

# YDLidar SDK API

V1.0.0

Generated by Doxygen 1.8.11



# Contents

<b>1</b>	<b>CYdLidar(YDLIDAR SDK API)</b>	<b>1</b>
<b>2</b>	<b>YDLIDAR DATASET</b>	<b>3</b>
<b>3</b>	<b>FlowChart</b>	<b>5</b>
<b>4</b>	<b>General FAQs</b>	<b>7</b>
<b>5</b>	<b>General FAQs_cn</b>	<b>9</b>
<b>6</b>	<b>Hardware FAQs</b>	<b>11</b>
<b>7</b>	<b>硬件FAQs:</b>	<b>13</b>
<b>8</b>	<b>FAQs</b>	<b>15</b>
<b>9</b>	<b>Software FAQs</b>	<b>17</b>
<b>10</b>	<b>软件FAQ</b>	<b>19</b>
<b>11</b>	<b>How to Build and Debug using VSCode</b>	<b>21</b>
<b>12</b>	<b>How to Build and Install</b>	<b>23</b>
<b>13</b>	<b>How to create a pull request</b>	<b>29</b>
<b>14</b>	<b>How to create a udev rules</b>	<b>31</b>
<b>15</b>	<b>Introduction</b>	<b>33</b>
<b>16</b>	<b>Github访问慢解决方案</b>	<b>35</b>

<b>17 Howto Guides</b>	<b>37</b>
<b>18 Quick Start Guides</b>	<b>39</b>
<b>19 Software Overview of YDLidar-SDK</b>	<b>41</b>
<b>20 YDLIDAR SDK Documents</b>	<b>43</b>
<b>21 Examining the Simple Lidar Tutorial</b>	<b>45</b>
<b>22 WritingLidarTutorial(c++)</b>	<b>47</b>
<b>23 WritingLidarTutorial(C)</b>	<b>53</b>
<b>24 WritingLidarTutorial(python)</b>	<b>59</b>
<b>25 YDLIDAR SDK Tutorials</b>	<b>63</b>
<b>26 YDLidar-SDK Communication Protocol</b>	<b>65</b>
<b>27 YDLIDAR SDK API for Developers</b>	<b>71</b>
<b>28 README</b>	<b>91</b>
<b>29 ETLidarDriver</b>	<b>95</b>
<b>30 YDIidarDriver</b>	<b>97</b>
<b>31 C API</b>	<b>99</b>
<b>32 Todo List</b>	<b>101</b>
<b>33 Namespace Index</b>	<b>107</b>
33.1 Namespace List . . . . .	107
<b>34 Hierarchical Index</b>	<b>109</b>
34.1 Class Hierarchy . . . . .	109
<b>35 Class Index</b>	<b>111</b>
35.1 Class List . . . . .	111

<b>36 File Index</b>	<b>113</b>
36.1 File List . . . . .	113
<b>37 Namespace Documentation</b>	<b>115</b>
37.1 etlidar_test Namespace Reference . . . . .	115
37.1.1 Variable Documentation . . . . .	115
37.1.1.1 laser . . . . .	115
37.1.1.2 r . . . . .	115
37.1.1.3 ret . . . . .	115
37.1.1.4 scan . . . . .	115
37.2 impl Namespace Reference . . . . .	116
37.2.1 Function Documentation . . . . .	116
37.2.1.1 getCurrentTime() . . . . .	116
37.2.1.2 getHDTimer() . . . . .	116
37.3 plot_tof_test Namespace Reference . . . . .	116
37.3.1 Function Documentation . . . . .	116
37.3.1.1 animate(num) . . . . .	116
37.3.2 Variable Documentation . . . . .	117
37.3.2.1 ani . . . . .	117
37.3.2.2 fig . . . . .	117
37.3.2.3 laser . . . . .	117
37.3.2.4 lidar_polar . . . . .	117
37.3.2.5 port . . . . .	117
37.3.2.6 ports . . . . .	117
37.3.2.7 ret . . . . .	117
37.3.2.8 RMAX . . . . .	117
37.3.2.9 scan . . . . .	117
37.4 plot_ydlidar_test Namespace Reference . . . . .	118
37.4.1 Function Documentation . . . . .	118
37.4.1.1 animate(num) . . . . .	118
37.4.2 Variable Documentation . . . . .	118

37.4.2.1	ani	118
37.4.2.2	fig	118
37.4.2.3	laser	118
37.4.2.4	lidar_polar	118
37.4.2.5	port	119
37.4.2.6	ports	119
37.4.2.7	ret	119
37.4.2.8	RMAX	119
37.4.2.9	scan	119
37.5	pytest Namespace Reference	119
37.6	serial Namespace Reference	119
37.7	serial::Serial Namespace Reference	119
37.8	setup Namespace Reference	120
37.8.1	Variable Documentation	120
37.8.1.1	YDLIDAR_SDK_BRANCH	120
37.8.1.2	YDLIDAR_SDK_REPO	120
37.9	test Namespace Reference	120
37.9.1	Variable Documentation	120
37.9.1.1	laser	120
37.9.1.2	port	120
37.9.1.3	ports	121
37.9.1.4	r	121
37.9.1.5	ret	121
37.9.1.6	scan	121
37.10	tof_test Namespace Reference	121
37.10.1	Variable Documentation	121
37.10.1.1	laser	121
37.10.1.2	port	121
37.10.1.3	ports	121
37.10.1.4	r	122

37.10.1.5 ret . . . . .	122
37.10.1.6 scan . . . . .	122
37.11 ydlidar Namespace Reference . . . . .	122
37.11.1 Detailed Description . . . . .	122
37.11.2 Function Documentation . . . . .	124
37.11.2.1 lidarPortList() . . . . .	124
37.11.2.2 os_init() . . . . .	124
37.11.2.3 os_isOk() . . . . .	125
37.11.2.4 os_shutdown() . . . . .	125
37.12 ydlidar::core Namespace Reference . . . . .	125
37.12.1 Detailed Description . . . . .	125
37.13 ydlidar::core::base Namespace Reference . . . . .	125
37.13.1 Function Documentation . . . . .	126
37.13.1.1 fileExists(const std::string &filename) . . . . .	126
37.13.1.2 init() . . . . .	126
37.13.1.3 ok() . . . . .	126
37.13.1.4 shutdown() . . . . .	127
37.14 ydlidar::core::common Namespace Reference . . . . .	127
37.14.1 Detailed Description . . . . .	128
37.14.2 Function Documentation . . . . .	128
37.14.2.1 ConvertLidarToUserSmample(int model, int rate) . . . . .	128
37.14.2.2 ConvertUserToLidarSmample(int model, int m_SampleRate, int defaultRate) . . . . .	129
37.14.2.3 hasIntensity(int model) . . . . .	130
37.14.2.4 hasSampleRate(int model) . . . . .	130
37.14.2.5 hasScanFrequencyCtrl(int model) . . . . .	130
37.14.2.6 hasZeroAngle(int model) . . . . .	131
37.14.2.7 isNetTOFLidar(int type) . . . . .	131
37.14.2.8 isNetTOFLidarByModel(int model) . . . . .	131
37.14.2.9 isOctaveLidar(int model) . . . . .	132
37.14.2.10 sOldVersionTOFLidar(int model, int Major, int Minor) . . . . .	132

37.14.2.11	isSerialNumbValid(const LaserDebug &info)	132
37.14.2.12	SupportLidar(int model)	133
37.14.2.13	SupportMotorCtrl(int model)	133
37.14.2.14	SupportScanFrequency(int model, double frequency)	133
37.14.2.15	sTOFLidar(int type)	134
37.14.2.16	sTOFLidarByModel(int model)	134
37.14.2.17	sTriangleLidar(int type)	134
37.14.2.18	sV1Protocol(uint8_t protocol)	134
37.14.2.19	sValidSampleRate(std::map< int, int > smap)	135
37.14.2.20	sValidValue(uint8_t value)	135
37.14.2.21	isVersionValid(const LaserDebug &info)	135
37.14.2.22	lidarModelDefaultSampleRate(int model)	136
37.14.2.23	lidarModelToString(int model)	136
37.14.2.24	ParseLaserDebugInfo(const LaserDebug &info, device_info &value)	136
37.14.2.25	parsePackageNode(const node_info &node, LaserDebug &info)	137
37.14.2.26	printfVersionInfo(const device_info &info, const std::string &port, int baudrate)	137
37.14.2.27	split(const std::string &s, char delim)	137
37.15	ydldidar::core::math Namespace Reference	138
37.15.1	Function Documentation	138
37.15.1.1	find_min_max_delta(double from, double left_limit, double right_limit, double &result_min_delta, double &result_max_delta)	138
37.15.1.2	from_degrees(double degrees)	139
37.15.1.3	normalize_angle(double angle)	139
37.15.1.4	normalize_angle_positive(double angle)	139
37.15.1.5	normalize_angle_positive_from_degree(double angle)	139
37.15.1.6	shortest_angular_distance(double from, double to)	140
37.15.1.7	shortest_angular_distance_with_limits(double from, double to, double left_limit, double right_limit, double &shortest_angle)	140
37.15.1.8	to_degrees(double radians)	140
37.15.1.9	two_pi_complement(double angle)	140
37.16	ydldidar::core::network Namespace Reference	141



37.17ydlidar::core::serial Namespace Reference . . . . .	141
37.17.1 Enumeration Type Documentation . . . . .	142
37.17.1.1 <code>bytesize_t</code> . . . . .	142
37.17.1.2 <code>flowcontrol_t</code> . . . . .	142
37.17.1.3 <code>parity_t</code> . . . . .	142
37.17.1.4 <code>stopbits_t</code> . . . . .	142
37.17.2 Function Documentation . . . . .	143
37.17.2.1 <code>is_standardbaudrate(unsigned long baudrate, speed_t &amp;baud)</code> . . . . .	143
37.17.2.2 <code>list_ports()</code> . . . . .	143
37.17.2.3 <code>set_common_props(termios *tio)</code> . . . . .	143
37.17.2.4 <code>set_databits(termios *tio, serial::bytesize_t databits)</code> . . . . .	143
37.17.2.5 <code>set_flowcontrol(termios *tio, serial::flowcontrol_t flowcontrol)</code> . . . . .	143
37.17.2.6 <code>set_parity(termios *tio, serial::parity_t parity)</code> . . . . .	143
37.17.2.7 <code>set_stopbits(termios *tio, serial::stopbits_t stopbits)</code> . . . . .	143
37.17.2.8 <code>timespec_from_ms(const uint32_t millis)</code> . . . . .	143
37.18ydlidar_test Namespace Reference . . . . .	143
37.18.1 Variable Documentation . . . . .	144
37.18.1.1 <code>laser</code> . . . . .	144
37.18.1.2 <code>port</code> . . . . .	144
37.18.1.3 <code>ports</code> . . . . .	144
37.18.1.4 <code>r</code> . . . . .	144
37.18.1.5 <code>ret</code> . . . . .	144
37.18.1.6 <code>scan</code> . . . . .	144

