range
- uint64_t stamp
  time stamp
- uint8_t scan_frequence
  scan frequency. invalid: 0
- uint8_t debugInfo
  debug information
- uint8_t index
  package index
- uint8_t error_package
  error package state
- uint8_t PakageSampleQuality
  intensity
- uint16_t PakageSampleDistance
  range
- uint16_t package_Head
  package header
- uint8_t package_CT
  package ct
- uint8_t nowPackageNum
  package number
- uint16_t packageFirstSampleAngle
  first sample angle
- uint16_t packageLastSampleAngle
  last sample angle
- uint16_t checkSum
  checksum
- PackageNode packageSample [0x100]
- uint16_t packageSampleDistance [0x100]
- uint8_t model
  LiDAR model.
- uint16_t firmware_version
  firmware version
- uint8_t hardware_version
  hardare version
- uint8_t serialnum [16]
  serial number

- uint8_t status
  health state
- uint16_t error_code
  error code
- uint8_t rate
  sample rate
- uint32_t frequency
  scan frequency
- uint8_t rotation
- uint8_t exposure
  low exposure
- uint8_t enable
  heart beat
- uint8_t flag
- uint8_t state
- int32_t angle
- uint8_t syncByte
- uint8_t cmd_flag
- uint8_t size
- uint8_t data
- uint8_t syncByte1
- uint8_t syncByte2
- uint32_t subType
- uint8_t type

39.16.1 Macro Definition Documentation

39.16.1.1 #define _countof(_Array) (int)(sizeof(_Array) / sizeof(_Array[0]))

Count the number of elements in a statically allocated array.

Definition at line 41 of file ydlidar_protocol.h.

39.16.1.2 #define GLASSNOISEINTENSITY 0xfe

Definition at line 64 of file ydlidar_protocol.h.

39.16.1.3 #define LIDAR_ANS_SYNC_BYTE1 0xA5

Definition at line 83 of file ydlidar_protocol.h.

39.16.1.4 #define LIDAR_ANS_SYNC_BYTE2 0x5A

Definition at line 84 of file ydlidar_protocol.h.

39.16.1.5 #define LIDAR_ANS_TYPE_DEVHEALTH 0x6

Definition at line 80 of file ydlidar_protocol.h.

39.16.1.6 #define LIDAR_ANS_TYPE_DEVINFO 0x4

Definition at line 79 of file ydlidar_protocol.h.

39.16.1.7 #define LIDAR_ANS_TYPE_MEASUREMENT 0x81

Definition at line 85 of file ydlidar_protocol.h.

39.16.1.8 #define LIDAR_CMD_ADD_EXPOSURE 0x96

Definition at line 116 of file ydlidar_protocol.h.

39.16.1.9 #define LIDAR_CMD_DIS_EXPOSURE 0x97

Definition at line 117 of file ydlidar_protocol.h.

39.16.1.10 #define LIDAR_CMD_DISABLE_CONST_FREQ 0x0F

Definition at line 111 of file ydlidar_protocol.h.

39.16.1.11 #define LIDAR_CMD_DISABLE_LOW_POWER 0x02

Definition at line 108 of file ydlidar_protocol.h.

39.16.1.12 #define LIDAR_CMD_ENABLE_CONST_FREQ 0x0E

Definition at line 110 of file ydlidar_protocol.h.

39.16.1.13 #define LIDAR_CMD_ENABLE_LOW_POWER 0x01

Definition at line 107 of file ydlidar_protocol.h.

39.16.1.14 #define LIDAR_CMD_FORCE_SCAN 0x61

Definition at line 73 of file ydlidar_protocol.h.

39.16.1.15 #define LIDAR_CMD_FORCE_STOP 0x00

Definition at line 75 of file ydlidar_protocol.h.

39.16.1.16 #define LIDAR_CMD_GET_AIMSPEED 0x0D

Definition at line 99 of file ydlidar_protocol.h.

39.16.1.17 #define LIDAR_CMD_GET_DEVICE_HEALTH 0x92

Definition at line 78 of file ydlidar_protocol.h.

39.16.1.18 #define LIDAR_CMD_GET_DEVICE_INFO 0x90

Definition at line 77 of file ydlidar_protocol.h.

39.16.1.19 #define LIDAR_CMD_GET_EAI 0x55

Definition at line 76 of file ydlidar_protocol.h.

39.16.1.20 #define LIDAR_CMD_GET_OFFSET_ANGLE 0x93

Definition at line 113 of file ydlidar_protocol.h.

39.16.1.21 #define LIDAR_CMD_GET_SAMPLING_RATE 0xD1

Definition at line 102 of file ydlidar_protocol.h.

39.16.1.22 #define LIDAR_CMD_RESET 0x80

Definition at line 74 of file ydlidar_protocol.h.

39.16.1.23 #define LIDAR_CMD_RUN_INVERSION 0x07

Definition at line 94 of file ydlidar_protocol.h.

39.16.1.24 #define LIDAR_CMD_RUN_POSITIVE 0x06

Definition at line 93 of file ydlidar_protocol.h.

39.16.1.25 #define LIDAR_CMD_SAVE_SET_EXPOSURE 0x94

Definition at line 114 of file ydlidar_protocol.h.

39.16.1.26 #define LIDAR_CMD_SCAN 0x60

Definition at line 72 of file ydlidar_protocol.h.

39.16.1.27 #define LIDAR_CMD_SET_AIMSPEED_ADD 0x0B

Definition at line 97 of file ydlidar_protocol.h.

39.16.1.28 #define LIDAR_CMD_SET_AIMSPEED_ADDMIC 0x09

Definition at line 95 of file ydlidar_protocol.h.

39.16.1.29 #define LIDAR_CMD_SET_AIMSPEED_DIS 0x0C

Definition at line 98 of file ydlidar_protocol.h.

39.16.1.30 #define LIDAR_CMD_SET_AIMSPEED_DISMIC 0x0A

Definition at line 96 of file ydlidar_protocol.h.

39.16.1.31 #define LIDAR_CMD_SET_LOW_EXPOSURE 0x95

Definition at line 115 of file ydlidar_protocol.h.

39.16.1.32 #define LIDAR_CMD_SET_SAMPLING_RATE 0xD0

Definition at line 101 of file ydlidar_protocol.h.

39.16.1.33 #define LIDAR_CMD_STATE_MODEL_MOTOR 0x05

Definition at line 109 of file ydlidar_protocol.h.

39.16.1.34 #define LIDAR_CMD_STOP 0x65

Definition at line 71 of file ydlidar_protocol.h.

39.16.1.35 #define LIDAR_CMD_SYNC_BYTE 0xA5

Definition at line 81 of file ydlidar_protocol.h.

39.16.1.36 #define LIDAR_CMDFLAG_HAS_PAYLOAD 0x80

Definition at line 82 of file ydlidar_protocol.h.

39.16.1.37 #define LIDAR_RESP_MEASUREMENT_ANGLE_SAMPLE_SHIFT 8

Definition at line 91 of file ydlidar_protocol.h.

39.16.1.38 #define LIDAR_RESP_MEASUREMENT_ANGLE_SHIFT 1

Definition at line 89 of file ydlidar_protocol.h.

39.16.1.39 #define LIDAR_RESP_MEASUREMENT_CHECKBIT (0x1<<0)

Definition at line 88 of file ydlidar_protocol.h.

39.16.1.40 #define LIDAR_RESP_MEASUREMENT_DISTANCE_SHIFT 2

Definition at line 90 of file ydlidar_protocol.h.

39.16.1.41 #define LIDAR_RESP_MEASUREMENT_QUALITY_SHIFT 2

Definition at line 87 of file ydlidar_protocol.h.

39.16.1.42 #define LIDAR_RESP_MEASUREMENT_SYNCBIT (0x1<<0)

Definition at line 86 of file ydlidar_protocol.h.

39.16.1.43 #define LIDAR_STATUS_ERROR 0x2

Definition at line 105 of file ydlidar_protocol.h.

39.16.1.44 #define LIDAR_STATUS_OK 0x0

Definition at line 103 of file ydlidar_protocol.h.

39.16.1.45 #define LIDAR_STATUS_WARNING 0x1

Definition at line 104 of file ydlidar_protocol.h.

39.16.1.46 #define M_PI 3.1415926

Definition at line 49 of file ydlidar_protocol.h.

39.16.1.47 #define Node_Default_Quality (10)

Default Node Quality.

Definition at line 132 of file ydlidar_protocol.h.

39.16.1.48 #define Node_NotSync 2

Normal Node.

Definition at line 136 of file ydlidar_protocol.h.

39.16.1.49 #define Node_Sync 1

Starting Node.

Definition at line 134 of file ydlidar_protocol.h.

39.16.1.50 #define PackagePaidBytes 10

Package Header Size.

Definition at line 138 of file ydlidar_protocol.h.

39.16.1.51 #define PackageSampleMaxLngth 0x100

Maximuum number of samples in a packet.

Definition at line 122 of file ydlidar_protocol.h.

39.16.1.52 #define PH 0x55AA

Package Header.

Definition at line 140 of file ydlidar_protocol.h.

39.16.1.53 #define SUNNOISEINTENSITY 0xff

Definition at line 57 of file ydlidar_protocol.h.

39.16.2 Typedef Documentation

39.16.2.1 typedef struct _dataFrame dataFrame

UDP Data format.

39.16.2.2 typedef struct _lidarConfig lidarConfig

39.16.3 Enumeration Type Documentation

39.16.3.1 enum CT

CT Package Type.

Enumerator

CT_Normal Normal package.
CT_RingStart Starting package.
CT_Tail

Definition at line 125 of file ydlidar_protocol.h.

39.16.3.2 enum ProtocolVer

ET LiDAR Protocol Type.

Enumerator

Protocol_V1 V1 version.
Protocol_V2 V2 version.

Definition at line 143 of file ydlidar_protocol.h.

39.16.4 Function Documentation

39.16.4.1 struct lidar_ans_header __attribute__ ((packed))

39.16.5 Variable Documentation

39.16.5.1 int32_t angle

Definition at line 942 of file ydlidar_protocol.h.

39.16.5.2 uint16_t angle_q6_checkbit

angle

Definition at line 944 of file ydlidar_protocol.h.

39.16.5.3 uint16_t checkSum

checksum

Definition at line 947 of file ydlidar_protocol.h.

39.16.5.4 uint8_t cmd_flag

Definition at line 943 of file ydlidar_protocol.h.

39.16.5.5 uint8_t data

Definition at line 945 of file ydlidar_protocol.h.

39.16.5.6 uint8_t debugInfo

debug information

Definition at line 948 of file ydlidar_protocol.h.

39.16.5.7 uint16_t distance_q2

range

Definition at line 945 of file ydlidar_protocol.h.

39.16.5.8 uint8_t enable

heart beat

Definition at line 942 of file ydlidar_protocol.h.

39.16.5.9 uint16_t error_code

error code

Definition at line 943 of file ydlidar_protocol.h.

39.16.5.10 uint8_t error_package

error package state

Definition at line 950 of file ydlidar_protocol.h.

39.16.5.11 uint8_t exposure

low exposure

Definition at line 942 of file ydlidar_protocol.h.

39.16.5.12 uint16_t firmware_version

firmware version

Definition at line 943 of file ydlidar_protocol.h.

39.16.5.13 uint8_t flag

Definition at line 942 of file ydlidar_protocol.h.

39.16.5.14 uint32_t frequency

scan frequency

Definition at line 942 of file ydlidar_protocol.h.

39.16.5.15 uint8_t hardware_version

hardare version

Definition at line 944 of file ydlidar_protocol.h.

39.16.5.16 uint8_t index

package index

Definition at line 949 of file ydlidar_protocol.h.

39.16.5.17 uint8_t model

LiDAR model.

Definition at line 942 of file ydlidar_protocol.h.

39.16.5.18 uint8_t nowPackageNum

package number

Definition at line 944 of file ydlidar_protocol.h.

39.16.5.19 uint8_t package_CT

package ct

Definition at line 943 of file ydlidar_protocol.h.

39.16.5.20 uint16_t package_Head

package header

Definition at line 942 of file ydlidar_protocol.h.

39.16.5.21 uint16_t packageFirstSampleAngle

first sample angle

Definition at line 945 of file ydlidar_protocol.h.

39.16.5.22 uint16_t packageLastSampleAngle

last sample angle

Definition at line 946 of file ydlidar_protocol.h.

39.16.5.23 PackageNode packageSample[0x100]

Definition at line 948 of file ydlidar_protocol.h.

