

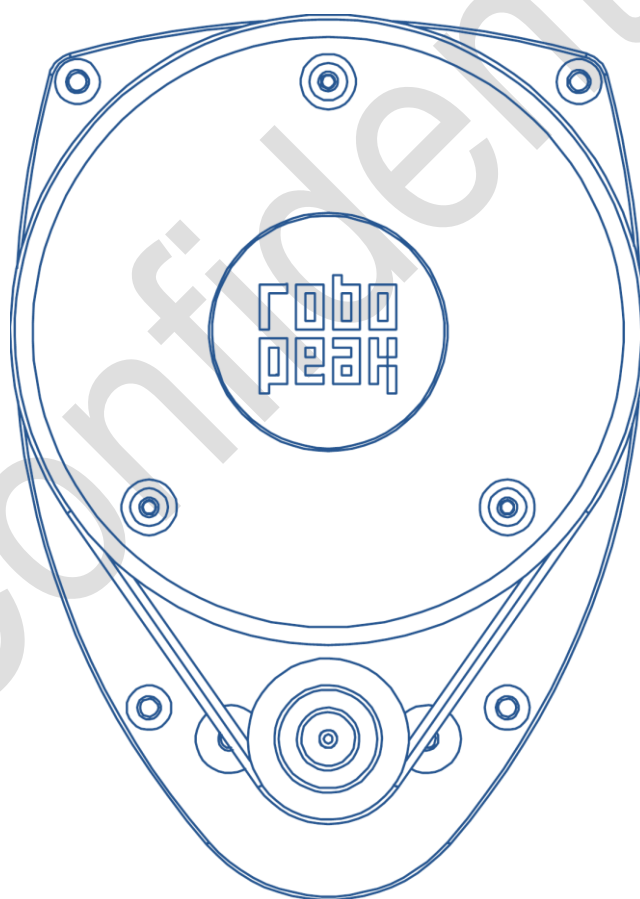


RPLIDAR

Low Cost 360 degree 2D Laser Scanner (LIDAR) System

Introduction to Standard SDK

2014-7  
Rev. 3



## Contents:

---

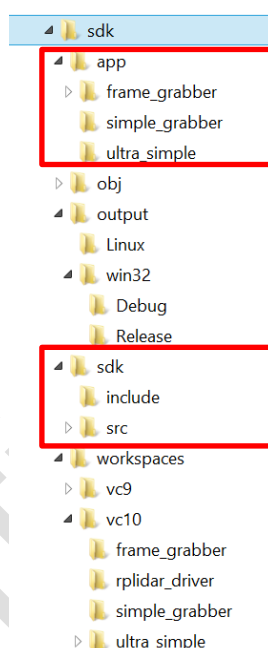
1. Introduction .....	2
SDK Organization .....	2
Build SDK and Demo Applications .....	3
Cross Compile .....	6
2. Demo Applications .....	7
ultra_simple .....	7
simple_grabber .....	8
frame_grabber .....	9
3. SDK Usage and Development Guide .....	11
Assumption .....	11
SDK usage .....	11
Runtime consistency .....	11
SDK Headers .....	12
SDK Initialization and Termination .....	12
Connecting to an RPLIDAR .....	13
Measurement Scan and Data Acquiring .....	13
Retrieving other information of an RPLIDAR .....	15
4. Revision History .....	16

## 1. Introduction

This document introduces RPLIDAR standard SDK. The SDK can be used in both Windows, Linux and MacOS (10.x) environment by using Microsoft Visual C++ 2008, 2010 and Makefile.

### SDK Organization

The RPLIDAR standard SDK organized as bellow:



The **workspaces** directory contains VS project files for SDK and demo applications.

The **sdk** directory contains the source code of RPLIDAR driver. The **include** folder contains all the header files required for applications use SDK. The **src** folder is the implementation of the SDK.

RoboPeak provides the following demo applications in the app directory:

- **ultra\_simple**

An ultra-simple command line application demonstrates the simplest way to connect to an RPLIDAR device and continuously fetching the scan data and outputting the

data to the console.

Users can quickly integrate RPLIDAR to their existing system based on this demo application.

- **Simple\_grabber**

A command line grab application. Each execution will grab two round of laser data and show as histogram.

- **Frame\_grabber**

A win32 GUI grab application. When press start scan button, it will start scan continuously and show the data in the UI.