<b>38 Class Documentation</b>	<b>145</b>
38.1 <code>_dataFrame</code> Struct Reference	145
38.1.1 Detailed Description	145
38.1.2 Member Data Documentation	145
38.1.2.1 <code>dataFormat</code>	145
38.1.2.2 <code>dataIndex</code>	146
38.1.2.3 <code>dataNum</code>	146
38.1.2.4 <code>deviceType</code>	146
38.1.2.5 <code>disScale</code>	146
38.1.2.6 <code>frameBuf</code>	146
38.1.2.7 <code>frameCrc</code>	146
38.1.2.8 <code>frameHead</code>	146
38.1.2.9 <code>frameIndex</code>	146
38.1.2.10 <code>frameType</code>	146
38.1.2.11 <code>headFrameFlag</code>	146
38.1.2.12 <code>startAngle</code>	147
38.1.2.13 <code>timestamp</code>	147
38.2 <code>_lidarConfig</code> Struct Reference	147
38.2.1 Detailed Description	148
38.2.2 Member Data Documentation	148
38.2.2.1 <code>correction_angle</code>	148
38.2.2.2 <code>correction_distance</code>	148
38.2.2.3 <code>dataRecvIp</code>	148
38.2.2.4 <code>dataRecvPort</code>	148
38.2.2.5 <code>deviceGatewayIp</code>	148
38.2.2.6 <code>deviceIp</code>	148
38.2.2.7 <code>deviceNetmask</code>	149
38.2.2.8 <code>dhcp_en</code>	149
38.2.2.9 <code>fov_end</code>	149
38.2.2.10 <code>fov_start</code>	149

38.2.2.11 laser_en . . . . .	149
38.2.2.12 laserScanFrequency . . . . .	149
38.2.2.13 motor_en . . . . .	149
38.2.2.14 motor_rpm . . . . .	149
38.2.2.15 trans_sel . . . . .	150
38.3 ydlidar::core::network::CActiveSocket Class Reference . . . . .	150
38.3.1 Detailed Description . . . . .	151
38.3.2 Constructor & Destructor Documentation . . . . .	151
38.3.2.1 CActiveSocket(CSocketType type=SocketTypeTcp) . . . . .	151
38.3.2.2 ~CActiveSocket() . . . . .	152
38.3.3 Member Function Documentation . . . . .	152
38.3.3.1 Open(const char *pAddr, uint16_t nPort) . . . . .	152
38.3.4 Friends And Related Function Documentation . . . . .	152
38.3.4.1 CPassiveSocket . . . . .	152
38.4 ydlidar::core::common::ChannelDevice Class Reference . . . . .	152
38.4.1 Detailed Description . . . . .	154
38.4.2 Constructor & Destructor Documentation . . . . .	154
38.4.2.1 ChannelDevice() . . . . .	154
38.4.2.2 ~ChannelDevice() . . . . .	154
38.4.3 Member Function Documentation . . . . .	154
38.4.3.1 available()=0 . . . . .	154
38.4.3.2 bindport(const char *, uint32_t) . . . . .	154
38.4.3.3 closePort()=0 . . . . .	154
38.4.3.4 DescribeError() . . . . .	155
38.4.3.5 flush()=0 . . . . .	155
38.4.3.6 getByteTime() . . . . .	155
38.4.3.7 isOpen()=0 . . . . .	155
38.4.3.8 open()=0 . . . . .	155
38.4.3.9 readData(uint8_t *data, size_t size)=0 . . . . .	156
38.4.3.10 readSize(size_t size=1)=0 . . . . .	156

38.4.3.11 setDTR(bool level=true) . . . . .	156
38.4.3.12 waitfordata(size_t data_count, uint32_t timeout=-1, size_t *returned_size=NULL)=0	157
38.4.3.13 writeData(const uint8_t *data, size_t size)=0 . . . . .	157
38.5 setup.CMakeBuild Class Reference . . . . .	158
38.5.1 Detailed Description . . . . .	159
38.5.2 Member Function Documentation . . . . .	159
38.5.2.1 build_extension(self, ext) . . . . .	159
38.5.2.2 clone(self) . . . . .	159
38.5.2.3 run(self) . . . . .	159
38.5.3 Member Data Documentation . . . . .	159
38.5.3.1 author . . . . .	159
38.5.3.2 author_email . . . . .	159
38.5.3.3 cmdclass . . . . .	159
38.5.3.4 description . . . . .	159
38.5.3.5 ext_modules . . . . .	160
38.5.3.6 long_description . . . . .	160
38.5.3.7 name . . . . .	160
38.5.3.8 url . . . . .	160
38.5.3.9 version . . . . .	160
38.5.3.10 zip_safe . . . . .	160
38.6 setup.CMakeExtension Class Reference . . . . .	160
38.6.1 Detailed Description . . . . .	161
38.6.2 Constructor & Destructor Documentation . . . . .	161
38.6.2.1 __init__(self, name, sourcedir="") . . . . .	161
38.6.3 Member Data Documentation . . . . .	161
38.6.3.1 sourcedir . . . . .	161
38.7 cmd_packet Struct Reference . . . . .	162
38.7.1 Detailed Description . . . . .	162
38.7.2 Member Data Documentation . . . . .	162
38.7.2.1 cmd_flag . . . . .	162

38.7.2.2	<a href="#">data</a>	162
38.7.2.3	<a href="#">size</a>	162
38.7.2.4	<a href="#">syncByte</a>	162
38.8	<a href="#">ydlidar::core::network::CPassiveSocket Class Reference</a>	163
38.8.1	<a href="#">Detailed Description</a>	164
38.8.2	<a href="#">Constructor &amp; Destructor Documentation</a>	164
38.8.2.1	<a href="#">CPassiveSocket(CSocketType type=SocketTypeTcp)</a>	164
38.8.2.2	<a href="#">~CPassiveSocket()</a>	164
38.8.3	<a href="#">Member Function Documentation</a>	164
38.8.3.1	<a href="#">Accept(void)</a>	164
38.8.3.2	<a href="#">BindMulticast(const char *pInterface, const char *pGroup, uint16_t nPort)</a>	164
38.8.3.3	<a href="#">Listen(const char *pAddr, uint16_t nPort, int32_t nConnectionBacklog=30000)</a>	165
38.8.3.4	<a href="#">Send(const uint8_t *pBuf, size_t bytesToSend)</a>	165
38.9	<a href="#">ydlidar::core::network::CSimpleSocket Class Reference</a>	166
38.9.1	<a href="#">Detailed Description</a>	170
38.9.2	<a href="#">Member Enumeration Documentation</a>	170
38.9.2.1	<a href="#">CShutdownMode</a>	170
38.9.2.2	<a href="#">CSocketError</a>	171
38.9.2.3	<a href="#">CSocketType</a>	171
38.9.3	<a href="#">Constructor &amp; Destructor Documentation</a>	171
38.9.3.1	<a href="#">CSimpleSocket(CSocketType type=SocketTypeTcp)</a>	171
38.9.3.2	<a href="#">CSimpleSocket(CSimpleSocket &amp;socket)</a>	172
38.9.3.3	<a href="#">~CSimpleSocket()</a>	172
38.9.4	<a href="#">Member Function Documentation</a>	172
38.9.4.1	<a href="#">available()</a>	172
38.9.4.2	<a href="#">BindInterface(const char *pInterface)</a>	172
38.9.4.3	<a href="#">bindport(const char *, uint32_t)</a>	172
38.9.4.4	<a href="#">Close(void)</a>	173
38.9.4.5	<a href="#">closePort()</a>	173
38.9.4.6	<a href="#">DescribeError(CSocketError err)</a>	173

38.9.4.7 DescribeError()	173
38.9.4.8 DisableNagleAlgoritm()	173
38.9.4.9 EnableNagleAlgoritm()	174
38.9.4.10 flush()	174
38.9.4.11 Flush()	174
38.9.4.12 GetBytesReceived(void)	174
38.9.4.13 GetBytesSent(void)	174
38.9.4.14 GetClientAddr()	175
38.9.4.15 GetClientPort()	175
38.9.4.16 GetConnectTimeoutSec(void)	175
38.9.4.17 GetConnectTimeoutUsec(void)	175
38.9.4.18 GetData(void)	175
38.9.4.19 GetMulticast()	176
38.9.4.20 GetReceiveTimeoutSec(void)	176
38.9.4.21 GetReceiveTimeoutUsec(void)	176
38.9.4.22 GetReceiveWindowSize()	176
38.9.4.23 GetSendTimeoutSec(void)	177
38.9.4.24 GetSendTimeoutUsec(void)	177
38.9.4.25 GetSendWindowSize()	177
38.9.4.26 GetServerAddr()	177
38.9.4.27 GetServerPort()	178
38.9.4.28 GetSocketDescriptor()	178
38.9.4.29 GetSocketDscp(void)	178
38.9.4.30 GetSocketError(void)	178
38.9.4.31 GetSocketType()	179
38.9.4.32 GetTotalTimeMs()	179
38.9.4.33 GetTotalTimeUsec()	179
38.9.4.34 Initialize(void)	179
38.9.4.35 IsNonblocking(void)	179
38.9.4.36 isOpen()	180

38.9.4.37 IsSocketValid(void) . . . . .	180
38.9.4.38 Open(const char *pAddr, uint16_t nPort) . . . . .	180
38.9.4.39 open() . . . . .	180
38.9.4.40 readData(uint8_t *data, size_t size) . . . . .	180
38.9.4.41 readSize(size_t size=1) . . . . .	181
38.9.4.42 Receive(int32_t nMaxBytes=1, uint8_t *pBuffer=0) . . . . .	181
38.9.4.43 Select(void) . . . . .	182
38.9.4.44 Select(int32_t nTimeoutSec, int32_t nTimeoutUsec) . . . . .	182
38.9.4.45 Send(const uint8_t *pBuf, size_t bytesToSend) . . . . .	182
38.9.4.46 Send(const struct iovec *sendVector, int32_t nNumItems) . . . . .	183
38.9.4.47 SendFile(int32_t nOutFd, int32_t nInFd, off_t *pOffset, int32_t nCount) . . . . .	183
38.9.4.48 SetBlocking(void) . . . . .	183
38.9.4.49 SetConnectTimeout(int32_t nConnectTimeoutSec, int32_t nConnectTimeoutUsec=0) . . . . .	184
38.9.4.50 SetMulticast(bool bEnable, uint8_t multicastTTL=1) . . . . .	184
38.9.4.51 SetNonblocking(void) . . . . .	184
38.9.4.52 SetOptionLinger(bool bEnable, uint16_t nTime) . . . . .	184
38.9.4.53 SetOptionReuseAddr() . . . . .	185
38.9.4.54 SetReceiveTimeout(int32_t nRecvTimeoutSec, int32_t nRecvTimeoutUsec=0) . . . . .	185
38.9.4.55 SetReceiveWindowSize(uint32_t nWindowSize) . . . . .	185
38.9.4.56 SetSendTimeout(int32_t nSendTimeoutSec, int32_t nSendTimeoutUsec=0) . . . . .	186
38.9.4.57 SetSendWindowSize(uint32_t nWindowSize) . . . . .	186
38.9.4.58 SetSocketDscp(int nDscp) . . . . .	186
38.9.4.59 SetSocketError(CSimpleSocket::CSocketError error) . . . . .	186
38.9.4.60 SetSocketHandle(SOCKET socket) . . . . .	187
38.9.4.61 SetSocketType(const CSocketType &type) . . . . .	187
38.9.4.62 Shutdown(CShutdownMode nShutdown) . . . . .	187
38.9.4.63 TranslateSocketError(void) . . . . .	187
38.9.4.64 WaitForData(size_t data_count, uint32_t timeout, size_t *returned_size) . . . . .	187
38.9.4.65 waitfordata(size_t data_count, uint32_t timeout=-1, size_t *returned_size=NULL) . . . . .	187
38.9.4.66 writeData(const uint8_t *data, size_t size) . . . . .	188