39.16.5.24 uint16_t packageSampleDistance[0x100]

Definition at line 948 of file ydlidar_protocol.h.

39.16.5.25 uint16_t PakageSampleDistance

range

Definition at line 943 of file ydlidar_protocol.h.

39.16.5.26 uint8_t PakageSampleQuality

intensity

Definition at line 942 of file ydlidar_protocol.h.

39.16.5.27 uint8_t rate

sample rate

Definition at line 942 of file ydlidar_protocol.h.

39.16.5.28 uint8_t rotation

Definition at line 942 of file ydlidar_protocol.h.

39.16.5.29 uint8_t scan_frequence

scan frequency. invalid: 0

Definition at line 947 of file ydlidar_protocol.h.

39.16.5.30 uint8_t serialnum[16]

serial number

Definition at line 945 of file ydlidar_protocol.h.

39.16.5.31 uint32_t size

Definition at line 944 of file ydlidar_protocol.h.

39.16.5.32 uint64_t stamp

time stamp

Definition at line 946 of file ydlidar_protocol.h.

39.16.5.33 uint8_t state

Definition at line 942 of file ydlidar_protocol.h.

39.16.5.34 uint8_t status

health state

Definition at line 942 of file ydlidar_protocol.h.

39.16.5.35 uint32_t subType

Definition at line 945 of file ydlidar_protocol.h.

39.16.5.36 uint8_t sync_flag

sync flag

Definition at line 942 of file ydlidar_protocol.h.

39.16.5.37 uint16_t sync_quality

intensity

Definition at line 943 of file ydlidar_protocol.h.

39.16.5.38 uint8_t syncByte

Definition at line 942 of file ydlidar_protocol.h.

39.16.5.39 uint8_t syncByte1

Definition at line 942 of file ydlidar_protocol.h.

39.16.5.40 uint8_t syncByte2

Definition at line 943 of file ydlidar_protocol.h.

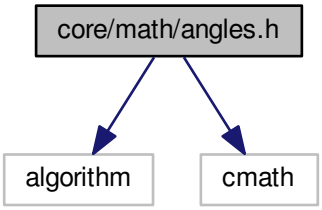
39.16.5.41 uint8_t type

Definition at line 946 of file ydlidar_protocol.h.

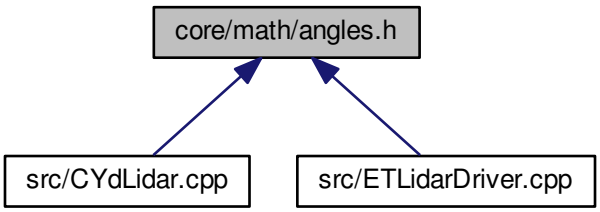
39.17 core/math/angles.h File Reference

```
#include <algorithm>
#include <cmath>
```

Include dependency graph for angles.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- ydlidar

  ydlidar
- ydlidar::core

  ydlidar core
- ydlidar::core::math

Macros

- #define M_PI 3.1415926

Functions

- static double ydlidar::core::math::from_degrees (double degrees)

  Convert degrees to radians.

- static double ydlidar::core::math::to_degrees (double radians)

  Convert radians to degrees.

- static double ydlidar::core::math::normalize_angle_positive (double angle)

  normalize_angle_positive

- static double ydlidar::core::math::normalize_angle_positive_from_degree (double angle)

  normalize_angle_positive_from_degree

- static double ydlidar::core::math::normalize_angle (double angle)

  normalize

- static double ydlidar::core::math::shortest_angular_distance (double from, double to)

  shortest_angular_distance

- static double ydlidar::core::math::two_pi_complement (double angle)

  returns the angle in [-2∗M_PI, 2∗M_PI] going the other way along the unit circle.

- static bool ydlidar::core::math::find_min_max_delta (double from, double left_limit, double right_limit, double &result_min_delta, double &result_max_delta)

  This function is only intended for internal use and not intended for external use. If you do use it, read the documentation very carefully. Returns the min and max amount (in radians) that can be moved from "from" angle to "left_limit" and "right_limit".

- static bool ydlidar::core::math::shortest_angular_distance_with_limits (double from, double to, double left_↩ limit, double right_limit, double &shortest_angle)

  Returns the delta from "from_angle" to "to_angle" making sure it does not violate limits specified by left_limit and right_limit. The valid interval of angular positions is [left_limit,right_limit]. E.g., [-0.25,0.25] is a 0.5 radians wide interval that contains 0. But [0.25,-0.25] is a 2∗M_PI-0.5 wide interval that contains M_PI (but not 0). The value of shortest_angle is the angular difference between "from" and "to" that lies within the defined valid interval. E.↩ g. shortest_angular_distance_with_limits(-0.5,0.5,0.25,-0.25,ss) evaluates ss to 2∗M_PI-1.0 and returns true while shortest_angular_distance_with_limits(-0.5,0.5,-0.25,0.25,ss) returns false since -0.5 and 0.5 do not lie in the interval [-0.25,0.25].

39.17.1 Macro Definition Documentation

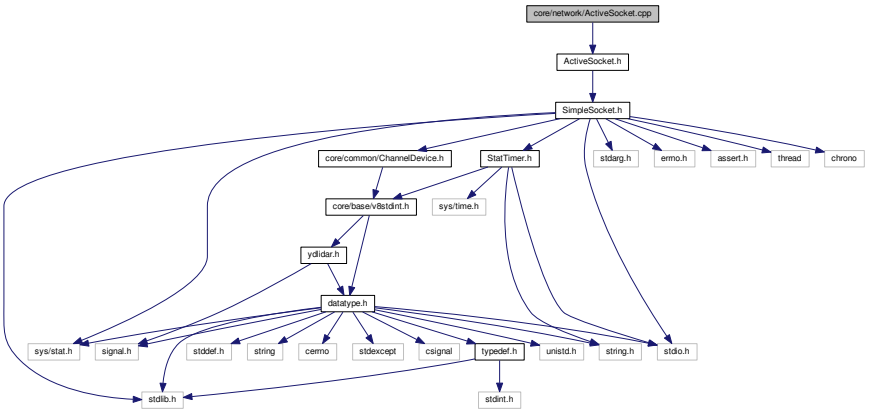
39.17.1.1 #define M_PI 3.1415926

Definition at line 39 of file angles.h.

39.18 core/network/ActiveSocket.cpp File Reference

```
#include "ActiveSocket.h"
```

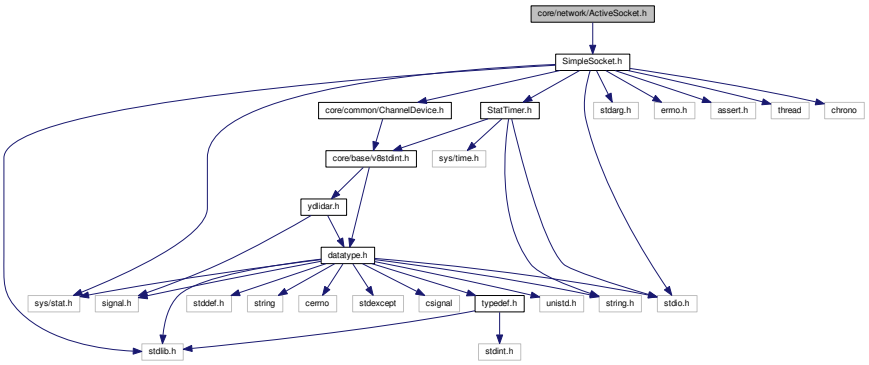
Include dependency graph for ActiveSocket.cpp:



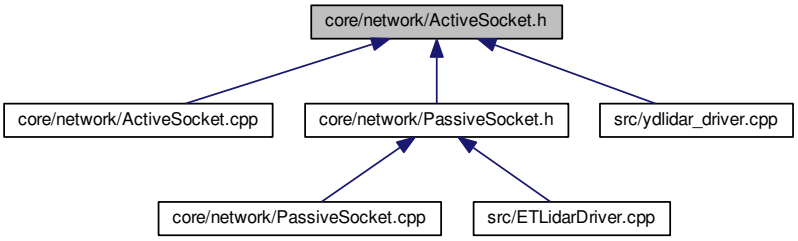
39.19 core/network/ActiveSocket.h File Reference

```
#include "SimpleSocket.h"
```

Include dependency graph for ActiveSocket.h:



This graph shows which files directly or indirectly include this file:



Classes

- class ydlidar::core::network::CActiveSocket

Namespaces

- ydlidar

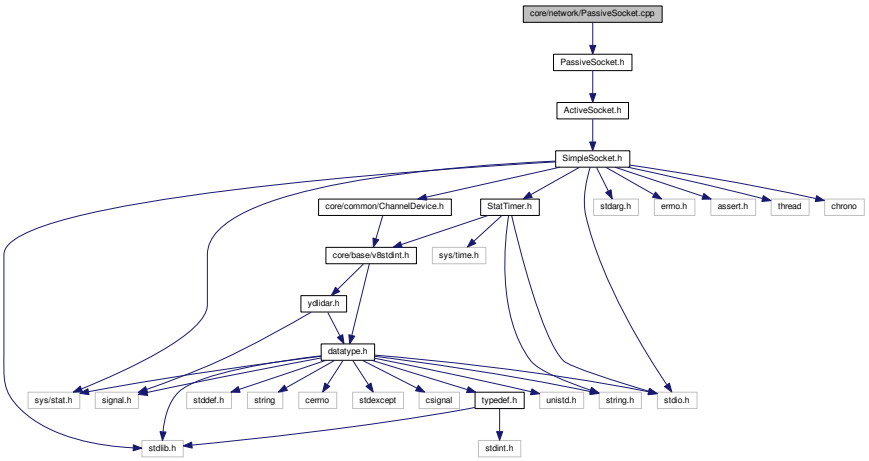
  ydlidar
- ydlidar::core

  ydlidar core
- ydlidar::core::network

39.20 core/network/PassiveSocket.cpp File Reference

```
#include "PassiveSocket.h"
```

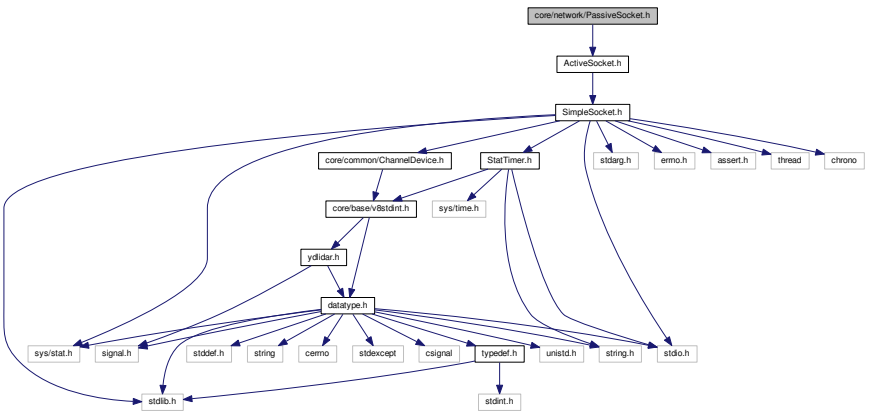
Include dependency graph for PassiveSocket.cpp:



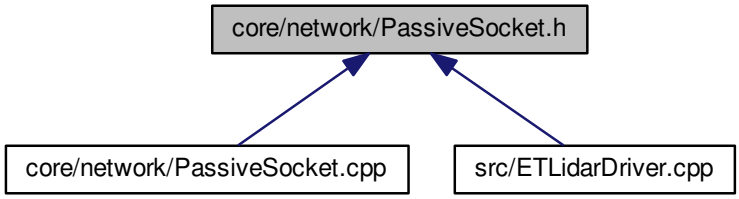
39.21 core/network/PassiveSocket.h File Reference

```
#include "ActiveSocket.h"
```

Include dependency graph for PassiveSocket.h:



This graph shows which files directly or indirectly include this file:



Classes

- class ydlidar::core::network::CPassiveSocket

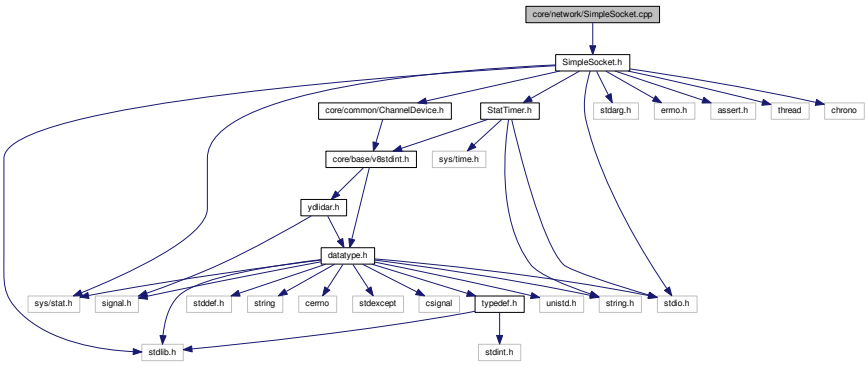
Namespaces

- ydlidar
  ydlidar
- ydlidar::core
  ydlidar core
- ydlidar::core::network

39.22 core/network/SimpleSocket.cpp File Reference

```
#include "SimpleSocket.h"
```

Include dependency graph for SimpleSocket.cpp:



Variables

- static volatile bool m_WSAStartup = false

39.22.1 Variable Documentation

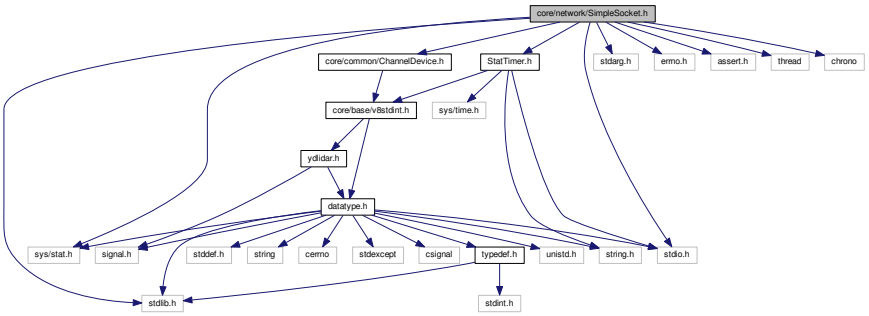
39.22.1.1 volatile bool m_WSAStartup = false [static]

Definition at line 49 of file SimpleSocket.cpp.

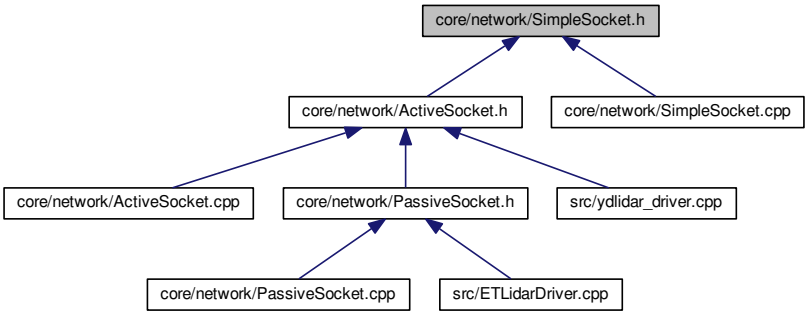
39.23 core/network/SimpleSocket.h File Reference

```
#include <sys/stat.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdarg.h>
#include <errno.h>
#include <assert.h>
#include <thread>
#include <chrono>
#include "StatTimer.h"
#include <core/common/ChannelDevice.h>
```

Include dependency graph for SimpleSocket.h:



This graph shows which files directly or indirectly include this file:



Classes

- class ydlidar::core::network::CSimpleSocket

Namespaces

- ydlidar

  ydlidar
- ydlidar::core

  ydlidar core
- ydlidar::core::network

Macros

- #define INVALID_SOCKET ~(0)
- #define SOCKET_SENDFILE_BLOCKSIZE 8192

39.23.1 Macro Definition Documentation

39.23.1.1 #define INVALID_SOCKET ~(0)

Definition at line 99 of file SimpleSocket.h.

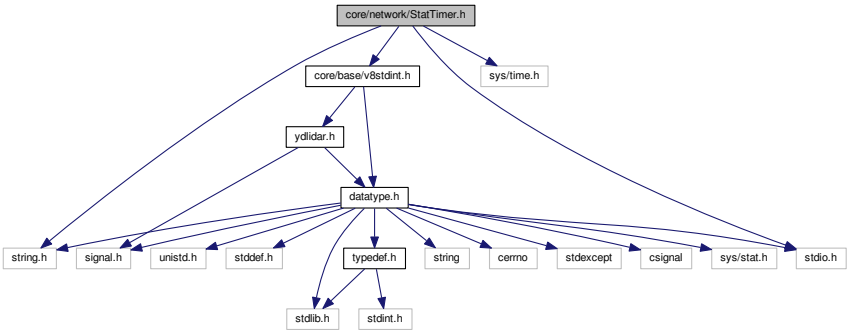
39.23.1.2 #define SOCKET_SENDFILE_BLOCKSIZE 8192

Definition at line 102 of file SimpleSocket.h.

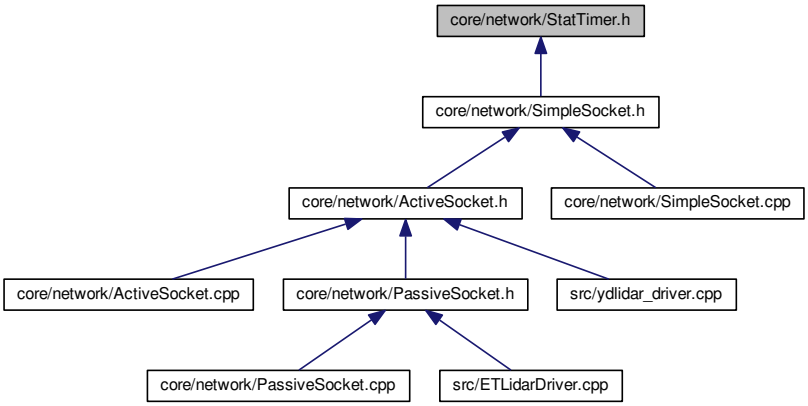
39.24 core/network/StatTimer.h File Reference

```
#include <string.h>
#include <stdio.h>
#include <sys/time.h>
#include <core/base/v8stdint.h>
```

Include dependency graph for StatTimer.h:



This graph shows which files directly or indirectly include this file:



Classes

- class CStatTimer

Macros

- #define GET_CLOCK_COUNT(x) gettimeofday(x, NULL)

39.24.1 Macro Definition Documentation

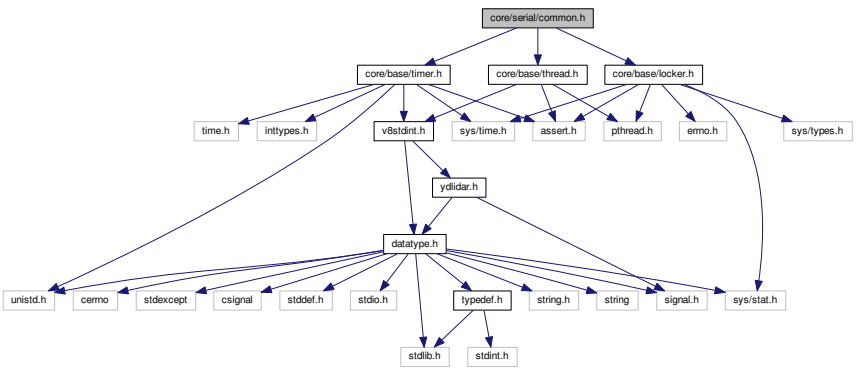
39.24.1.1 #define GET_CLOCK_COUNT(*x*) gettimeofday(x, NULL)

Definition at line 80 of file StatTimer.h.

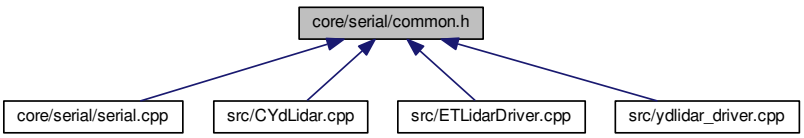
39.25 core/serial/common.h File Reference

```
#include <core/base/thread.h>
#include <core/base/locker.h>
#include <core/base/timer.h>
```

Include dependency graph for common.h:



This graph shows which files directly or indirectly include this file:

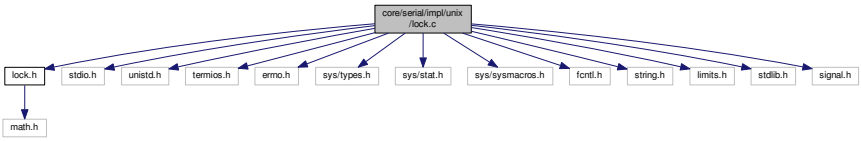


39.26 core/serial/impl/unix/list_ports_linux.cpp File Reference

39.27 core/serial/impl/unix/lock.c File Reference

```
#include "lock.h"
#include <stdio.h>
#include <unistd.h>
#include <termios.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/sysmacros.h>
#include <fcntl.h>
#include <string.h>
#include <limits.h>
#include <stdlib.h>
#include <signal.h>
```

Include dependency graph for lock.c:



Functions

- int fhs_lock (const char *filename, int pid)
- int uucp_lock (const char *filename, int pid)
- int check_lock_status (const char *filename)
- void fhs_unlock (const char *filename, int openpid)
- void uucp_unlock (const char *filename, int openpid)
- int check_lock_pid (const char *file, int openpid)
- int check_group_uucp ()
- int is_device_locked (const char *port_filename)

39.27.1 Function Documentation

39.27.1.1 int check_group_uucp ( )

Definition at line 502 of file lock.c.

39.27.1.2 int check_lock_pid (const char * *file*, int *openpid*)

Definition at line 449 of file lock.c.

39.27.1.3 int check_lock_status (const char * *filename*)

Definition at line 341 of file lock.c.

39.27.1.4 int fhs_lock (const char * *filename*, int *pid*)

Definition at line 199 of file lock.c.

39.27.1.5 void fhs_unlock (const char * *filename*, int *openpid*)

Definition at line 378 of file lock.c.

39.27.1.6 int is_device_locked (const char * *port_filename*)

Definition at line 649 of file lock.c.

39.27.1.7 int uucp_lock (const char * *filename*, int *pid*)

Definition at line 281 of file lock.c.

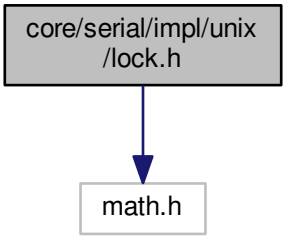
39.27.1.8 void uucp_unlock (const char * *filename*, int *openpid*)

Definition at line 408 of file lock.c.

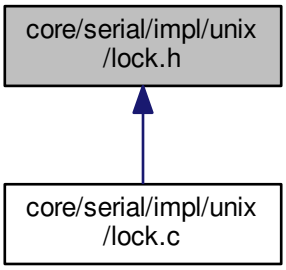
39.28 core/serial/impl/unix/lock.h File Reference

```
#include <math.h>
```

Include dependency graph for lock.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define LOCK system_does_not_lock
- #define UNLOCK system_does_not_unlock

Functions

- int check_group_uucp ()
- int check_lock_pid (const char *file, int openpid)
- int lock_device (const char *)
- void unlock_device (const char *)
- int is_device_locked (const char *)
- int check_lock_status (const char *)

- int lfs_unlock (const char *, int)
- int lfs_lock (const char *, int)
- int lib_lock_dev_unlock (const char *, int)
- int lib_lock_dev_lock (const char *, int)
- void fhs_unlock (const char *, int)
- int fhs_lock (const char *, int)
- void uucp_unlock (const char *, int)
- int uucp_lock (const char *, int)

39.28.1 Macro Definition Documentation

39.28.1.1 #define LOCK system_does_not_lock

Definition at line 206 of file lock.h.

39.28.1.2 #define UNLOCK system_does_not_unlock

Definition at line 207 of file lock.h.

39.28.2 Function Documentation

39.28.2.1 int check_group_uucp ( )

Definition at line 502 of file lock.c.

39.28.2.2 int check_lock_pid (const char * *file*, int *openpid*)

Definition at line 449 of file lock.c.

39.28.2.3 int check_lock_status (const char *)

Definition at line 341 of file lock.c.

39.28.2.4 int fhs_lock (const char *, int)

Definition at line 199 of file lock.c.

39.28.2.5 void fhs_unlock (const char *, int)

Definition at line 378 of file lock.c.

39.28.2.6 int is_device_locked (const char *)

Definition at line 649 of file lock.c.