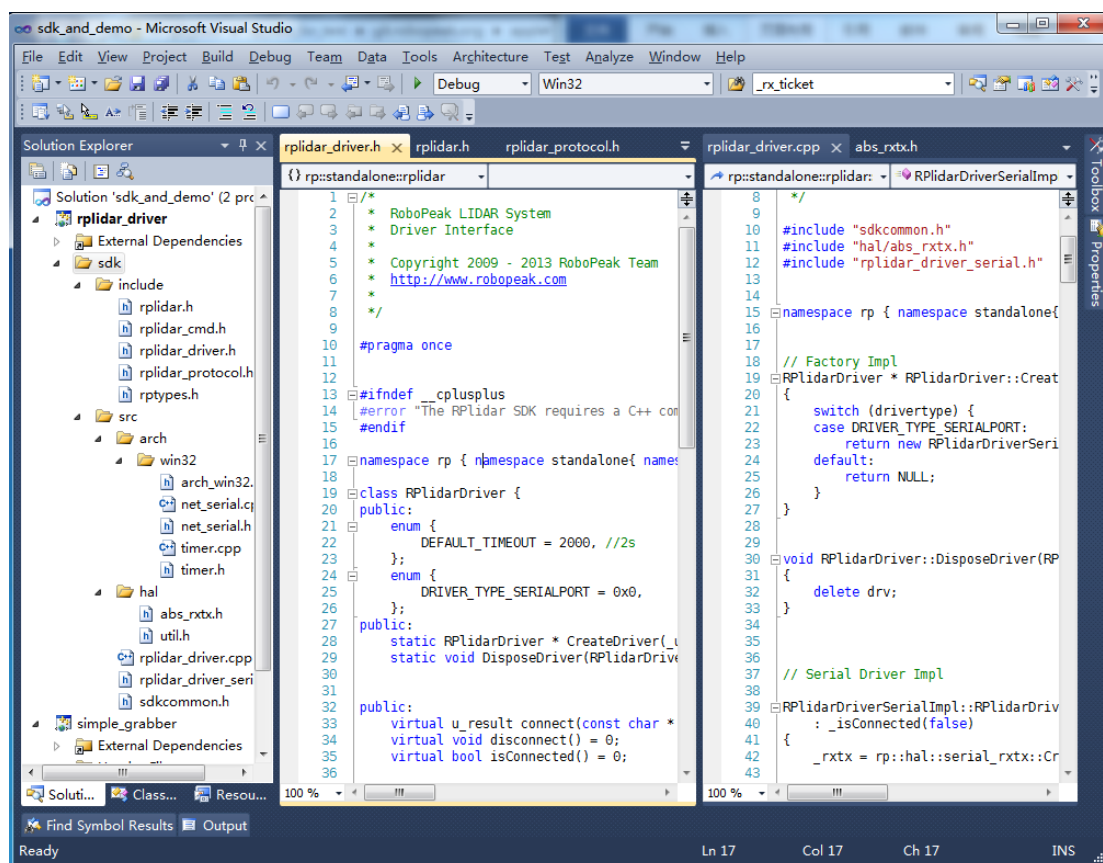
After compilation, there will be two more folders in the SDK: obj and output. Output folder contains generated SDK static library (.lib or .a) and demo application executable files (exe or elf). obj folder contains intermediate files generated during compilation.

## Build SDK and Demo Applications

If you're developing under Windows, please open VS solution file under workspaces\vc10: sdk\_and\_demo.sln. It has included SDK project and all demo application projects.

# Low Cost 360 degree 2D Laser Scanner (LIDAR) System

## Introduction to Standard SDK



Now, you can use VS to build SDK and demo applications easily. Depends on your build configure. The generated binary can be found under output\win32\Release or output\win32\Debug.

If you're developing under Linux or MacOS, just type "make" under the root of SDK directory. It will do Release build by default, you can also typing "make DEBUG=1" to do Debug build. The generated binary can be found under the following folders:

### Linux

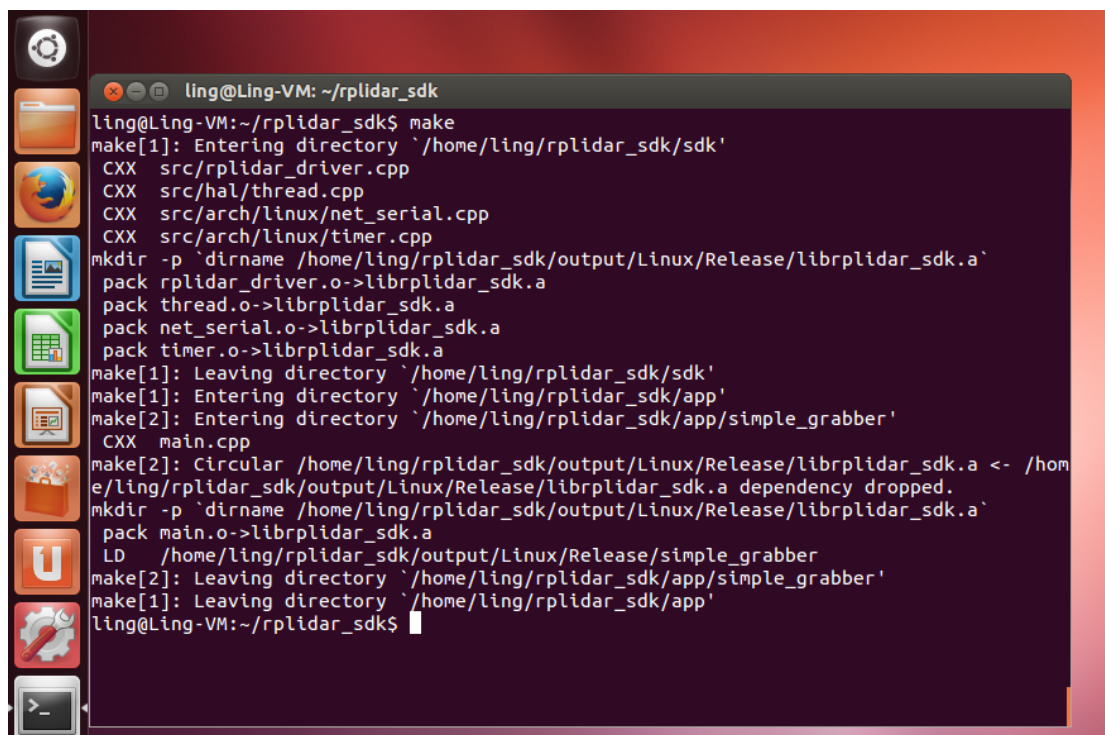
- output\Linux\Release
- output\Linux\Debug.

### MacOS

- output\Darwin\Release
- output\Darwin\Debug.