38.9.4.67 WSACleanUp()	188
38.9.5 Member Data Documentation	188
38.9.5.1 m_addr	188
38.9.5.2 m_bIsBlocking	188
38.9.5.3 m_bIsMulticast	189
38.9.5.4 m_errorFds	189
38.9.5.5 m_nBufferSize	189
38.9.5.6 m_nBytesReceived	189
38.9.5.7 m_nBytesSent	189
38.9.5.8 m_nFlags	189
38.9.5.9 m_nSocketDomain	189
38.9.5.10 m_nSocketType	189
38.9.5.11 m_open	190
38.9.5.12 m_pBuffer	190
38.9.5.13 m_port	190
38.9.5.14 m_readFds	190
38.9.5.15 m_socket	190
38.9.5.16 m_socketErrno	190
38.9.5.17 m_stClientSockaddr	190
38.9.5.18 m_stConnectTimeout	190
38.9.5.19 m_stLinger	191
38.9.5.20 m_stMulticastGroup	191
38.9.5.21 m_stRecvTimeout	191
38.9.5.22 m_stSendTimeout	191
38.9.5.23 m_stServerSockaddr	191
38.9.5.24 m_timer	191
38.9.5.25 m_writeFds	191
38.10CStatTimer Class Reference	192
38.10.1 Detailed Description	192
38.10.2 Constructor & Destructor Documentation	192



38.10.2.1 CStatTimer()	192
38.10.2.2 ~CStatTimer()	192
38.10.3 Member Function Documentation	192
38.10.3.1 GetCurrentTime()	192
38.10.3.2 GetEndTime()	193
38.10.3.3 GetMicroSeconds()	193
38.10.3.4 GetMilliSeconds()	193
38.10.3.5 GetSeconds()	193
38.10.3.6 GetStartTime()	193
38.10.3.7 Initialize()	193
38.10.3.8 SetEndTime()	193
38.10.3.9 SetStartTime()	193
38.11 CYdLidar Class Reference	193
38.11.1 Detailed Description	194
38.11.2 Constructor & Destructor Documentation	205
38.11.2.1 CYdLidar()	205
38.11.2.2 ~CYdLidar()	205
38.11.3 Member Function Documentation	205
38.11.3.1 DescribeError() const	205
38.11.3.2 disconnecting()	206
38.11.3.3 doProcessSimple(LaserScan &outscan)	206
38.11.3.4 getlidaropt(int optname, void *optval, int optlen)	206
38.11.3.5 GetLidarVersion(LidarVersion &version)	208
38.11.3.6 initialize()	208
38.11.3.7 setlidaropt(int optname, const void *optval, int optlen)	208
38.11.3.8 turnOff()	210
38.11.3.9 turnOn()	210
38.12 dataframe Class Reference	211
38.12.1 Detailed Description	211
38.13 device_health Struct Reference	211

38.13.1 Detailed Description . . . . .	211
38.13.2 Member Data Documentation . . . . .	211
38.13.2.1 error_code . . . . .	211
38.13.2.2 status . . . . .	212
38.14 device_info Struct Reference . . . . .	212
38.14.1 Detailed Description . . . . .	212
38.14.2 Member Data Documentation . . . . .	212
38.14.2.1 firmware_version . . . . .	212
38.14.2.2 hardware_version . . . . .	212
38.14.2.3 model . . . . .	213
38.14.2.4 serialnum . . . . .	213
38.15 ydlidar::core::common::DriverInterface Class Reference . . . . .	213
38.15.1 Detailed Description . . . . .	217
38.15.2 Member Enumeration Documentation . . . . .	217
38.15.2.1 anonymous enum . . . . .	217
38.15.2.2 anonymous enum . . . . .	218
38.15.2.3 anonymous enum . . . . .	218
38.15.3 Constructor & Destructor Documentation . . . . .	218
38.15.3.1 DriverInterface() . . . . .	218
38.15.3.2 ~DriverInterface() . . . . .	218
38.15.4 Member Function Documentation . . . . .	218
38.15.4.1 connect(const char *port_path, uint32_t baudrate=8000)=0 . . . . .	218
38.15.4.2 DescribeError(bool isTCP=true)=0 . . . . .	219
38.15.4.3 disconnect()=0 . . . . .	219
38.15.4.4 getDeviceInfo(device_info &info, uint32_t timeout=DEFAULT_TIMEOUT)=0 . . . . .	219
38.15.4.5 getFinishedScanCfg() const . . . . .	220
38.15.4.6 getHealth(device_health &health, uint32_t timeout=DEFAULT_TIMEOUT)=0 . . . . .	220
38.15.4.7 getLidarType() const . . . . .	220
38.15.4.8 getPointTime() const . . . . .	221
38.15.4.9 getSamplingRate(sampling_rate &rate, uint32_t timeout=DEFAULT_TIMEOUT)=0 . . . . .	221

38.15.4.10	getScanFrequency(scan_frequency &frequency, uint32_t timeout=DEFAULT_← TIMEOUT)=0 . . . . .	221
38.15.4.11	getSDKVersion()=0 . . . . .	222
38.15.4.12	getSingleChannel() const . . . . .	222
38.15.4.13	getSupportMotorDtrCtrl() const . . . . .	222
38.15.4.14	getZeroOffsetAngle(offset_angle &angle, uint32_t timeout=DEFAULT_TIMEO← UT)=0 . . . . .	222
38.15.4.15	grabScanData(node_info *nodebuffer, size_t &count, uint32_t timeout=DEFAU← LT_TIMEOUT)=0 . . . . .	223
38.15.4.16	isconnected() const =0 . . . . .	223
38.15.4.17	isscanning() const =0 . . . . .	224
38.15.4.18	setAutoReconnect(const bool &enable)=0 . . . . .	224
38.15.4.19	setIntensities(const bool &isintensities)=0 . . . . .	224
38.15.4.20	setLidarType(int v) . . . . .	224
38.15.4.21	setPointTime(uint32_t v) . . . . .	224
38.15.4.22	setSamplingRate(sampling_rate &rate, uint32_t timeout=DEFAULT_TIMEOUT)=0	225
38.15.4.23	setScanFrequencyAdd(scan_frequency &frequency, uint32_t timeout=DEFAU← LT_TIMEOUT)=0 . . . . .	225
38.15.4.24	setScanFrequencyAddMic(scan_frequency &frequency, uint32_t timeout=DEF← AULT_TIMEOUT)=0 . . . . .	226
38.15.4.25	setScanFrequencyDis(scan_frequency &frequency, uint32_t timeout=DEFAUL← T_TIMEOUT)=0 . . . . .	226
38.15.4.26	setScanFrequencyDisMic(scan_frequency &frequency, uint32_t timeout=DEFA← ULT_TIMEOUT)=0 . . . . .	227
38.15.4.27	setSingleChannel(bool v) . . . . .	227
38.15.4.28	setSupportMotorDtrCtrl(bool v) . . . . .	227
38.15.4.29	startScan(bool force=false, uint32_t timeout=DEFAULT_TIMEOUT)=0 . . . . .	227
38.15.4.30	stop()=0 . . . . .	228
38.15.5	Member Data Documentation . . . . .	228
38.15.5.1	_cmd_lock . . . . .	228
38.15.5.2	_dataEvent . . . . .	228
38.15.5.3	_lock . . . . .	229
38.15.5.4	_thread . . . . .	229

38.15.5.5 isAutoconnting . . . . .	229
38.15.5.6 isAutoReconnect . . . . .	229
38.15.5.7 m_baudrate . . . . .	229
38.15.5.8 m_config . . . . .	229
38.15.5.9 m_intensities . . . . .	229
38.15.5.10m_isConnected . . . . .	229
38.15.5.11m_isScanning . . . . .	230
38.15.5.12m_LidarType . . . . .	230
38.15.5.13m_PointTime . . . . .	230
38.15.5.14m_SingleChannel . . . . .	231
38.15.5.15m_SupportMotorDtrCtrl . . . . .	231
38.15.5.16package_Sample_Index . . . . .	232
38.15.5.17retryCount . . . . .	232
38.15.5.18scan_node_buf . . . . .	232
38.15.5.19scan_node_count . . . . .	232
38.15.5.20serial_port . . . . .	232
38.16ydliar::ETLidarDriver Class Reference . . . . .	233
38.16.1 Detailed Description . . . . .	235
38.16.2 Constructor & Destructor Documentation . . . . .	235
38.16.2.1 ETLidarDriver() . . . . .	235
38.16.2.2 ~ETLidarDriver() . . . . .	235
38.16.3 Member Function Documentation . . . . .	235
38.16.3.1 connect(const char *port_path, uint32_t baudrate=8000) . . . . .	235
38.16.3.2 DescribeError(bool isTCP=true) . . . . .	236
38.16.3.3 disconnect() . . . . .	236
38.16.3.4 getDeviceInfo(device_info &info, uint32_t timeout=DEFAULT_TIMEOUT) . . . . .	236
38.16.3.5 getFinishedScanCfg() . . . . .	237
38.16.3.6 getHealth(device_health &health, uint32_t timeout=DEFAULT_TIMEOUT) . . . . .	237
38.16.3.7 getSamplingRate(sampling_rate &rate, uint32_t timeout=DEFAULT_TIMEOUT) . . . . .	238
38.16.3.8 getScanCfg(lidarConfig &config, const std::string &ip_address="") . . . . .	238

38.16.3.9 getScanFrequency(scan_frequency &frequency, uint32_t timeout=DEFAULT_↵ TIMEOUT) . . . . .	238
38.16.3.10getSDKVersion() . . . . .	239
38.16.3.11getZeroOffsetAngle(offset_angle &angle, uint32_t timeout=DEFAULT_TIMEO↵ UT) . . . . .	239
38.16.3.12grabScanData(node_info *nodebuffer, size_t &count, uint32_t timeout=DEFAU↵ LT_TIMEOUT) . . . . .	240
38.16.3.13sconnected() const . . . . .	240
38.16.3.14sscanning() const . . . . .	241
38.16.3.15setAutoReconnect(const bool &enable) . . . . .	241
38.16.3.16setIntensities(const bool &isintensities) . . . . .	241
38.16.3.17setSamplingRate(sampling_rate &rate, uint32_t timeout=DEFAULT_TIMEOUT) . . . . .	242
38.16.3.18setScanFrequencyAdd(scan_frequency &frequency, uint32_t timeout=DEFAU↵ LT_TIMEOUT) . . . . .	242
38.16.3.19setScanFrequencyAddMic(scan_frequency &frequency, uint32_t timeout=DEF↵ AULT_TIMEOUT) . . . . .	243
38.16.3.20setScanFrequencyDis(scan_frequency &frequency, uint32_t timeout=DEFAUL↵ T_TIMEOUT) . . . . .	243
38.16.3.21setScanFrequencyDisMic(scan_frequency &frequency, uint32_t timeout=DEFA↵ ULT_TIMEOUT) . . . . .	244
38.16.3.22startScan(bool force=false, uint32_t timeout=DEFAULT_TIMEOUT) . . . . .	244
38.16.3.23stop() . . . . .	245
38.16.3.24updateScanCfg(const lidarConfig &config) . . . . .	245
38.17ydlidar::core::base::Event Class Reference . . . . .	246
38.17.1 Detailed Description . . . . .	246
38.17.2 Member Enumeration Documentation . . . . .	246
38.17.2.1 anonymous enum . . . . .	246
38.17.3 Constructor & Destructor Documentation . . . . .	247
38.17.3.1 Event(bool isAutoReset=true, bool isSignal=false) . . . . .	247
38.17.3.2 ~Event() . . . . .	247
38.17.4 Member Function Documentation . . . . .	247
38.17.4.1 release() . . . . .	247
38.17.4.2 set(bool isSignal=true) . . . . .	247

