YDLIDAR

# YDLIDAR SDK MANUAL

DOC#：01.13.000000

## CONTENTS

## YDLIDAR SDK COMMON INTERFACE DEFINITION

The common interfaces of YDLIDAR's driver class YDlidarDriver under Linux are as follows:

CHART 1 YDLIDAR SDK API

| Item | Platform | Interface function |
|---|---|---|
| Create an instance | Common | static void initDriver() |
| Get an instance | Common | static YDlidarDriver* singleton() |
| Destroy instance | Common | static void done() |
| Open serial port | Common | result_t connect(const char * port_path, uint32_t baudrate); |
| Close the serial port | Common | void disconnect(); |
| Start scanning | Common | result_t startScan(bool force = false, uint32_t timeout = DEFAULT_TIMEOUT) ; |
| Stop scanning | Common | result_t stop(); |
| Get status information | Common | result_t getHealth(device_health & health, uint32_t timeout = DEFAULT_TIMEOUT) |
| Get device information | Common | result_t getDeviceInfo(device_info & info, uint32_t timeout = DEFAULT_TIMEOUT); |
| Get a circle of point cloud data | Common | result_t grabScanData(node_info * nodebuffer, size_t & count, uint32_t timeout = DEFAULT_TIMEOUT) ; |
| Point cloud data angle regularization | Common | result_t ascendScanData(node_info * nodebuffer, size_t count); |
| Device restart | Common | result_t reset(uint32_t timeout = DEFAULT_TIMEOUT); |
| Get SDK version information | Common | static std::string getSDKVersion(); |
| Get sample rate | G4/G6 | result_t getSamplingRate(sampling_rate & rate, uint32_t timeout = DEFAULT_TIMEOUT); |
| Set sample rate | G4/G6 | result_t setSamplingRate(sampling_rate & rate, uint32_t timeout = DEFAULT_TIMEOUT); |
| Get scan frequency | G4/G6 | result_t getScanFrequency(scan_frequency & frequency, uint32_t timeout = DEFAULT_TIMEOUT); |
| Increasing 1Hz scanning frequency | G4/G6 | result_t setScanFrequencyAdd(scan_frequency & frequency, uint32_t timeout = DEFAULT_TIMEOUT); |

| | | |
|---|---|---|
| Reducing 1Hz scanning frequency | G4/G6 | result_t setScanFrequencyDis(scan_frequency & frequency, uint32_t timeout = DEFAULT_TIMEOUT); |
| Scanning 0.1Hz frequency increases | G4/G6 | result_t setScanFrequencyAddMic(scan_frequency & frequency, uint32_t timeout = DEFAULT_TIMEOUT); |
| Scanning 0.1Hz frequency decreases | G4/G6 | result_t setScanFrequencyDisMic(scan_frequency & frequency, uint32_t timeout = DEFAULT_TIMEOUT); |
| Get Lidar scanning direction | G4/G6 | result_t setRotationPositive(scan_rotation & rotation, uint32_t timeout = DEFAULT_TIMEOUT); |
| Set Lidar scanning direction | G4/G6 | result_t setRotationInversion(scan_rotation & rotation, uint32_t timeout = DEFAULT_TIMEOUT); |
| Low power mode enable | G4/G6 | result_t enableLowerPower(function_state & state, uint32_t timeout = DEFAULT_TIMEOUT); |
| Low power mode off | G4/G6 | result_t disableLowerPower(function_state & state, uint32_t timeout = DEFAULT_TIMEOUT); |
| Constant frequency mode enable | G4/G6 | result_t enableConstFreq(function_state & state, uint32_t timeout = DEFAULT_TIMEOUT); |
| Constant frequency mode is off | G4/G6 | result_t disableConstFreq(function_state & state, uint32_t timeout = DEFAULT_TIMEOUT); |
| Query power-down mode status | G4/G6 | const bool getHeartBeat(); |
| Set the power-down mode status | G4/G6 | void setHeartBeat(const bool enable); |
| Send online signal | G4/G6 | result_t sendHeartBeat(); |
| Starter motor | X4/S4 | result_t startMotor(); |
| Stop motor | X4/S4 | result_t stopMotor(); |
| Set the signal strength flag | S4 | void setIntensities(const bool isintensities); |

*Note: result_t is a macro definition of int.*

# YDLIDAR SDK COMMON INTERFACE DESCRIPTION

## Create an instance

```
static void initDriver()
```

The initDriver() static method creates a driver instance with no return value.

## Get the driver instance

```
static YDlidarDriver* singleton()
```

Singleton () gets the driver instance, the return value is the driver instance pointer.

## Destroy instance

```
static void done()
```

## Open serial port

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

Connect () is the serial port name and baud rate, and the baud rate should correspond to your Lidar model.

## Close serial port

```
void disconnect()
```

disconnect() closes the serial port and there is no return value.

## Get status information

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

Device_health is the device state structure. getHealth() parameter can be used as the state structure instance. The return value is 0, -1, and -2.

0 means the data is correct, -1 means data acquisition fails. -2 means data is timed out.

## Get device information

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

Device_info is the device information structure, and getDeviceInfo () can be used as an instance of the device information structure. The return value is 0, -1, and -2.

0 means the data is correct, -1 means data acquisition fails. -2 means data is timed out.

## Start scanning

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

startScan () does not need to pass parameters. The return value is 0, -1, and -2.

0 means scanning successfully, -1 means the scan command failed to be sent. -2 means the command is timed out.

## Stop scanning

```
result_t stop()
```

stop () does not need to pass parameters. The return value is 0, -1, and -2.

0 means stop scanning successfully, -1 means the stop command failed to be sent. -2 means the command is timed out.

## Grab a circle of Lidar data

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

Node_info is the Lidar data structure. GrabScanData() is used as the Lidar data structure instance and the number of Lidar data. The return value is 0, -1 and -2.

0 means the data is successfully obtained. -1 indicates that the data acquisition failed. -2 means the data acquisition is timeout.

## Point cloud data angle regularization

```
result_t ascendScanData(node_info * nodebuffer, size_t count)
```

Node_info is the Lidar data structure, count is the number of points of the point cloud returned, and ascendScanData() adjusts the data of the Lidar output angle greater than 360 degrees and less than 0 degrees to the range of 0-360. The return value is 0, -1 and -2.

0 means the data is successfully obtained. -1 indicates that the data acquisition failed. -2 means the data acquisition is timeout.

## Device restart

```
result_t reset(uint32_t timeout = DEFAULT_TIMEOUT)
```

Reset() does not need to pass parameters. When the return value is 0, the device resets successfully.

### Get SDK version information

```
static std::string getSDKVersion()
```

getSDKVersion() does not need to pass arguments, the return value is the SDK version number.

## OTHER INTERFACE FUNCTION DESCRIPTION

In addition to the general interface, different radars have interface functions that are specific to them. The specific information needs to be combined with the radar development manual to understand; this article will not elaborate.