39.28.2.7 int lfs_lock (const char *, int)

39.28.2.8 int lfs_unlock (const char *, int)

39.28.2.9 int lib_lock_dev_lock (const char *, int)

39.28.2.10 int lib_lock_dev_unlock (const char *, int)

39.28.2.11 int lock_device (const char *)

39.28.2.12 void unlock_device (const char *)

39.28.2.13 int uucp_lock (const char *, int)

Definition at line 281 of file lock.c.

39.28.2.14 void uucp_unlock (const char *, int)

Definition at line 408 of file lock.c.

39.29 core/serial/impl/unix/unix.h File Reference

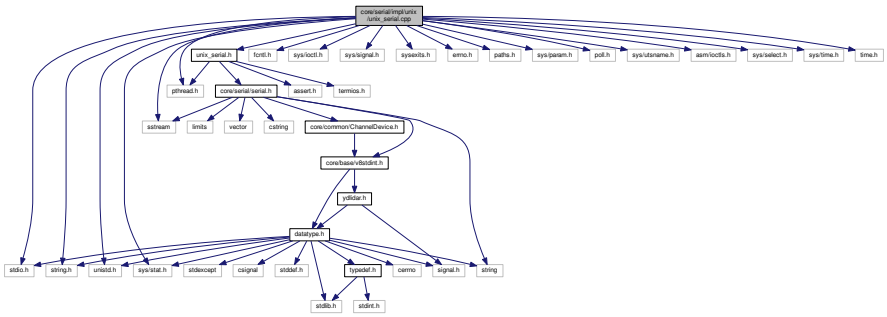
```
#include <stdio.h>
#include <stdint.h>
#include <string.h>
#include <stdlib.h>
#include <assert.h>
#include <math.h>
#include <time.h>
#include <stdarg.h>
#include <iostream>
#include <string>
#include <unistd.h>
#include <errno.h>
#include <pthread.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <sys/select.h>
```

Include dependency graph for unix.h:

39.30 core/serial/impl/unix/unix_serial.cpp File Reference

```
#include <stdio.h>
#include <string.h>
#include <sstream>
#include <unistd.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <sys/signal.h>
#include <sysexits.h>
#include <errno.h>
#include <paths.h>
#include <sys/param.h>
#include <pthread.h>
#include <poll.h>
#include <sys/utsname.h>
#include <asm/ioctls.h>
#include <sys/select.h>
#include <sys/time.h>
#include <sys/stat.h>
#include <time.h>
#include "unix_serial.h"
```

Include dependency graph for unix_serial.cpp:



Classes

- struct ydlidar::core::serial::termios2

Namespaces

- ydlidar

  ydlidar
- ydlidar::core

  ydlidar core
- ydlidar::core::serial

Macros

- #define TIOCINQ 0x541B
- #define SNCCS 19
- #define TCGETS2 _IOR('T', 0x2A, struct termios2)
- #define TCSETS2 _IOW('T', 0x2B, struct termios2)
- #define BOTHER 0010000

Functions

- timespec ydlidar::core::serial::timespec_from_ms (const uint32_t millis)
- static void ydlidar::core::serial::set_common_props (termios *tio)
- static void ydlidar::core::serial::set_databits (termios *tio, serial::bytesize_t databits)
- static void ydlidar::core::serial::set_parity (termios *tio, serial::parity_t parity)
- static void ydlidar::core::serial::set_stopbits (termios *tio, serial::stopbits_t stopbits)
- static void ydlidar::core::serial::set_flowcontrol (termios *tio, serial::flowcontrol_t flowcontrol)
- static bool ydlidar::core::serial::is_standardbaudrate (unsigned long baudrate, speed_t &baud)

39.30.1 Macro Definition Documentation

39.30.1.1 #define BOTHER 0010000

Definition at line 199 of file unix_serial.cpp.

39.30.1.2 #define SNCCS 19

Definition at line 176 of file unix_serial.cpp.

39.30.1.3 #define TCGETS2 _IOR('T', 0x2A, struct termios2)

Definition at line 191 of file unix_serial.cpp.

39.30.1.4 #define TCSETS2 _IOW('T', 0x2B, struct termios2)

Definition at line 195 of file unix_serial.cpp.

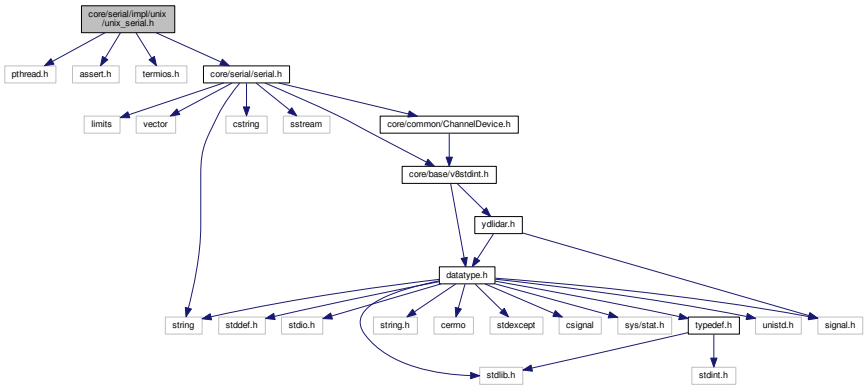
39.30.1.5 #define TIOCINQ 0x541B

Definition at line 49 of file unix_serial.cpp.

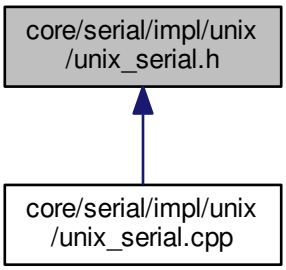
39.31 core/serial/impl/unix/unix_serial.h File Reference

```
#include <pthread.h>
#include <assert.h>
#include <termios.h>
#include <core/serial/serial.h>
```

Include dependency graph for unix_serial.h:



This graph shows which files directly or indirectly include this file:



Classes

- class ydlidar::core::serial::MillisecondTimer
- class serial::Serial::SerialImpl

Namespaces

- ydlidar

  ydlidar
- ydlidar::core

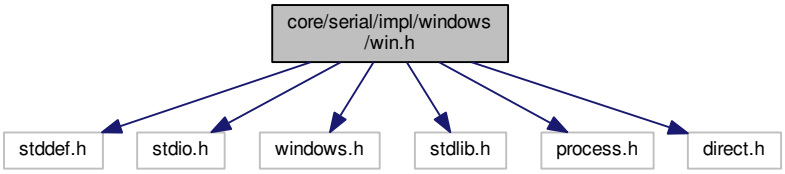
  ydlidar core
- ydlidar::core::serial

39.32 core/serial/impl/windows/list_ports_win.cpp File Reference

39.33 core/serial/impl/windows/win.h File Reference

```
#include <stddef.h>
#include <stdio.h>
#include <windows.h>
#include <stdlib.h>
#include <process.h>
#include <direct.h>
```

Include dependency graph for win.h:



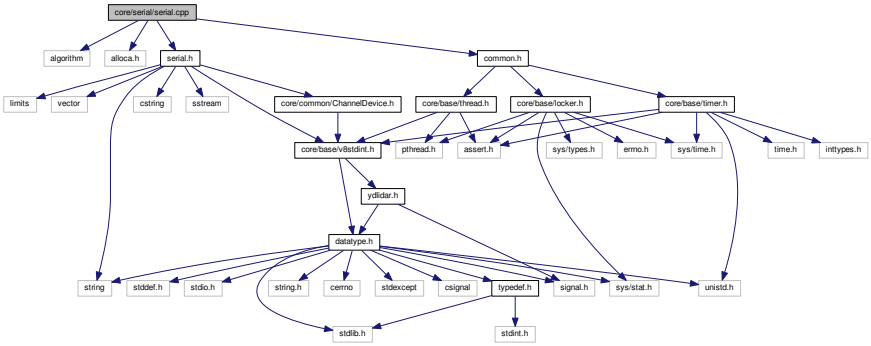
39.34 core/serial/impl/windows/win_serial.cpp File Reference

39.35 core/serial/impl/windows/win_serial.h File Reference

39.36 core/serial/serial.cpp File Reference

```
#include <algorithm>
#include <alloca.h>
#include "serial.h"
#include "common.h"
```

Include dependency graph for serial.cpp:



Classes

- class serial::Serial::ScopedReadLock
- class serial::Serial::ScopedWriteLock

Namespaces

- ydlidar

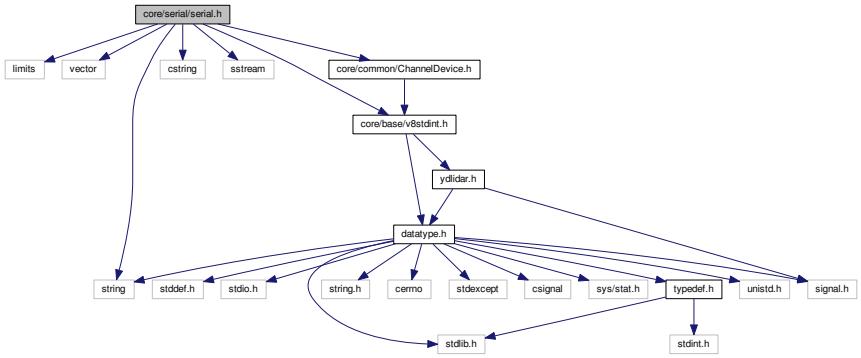
  ydlidar
- ydlidar::core

  ydlidar core
- ydlidar::core::serial

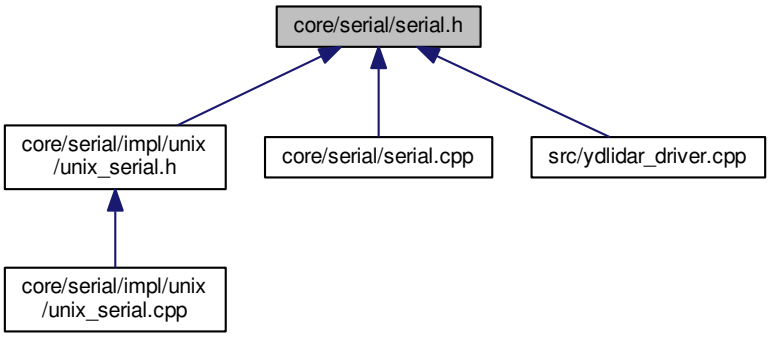
39.37 core/serial/serial.h File Reference

```
#include <limits>
#include <vector>
#include <string>
#include <cstring>
#include <sstream>
#include <core/base/v8stdint.h>
#include <core/common/ChannelDevice.h>
```

Include dependency graph for serial.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct ydlidar::core::serial::Timeout
- class ydlidar::core::serial::Serial
- struct ydlidar::core::serial::PortInfo

Namespaces

- ydlidar

  ydlidar
- ydlidar::core

  ydlidar core
- ydlidar::core::serial

Enumerations

- enum ydlidar::core::serial::bytesize_t { ydlidar::core::serial::fivebits = 5, ydlidar::core::serial::sixbits = 6, ydlidar::core::serial::sevenbits = 7, ydlidar::core::serial::eightbits = 8 }
- enum ydlidar::core::serial::parity_t { ydlidar::core::serial::parity_none = 0, ydlidar::core::serial::parity_odd = 1, ydlidar::core::serial::parity_even = 2, ydlidar::core::serial::parity_mark = 3, ydlidar::core::serial::parity_space = 4 }
- enum ydlidar::core::serial::stopbits_t { ydlidar::core::serial::stopbits_one = 1, ydlidar::core::serial::stopbits_two = 2, ydlidar::core::serial::stopbits_one_point_five }
- enum ydlidar::core::serial::flowcontrol_t { ydlidar::core::serial::flowcontrol_none = 0, ydlidar::core::serial::flowcontrol_software, ydlidar::core::serial::flowcontrol_hardware }

Functions

- std::vector< PortInfo > ydlidar::core::serial::list_ports ()