38.17.4.3 wait(unsigned long timeout=0xFFFFFFFF) . . . . .	247
38.17.5 Member Data Documentation . . . . .	247
38.17.5.1 _cond_cattr . . . . .	247
38.17.5.2 _cond_locker . . . . .	247
38.17.5.3 _cond_var . . . . .	247
38.17.5.4 _is_signalled . . . . .	247
38.17.5.5 _isAutoReset . . . . .	248
38.18function_state Struct Reference . . . . .	248
38.18.1 Detailed Description . . . . .	248
38.18.2 Member Data Documentation . . . . .	248
38.18.2.1 state . . . . .	248
38.19LaserConfig Struct Reference . . . . .	248
38.19.1 Detailed Description . . . . .	249
38.19.2 Member Data Documentation . . . . .	249
38.19.2.1 angle_increment . . . . .	249
38.19.2.2 max_angle . . . . .	249
38.19.2.3 max_range . . . . .	250
38.19.2.4 min_angle . . . . .	250
38.19.2.5 min_range . . . . .	250
38.19.2.6 scan_time . . . . .	250
38.19.2.7 time_increment . . . . .	250
38.20LaserDebug Struct Reference . . . . .	250
38.20.1 Detailed Description . . . . .	251
38.20.2 Member Data Documentation . . . . .	251
38.20.2.1 MaxDebugIndex . . . . .	251
38.20.2.2 W1F6GNoise_W1F5SNoise_W1F4MotorCtl_W4F0SnYear . . . . .	251
38.20.2.3 W2F5Output2K4K5K_W5F0Date . . . . .	251
38.20.2.4 W3F4BoradHardVer_W4F0Moth . . . . .	251
38.20.2.5 W3F4CusHardVer_W4F0CusSoftVer . . . . .	251
38.20.2.6 W3F4CusMajor_W4F0CusMinor . . . . .	252

38.20.2.7 W3F4HardwareVer_W4F0FirewareMajor . . . . .	252
38.20.2.8 W4F3Model_W3F0DebugInfTranVer . . . . .	252
38.20.2.9 W7F0FirewareMinor . . . . .	252
38.20.2.10W7F0Health . . . . .	252
38.20.2.11W7F0LaserCurrent . . . . .	252
38.20.2.12W7F0SnNumH . . . . .	252
38.20.2.13W7F0SnNumL . . . . .	252
38.21LaserFan Struct Reference . . . . .	253
38.21.1 Detailed Description . . . . .	253
38.21.2 Member Data Documentation . . . . .	254
38.21.2.1 config . . . . .	254
38.21.2.2 npoints . . . . .	254
38.21.2.3 points . . . . .	254
38.21.2.4 stamp . . . . .	254
38.22LaserPoint Struct Reference . . . . .	255
38.22.1 Detailed Description . . . . .	255
38.22.2 Member Data Documentation . . . . .	255
38.22.2.1 angle . . . . .	255
38.22.2.2 intensity . . . . .	255
38.22.2.3 range . . . . .	255
38.23LaserScan Struct Reference . . . . .	256
38.23.1 Detailed Description . . . . .	256
38.23.2 Member Data Documentation . . . . .	257
38.23.2.1 config . . . . .	257
38.23.2.2 points . . . . .	257
38.23.2.3 stamp . . . . .	257
38.24lidar_ans_header Struct Reference . . . . .	258
38.24.1 Detailed Description . . . . .	258
38.24.2 Member Data Documentation . . . . .	258
38.24.2.1 size . . . . .	258

38.24.2.2 subType . . . . .	258
38.24.2.3 syncByte1 . . . . .	258
38.24.2.4 syncByte2 . . . . .	258
38.24.2.5 type . . . . .	258
38.25lidarConfig Class Reference . . . . .	259
38.25.1 Detailed Description . . . . .	259
38.26LidarPort Struct Reference . . . . .	259
38.26.1 Detailed Description . . . . .	259
38.26.2 Member Data Documentation . . . . .	260
38.26.2.1 port . . . . .	260
38.27LidarTest Class Reference . . . . .	260
38.27.1 Detailed Description . . . . .	261
38.27.2 Constructor & Destructor Documentation . . . . .	261
38.27.2.1 LidarTest() . . . . .	261
38.27.2.2 ~LidarTest() . . . . .	261
38.27.3 Member Function Documentation . . . . .	261
38.27.3.1 SetUp() . . . . .	261
38.27.3.2 TearDown() . . . . .	261
38.27.4 Member Data Documentation . . . . .	261
38.27.4.1 m_lidar . . . . .	261
38.28LidarVersion Struct Reference . . . . .	261
38.28.1 Detailed Description . . . . .	262
38.28.2 Member Data Documentation . . . . .	262
38.28.2.1 hardware . . . . .	262
38.28.2.2 sn . . . . .	262
38.28.2.3 soft_major . . . . .	262
38.28.2.4 soft_minor . . . . .	262
38.28.2.5 soft_patch . . . . .	262
38.29ydlidar::core::base::Locker Class Reference . . . . .	263
38.29.1 Detailed Description . . . . .	263



38.29.2 Member Enumeration Documentation . . . . .	263
38.29.2.1 LOCK_STATUS . . . . .	263
38.29.3 Constructor & Destructor Documentation . . . . .	263
38.29.3.1 Locker() . . . . .	263
38.29.3.2 ~Locker() . . . . .	264
38.29.4 Member Function Documentation . . . . .	264
38.29.4.1 getLockHandle() . . . . .	264
38.29.4.2 init() . . . . .	264
38.29.4.3 lock(unsigned long timeout=0xFFFFFFFF) . . . . .	264
38.29.4.4 release() . . . . .	264
38.29.4.5 unlock() . . . . .	264
38.29.5 Member Data Documentation . . . . .	264
38.29.5.1 _lock . . . . .	264
38.30 ydlidar::core::serial::MillisecondTimer Class Reference . . . . .	264
38.30.1 Detailed Description . . . . .	265
38.30.2 Constructor & Destructor Documentation . . . . .	265
38.30.2.1 MillisecondTimer(const uint32_t millis) . . . . .	265
38.30.3 Member Function Documentation . . . . .	265
38.30.3.1 remaining() . . . . .	265
38.31 node_info Struct Reference . . . . .	265
38.31.1 Detailed Description . . . . .	266
38.31.2 Member Data Documentation . . . . .	266
38.31.2.1 angle_q6_checkbit . . . . .	266
38.31.2.2 debugInfo . . . . .	266
38.31.2.3 distance_q2 . . . . .	266
38.31.2.4 error_package . . . . .	266
38.31.2.5 index . . . . .	266
38.31.2.6 scan_frequence . . . . .	266
38.31.2.7 stamp . . . . .	267
38.31.2.8 sync_flag . . . . .	267

38.31.2.9 sync_quality . . . . .	267
38.32node_package Struct Reference . . . . .	267
38.32.1 Detailed Description . . . . .	268
38.32.2 Member Data Documentation . . . . .	268
38.32.2.1 checksum . . . . .	268
38.32.2.2 nowPackageNum . . . . .	268
38.32.2.3 package_CT . . . . .	268
38.32.2.4 package_Head . . . . .	268
38.32.2.5 packageFirstSampleAngle . . . . .	269
38.32.2.6 packageLastSampleAngle . . . . .	269
38.32.2.7 packageSample . . . . .	269
38.33node_packages Struct Reference . . . . .	269
38.33.1 Detailed Description . . . . .	269
38.33.2 Member Data Documentation . . . . .	270
38.33.2.1 checksum . . . . .	270
38.33.2.2 nowPackageNum . . . . .	270
38.33.2.3 package_CT . . . . .	270
38.33.2.4 package_Head . . . . .	270
38.33.2.5 packageFirstSampleAngle . . . . .	270
38.33.2.6 packageLastSampleAngle . . . . .	270
38.33.2.7 packageSampleDistance . . . . .	270
38.34offset_angle Struct Reference . . . . .	271
38.34.1 Detailed Description . . . . .	271
38.34.2 Member Data Documentation . . . . .	271
38.34.2.1 angle . . . . .	271
38.35PackageNode Struct Reference . . . . .	271
38.35.1 Detailed Description . . . . .	271
38.35.2 Member Data Documentation . . . . .	272
38.35.2.1 PackageSampleDistance . . . . .	272
38.35.2.2 PackageSampleQuality . . . . .	272

38.36ydlidar::core::serial::PortInfo Struct Reference . . . . .	272
38.36.1 Detailed Description . . . . .	273
38.36.2 Member Data Documentation . . . . .	273
38.36.2.1 description . . . . .	273
38.36.2.2 device_id . . . . .	273
38.36.2.3 hardware_id . . . . .	273
38.36.2.4 port . . . . .	273
38.37pytest.PyTestTestCase Class Reference . . . . .	274
38.37.1 Detailed Description . . . . .	274
38.37.2 Member Function Documentation . . . . .	274
38.37.2.1 testOSInitIsWrappedCorrectly(self) . . . . .	274
38.37.2.2 testParamtersIsWrappedCorrectly(self) . . . . .	275
38.38sampling_rate Struct Reference . . . . .	275
38.38.1 Detailed Description . . . . .	275
38.38.2 Member Data Documentation . . . . .	275
38.38.2.1 rate . . . . .	275
38.39scan_exposure Struct Reference . . . . .	275
38.39.1 Detailed Description . . . . .	276
38.39.2 Member Data Documentation . . . . .	276
38.39.2.1 exposure . . . . .	276
38.40scan_frequency Struct Reference . . . . .	276
38.40.1 Detailed Description . . . . .	276
38.40.2 Member Data Documentation . . . . .	277
38.40.2.1 frequency . . . . .	277
38.41scan_heart_beat Struct Reference . . . . .	277
38.41.1 Detailed Description . . . . .	277
38.41.2 Member Data Documentation . . . . .	277
38.41.2.1 enable . . . . .	277
38.42scan_points Struct Reference . . . . .	277
38.42.1 Detailed Description . . . . .	278

38.42.2 Member Data Documentation . . . . .	278
38.42.2.1 flag . . . . .	278
38.43scan_rotation Struct Reference . . . . .	278
38.43.1 Detailed Description . . . . .	278
38.43.2 Member Data Documentation . . . . .	278
38.43.2.1 rotation . . . . .	278
38.44ydlidar::core::base::ScopedLocker Class Reference . . . . .	279
38.44.1 Detailed Description . . . . .	279
38.44.2 Constructor & Destructor Documentation . . . . .	279
38.44.2.1 ScopedLocker(Locker &l) . . . . .	279
38.44.2.2 ~ScopedLocker() . . . . .	279
38.44.3 Member Function Documentation . . . . .	280
38.44.3.1 forceUnlock() . . . . .	280
38.44.4 Member Data Documentation . . . . .	280
38.44.4.1 _binded . . . . .	280
38.45serial::Serial::ScopedReadLock Class Reference . . . . .	280
38.45.1 Detailed Description . . . . .	280
38.45.2 Constructor & Destructor Documentation . . . . .	280
38.45.2.1 ScopedReadLock(Serial::SerialImpl *pimpl) . . . . .	280
38.45.2.2 ~ScopedReadLock() . . . . .	280
38.46serial::Serial::ScopedWriteLock Class Reference . . . . .	281
38.46.1 Detailed Description . . . . .	281
38.46.2 Constructor & Destructor Documentation . . . . .	281
38.46.2.1 ScopedWriteLock(Serial::SerialImpl *pimpl) . . . . .	281
38.46.2.2 ~ScopedWriteLock() . . . . .	281
38.47ydlidar::core::serial::Serial Class Reference . . . . .	281
38.47.1 Detailed Description . . . . .	283
38.47.2 Member Enumeration Documentation . . . . .	284
38.47.2.1 SerialPortError . . . . .	284
38.47.3 Constructor & Destructor Documentation . . . . .	284