# Low Cost 360 degree 2D Laser Scanner (LIDAR) System

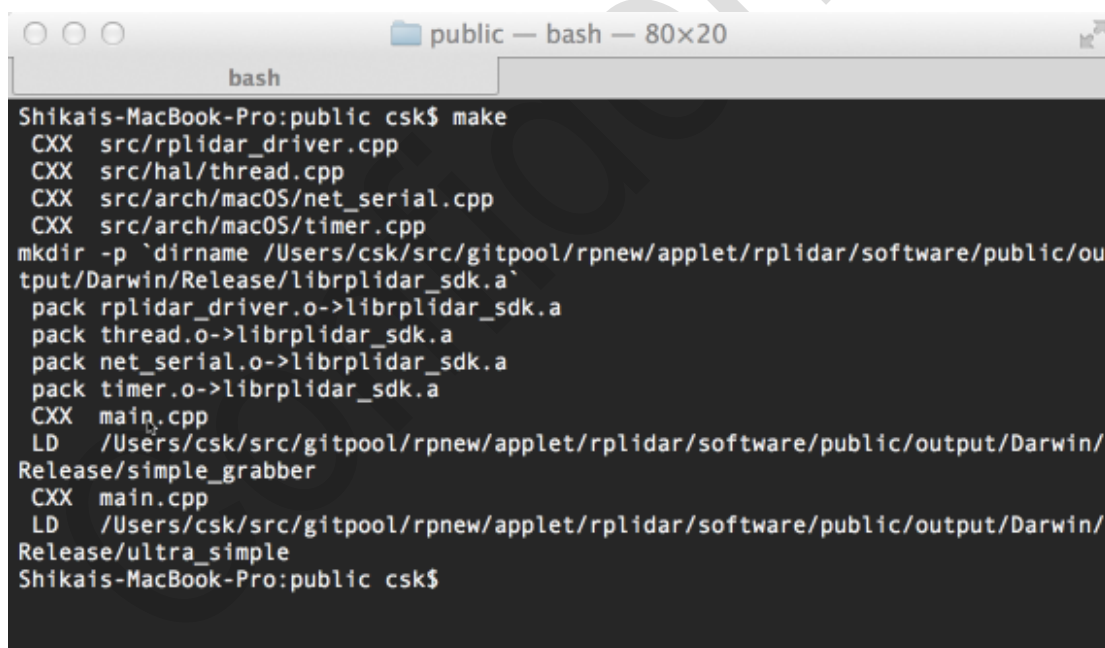
## Introduction to Standard SDK



```

ling@Ling-VM: ~/rplidar_sdk
ling@Ling-VM:~/rplidar_sdk$ make
make[1]: Entering directory `/home/ling/rplidar_sdk/sdk'
CXX src/rplidar_driver.cpp
CXX src/hal/thread.cpp
CXX src/arch/linux/net_serial.cpp
CXX src/arch/linux/timer.cpp
mkdir -p `dirname /home/ling/rplidar_sdk/output/Linux/Release/librplidar_sdk.a`
pack rplidar_driver.o->librplidar_sdk.a
pack thread.o->librplidar_sdk.a
pack net_serial.o->librplidar_sdk.a
pack timer.o->librplidar_sdk.a
make[1]: Leaving directory `/home/ling/rplidar_sdk/sdk'
make[1]: Entering directory `/home/ling/rplidar_sdk/app'
make[2]: Entering directory `/home/ling/rplidar_sdk/app/simple_grabber'
CXX main.cpp
make[2]: Circular /home/ling/rplidar_sdk/output/Linux/Release/librplidar_sdk.a <- /home/ling/rplidar_sdk/output/Linux/Release/librplidar_sdk.a dependency dropped.
mkdir -p `dirname /home/ling/rplidar_sdk/output/Linux/Release/librplidar_sdk.a`
pack main.o->librplidar_sdk.a
LD /home/ling/rplidar_sdk/output/Linux/Release/simple_grabber
make[2]: Leaving directory `/home/ling/rplidar_sdk/app/simple_grabber'
make[1]: Leaving directory `/home/ling/rplidar_sdk/app'
ling@Ling-VM:~/rplidar_sdk$

```



```

public — bash — 80x20
bash
Shikais-MacBook-Pro:public csk$ make
CXX src/rplidar_driver.cpp
CXX src/hal/thread.cpp
CXX src/arch/macOS/net_serial.cpp
CXX src/arch/macOS/timer.cpp
mkdir -p `dirname /Users/csk/src/gitpool/rpnew/applet/rplidar/software/public/output/Darwin/Release/librplidar_sdk.a`
pack rplidar_driver.o->librplidar_sdk.a
pack thread.o->librplidar_sdk.a
pack net_serial.o->librplidar_sdk.a
pack timer.o->librplidar_sdk.a
CXX main.cpp
LD /Users/csk/src/gitpool/rpnew/applet/rplidar/software/public/output/Darwin/Release/simple_grabber
CXX main.cpp
LD /Users/csk/src/gitpool/rpnew/applet/rplidar/software/public/output/Darwin/Release/ultra_simple
Shikais-MacBook-Pro:public csk$

```

## Cross Compile

The SDK build system allows you to generate binaries which run on another platform/system using the cross-compiling feature.

**NOTICE: this feature only works with Make build system.**

The cross compile process can be triggered by invoking the `cross_compile.sh` script under the SDK root folder. The common usage is:

```
CROSS_COMPILE_PREFIX=<COMPILE_PREFIX> ./cross_compile.sh
```