39.38 doc/Dataset.md File Reference

39.39 doc/Diagram.md File Reference

39.40 doc/FAQs/General_FAQs.md File Reference

39.41 doc/FAQs/General_FAQs_cn.md File Reference

39.42 doc/FAQs/Hardware_FAQs.md File Reference

39.43 doc/FAQs/Hardware_FAQs_cn.md File Reference

39.44 doc/FAQs/README.md File Reference

39.45 doc/howto/README.md File Reference

39.46 doc/quickstart/README.md File Reference

39.47 doc/README.md File Reference

39.48 README.md File Reference

39.49 doc/FAQs/Software_FAQs.md File Reference

39.50 doc/FAQs/Software_FAQs_cn.md File Reference

39.51 doc/howto/how_to_build_and_debug_using_vscode.md File Reference

39.52 doc/howto/how_to_build_and_install.md File Reference

39.53 doc/howto/how_to_create_a_pull.md File Reference

39.54 doc/howto/how_to_create_a_udev_rules.md File Reference

39.55 doc/howto/how_to_gerenrate_vs_project_by_cmake.md File Reference

39.56 doc/howto/how_to_solve_slow_pull_from_cn.md File Reference

39.57 doc/quickstart/ydlidar_sdk_software_installation_guide.md File Reference

39.58 doc/Tutorials.md File Reference

39.59 doc/tutorials/examine_the_simple_lidar_tutorial.md File Reference

39.60 doc/tutorials/writing_lidar_tutorial_c++.md File Reference

39.61 doc/tutorials/writing_lidar_tutorial_c.md File Reference

39.62 doc/tutorials/writing_lidar_tutorial_python.md File Reference

39.63 doc/YDLidar-SDK-Communication-Protocol.md File Reference

39.64 doc/YDLIDAR_SDK_API_for_Developers.md File Reference

39.65 python/examples/etlidar_test.py File Reference

Namespaces

- etlidar_test

Variables

- etlidar_test.laser = ydlidar.CYdLidar();
- etlidar_test.ret = laser.initialize();
- etlidar_test.scan = ydlidar.LaserScan();
- etlidar_test.r = laser.doProcessSimple(scan);

39.66 python/examples/plot_tof_test.py File Reference

Namespaces

- plot_tof_test

Functions

- def plot_tof_test.animate (num)

Variables

- float plot_tof_test.RMAX = 32.0
- plot_tof_test.fig = plt.figure()
- plot_tof_test.lidar_polar = plt.subplot(polar=True)
- plot_tof_test.ports = ydlidar.lidarPortList();
- string plot_tof_test.port = "/dev/ydlidar"
- plot_tof_test.laser = ydlidar.CYdLidar();
- plot_tof_test.scan = ydlidar.LaserScan()
- plot_tof_test.ret = laser.initialize();
- plot_tof_test.ani = animation.FuncAnimation(fig, animate, interval=50)

39.67 python/examples/plot_ydlidar_test.py File Reference

Namespaces

- plot_ydlidar_test

Functions

- def plot_ydlidar_test.animate (num)

Variables

- float plot_ydlidar_test.RMAX = 32.0
- plot_ydlidar_test.fig = plt.figure()
- plot_ydlidar_test.lidar_polar = plt.subplot(polar=True)
- plot_ydlidar_test.ports = ydlidar.lidarPortList();
- string plot_ydlidar_test.port = "/dev/ydlidar"
- plot_ydlidar_test.laser = ydlidar.CYdLidar();
- plot_ydlidar_test.scan = ydlidar.LaserScan()
- plot_ydlidar_test.ret = laser.initialize();
- plot_ydlidar_test.ani = animation.FuncAnimation(fig, animate, interval=50)

39.68 python/examples/test.py File Reference

Namespaces

- test

Variables

- test.ports = ydlidar.lidarPortList();
- string test.port = "/dev/ydlidar"
- test.laser = ydlidar.CYdLidar();
- test.ret = laser.initialize();
- test.scan = ydlidar.LaserScan();
- test.r = laser.doProcessSimple(scan);

39.69 python/examples/tof_test.py File Reference

Namespaces

- tof_test

Variables

- tof_test.ports = ydlidar.lidarPortList();
- string tof_test.port = "/dev/ydlidar"
- tof_test.laser = ydlidar.CYdLidar();
- tof_test.ret = laser.initialize();
- tof_test.scan = ydlidar.LaserScan()
- tof_test.r = laser.doProcessSimple(scan);

39.70 python/examples/ydlidar_test.py File Reference

Namespaces

- ydlidar_test

Variables

- ydlidar_test.ports = ydlidar.lidarPortList();
- string ydlidar_test.port = "/dev/ydlidar"
- ydlidar_test.laser = ydlidar.CYdLidar();
- ydlidar_test.ret = laser.initialize();
- ydlidar_test.scan = ydlidar.LaserScan();
- ydlidar_test.r = laser.doProcessSimple(scan);

39.71 python/test/pytest.py File Reference

Classes

- class pytest.PyTestTestCase

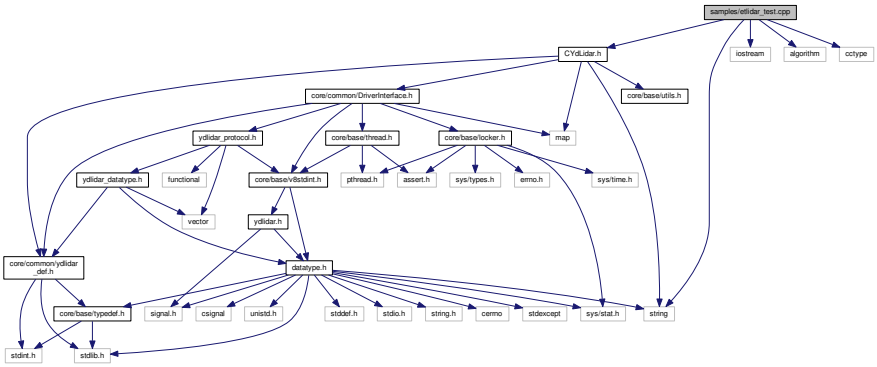
Namespaces

- pytest

39.72 samples/etlidar_test.cpp File Reference

```
#include "CYdLidar.h"
#include <iostream>
#include <string>
#include <algorithm>
#include <cctype>
```

Include dependency graph for etlidar_test.cpp:



Functions

- int main (int argc, char ∗argv[ ])

  etlidar test

39.72.1 Function Documentation

39.72.1.1 int main (int *argc,* char ∗ *argv[ ]*)

etlidar test

Parameters

argc	
argv	

Returns

Flow chart

Step1: instance CYdLidar.
Step2: set paramters.

Step3: initialize SDK and LiDAR.(CYdLidar::initialize)
Step4: Start the device scanning routine which runs on a separate thread and enable motor.(CYdLidar::turnOn)
Step5: Get the LiDAR Scan Data.(CYdLidar::doProcessSimple)
Step6: Stop the device scanning thread and disable motor.(CYdLidar::turnOff)
Step7: Uninitialize the SDK and Disconnect the LiDAR.(CYdLidar::disconnecting)

string property////////////////// lidar port

ignore array

int property////////////////// lidar baudrate

tof lidar

device type

sample rate

abnormal count

bool property////////////////// fixed angle resolution

rotate 180

Counterclockwise

one-way communication

intensity

Motor DTR

float property////////////////// unit: °

unit: m

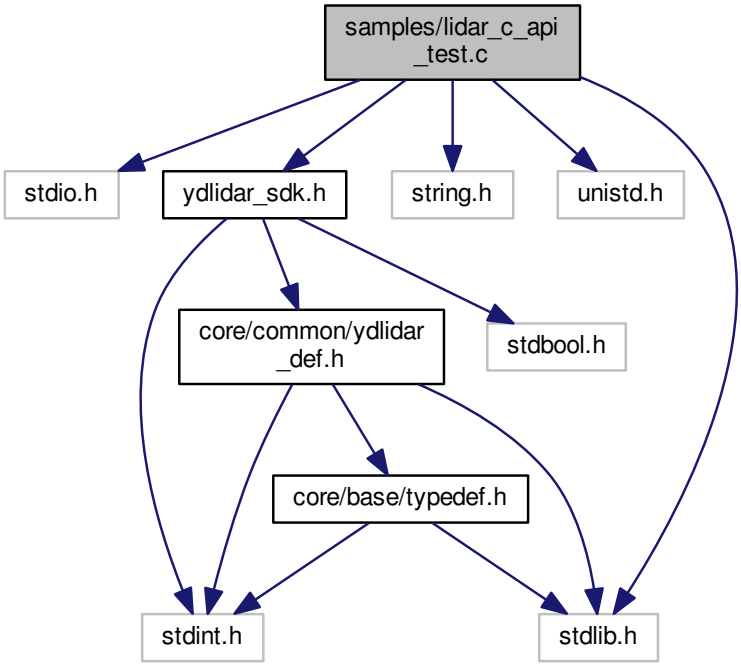
unit: Hz

Definition at line 60 of file etlidar_test.cpp.

39.73 samples/lidar_c_api_test.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include "ydlidar_sdk.h"
```

Include dependency graph for lidar_c_api_test.c:



Functions

- int main (int argc, const char *argv[])

39.73.1 Function Documentation

39.73.1.1 int main (int *argc*, const char * *argv[]*)

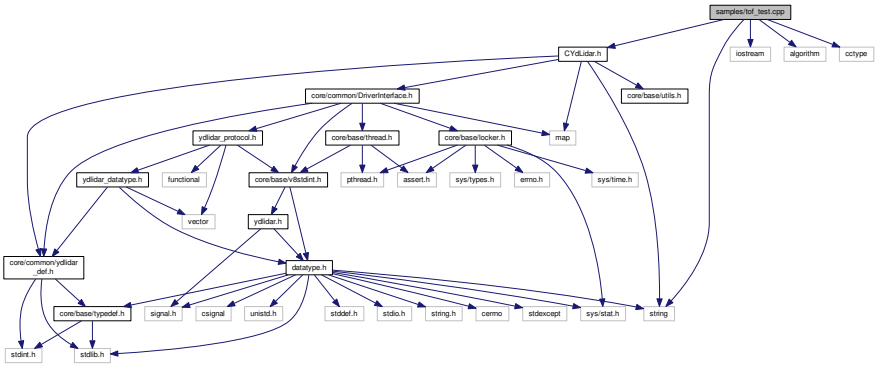
last port

Definition at line 39 of file lidar_c_api_test.c.

39.74 samples/tof_test.cpp File Reference

```
#include "CYdLidar.h"
#include <iostream>
#include <string>
#include <algorithm>
#include <cctype>
```

Include dependency graph for tof_test.cpp:



Functions

- int main (int argc, char ∗argv[ ])

  tof Lidar test

39.74.1 Function Documentation

39.74.1.1 int main (int *argc*, char ∗ *argv[ ]*)

tof Lidar test

Parameters

argc	
argv	

Returns

Flow chart

Step1: instance CYdLidar.
Step2: set paramters.
Step3: initialize SDK and LiDAR.(CYdLidar::initialize)
Step4: Start the device scanning routine which runs on a separate thread and enable motor.(CYdLidar::turnOn)
Step5: Get the LiDAR Scan Data.(CYdLidar::doProcessSimple)
Step6: Stop the device scanning thread and disable motor.(CYdLidar::turnOff)

Step7: Uninitialize the SDK and Disconnect the LiDAR.(CYdLidar::disconnecting)

instance

string property////////////////// lidar port

ignore array

int property////////////////// lidar baudrate

tof lidar

device type

sample rate

abnormal count

bool property////////////////// fixed angle resolution

rotate 180

Counterclockwise

one-way communication

intensity

Motor DTR

float property////////////////// unit: °

unit: m

unit: Hz

initialize SDK and LiDAR.

Start the device scanning routine which runs on a separate thread and enable motor.

Turn On success and loop

Stop the device scanning thread and disable motor.

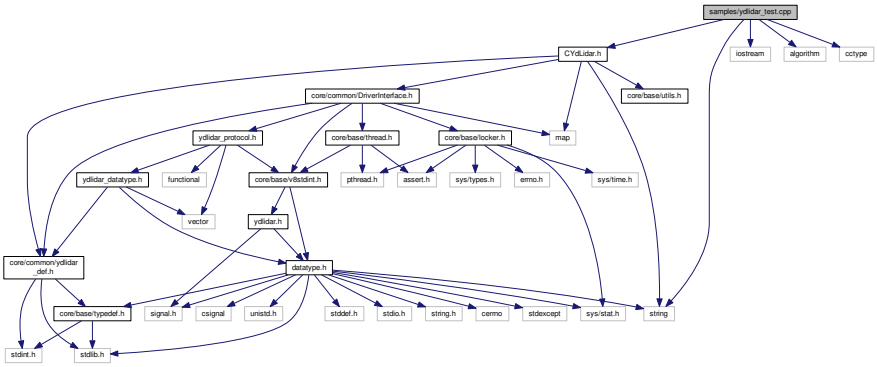
Uninitialize the SDK and Disconnect the LiDAR.

Definition at line 60 of file tof_test.cpp.

39.75 samples/ydlidar_test.cpp File Reference