38.47.3.1 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) . . . . .	284
38.47.3.2 ~Serial() . . . . .	285
38.47.4 Member Function Documentation . . . . .	285
38.47.4.1 available() . . . . .	285
38.47.4.2 closePort() . . . . .	285
38.47.4.3 DescribeError(SerialPortError err) . . . . .	285
38.47.4.4 DescribeError() . . . . .	286
38.47.4.5 flush() . . . . .	286
38.47.4.6 flushInput() . . . . .	286
38.47.4.7 flushOutput() . . . . .	286
38.47.4.8 getBaudrate() const . . . . .	286
38.47.4.9 getBytesize() const . . . . .	287
38.47.4.10getByteTime() . . . . .	287
38.47.4.11getCD() . . . . .	287
38.47.4.12getCTS() . . . . .	287
38.47.4.13getDSR() . . . . .	287
38.47.4.14getFlowcontrol() const . . . . .	287
38.47.4.15getParity() const . . . . .	288
38.47.4.16getPort() const . . . . .	288
38.47.4.17getRI() . . . . .	288
38.47.4.18getStopbits() const . . . . .	288
38.47.4.19getSystemError(int systemErrorCode=-1) const . . . . .	288
38.47.4.20getTimeout() const . . . . .	289
38.47.4.21isOpen() . . . . .	289
38.47.4.22open() . . . . .	289
38.47.4.23read(uint8_t *buffer, size_t size) . . . . .	290
38.47.4.24read(std::vector< uint8_t > &buffer, size_t size=1) . . . . .	290
38.47.4.25read(std::string &buffer, size_t size=1) . . . . .	290
38.47.4.26readData(uint8_t *data, size_t size) . . . . .	291

38.47.4.27	<code>readline(std::string &amp;buffer, size_t size=65536, std::string eol=""\n")</code>	291
38.47.4.28	<code>readline(size_t size=65536, std::string eol=""\n")</code>	292
38.47.4.29	<code>readlines(size_t size=65536, std::string eol=""\n")</code>	292
38.47.4.30	<code>readSize(size_t size=1)</code>	292
38.47.4.31	<code>sendBreak(int duration)</code>	293
38.47.4.32	<code>setBaudrate(uint32_t baudrate)</code>	293
38.47.4.33	<code>setBreak(bool level=true)</code>	293
38.47.4.34	<code>setBytesize(bytesize_t bytesize)</code>	293
38.47.4.35	<code>setDTR(bool level=true)</code>	294
38.47.4.36	<code>setFlowcontrol(flowcontrol_t flowcontrol)</code>	294
38.47.4.37	<code>setParity(parity_t parity)</code>	294
38.47.4.38	<code>setPort(const std::string &amp;port)</code>	294
38.47.4.39	<code>setRTS(bool level=true)</code>	295
38.47.4.40	<code>setStopbits(stopbits_t stopbits)</code>	295
38.47.4.41	<code>setTimeout(Timeout &amp;timeout)</code>	295
38.47.4.42	<code>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)</code>	296
38.47.4.43	<code>waitByteTimes(size_t count)</code>	296
38.47.4.44	<code>waitForChange()</code>	296
38.47.4.45	<code>waitfordata(size_t data_count, uint32_t timeout, size_t *returned_size)</code>	296
38.47.4.46	<code>waitReadable()</code>	297
38.47.4.47	<code>write(const uint8_t *data, size_t size)</code>	297
38.47.4.48	<code>write(const std::vector&lt; uint8_t &gt; &amp;data)</code>	297
38.47.4.49	<code>write(const std::string &amp;data)</code>	298
38.47.4.50	<code>writeData(const uint8_t *data, size_t size)</code>	298
38.48	<code>serial::Serial::SerialImpl Class Reference</code>	298
38.48.1	Detailed Description	299
38.48.2	Constructor & Destructor Documentation	300
38.48.2.1	<code>SerialImpl(const string &amp;port, unsigned long baudrate, bytesize_t bytesize, parity_t parity, stopbits_t stopbits, flowcontrol_t flowcontrol)</code>	300
38.48.2.2	<code>~SerialImpl()</code>	300

38.48.3 Member Function Documentation	300
38.48.3.1 available()	300
38.48.3.2 close()	300
38.48.3.3 flush()	300
38.48.3.4 flushInput()	300
38.48.3.5 flushOutput()	300
38.48.3.6 getBaudrate() const	300
38.48.3.7 getBytesize() const	300
38.48.3.8 getByteTime()	301
38.48.3.9 getCD()	301
38.48.3.10 getCTS()	301
38.48.3.11 getDSR()	301
38.48.3.12 getFlowcontrol() const	301
38.48.3.13 getParity() const	301
38.48.3.14 getPort() const	301
38.48.3.15 getRI()	301
38.48.3.16 getStopbits() const	301
38.48.3.17 getSystemError(int systemErrorCode) const	301
38.48.3.18 getTermios(termios *tio)	302
38.48.3.19 getTimeout() const	302
38.48.3.20 isOpen() const	302
38.48.3.21 open()	302
38.48.3.22 read(uint8_t *buf, size_t size=1)	302
38.48.3.23 readLock()	302
38.48.3.24 readUnlock()	302
38.48.3.25 sendBreak(int duration)	302
38.48.3.26 setBaudrate(unsigned long baudrate)	302
38.48.3.27 setBreak(bool level)	302
38.48.3.28 setBytesize(bytesize_t bytesize)	303
38.48.3.29 setCustomBaudRate(unsigned long baudrate)	303

38.48.3.30	setDTR(bool level)	303
38.48.3.31	setFlowcontrol(flowcontrol_t flowcontrol)	303
38.48.3.32	setParity(parity_t parity)	303
38.48.3.33	setPort(const string &port)	303
38.48.3.34	setRTS(bool level)	303
38.48.3.35	setStandardBaudRate(speed_t baudrate)	303
38.48.3.36	setStopbits(stopbits_t stopbits)	303
38.48.3.37	setTermios(const termios *tio)	303
38.48.3.38	setTimeout(Timeout &timeout)	304
38.48.3.39	waitByteTimes(size_t count)	304
38.48.3.40	waitForChange()	304
38.48.3.41	waitfordata(size_t data_count, uint32_t timeout, size_t *returned_size)	304
38.48.3.42	waitReadable(uint32_t timeout)	304
38.48.3.43	write(const uint8_t *data, size_t length)	304
38.48.3.44	writeLock()	304
38.48.3.45	writeUnlock()	304
38.49	string_t Struct Reference	305
38.49.1	Detailed Description	305
38.49.2	Member Data Documentation	305
38.49.2.1	data	305
38.50	ydliar::core::serial::termios2 Struct Reference	305
38.50.1	Detailed Description	305
38.50.2	Member Data Documentation	306
38.50.2.1	c_cc	306
38.50.2.2	c_cflag	306
38.50.2.3	c_iflag	306
38.50.2.4	c_ispeed	306
38.50.2.5	c_lflag	306
38.50.2.6	c_line	306
38.50.2.7	c_oflag	306



38.50.2.8 c_ospeed . . . . .	306
38.51 ydlidar::core::base::Thread Class Reference . . . . .	306
38.51.1 Detailed Description . . . . .	307
38.51.2 Constructor & Destructor Documentation . . . . .	307
38.51.2.1 Thread() . . . . .	307
38.51.2.2 ~Thread() . . . . .	307
38.51.2.3 Thread(thread_proc_t proc, void *param) . . . . .	307
38.51.3 Member Function Documentation . . . . .	308
38.51.3.1 createThread(thread_proc_t proc, void *param=NULL) . . . . .	308
38.51.3.2 createThreadAux(void *param) . . . . .	308
38.51.3.3 getHandle() . . . . .	308
38.51.3.4 getParam() . . . . .	308
38.51.3.5 join(unsigned long timeout=-1) . . . . .	308
38.51.3.6 operator==(const Thread &right) . . . . .	308
38.51.3.7 terminate() . . . . .	308
38.51.3.8 ThreadCreateObjectFunctor(CLASS *pthis) . . . . .	308
38.51.4 Member Data Documentation . . . . .	308
38.51.4.1 _func . . . . .	308
38.51.4.2 _handle . . . . .	309
38.51.4.3 _param . . . . .	309
38.52 ydlidar::core::serial::Timeout Struct Reference . . . . .	309
38.52.1 Detailed Description . . . . .	309
38.52.2 Constructor & Destructor Documentation . . . . .	310
38.52.2.1 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) . . . . .	310
38.52.3 Member Function Documentation . . . . .	310
38.52.3.1 max() . . . . .	310
38.52.3.2 simpleTimeout(uint32_t timeout) . . . . .	310
38.52.4 Member Data Documentation . . . . .	310
38.52.4.1 inter_byte_timeout . . . . .	310

38.52.4.2 read_timeout_constant . . . . .	310
38.52.4.3 read_timeout_multiplier . . . . .	310
38.52.4.4 write_timeout_constant . . . . .	311
38.52.4.5 write_timeout_multiplier . . . . .	311
38.53 YDLidar Struct Reference . . . . .	311
38.53.1 Detailed Description . . . . .	311
38.53.2 Member Data Documentation . . . . .	311
38.53.2.1 lidar . . . . .	311
38.54 ydlidar::YDLidarDriver Class Reference . . . . .	312
38.54.1 Detailed Description . . . . .	315
38.54.2 Constructor & Destructor Documentation . . . . .	315
38.54.2.1 YDLidarDriver(uint8_t type=YDLIDAR_TYPE_SERIAL) . . . . .	315
38.54.2.2 ~YDLidarDriver() . . . . .	315
38.54.3 Member Function Documentation . . . . .	316
38.54.3.1 ascendScanData(node_info *nodebuffer, size_t count) . . . . .	316
38.54.3.2 cacheScanData() . . . . .	317
38.54.3.3 checkAutoConnecting() . . . . .	317
38.54.3.4 checkDeviceInfo(uint8_t *recvBuffer, uint8_t byte, int recvPos, int recvSize, int pos) . . . . .	317
38.54.3.5 checkTransDelay() . . . . .	318
38.54.3.6 clearDTR() . . . . .	318
38.54.3.7 connect(const char *port_path, uint32_t baudrate) . . . . .	318
38.54.3.8 createThread() . . . . .	319
38.54.3.9 DescribeError(bool isTCP=true) . . . . .	319
38.54.3.10 disableDataGrabbing() . . . . .	319
38.54.3.11 disconnect() . . . . .	319
38.54.3.12 flushSerial() . . . . .	320
38.54.3.13 getAutoZeroOffsetAngle(offset_angle &angle, uint32_t timeout=DEFAULT_TIMEOUT) . . . . .	320
38.54.3.14 getData(uint8_t *data, size_t size) . . . . .	320
38.54.3.15 getDeviceInfo(device_info &info, uint32_t timeout=DEFAULT_TIMEOUT) . . . . .	321
38.54.3.16 getHealth(device_health &health, uint32_t timeout=DEFAULT_TIMEOUT) . . . . .	321