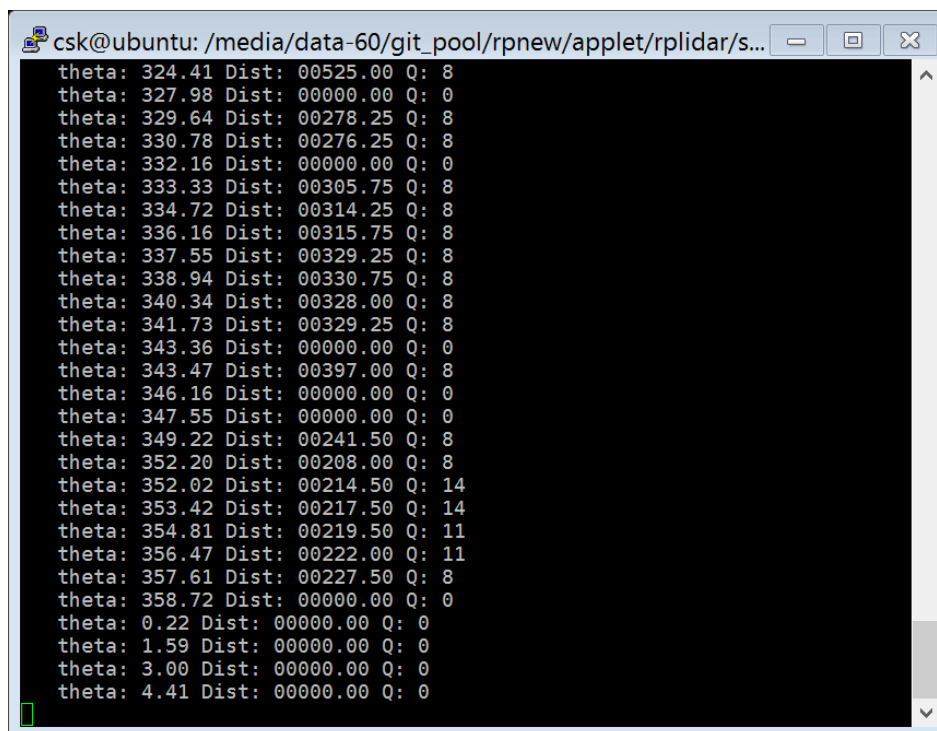
e.g. `CROSS_COMPILE_PREFIX=arm-linux-gnueabihf ./cross_compile.sh`

Confidential

## 2. Demo Applications

### ultra\_simple

The demo application simply connects to an RPLIDAR device and outputs the scan data to the console:



The screenshot shows a terminal window titled 'csk@ubuntu: /media/data-60/git\_pool/rpnew/applet/rplidar/s...'. The terminal displays a series of scan data points in the following format: 'theta: [value] Dist: [value] Q: [value]'. The data points are as follows:

theta	Dist	Q
324.41	00525.00	8
327.98	00000.00	0
329.64	00278.25	8
330.78	00276.25	8
332.16	00000.00	0
333.33	00305.75	8
334.72	00314.25	8
336.16	00315.75	8
337.55	00329.25	8
338.94	00330.75	8
340.34	00328.00	8
341.73	00329.25	8
343.36	00000.00	0
343.47	00397.00	8
346.16	00000.00	0
347.55	00000.00	0
349.22	00241.50	8
352.20	00208.00	8
352.02	00214.50	14
353.42	00217.50	14
354.81	00219.50	11
356.47	00222.00	11
357.61	00227.50	8
358.72	00000.00	0
0.22	00000.00	0
1.59	00000.00	0
3.00	00000.00	0
4.41	00000.00	0

### Steps:

1) Connect RPLIDAR to pc using provided USB cable. (USB to serial chip embedded in RPLIDAR development kit)

2) Start application use the command:

#### ● Windows

`ultra_simple <com_port>`

Note: if the com number is larger than 9, e.g. com11, then you should start application use command like this: `ultra_grabber \\.\com11`



- **Linux**

```
ultra_simple <tty device>
```

e.g. `ultra_simple /dev/ttyUSB0`.

- **Linux**

ultra\_simple <usb tty device>

e.g. `ultra_simple /dev/tty.SLAB_USBtoUART`.

simple\_grabber

This demo application will get RPLIDAR's serial number, firmware version and healthy status. Then the demo app grabs two round of scan data and show the range data as histogram in the command line mode. User can print all scan data if needed.

[illegible]

### Steps:

1) Connect RPLIDAR to pc using provided USB cable. (USB to serial chip embedded in RPLIDAR development kit)

2) Start application use command: `simple_grabber <com number>`

- **Windows**

`simple_grabber <com_port>`

Note: if the com number is larger than 9, e.g. com11, then you should start application use command like this: `ultra_grabber \\.\com11`