```
#include "CYdLidar.h"
#include <iostream>
#include <string>
#include <algorithm>
#include <cctype>
```

Include dependency graph for ydlidar_test.cpp:



Functions

- int main (int argc, char ∗argv[ ])

  ydlidar test

39.75.1 Function Documentation

39.75.1.1 int main (int *argc*, char ∗ *argv[ ]*)

ydlidar test

Parameters

argc	
argv	

Returns

Flow chart

Step1: instance CYdLidar.
Step2: set paramters.
Step3: initialize SDK and LiDAR.(CYdLidar::initialize)
Step4: Start the device scanning routine which runs on a separate thread and enable motor.(CYdLidar::turnOn)
Step5: Get the LiDAR Scan Data.(CYdLidar::doProcessSimple)

Step6: Stop the device scanning thread and disable motor.(CYdLidar::turnOff)
Step7: Uninitialize the SDK and Disconnect the LiDAR.(CYdLidar::disconnecting)

string property/////////////////// lidar port

ignore array

int property/////////////////// lidar baudrate

tof lidar

device type

sample rate

abnormal count

bool property/////////////////// fixed angle resolution

rotate 180

Counterclockwise

one-way communication

intensity

Motor DTR

float property/////////////////// unit: °

unit: m

unit: Hz

Definition at line 61 of file ydlidar_test.cpp.

39.76 setup.py File Reference

Classes

- class setup.CMakeExtension
- class setup.CMakeBuild

Namespaces

- setup

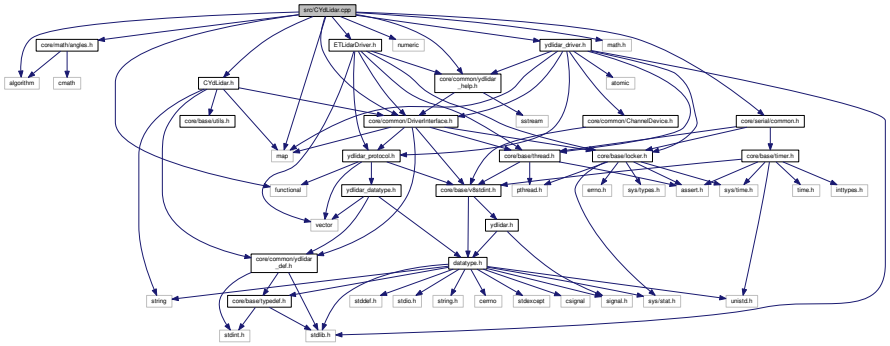
Variables

- string setup.YDLIDAR_SDK_REPO = "https://github.com/YDLIDAR/YDLidar-SDK"
- string setup.YDLIDAR_SDK_BRANCH = "master"

39.77 src/CYdLidar.cpp File Reference

```
#include "CYdLidar.h"
#include <core/serial/common.h>
#include <map>
#include <core/math/angles.h>
#include <numeric>
#include <algorithm>
#include "ydlidar_driver.h"
#include <math.h>
#include <functional>
#include <core/common/DriverInterface.h>
#include <core/common/ydlidar_help.h>
#include "ETLidarDriver.h"
```

Include dependency graph for CYdLidar.cpp:



Namespaces

- ydlidar

  ydlidar

Functions

- void removeExceptionSample (std::map< int, int > &smap)
- void ydlidar::os_init ()

  system signal initialize
- bool ydlidar::os_isOk ()

  Whether system signal is initialized.
- void ydlidar::os_shutdown ()

  shutdown system signal
- std::map< std::string, std::string > ydlidar::lidarPortList ()

  lidarPortList

39.77.1 Function Documentation

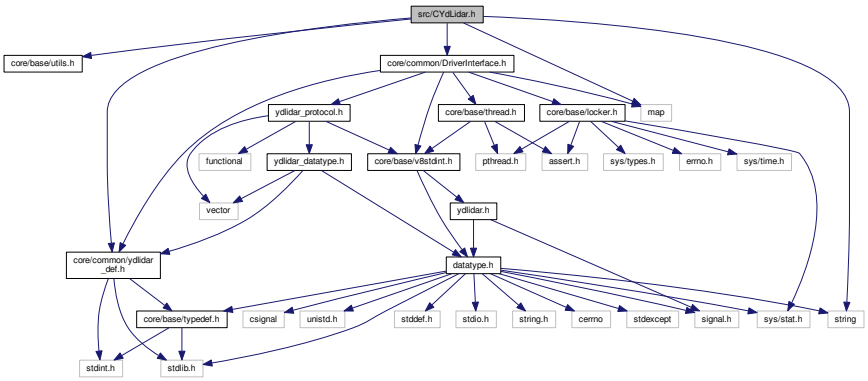
39.77.1.1 void removeExceptionSample (std::map< int, int > & *smap*) [inline]

Definition at line 909 of file CYdLidar.cpp.

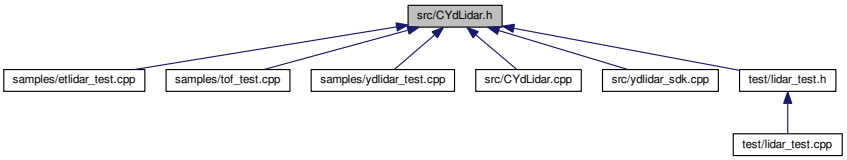
39.78 src/CYdLidar.h File Reference

```
#include <core/base/utils.h>
#include <core/common/ydlidar_def.h>
#include <core/common/DriverInterface.h>
#include <string>
#include <map>
```

Include dependency graph for CYdLidar.h:



This graph shows which files directly or indirectly include this file:



Classes

- class CYdLidar

  Set and Get LiDAR Maximum effective range.

Namespaces

- ydlidar

  ydlidar

Functions

- void ydlidar::os_init ()

  system signal initialize
- bool ydlidar::os_isOk ()

  Whether system signal is initialized.
- void ydlidar::os_shutdown ()

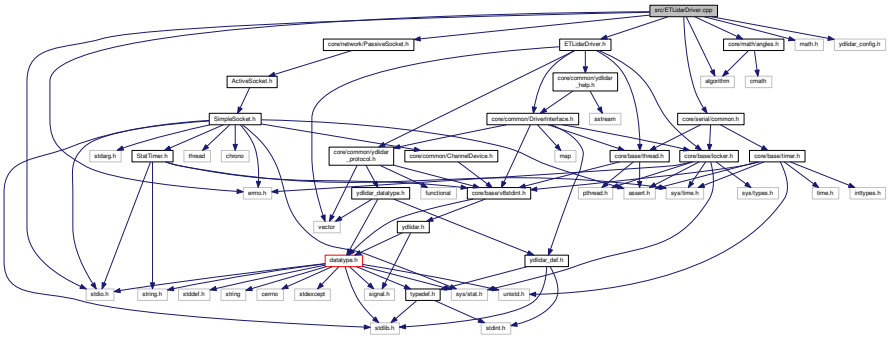
  shutdown system signal
- std::map< std::string, std::string > ydlidar::lidarPortList ()

  lidarPortList

39.79 src/ETLidarDriver.cpp File Reference

```
#include <core/network/PassiveSocket.h>
#include "ETLidarDriver.h"
#include <stdio.h>
#include <errno.h>
#include <core/serial/common.h>
#include <core/math/angles.h>
#include <math.h>
#include <algorithm>
#include <ydlidar_config.h>
```

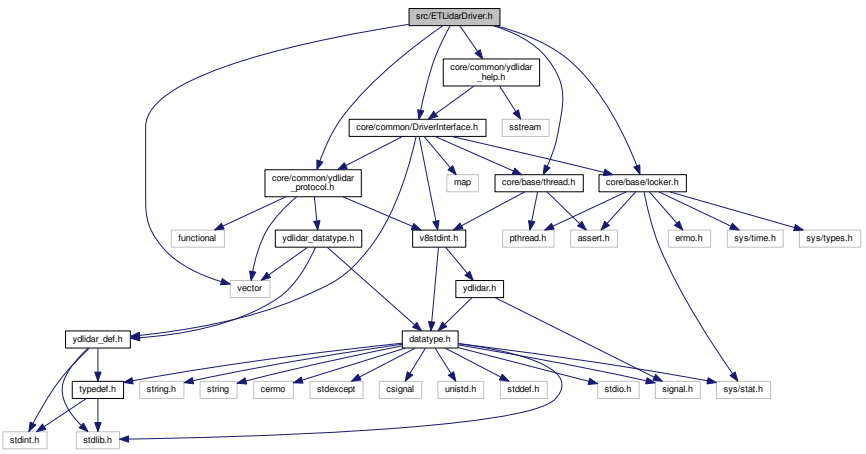
Include dependency graph for ETLidarDriver.cpp:



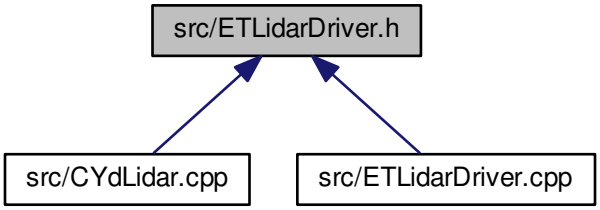
39.80 src/ETLidarDriver.h File Reference

```
#include <core/base/thread.h>
#include <core/base/locker.h>
#include <vector>
#include <core/common/ydlidar_protocol.h>
#include <core/common/DriverInterface.h>
#include <core/common/ydlidar_help.h>
```

Include dependency graph for ETLidarDriver.h:



This graph shows which files directly or indirectly include this file:



Classes

- class ydlidar::ETLidarDriver

Namespaces

- ydlidar

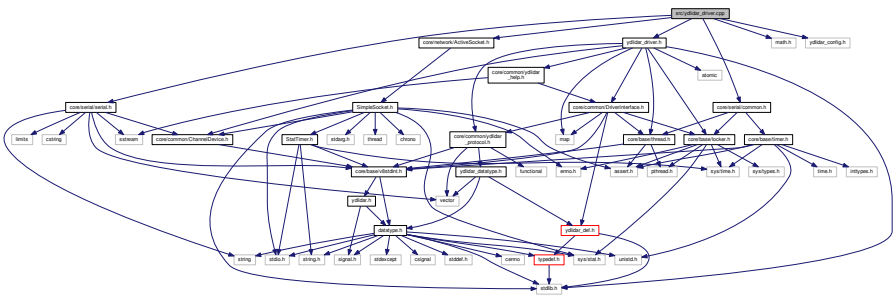
  ydlidar
- ydlidar::core

  ydlidar core
- ydlidar::core::network

39.81 src/ydlidar_driver.cpp File Reference

```
#include <core/serial/serial.h>
#include <core/network/ActiveSocket.h>
#include "ydlidar_driver.h"
#include <core/serial/common.h>
#include <math.h>
#include <ydlidar_config.h>
```

Include dependency graph for ydlidar_driver.cpp:



Namespaces

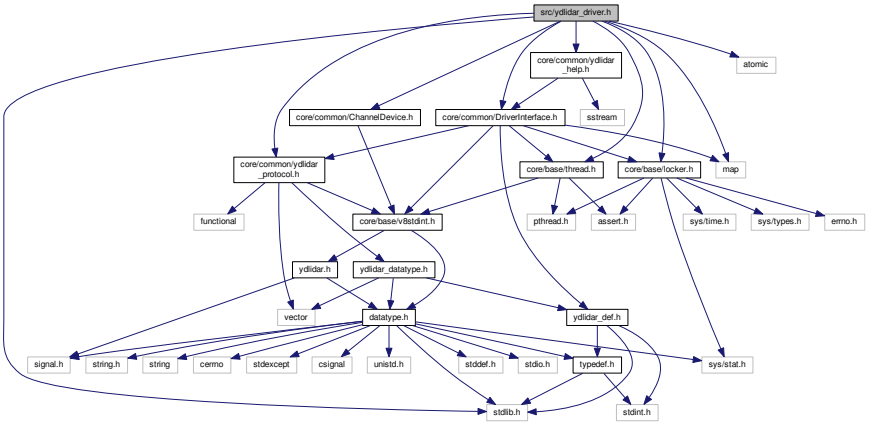
- ydlidar

  ydlidar

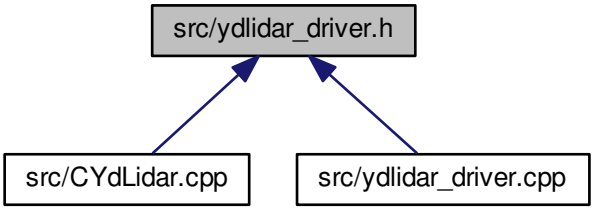
39.82 src/ydlidar_driver.h File Reference