38.54.3.17	getSamplingRate(sampling_rate &rate, uint32_t timeout=DEFAULT_TIMEOUT)	322
38.54.3.18	getScanFrequency(scan_frequency &frequency, uint32_t timeout=DEFAULT_TIMEOUT)	322
38.54.3.19	getSDKVersion()	323
38.54.3.20	getZeroOffsetAngle(offset_angle &angle, uint32_t timeout=DEFAULT_TIMEOUT)	323
38.54.3.21	grabScanData(node_info *nodebuffer, size_t &count, uint32_t timeout=DEFAULT_TIMEOUT)	324
38.54.3.22	isconnected() const	324
38.54.3.23	isScanning() const	325
38.54.3.24	lidarPortList()	325
38.54.3.25	reset(uint32_t timeout=DEFAULT_TIMEOUT)	325
38.54.3.26	sendCommand(uint8_t cmd, const void *payload=NULL, size_t payloadsize=0)	326
38.54.3.27	sendData(const uint8_t *data, size_t size)	326
38.54.3.28	setAutoReconnect(const bool &enable)	327
38.54.3.29	setDTR()	327
38.54.3.30	setIntensities(const bool &isintensities)	327
38.54.3.31	setSamplingRate(sampling_rate &rate, uint32_t timeout=DEFAULT_TIMEOUT)	327
38.54.3.32	setScanFrequencyAdd(scan_frequency &frequency, uint32_t timeout=DEFAULT_TIMEOUT)	328
38.54.3.33	setScanFrequencyAddMic(scan_frequency &frequency, uint32_t timeout=DEFAULT_TIMEOUT)	328
38.54.3.34	setScanFrequencyDis(scan_frequency &frequency, uint32_t timeout=DEFAULT_TIMEOUT)	329
38.54.3.35	setScanFrequencyDisMic(scan_frequency &frequency, uint32_t timeout=DEFAULT_TIMEOUT)	330
38.54.3.36	startAutoScan(bool force=false, uint32_t timeout=DEFAULT_TIMEOUT)	330
38.54.3.37	startMotor()	331
38.54.3.38	startScan(bool force=false, uint32_t timeout=DEFAULT_TIMEOUT)	331
38.54.3.39	stop()	332
38.54.3.40	stopMotor()	332
38.54.3.41	stopScan(uint32_t timeout=DEFAULT_TIMEOUT)	332
38.54.3.42	waitDevicePackage(uint32_t timeout=DEFAULT_TIMEOUT)	333
38.54.3.43	waitForData(size_t data_count, uint32_t timeout=DEFAULT_TIMEOUT, size_t *returned_size=NULL)	333
38.54.3.44	waitPackage(node_info *node, uint32_t timeout=DEFAULT_TIMEOUT)	334
38.54.3.45	waitResponseHeader(lidar_ans_header *header, uint32_t timeout=DEFAULT_TIMEOUT)	334
38.54.3.46	waitScanData(node_info *nodebuffer, size_t &count, uint32_t timeout=DEFAULT_TIMEOUT)	335

<b>39 File Documentation</b>	<b>337</b>
39.1 core/base/datatype.h File Reference	337
39.1.1 Macro Definition Documentation	338
39.1.1.1 __attribute__	338
39.1.1.2 __small_endian	338
39.1.1.3 __WORDSIZE	338
39.1.1.4 _access	338
39.1.1.5 _itoa	339
39.1.1.6 DATA_FRAME	339
39.1.1.7 DEFAULT_INTENSITY	339
39.1.1.8 DSL	339
39.1.1.9 FRAME_PREAMBLE	339
39.1.1.10 INVALID_TIMESTAMP	339
39.1.1.11 IS_FAIL	339
39.1.1.12 IS_OK	339
39.1.1.13 IS_TIMEOUT	339
39.1.1.14 LIDAR_2D	339
39.1.1.15 RESULT_FAIL	340
39.1.1.16 RESULT_OK	340
39.1.1.17 RESULT_TIMEOUT	340
39.1.1.18 UNUSED	340
39.1.1.19 valLastName	340
39.1.1.20 valName	340
39.1.2 Typedef Documentation	340
39.1.2.1 result_t	340
39.2 core/base/locker.h File Reference	341
39.3 core/base/thread.h File Reference	342
39.3.1 Macro Definition Documentation	343
39.3.1.1 CLASS_THREAD	343
39.4 core/base/timer.cpp File Reference	343

39.5	core/base/timer.h File Reference	343
39.5.1	Macro Definition Documentation	345
39.5.1.1	BEGIN_STATIC_CODE	345
39.5.1.2	END_STATIC_CODE	345
39.5.1.3	getms	345
39.5.1.4	getTime	345
39.5.2	Function Documentation	345
39.5.2.1	delay(uint32_t ms)	345
39.6	core/base/typedef.h File Reference	345
39.7	core/base/utils.h File Reference	346
39.7.1	Macro Definition Documentation	346
39.7.1.1	YDLIDAR_API	346
39.8	core/base/v8stdint.h File Reference	346
39.8.1	Macro Definition Documentation	348
39.8.1.1	CLEARERR	348
39.8.1.2	DEFAULT_CONNECTION_TIMEOUT_SEC	348
39.8.1.3	DEFAULT_CONNECTION_TIMEOUT_USEC	348
39.8.1.4	DEFAULT_REV_TIMEOUT_SEC	348
39.8.1.5	DEFAULT_REV_TIMEOUT_USEC	348
39.8.1.6	FALSE	348
39.8.1.7	FCLOSE	348
39.8.1.8	FEOF	348
39.8.1.9	FERROR	348
39.8.1.10	FFLUSH	349
39.8.1.11	FILE_HANDLE	349
39.8.1.12	FILENO	349
39.8.1.13	FOPEN	349
39.8.1.14	FPRINTF	349
39.8.1.15	FSTAT	349
39.8.1.16	htonll	349

39.8.1.17 LSTAT . . . . .	349
39.8.1.18 MICROSECONDS_CONVERSION . . . . .	349
39.8.1.19 MILLISECONDS_CONVERSION . . . . .	349
39.8.1.20 NANOSECONDS_CONVERSION . . . . .	350
39.8.1.21 ntohs . . . . .	350
39.8.1.22 PRINTF . . . . .	350
39.8.1.23 SNPRINTF . . . . .	350
39.8.1.24 STAT_BLK_SIZE . . . . .	350
39.8.1.25 STRTOULL . . . . .	350
39.8.1.26 STRUCT_STAT . . . . .	350
39.8.1.27 TRUE . . . . .	350
39.8.1.28 VPRINTF . . . . .	350
39.8.2 Typedef Documentation . . . . .	350
39.8.2.1 thread_proc_t . . . . .	350
39.9 core/base/ydlidar.h File Reference . . . . .	351
39.9.1 Macro Definition Documentation . . . . .	352
39.9.1.1 PropertyBuilderByName . . . . .	352
39.9.2 Typedef Documentation . . . . .	352
39.9.2.1 signal_handler_t . . . . .	352
39.9.3 Function Documentation . . . . .	352
39.9.3.1 set_signal_handler(int signal_value, signal_handler_t signal_handler) . . . . .	352
39.9.3.2 signal_handler(int signal_value) . . . . .	353
39.9.3.3 trigger_interrupt_guard_condition(int signal_value) . . . . .	353
39.9.4 Variable Documentation . . . . .	353
39.9.4.1 g_signal_status . . . . .	353
39.9.4.2 old_signal_handler . . . . .	353
39.10core/common/ChannelDevice.h File Reference . . . . .	353
39.11core/common/DriverInterface.h File Reference . . . . .	354
39.12core/common/ydlidar_datatype.h File Reference . . . . .	355
39.13core/common/ydlidar_def.cpp File Reference . . . . .	356

39.13.1 Function Documentation . . . . .	356
39.13.1.1 LaserFanDestroy(LaserFan *to_destroy) . . . . .	356
39.13.1.2 LaserFanInit(LaserFan *to_init) . . . . .	356
39.14core/common/ydlidar_def.h File Reference . . . . .	357
39.14.1 Enumeration Type Documentation . . . . .	358
39.14.1.1 DeviceTypeID . . . . .	358
39.14.1.2 LidarProperty . . . . .	359
39.14.1.3 LidarTypeID . . . . .	359
39.14.2 Function Documentation . . . . .	359
39.14.2.1 LaserFanDestroy(LaserFan *to_destroy) . . . . .	359
39.14.2.2 LaserFanInit(LaserFan *to_init) . . . . .	359
39.15core/common/ydlidar_help.h File Reference . . . . .	360
39.16core/common/ydlidar_protocol.h File Reference . . . . .	362
39.16.1 Macro Definition Documentation . . . . .	366
39.16.1.1 _countof . . . . .	366
39.16.1.2 GLASSNOISEINTENSITY . . . . .	366
39.16.1.3 LIDAR_ANS_SYNC_BYTE1 . . . . .	366
39.16.1.4 LIDAR_ANS_SYNC_BYTE2 . . . . .	366
39.16.1.5 LIDAR_ANS_TYPE_DEVHEALTH . . . . .	367
39.16.1.6 LIDAR_ANS_TYPE_DEVINFO . . . . .	367
39.16.1.7 LIDAR_ANS_TYPE_MEASUREMENT . . . . .	367
39.16.1.8 LIDAR_CMD_ADD_EXPOSURE . . . . .	367
39.16.1.9 LIDAR_CMD_DIS_EXPOSURE . . . . .	367
39.16.1.10LIDAR_CMD_DISABLE_CONST_FREQ . . . . .	367
39.16.1.11LIDAR_CMD_DISABLE_LOW_POWER . . . . .	367
39.16.1.12LIDAR_CMD_ENABLE_CONST_FREQ . . . . .	367
39.16.1.13LIDAR_CMD_ENABLE_LOW_POWER . . . . .	367
39.16.1.14LIDAR_CMD_FORCE_SCAN . . . . .	367
39.16.1.15LIDAR_CMD_FORCE_STOP . . . . .	368
39.16.1.16LIDAR_CMD_GET_AIMSPEED . . . . .	368

39.16.1.17	LIDAR_CMD_GET_DEVICE_HEALTH . . . . .	368
39.16.1.18	LIDAR_CMD_GET_DEVICE_INFO . . . . .	368
39.16.1.19	LIDAR_CMD_GET_EAI . . . . .	368
39.16.1.20	LIDAR_CMD_GET_OFFSET_ANGLE . . . . .	368
39.16.1.21	LIDAR_CMD_GET_SAMPLING_RATE . . . . .	368
39.16.1.22	LIDAR_CMD_RESET . . . . .	368
39.16.1.23	LIDAR_CMD_RUN_INVERSION . . . . .	368
39.16.1.24	LIDAR_CMD_RUN_POSITIVE . . . . .	368
39.16.1.25	LIDAR_CMD_SAVE_SET_EXPOSURE . . . . .	369
39.16.1.26	LIDAR_CMD_SCAN . . . . .	369
39.16.1.27	LIDAR_CMD_SET_AIMSPEED_ADD . . . . .	369
39.16.1.28	LIDAR_CMD_SET_AIMSPEED_ADDMIC . . . . .	369
39.16.1.29	LIDAR_CMD_SET_AIMSPEED_DIS . . . . .	369
39.16.1.30	LIDAR_CMD_SET_AIMSPEED_DISMIC . . . . .	369
39.16.1.31	LIDAR_CMD_SET_LOW_EXPOSURE . . . . .	369
39.16.1.32	LIDAR_CMD_SET_SAMPLING_RATE . . . . .	369
39.16.1.33	LIDAR_CMD_STATE_MODEL_MOTOR . . . . .	369
39.16.1.34	LIDAR_CMD_STOP . . . . .	369
39.16.1.35	LIDAR_CMD_SYNC_BYTE . . . . .	370
39.16.1.36	LIDAR_CMDFLAG_HAS_PAYLOAD . . . . .	370
39.16.1.37	LIDAR_RESP_MEASUREMENT_ANGLE_SAMPLE_SHIFT . . . . .	370
39.16.1.38	LIDAR_RESP_MEASUREMENT_ANGLE_SHIFT . . . . .	370
39.16.1.39	LIDAR_RESP_MEASUREMENT_CHECKBIT . . . . .	370
39.16.1.40	LIDAR_RESP_MEASUREMENT_DISTANCE_SHIFT . . . . .	370
39.16.1.41	LIDAR_RESP_MEASUREMENT_QUALITY_SHIFT . . . . .	370
39.16.1.42	LIDAR_RESP_MEASUREMENT_SYNCBIT . . . . .	370
39.16.1.43	LIDAR_STATUS_ERROR . . . . .	370
39.16.1.44	LIDAR_STATUS_OK . . . . .	370
39.16.1.45	LIDAR_STATUS_WARNING . . . . .	371
39.16.1.46	M_PI . . . . .	371