- **Linux**

`simple_grabber <tty device>`

e.g. `ultra_simple /dev/ttyUSB0.`

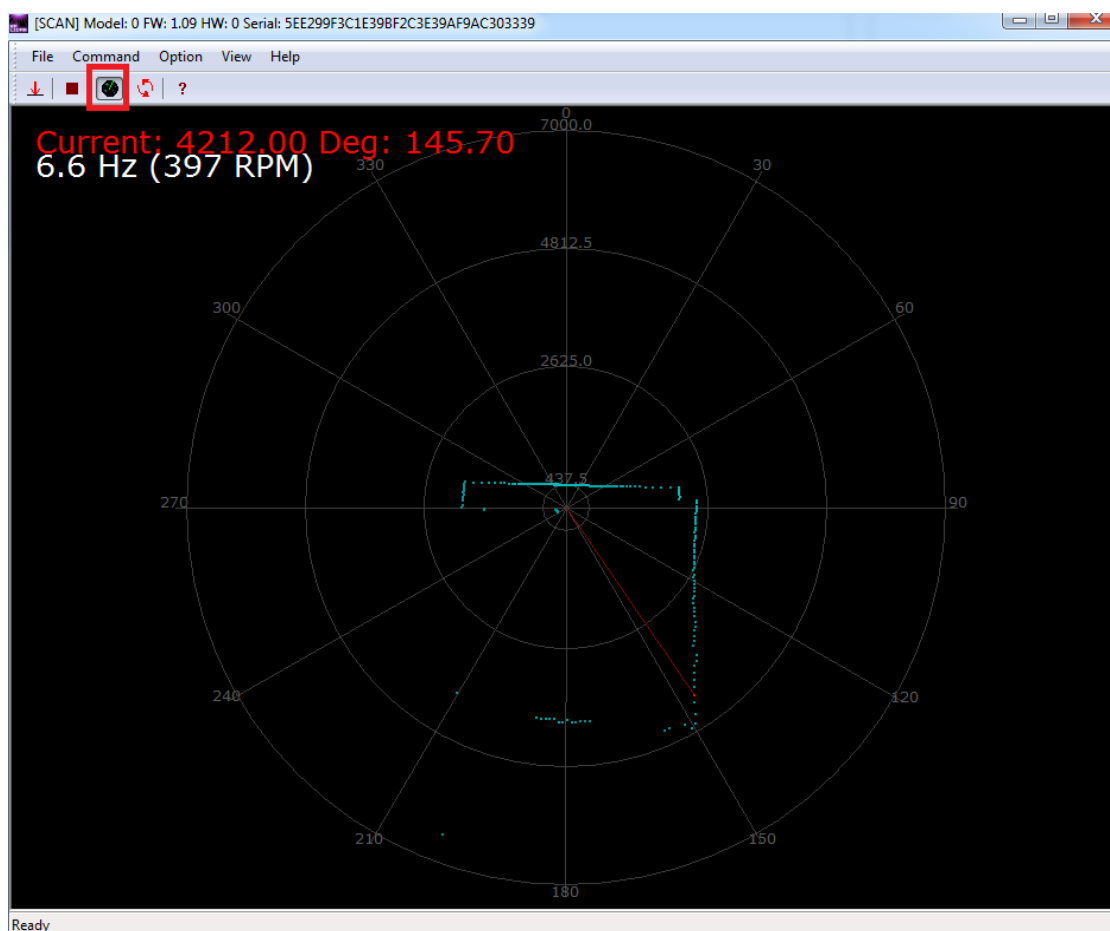
- **Linux**

`simple_grabber <usb tty device>`

e.g. `ultra_simple /dev/tty.SLAB_USBtoUART.`

`frame_grabber`

This demo application can show real-time laser scan data in the GUI with 0-360 degree environment range data. Note, this demo application only has win32 version.



### Steps:

- 1) Connect RPLIDAR to pc using provided USB cable. (USB to serial chip embedded in RPLIDAR development kit)
- 2) Choose correct com port number through com port selection dialog.
- 3) Press start scan button (marked in red in the figure above) to start.

### 3. SDK Usage and Development Guide

---

#### Assumption

This document assume developer has some knowledge about C++ development.

We strongly recommend developer understanding RPLIDAR's communication protocol and working mode before starting develop use RPLIDAR SDK.

#### SDK usage

RPLIDAR standard SDK provided with static library to facility develop intergrading SDK into their own project. It's also possible to build SDK as dynamic library by changing project configurations.

When develop with RPLIDAR SDK, you only need to include SDK's external header files (under sdk\include) into your source code and link your application with SDK's static library (rplidar\_driver.lib or rplidar\_driver.a).

For VS developer, you can also include SDK's VC project into your solution.

For Linux developer, please refer to simple\_grabber's Makefile.