```
#include <stdlib.h>
#include <atomic>
#include <map>
#include <core/common/ChannelDevice.h>
#include <core/base/locker.h>
#include <core/base/thread.h>
#include <core/common/ydlidar_protocol.h>
#include <core/common/DriverInterface.h>
#include <core/common/ydlidar_help.h>
```

Include dependency graph for ydlidar_driver.h:



This graph shows which files directly or indirectly include this file:



Classes

- class ydlidar::YDlidarDriver

Namespaces

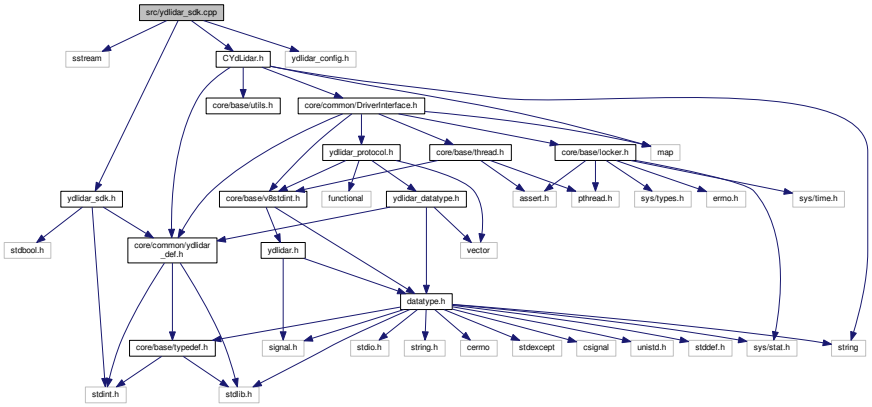
- ydlidar

  ydlidar

39.83 src/ydlidar_sdk.cpp File Reference

```
#include <sstream>
#include "ydlidar_sdk.h"
#include "CYdLidar.h"
#include "ydlidar_config.h"
```

Include dependency graph for ydlidar_sdk.cpp:



Functions

- YDLidar * lidarCreate ()

  create a Lidar instance
- void lidarDestroy (YDLidar **lidar)

  Destroy Lidar instance by lidarCreate create.
- bool setlidaropt (YDLidar *lidar, int optname, const void *optval, int optlen)

  set lidar properties
- bool getlidaropt (YDLidar *lidar, int optname, void *optval, int optlen)

  get lidar property
- void GetSdkVersion (char *version)
- bool initialize (YDLidar *lidar)
- void GetLidarVersion (YDLidar *lidar, LidarVersion *version)

  Return LiDAR's version information in a numeric form.
- bool turnOn (YDLidar *lidar)
- bool doProcessSimple (YDLidar *lidar, LaserFan *outscan)

  Get the LiDAR Scan Data. turnOn is successful before doProcessSimple scan data.
- bool turnOff (YDLidar *lidar)

  Stop the device scanning thread and disable motor.
- void disconnecting (YDLidar *lidar)

  Uninitialize the SDK and Disconnect the LiDAR.
- const char * DescribeError (YDLidar *lidar)

Get the last error information of a (socket or serial)

- void os_init ()

  initialize system signals

- bool os_isOk ()

  isOk

- void os_shutdown ()

  os_shutdown

- int lidarPortList (LidarPort *ports)

  get lidar serial port

39.83.1 Function Documentation

39.83.1.1 const char* DescribeError (YDLidar * *lidar*)

Get the last error information of a (socket or serial)

Returns

a human-readable description of the given error information or the last error information of a (socket or serial)

Definition at line 179 of file ydlidar_sdk.cpp.

39.83.1.2 void disconnecting (YDLidar * *lidar*)

Uninitialize the SDK and Disconnect the LiDAR.

Definition at line 167 of file ydlidar_sdk.cpp.

39.83.1.3 bool doProcessSimple (YDLidar * *lidar,* LaserFan * *outscan*)

Get the LiDAR Scan Data. turnOn is successful before doProcessSimple scan data.

Parameters

in	*lidar*	LiDAR instance
out	*outscan*	LiDAR Scan Data

Returns

true if successfully started, otherwise false.

Definition at line 129 of file ydlidar_sdk.cpp.

39.83.1.4 bool getlidaropt (YDLidar * *lidar,* int *optname,* void * *optval,* int *optlen*)

get lidar property

Parameters

lidar	a lidar instance
optname	option name

Todo string properties

- LidarPropSerialPort
- LidarPropIgnoreArray

Note

get string property example

```
1 CYdLidar laser;
2 char lidar_port[30];
3 laser.getlidaropt(LidarPropSerialPort,lidar_port, sizeof(lidar_port));
```

Todo int properties

- LidarPropSerialBaudrate
- LidarPropLidarType
- LidarPropDeviceType
- LidarPropSampleRate

Note

get int property example

```
1 CYdLidar laser;
2 int lidar_baudrate;
3 laser.getlidaropt(LidarPropSerialPort,&lidar_baudrate, sizeof(int));
```

Todo bool properties

- LidarPropFixedResolution
- LidarPropReversion
- LidarPropInverted
- LidarPropAutoReconnect
- LidarPropSingleChannel
- LidarPropIntenstiy

Note

get bool property example

```
1 CYdLidar laser;
2 bool lidar_fixedresolution;
3 laser.getlidaropt(LidarPropSerialPort,&lidar_fixedresolution, sizeof(bool));
```

Todo float properties

- LidarPropMaxRange
- LidarPropMinRange
- LidarPropMaxAngle
- LidarPropMinAngle
- LidarPropScanFrequency

Note

set float property example

```
1 CYdLidar laser;
2 float lidar_maxrange;
3 laser.getlidaropt(LidarPropSerialPort,&lidar_maxrange, sizeof(float));
```

Parameters

optval	option value • std::string(or char*) • int • bool • float
optlen	option length • data type size

Returns

true if the Property is get successfully, otherwise false.

See also

LidarProperty

Definition at line 70 of file ydlidar_sdk.cpp.

39.83.1.5 void GetLidarVersion (YDLidar * *lidar*, LidarVersion * *version*)

Return LiDAR's version information in a numeric form.

Parameters

version	Pointer to a version structure for returning the version information.

Definition at line 103 of file ydlidar_sdk.cpp.

39.83.1.6 void GetSdkVersion (char * *version*)

Return SDK's version information in a numeric form.

Parameters

version	Pointer to a version for returning the version information.

Definition at line 85 of file ydlidar_sdk.cpp.

39.83.1.7 bool initialize (YDLidar * *lidar*)

Initialize the SDK.

Returns

true if successfully initialized, otherwise false.

Definition at line 89 of file ydlidar_sdk.cpp.

39.83.1.8 YDLidar* lidarCreate (void)

create a Lidar instance

"YDLIDAR_C_API"

YDLIDAR_C_API

Note

call lidarDestroy destroy

Returns

created instance

Definition at line 30 of file ydlidar_sdk.cpp.

39.83.1.9 void lidarDestroy (YDLidar ** *lidar*)

Destroy Lidar instance by lidarCreate create.

Parameters

lidar	CYdLidar instance

Definition at line 38 of file ydlidar_sdk.cpp.

39.83.1.10 int lidarPortList (LidarPort * *ports*)

get lidar serial port

Parameters

ports	serial port lists

Returns

valid port number

Definition at line 208 of file ydlidar_sdk.cpp.

39.83.1.11 void os_init ( )

initialize system signals

initialize system signals

Definition at line 195 of file ydlidar_sdk.cpp.

39.83.1.12 bool os_isOk ( )

isOk

Returns

true if successfully initialize, otherwise false.

isOk

Returns

Definition at line 199 of file ydlidar_sdk.cpp.

39.83.1.13 void os_shutdown ( )

os_shutdown

os_shutdown

Definition at line 203 of file ydlidar_sdk.cpp.

39.83.1.14 bool setlidaropt (YDLidar * *lidar*, int *optname*, const void * *optval*, int *optlen*)

set lidar properties

Parameters

lidar	a lidar instance
optname	option name

Todo string properties

- LidarPropSerialPort

- LidarPropIgnoreArray

Note

set string property example

```
1 CYdLidar laser;
2 std::string lidar_port = "/dev/ydlidar";
3 laser.setlidaropt(LidarPropSerialPort,lidar_port.c_str(), lidar_port.size());
```

Todo int properties

- LidarPropSerialBaudrate
- LidarPropLidarType
- LidarPropDeviceType
- LidarPropSampleRate

Note

set int property example

```
1 CYdLidar laser;
2 int lidar_baudrate = 230400;
3 laser.setlidaropt(LidarPropSerialPort,&lidar_baudrate, sizeof(int));
```

Todo bool properties

- LidarPropFixedResolution
- LidarPropReversion
- LidarPropInverted
- LidarPropAutoReconnect
- LidarPropSingleChannel
- LidarPropIntenstiy

Note

set bool property example

```
1 CYdLidar laser;
2 bool lidar_fixedresolution = true;
3 laser.setlidaropt(LidarPropSerialPort,&lidar_fixedresolution, sizeof(bool));
```

Todo float properties

- LidarPropMaxRange
- LidarPropMinRange
- LidarPropMaxAngle
- LidarPropMinAngle
- LidarPropScanFrequency

Note

set float property example

```
1 CYdLidar laser;
2 float lidar_maxrange = 16.0f;
3 laser.setlidaropt(LidarPropSerialPort,&lidar_maxrange, sizeof(float));
```

Parameters

optval	option value • std::string(or char*) • int • bool • float
optlen	option length • data type size

Returns

true if the Property is set successfully, otherwise false.

See also

LidarProperty

Definition at line 56 of file ydlidar_sdk.cpp.

39.83.1.15 bool turnOff (YDLidar * *lidar*)

Stop the device scanning thread and disable motor.

Returns

true if successfully Stoped, otherwise false.

Definition at line 153 of file ydlidar_sdk.cpp.

39.83.1.16 bool turnOn (YDLidar * *lidar*)

Start the device scanning routine which runs on a separate thread.

Returns

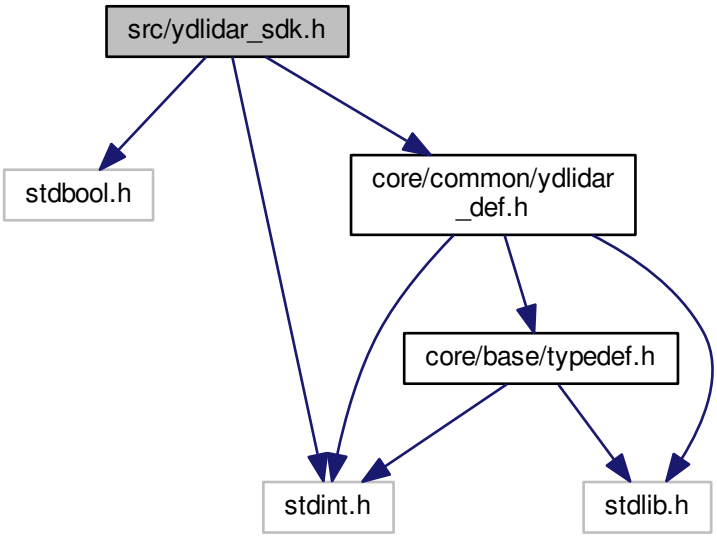
true if successfully started, otherwise false.

Definition at line 115 of file ydlidar_sdk.cpp.

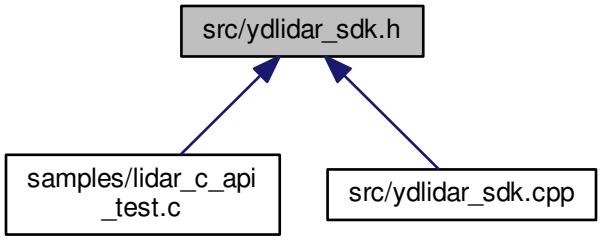
39.84 src/ydlidar_sdk.h File Reference