39.16.1.47Node_Default_Quality . . . . .	371
39.16.1.48Node_NotSync . . . . .	371
39.16.1.49Node_Sync . . . . .	371
39.16.1.50PackagePaidBytes . . . . .	371
39.16.1.51PackageSampleMaxLngth . . . . .	371
39.16.1.52PH . . . . .	371
39.16.1.53SUNNOISEINTENSITY . . . . .	372
39.16.2 Typedef Documentation . . . . .	372
39.16.2.1 dataFrame . . . . .	372
39.16.2.2 lidarConfig . . . . .	372
39.16.3 Enumeration Type Documentation . . . . .	372
39.16.3.1 CT . . . . .	372
39.16.3.2 ProtocolVer . . . . .	372
39.16.4 Function Documentation . . . . .	372
39.16.4.1 __attribute__((packed)) . . . . .	372
39.16.5 Variable Documentation . . . . .	372
39.16.5.1 angle . . . . .	372
39.16.5.2 angle_q6_checkbit . . . . .	373
39.16.5.3 checkSum . . . . .	373
39.16.5.4 cmd_flag . . . . .	373
39.16.5.5 data . . . . .	373
39.16.5.6 debugInfo . . . . .	373
39.16.5.7 distance_q2 . . . . .	373
39.16.5.8 enable . . . . .	373
39.16.5.9 error_code . . . . .	373
39.16.5.10error_package . . . . .	374
39.16.5.11exposure . . . . .	374
39.16.5.12firmware_version . . . . .	374
39.16.5.13flag . . . . .	374
39.16.5.14frequency . . . . .	374

39.16.5.15hardware_version . . . . .	374
39.16.5.16index . . . . .	374
39.16.5.17model . . . . .	374
39.16.5.18nowPackageNum . . . . .	375
39.16.5.19package_CT . . . . .	375
39.16.5.20package_Head . . . . .	375
39.16.5.21packageFirstSampleAngle . . . . .	375
39.16.5.22packageLastSampleAngle . . . . .	375
39.16.5.23packageSample . . . . .	375
39.16.5.24packageSampleDistance . . . . .	375
39.16.5.25PackageSampleDistance . . . . .	375
39.16.5.26PackageSampleQuality . . . . .	376
39.16.5.27rate . . . . .	376
39.16.5.28rotation . . . . .	376
39.16.5.29scan_frequency . . . . .	376
39.16.5.30serialnum . . . . .	376
39.16.5.31size . . . . .	376
39.16.5.32stamp . . . . .	376
39.16.5.33state . . . . .	376
39.16.5.34status . . . . .	377
39.16.5.35subType . . . . .	377
39.16.5.36sync_flag . . . . .	377
39.16.5.37sync_quality . . . . .	377
39.16.5.38syncByte . . . . .	377
39.16.5.39syncByte1 . . . . .	377
39.16.5.40syncByte2 . . . . .	377
39.16.5.41type . . . . .	377
39.17core/math/angles.h File Reference . . . . .	378
39.17.1 Macro Definition Documentation . . . . .	379
39.17.1.1 M_PI . . . . .	379

39.18	core/network/ActiveSocket.cpp File Reference . . . . .	380
39.19	core/network/ActiveSocket.h File Reference . . . . .	380
39.20	core/network/PassiveSocket.cpp File Reference . . . . .	381
39.21	core/network/PassiveSocket.h File Reference . . . . .	381
39.22	core/network/SimpleSocket.cpp File Reference . . . . .	382
39.22.1	Variable Documentation . . . . .	383
39.22.1.1	m_WSAStartup . . . . .	383
39.23	core/network/SimpleSocket.h File Reference . . . . .	383
39.23.1	Macro Definition Documentation . . . . .	384
39.23.1.1	INVALID_SOCKET . . . . .	384
39.23.1.2	SOCKET_SENDFILE_BLOCKSIZE . . . . .	384
39.24	core/network/StatTimer.h File Reference . . . . .	384
39.24.1	Macro Definition Documentation . . . . .	385
39.24.1.1	GET_CLOCK_COUNT . . . . .	385
39.25	core/serial/common.h File Reference . . . . .	385
39.26	core/serial/impl/unix/list_ports_linux.cpp File Reference . . . . .	386
39.27	core/serial/impl/unix/lock.c File Reference . . . . .	386
39.27.1	Function Documentation . . . . .	387
39.27.1.1	check_group_uucp() . . . . .	387
39.27.1.2	check_lock_pid(const char *file, int openpid) . . . . .	387
39.27.1.3	check_lock_status(const char *filename) . . . . .	387
39.27.1.4	fhs_lock(const char *filename, int pid) . . . . .	387
39.27.1.5	fhs_unlock(const char *filename, int openpid) . . . . .	387
39.27.1.6	is_device_locked(const char *port_filename) . . . . .	387
39.27.1.7	uucp_lock(const char *filename, int pid) . . . . .	387
39.27.1.8	uucp_unlock(const char *filename, int openpid) . . . . .	387
39.28	core/serial/impl/unix/lock.h File Reference . . . . .	388
39.28.1	Macro Definition Documentation . . . . .	389
39.28.1.1	LOCK . . . . .	389
39.28.1.2	UNLOCK . . . . .	389

39.28.2 Function Documentation . . . . .	389
39.28.2.1 check_group_uucp() . . . . .	389
39.28.2.2 check_lock_pid(const char *file, int openpid) . . . . .	389
39.28.2.3 check_lock_status(const char *) . . . . .	389
39.28.2.4 fhs_lock(const char *, int) . . . . .	389
39.28.2.5 fhs_unlock(const char *, int) . . . . .	389
39.28.2.6 is_device_locked(const char *) . . . . .	390
39.28.2.7 lfs_lock(const char *, int) . . . . .	390
39.28.2.8 lfs_unlock(const char *, int) . . . . .	390
39.28.2.9 lib_lock_dev_lock(const char *, int) . . . . .	390
39.28.2.10 lib_lock_dev_unlock(const char *, int) . . . . .	390
39.28.2.11 lock_device(const char *) . . . . .	390
39.28.2.12 unlock_device(const char *) . . . . .	390
39.28.2.13 uucp_lock(const char *, int) . . . . .	390
39.28.2.14 uucp_unlock(const char *, int) . . . . .	390
39.29 core/serial/impl/unix/unix.h File Reference . . . . .	390
39.30 core/serial/impl/unix/unix_serial.cpp File Reference . . . . .	391
39.30.1 Macro Definition Documentation . . . . .	392
39.30.1.1 BOTHER . . . . .	392
39.30.1.2 SNCCS . . . . .	392
39.30.1.3 TCGETS2 . . . . .	392
39.30.1.4 TCSETS2 . . . . .	392
39.30.1.5 TIOCINQ . . . . .	392
39.31 core/serial/impl/unix/unix_serial.h File Reference . . . . .	393
39.32 core/serial/impl/windows/list_ports_win.cpp File Reference . . . . .	394
39.33 core/serial/impl/windows/win.h File Reference . . . . .	394
39.34 core/serial/impl/windows/win_serial.cpp File Reference . . . . .	394
39.35 core/serial/impl/windows/win_serial.h File Reference . . . . .	394
39.36 core/serial/serial.cpp File Reference . . . . .	394
39.37 core/serial/serial.h File Reference . . . . .	395

39.38doc/Dataset.md File Reference . . . . .	397
39.39doc/Diagram.md File Reference . . . . .	397
39.40doc/FAQs/General_FAQs.md File Reference . . . . .	397
39.41doc/FAQs/General_FAQs_cn.md File Reference . . . . .	397
39.42doc/FAQs/Hardware_FAQs.md File Reference . . . . .	397
39.43doc/FAQs/Hardware_FAQs_cn.md File Reference . . . . .	397
39.44doc/FAQs/README.md File Reference . . . . .	397
39.45doc/howto/README.md File Reference . . . . .	397
39.46doc/quickstart/README.md File Reference . . . . .	397
39.47doc/README.md File Reference . . . . .	397
39.48README.md File Reference . . . . .	397
39.49doc/FAQs/Software_FAQs.md File Reference . . . . .	397
39.50doc/FAQs/Software_FAQs_cn.md File Reference . . . . .	397
39.51doc/howto/how_to_build_and_debug_using_vscode.md File Reference . . . . .	397
39.52doc/howto/how_to_build_and_install.md File Reference . . . . .	397
39.53doc/howto/how_to_create_a_pull.md File Reference . . . . .	397
39.54doc/howto/how_to_create_a_udev_rules.md File Reference . . . . .	397
39.55doc/howto/how_to_gerenrate_vs_project_by_cmake.md File Reference . . . . .	398
39.56doc/howto/how_to_solve_slow_pull_from_cn.md File Reference . . . . .	398
39.57doc/quickstart/ydlidar_sdk_software_installation_guide.md File Reference . . . . .	398
39.58doc/Tutorials.md File Reference . . . . .	398
39.59doc/tutorials/examine_the_simple_lidar_tutorial.md File Reference . . . . .	398
39.60doc/tutorials/writing_lidar_tutorial_c++.md File Reference . . . . .	398
39.61doc/tutorials/writing_lidar_tutorial_c.md File Reference . . . . .	398
39.62doc/tutorials/writing_lidar_tutorial_python.md File Reference . . . . .	398
39.63doc/YDLidar-SDK-Communication-Protocol.md File Reference . . . . .	398
39.64doc/YDLIDAR_SDK_API_for_Developers.md File Reference . . . . .	398
39.65python/examples/etlidar_test.py File Reference . . . . .	398
39.66python/examples/plot_tof_test.py File Reference . . . . .	398
39.67python/examples/plot_ydlidar_test.py File Reference . . . . .	399