#### Runtime consistency

For windows developer: the SDK static library uses VC10 MD C runtime library. If your project used different C runtime library may leads to compilation failure or unpredictable behavior. Then please change SDK settings accordingly.

## SDK Headers

- **rplidar.h**

Usually, you only need to include this file to get all functions of RPLIDAR SDK.

- **rplidar\_driver.h**

This header defines class RPLidarDriver. Please refer to demo applications to understand how to use it.

- **rplidar\_protocol.h**

This header defines low-level data structures and constants for RPLIDAR protocol.

- **rplidar\_cmd.h**

This header defines request/answer data structures and constants for RPLIDAR protocol.

- **rptypes.h**

This header defines platform-independent data structures and constants.

## SDK Initialization and Termination

User programs are required to create an RPLidarDriver instance before communicating with an RPLIDAR device. The following static function is used:

```
RPLidarDriver *RPLidarDriver::CreateDriver (_u32 drivertype)
```

Each RPLidarDriver instance is bind to only one RPLIDAR device at the same time. But user programs can freely allocate arbitrary number of RPLidarDriver instances and

make them communicates with multiple RPLIAR devices concurrently.

Once user programs finish operations, all previously created RPlidarDriver instances should be released explicitly using the following static function in order to free system memory.

```
RPlidarDriver::DisposeDriver(RPlidarDriver * drv)
```

## Connecting to an RPLIDAR

After creating an RPlidarDriver instance, the user program should invoke the connect() function firstly in order to open the serial port and make connection with the RPLIDAR device. All RPLIDAR operations require a connection has been established between the system and the RPLIDAR device.

```
u_result RPlidarDriver::connect(const char * port_path, _u32 baudrate, _u32 flag = 0)
```

The function returns RESULT\_OK for success operation.

Once the user program finishes operation, it can call the disconnect() function to close the connection and release the serial port device.

## Measurement Scan and Data Acquiring

The following functions are related to the measurement scan operation and help user programs to acquire the measurement data:

Function Name	Brief description
<b>startScan()</b>	Request the RPLIDAR core to start measurement scan operation and send out result data continuously
<b>stop()</b>	Request the RPLIDAR core to stop the measurement scan operation.
<b>grabScanData()</b>	Grab a complete 360-degrees' scan data sequence.

The user program should invoke the startScan() firstly to make the RPLIDAR core enter measurement scan operation. Once the rotation speed of RPLIDAR core becomes stable, RPLIDAR core will output the measurement scan data continuously.

The `startScan()` function will start a background worker thread to receive the measurement scan data sequence sent from RPLIDAR asynchronously. The received data sequence is stored in the driver's internal cache for the `grabScanData()` function to fetch.

User programs can use the `grabScanData()` function to retrieve the scan data sequence previously received and cached by the driver. This function always returns a latest and complete 360-degrees' measurement scan data sequence. After each `grabScanData()` call, the internal data cache will be cleared to ensure the `grabScanData()` won't get duplicated data.

In case a complete 360-degrees' scan sequence hasn't been available at the time when `grabScanData()` is called, the function will wait until a complete scan data is received by the driver or the given timeout duration is expired. User programs can tune this timeout value to meet different application requirements.

**Note: the `startScan()` and `stop()` functions don't control the scanning motor of the RPLIDAR directly. The host system should control the scanning motor to rotate or stop via the PWM pin.**

Please refer to the comments in the header files and the implementation of SDK demo applications for details.

## Retrieving other information of an RPLIDAR

The user program can retrieve other information of an RPLIDAR via the following functions. Please refer to the comments in the header files and the implementation of SDK demo applications for details.

Function Name	Brief description
<b>getHealth()</b>	Get the healthy status of an RPLIDAR
<b>getDeviceInfo()</b>	Retrieve the device information, e.g. serial number, firmware version etc, from an RPLIDAR
<b>getFrequency()</b>	Calculate an RPLIDAR's scanning speed from a complete scan sequence.



## 4. Revision History

---

Date	Content
2013-3-5	Initial draft
2014-1-25	<ol style="list-style-type: none"><li>1. Translated into English</li><li>2. Add Linux support</li></ol>
2014-3-8	<ol style="list-style-type: none"><li>1. Added descriptions of the ultra_simple app</li><li>2. Added descriptions of the major SDK functions</li></ol>
2014-7-25	<ol style="list-style-type: none"><li>1. Added descriptions of MacOS usage</li><li>2. Added descriptions of cross-compiling feature</li></ol>