```
#include <stdbool.h>
#include <stdint.h>
#include <core/common/ydlidar_def.h>
```

Include dependency graph for ydlidar_sdk.h:



This graph shows which files directly or indirectly include this file:



Functions

- YDLidar * lidarCreate (void)

  create a Lidar instance
- void lidarDestroy (YDLidar **lidar)

Destroy Lidar instance by lidarCreate create.

- bool setlidaropt (YDLidar *lidar, int optname, const void *optval, int optlen)

  set lidar properties
- bool getlidaropt (YDLidar *lidar, int optname, void *optval, int optlen)

  get lidar property
- void GetSdkVersion (char *version)
- bool initialize (YDLidar *lidar)
- void GetLidarVersion (YDLidar *lidar, LidarVersion *version)

  Return LiDAR's version information in a numeric form.
- bool turnOn (YDLidar *lidar)
- bool doProcessSimple (YDLidar *lidar, LaserFan *outscan)

  Get the LiDAR Scan Data. turnOn is successful before doProcessSimple scan data.
- bool turnOff (YDLidar *lidar)

  Stop the device scanning thread and disable motor.
- void disconnecting (YDLidar *lidar)

  Uninitialize the SDK and Disconnect the LiDAR.
- const char * DescribeError (YDLidar *lidar)

  Get the last error information of a (socket or serial)
- void os_init ()

  initialize system signals
- bool os_isOk ()

  isOk
- void os_shutdown ()

  os_shutdown
- int lidarPortList (LidarPort *ports)

  get lidar serial port

39.84.1 Function Documentation

39.84.1.1 const char* DescribeError (YDLidar * *lidar*)

Get the last error information of a (socket or serial)

Returns

a human-readable description of the given error information or the last error information of a (socket or serial)

Definition at line 179 of file ydlidar_sdk.cpp.

39.84.1.2 void disconnecting (YDLidar * *lidar*)

Uninitialize the SDK and Disconnect the LiDAR.

Definition at line 167 of file ydlidar_sdk.cpp.

39.84.1.3 bool doProcessSimple (YDLidar * *lidar*, LaserFan * *outscan*)

Get the LiDAR Scan Data. turnOn is successful before doProcessSimple scan data.

Parameters

in	*lidar*	LiDAR instance
out	*outscan*	LiDAR Scan Data

Returns

true if successfully started, otherwise false.

Definition at line 129 of file ydlidar_sdk.cpp.

39.84.1.4 bool getlidaropt (YDLidar * *lidar*, int *optname*, void * *optval*, int *optlen*)

get lidar property

Parameters

lidar	a lidar instance
optname	option name

Todo string properties

- LidarPropSerialPort
- LidarPropIgnoreArray

Note

get string property example

```
1 CYdLidar laser;
2 char lidar_port[30];
3 laser.getlidaropt(LidarPropSerialPort,lidar_port, sizeof(lidar_port));
```

Todo int properties

- LidarPropSerialBaudrate
- LidarPropLidarType
- LidarPropDeviceType
- LidarPropSampleRate

Note

get int property example

```
1 CYdLidar laser;
2 int lidar_baudrate;
3 laser.getlidaropt(LidarPropSerialPort,&lidar_baudrate, sizeof(int));
```

Todo bool properties

- LidarPropFixedResolution

- LidarPropReversion
- LidarPropInverted
- LidarPropAutoReconnect
- LidarPropSingleChannel
- LidarPropIntenstiy

Note

get bool property example

```
1 CYdLidar laser;
2 bool lidar_fixedresolution;
3 laser.getlidaropt(LidarPropSerialPort,&lidar_fixedresolution, sizeof(bool));
```

Todo float properties

- LidarPropMaxRange
- LidarPropMinRange
- LidarPropMaxAngle
- LidarPropMinAngle
- LidarPropScanFrequency

Note

set float property example

```
1 CYdLidar laser;
2 float lidar_maxrange;
3 laser.getlidaropt(LidarPropSerialPort,&lidar_maxrange, sizeof(float));
```

Parameters

optval	option value • std::string(or char∗) • int • bool • float
optlen	option length • data type size

Returns

true if the Property is get successfully, otherwise false.

See also

LidarProperty

Definition at line 70 of file ydlidar_sdk.cpp.

39.84.1.5 void GetLidarVersion (YDLidar * *lidar,* LidarVersion * *version*)

Return LiDAR's version information in a numeric form.

Parameters

version	Pointer to a version structure for returning the version information.

Definition at line 103 of file ydlidar_sdk.cpp.

39.84.1.6 void GetSdkVersion (char * *version*)

Return SDK's version information in a numeric form.

Parameters

version	Pointer to a version for returning the version information.

Definition at line 85 of file ydlidar_sdk.cpp.

39.84.1.7 bool initialize (YDLidar * *lidar*)

Initialize the SDK.

Returns

true if successfully initialized, otherwise false.

Definition at line 89 of file ydlidar_sdk.cpp.

39.84.1.8 YDLidar* lidarCreate (void)

create a Lidar instance

"YDLIDAR_C_API"

YDLIDAR_C_API

Note

call lidarDestroy destroy

Returns

created instance

Definition at line 30 of file ydlidar_sdk.cpp.

39.84.1.9 void lidarDestroy (YDLidar ** *lidar*)

Destroy Lidar instance by lidarCreate create.

Parameters

lidar	CYdLidar instance

Definition at line 38 of file ydlidar_sdk.cpp.

39.84.1.10 int lidarPortList (LidarPort * *ports*)

get lidar serial port

Parameters

ports	serial port lists

Returns

valid port number

Definition at line 208 of file ydlidar_sdk.cpp.

39.84.1.11 void os_init ( )

initialize system signals

initialize system signals

Definition at line 1378 of file CYdLidar.cpp.

39.84.1.12 bool os_isOk ( )

isOk

Returns

true if successfully initialize, otherwise false.

isOk

Returns

Definition at line 1382 of file CYdLidar.cpp.

39.84.1.13 void os_shutdown ( )

os_shutdown

os_shutdown

Definition at line 1386 of file CYdLidar.cpp.

39.84.1.14 bool setlidaropt (YDLidar * *lidar,* int *optname,* const void * *optval,* int *optlen*)

set lidar properties

Parameters

lidar	a lidar instance
optname	option name

Todo string properties

- LidarPropSerialPort
- LidarPropIgnoreArray

Note

set string property example

```
1 CYdLidar laser;
2 std::string lidar_port = "/dev/ydlidar";
3 laser.setlidaropt(LidarPropSerialPort,lidar_port.c_str(), lidar_port.size());
```

Todo int properties

- LidarPropSerialBaudrate
- LidarPropLidarType
- LidarPropDeviceType
- LidarPropSampleRate

Note

set int property example

```
1 CYdLidar laser;
2 int lidar_baudrate = 230400;
3 laser.setlidaropt(LidarPropSerialPort,&lidar_baudrate, sizeof(int));
```

Todo bool properties

- LidarPropFixedResolution
- LidarPropReversion
- LidarPropInverted
- LidarPropAutoReconnect
- LidarPropSingleChannel
- LidarPropIntenstiy

Note

set bool property example

```
1 CYdLidar laser;
2 bool lidar_fixedresolution = true;
3 laser.setlidaropt(LidarPropSerialPort,&lidar_fixedresolution, sizeof(bool));
```

Todo float properties

- LidarPropMaxRange
- LidarPropMinRange
- LidarPropMaxAngle
- LidarPropMinAngle
- LidarPropScanFrequency

Note

set float property example

```
1 CYdLidar laser;
2 float lidar_maxrange = 16.0f;
3 laser.setlidaropt(LidarPropSerialPort,&lidar_maxrange, sizeof(float));
```

Parameters

optval	option value • std::string(or char*) • int • bool • float
optlen	option length • data type size

Returns

true if the Property is set successfully, otherwise false.

See also

LidarProperty

Definition at line 56 of file ydlidar_sdk.cpp.

39.84.1.15 bool turnOff (YDLidar * *lidar*)

Stop the device scanning thread and disable motor.

Returns

true if successfully Stoped, otherwise false.

Definition at line 153 of file ydlidar_sdk.cpp.

39.84.1.16 bool turnOn (YDLidar * *lidar*)

Start the device scanning routine which runs on a separate thread.

Returns

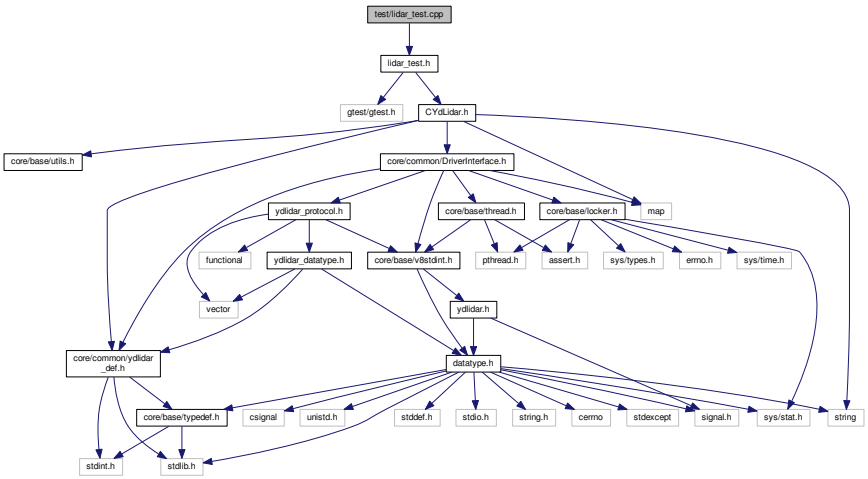
true if successfully started, otherwise false.

Definition at line 115 of file ydlidar_sdk.cpp.

39.85 test/lidar_test.cpp File Reference

```
#include "lidar_test.h"
```

Include dependency graph for lidar_test.cpp:



Functions

- TEST_F (LidarTest, SystemSignal)
- TEST_F (LidarTest, SerialPort)
- TEST_F (LidarTest, SerialBaudrate)
- TEST_F (LidarTest, SingleChannel)
- TEST_F (LidarTest, ScanFequency)
- TEST_F (LidarTest, TurnOn)
- int main (int argc, char **argv)

39.85.1 Function Documentation

39.85.1.1 int main (int *argc*, char ** *argv*)

Definition at line 75 of file lidar_test.cpp.

39.85.1.2 TEST_F (LidarTest , SystemSignal)

Definition at line 23 of file lidar_test.cpp.

39.85.1.3 TEST_F (LidarTest , SerialPort)

Definition at line 30 of file lidar_test.cpp.

39.85.1.4 TEST_F (LidarTest , SerialBaudrate)

Definition at line 37 of file lidar_test.cpp.

39.85.1.5 TEST_F (LidarTest , SingleChannel)

Definition at line 43 of file lidar_test.cpp.

39.85.1.6 TEST_F (LidarTest , ScanFequency)

Definition at line 49 of file lidar_test.cpp.

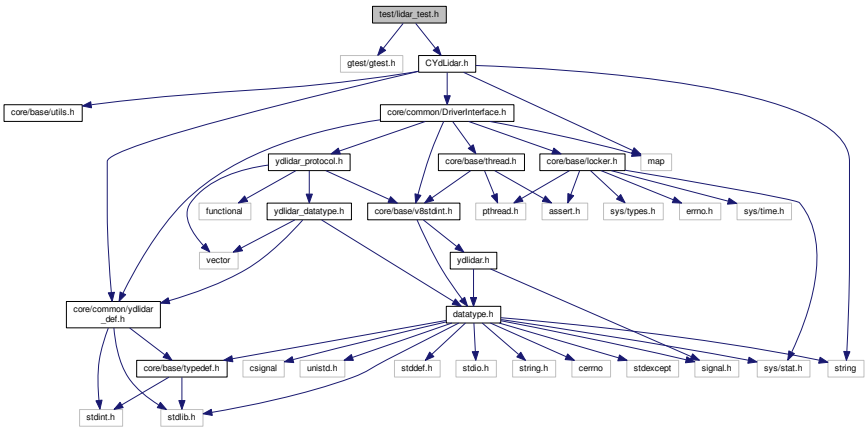
39.85.1.7 TEST_F (LidarTest , TurnOn)

Definition at line 56 of file lidar_test.cpp.

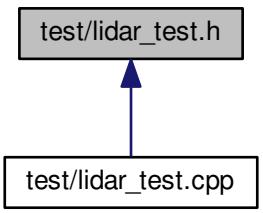
39.86 test/lidar_test.h File Reference

```
#include "gtest/gtest.h"
#include "CYdLidar.h"
```

Include dependency graph for lidar_test.h:



This graph shows which files directly or indirectly include this file:



Classes

- class LidarTest

Index