39.68python/examples/test.py File Reference . . . . .	399
39.69python/examples/tof_test.py File Reference . . . . .	400
39.70python/examples/ydlidar_test.py File Reference . . . . .	400
39.71python/test/pytest.py File Reference . . . . .	400
39.72samples/etlidar_test.cpp File Reference . . . . .	401
39.72.1 Function Documentation . . . . .	401
39.72.1.1 main(int argc, char *argv[]) . . . . .	401
39.73samples/lidar_c_api_test.c File Reference . . . . .	403
39.73.1 Function Documentation . . . . .	403
39.73.1.1 main(int argc, const char *argv[]) . . . . .	403
39.74samples/tof_test.cpp File Reference . . . . .	404
39.74.1 Function Documentation . . . . .	404
39.74.1.1 main(int argc, char *argv[]) . . . . .	404
39.75samples/ydlidar_test.cpp File Reference . . . . .	406
39.75.1 Function Documentation . . . . .	406
39.75.1.1 main(int argc, char *argv[]) . . . . .	406
39.76setup.py File Reference . . . . .	407
39.77src/CYdLidar.cpp File Reference . . . . .	408
39.77.1 Function Documentation . . . . .	408
39.77.1.1 removeExceptionSample(std::map< int, int > &smap) . . . . .	408
39.78src/CYdLidar.h File Reference . . . . .	409
39.79src/ETLidarDriver.cpp File Reference . . . . .	410
39.80src/ETLidarDriver.h File Reference . . . . .	410
39.81src/ydlidar_driver.cpp File Reference . . . . .	411
39.82src/ydlidar_driver.h File Reference . . . . .	412
39.83src/ydlidar_sdk.cpp File Reference . . . . .	413
39.83.1 Function Documentation . . . . .	414
39.83.1.1 DescribeError(YDLidar *lidar) . . . . .	414
39.83.1.2 disconnecting(YDLidar *lidar) . . . . .	414
39.83.1.3 doProcessSimple(YDLidar *lidar, LaserFan *outscan) . . . . .	414

39.83.1.4	getlidaropt(YDLidar *lidar, int optname, void *optval, int optlen)	414
39.83.1.5	GetLidarVersion(YDLidar *lidar, LidarVersion *version)	416
39.83.1.6	GetSdkVersion(char *version)	416
39.83.1.7	initialize(YDLidar *lidar)	416
39.83.1.8	lidarCreate()	417
39.83.1.9	lidarDestroy(YDLidar **lidar)	417
39.83.1.10	lidarPortList(LidarPort *ports)	417
39.83.1.11	los_init()	418
39.83.1.12	bs_isOk()	418
39.83.1.13	bs_shutdown()	418
39.83.1.14	setlidaropt(YDLidar *lidar, int optname, const void *optval, int optlen)	418
39.83.1.15	turnOff(YDLidar *lidar)	420
39.83.1.16	turnOn(YDLidar *lidar)	420
39.84	src/ydlidar_sdk.h File Reference	421
39.84.1	Function Documentation	422
39.84.1.1	DescribeError(YDLidar *lidar)	422
39.84.1.2	disconnecting(YDLidar *lidar)	422
39.84.1.3	doProcessSimple(YDLidar *lidar, LaserFan *outscan)	422
39.84.1.4	getlidaropt(YDLidar *lidar, int optname, void *optval, int optlen)	423
39.84.1.5	GetLidarVersion(YDLidar *lidar, LidarVersion *version)	425
39.84.1.6	GetSdkVersion(char *version)	425
39.84.1.7	initialize(YDLidar *lidar)	425
39.84.1.8	lidarCreate(void)	425
39.84.1.9	lidarDestroy(YDLidar **lidar)	425
39.84.1.10	lidarPortList(LidarPort *ports)	426
39.84.1.11	los_init()	426
39.84.1.12	bs_isOk()	426
39.84.1.13	bs_shutdown()	426
39.84.1.14	setlidaropt(YDLidar *lidar, int optname, const void *optval, int optlen)	426
39.84.1.15	turnOff(YDLidar *lidar)	428
39.84.1.16	turnOn(YDLidar *lidar)	428
39.85	test/lidar_test.cpp File Reference	429
39.85.1	Function Documentation	429
39.85.1.1	main(int argc, char **argv)	429
39.85.1.2	TEST_F(LidarTest, SystemSignal)	429
39.85.1.3	TEST_F(LidarTest, SerialPort)	429
39.85.1.4	TEST_F(LidarTest, SerialBaudrate)	430
39.85.1.5	TEST_F(LidarTest, SingleChannel)	430
39.85.1.6	TEST_F(LidarTest, ScanFrequency)	430
39.85.1.7	TEST_F(LidarTest, TurnOn)	430
39.86	test/lidar_test.h File Reference	430





## Chapter 1

# CYdLidar(YDLIDAR SDK API)

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

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

- [YDlidarDriver](#)
- [ETLidarDriver](#)
- [C API](#) C API

### Copyright

Copyright (c) 2018-2020 EAIBOT

Jump to the [::CYdLidar](#) interface documentation.



## Chapter 2

# YDLIDAR DATASET

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
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

```
1 st=>start: Start
2 op=>operation: Set Paramamters and Initialize
3 opl=>operation: TrunOn
4 tr=>operation: Try Again
5 op2=>operation: doProcessSimple
6 op3=>operation: TrunOff
7 op4=>operation: disconnecting
8 cond=>condition: success Yes or No?
9 cond1=>condition: success Yes or No?
10 cond2=>condition: success Yes or No?
11 cond3=>condition: LOOP Yes or No?
12 cond4=>condition: TryAgain Yes or No?
13 e=>end: End
14 en=>end: End
15
16 st(left)->op->cond
17 cond(yes)->opl->cond1
18 cond(no)->op3->op4->e
19 cond1(yes)->op2->cond3
20 cond3(yes)->op2
21 cond3(no, left)->op3->op4->e
22 cond1(no, right)->tr(bottom)->cond4
23 cond4(yes)->op3
24 cond4(no)->op3(right)->op4(right)->e
```

### sequenceDiagram

```
1 sequenceDiagram
2 note over UserProgram: Set Paramters
3 note over UserProgram: Initialize SDK
4 UserProgram->>Command: Get LiDAR Information
5 Command-->>UserProgram: Device connected and Devce Information received
6 note over UserProgram: TurnOn
7 UserProgram->>Command: Start LiDAR
8 Command-->>UserProgram: LiDAR Started successfully
9 UserProgram->>LaserScan: Get Laser Scan Data
10 LaserScan-->>UserProgram: Laser Scan Data Recvied
11 note over UserProgram: doProcessSimple
12 loop Laser Scan Data
13   LaserScan->>UserProgram: doProcessSimple
14 end
15 note over UserProgram: TurnOn
16 UserProgram->>Command: TurnOff
17 note over UserProgram: disconnecting
18 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: `python setup.py build`
3. install sdk: `python 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

- [General FAQs](#)
- [General FAQs cn](#)
- [Hardware FAQs](#)
- [Hardware FAQs cn](#)
- [Software FAQs](#)
- [Software FAQs cn](#)





## 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

- 1. [Install CMake](#)
- 2. [Build YDLidar-SDK](#)
- 3. [Run Samples](#)
- 4. [Build in VSCode](#)

### 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 separately:

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 generate 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 Ubuntu 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 your 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.



### **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](#)
- [Create The New UDEV Rules](#)
  - [Create new udev file](#)
  - [Query serial port number through udevadm](#)
  - [Create UDEV Permission Rule For tty Devices](#)
- [Restart The UDEV Service](#)

### 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-udev` 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 describes steps to run YDLidar-SDK on Visual Studio 2017.

The YDLidar-SDK version used in this document is the latest 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](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](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](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 Tutorial](#)

### Running the Lidar Tutorial

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 \(C++\)](#)
  - [create beginner\\_tutorials directories](#)
  - [The Code Explained](#)
- [Building your project](#)

### 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](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 available 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.

```

1 {c++}
2 //////////////////////////////////////////////////int property/////////////////////////////////
3 /// lidar baudrate
4 int optval = 230400;
5 laser.setlidaropt(LidarPropSerialBaudrate, &optval, sizeof(int));
6 /// tof lidar
7 optval = TYPE_TRIANGLE;
8 laser.setlidaropt(LidarPropLidarType, &optval, sizeof(int));
9 /// device type
10 optval = YDLIDAR_TYPE_SERIAL;
11 laser.setlidaropt(LidarPropDeviceType, &optval, sizeof(int));
12 /// sample rate
13 optval = 9;
14 laser.setlidaropt(LidarPropSampleRate, &optval, sizeof(int));
15 /// abnormal count
16 optval = 4;
17 laser.setlidaropt(LidarPropAbnormalCheckCount, &optval, sizeof(int));

```

Set Lidar string int paramters.

```

1 {c++}
2 //////////////////////////////////////////////////bool property/////////////////////////////////
3 /// fixed angle resolution
4 bool b_optvalue = false;
5 laser.setlidaropt(LidarPropFixedResolution, &b_optvalue, sizeof(bool));
6 /// rotate 180
7 laser.setlidaropt(LidarPropReversion, &b_optvalue, sizeof(bool));
8 /// Counterclockwise
9 laser.setlidaropt(LidarPropInverted, &b_optvalue, sizeof(bool));
10 b_optvalue = true;
11 laser.setlidaropt(LidarPropAutoReconnect, &b_optvalue, sizeof(bool));
12 /// one-way communication
13 b_optvalue = false;
14 laser.setlidaropt(LidarPropSingleChannel, &b_optvalue, sizeof(bool));
15 /// intensity
16 b_optvalue = false;
17 laser.setlidaropt(LidarPropIntenstiy, &b_optvalue, sizeof(bool));
18 /// Motor DTR
19 b_optvalue = false;
20 laser.setlidaropt(LidarPropSupportMotorDtrCtrl, &b_optvalue, sizeof(bool));

```

Set Lidar bool property paramters.

```

1 {c++}
2 //////////////////////////////////////////////////float property/////////////////////////////////
3 /// unit: °
4 float f_optvalue = 180.0f;
5 laser.setlidaropt(LidarPropMaxAngle, &f_optvalue, sizeof(float));
6 f_optvalue = -180.0f;
7 laser.setlidaropt(LidarPropMinAngle, &f_optvalue, sizeof(float));
8 /// unit: m
9 f_optvalue = 16.f;
10 laser.setlidaropt(LidarPropMaxRange, &f_optvalue, sizeof(float));
11 f_optvalue = 0.1f;
12 laser.setlidaropt(LidarPropMinRange, &f_optvalue, sizeof(float));
13 /// unit: Hz
14 f_optvalue = 10.f;
15 laser.setlidaropt(LidarPropScanFrequency, &f_optvalue, sizeof(float));

```

Set Lidar float property paramters.

```

1 {c++}
2 // initialize SDK and LiDAR
3 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 supply 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\\_sdk\\_tutorials/blob/master/cpp\\_tutorials/lidar\\_tutorial/CMakeLists.txt](https://github.com/YDLIDAR/ydlidar_sdk_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 \(C\)](#)
  - [create beginner\\_tutorials directories](#)
  - [The Code Explained](#)
- [Building your project](#)

### 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](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"

#ifdef _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 available Lidar ports.

```

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

```

Set Lidar string property parameters.

```

//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 supply 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\\_sdk\\_tutorials/blob/master/c\\_tutorials/lidar\\_tutorial/CMakeLists.txt](https://github.com/YDLIDAR/ydlidar_sdk_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 \(Python\)](#)
  - [create beginner\\_tutorials directories](#)
  - [The Code Explained](#)

### 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](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 available 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 supply 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](#)
  - [Writing a Simple Lidar Tutorial \(C++\)](#)
  - [Writing a Simple Lidar Tutorial \(Python\)](#)
  - [Writing a Simple Lidar Tutorial \(C\)](#)
  - [Examining the simple lidar tutorial](#)

### 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
<b>PH(2B)</b>	Packet header	2 Byte in length, Fixed at 0x55AA, low is front, high in back.
<b>CT(1B)</b>	Package type	Indicates the current packet type. (0x00 = CT & 0x01): Normal Point cloud packet. (0x01 = CT & 0x01): Zero packet.
<b>LSN(1B)</b>	Sample Data Number	Indicates the number of sampling points contained in the current packet. There is only once zero point of data in the zero packet. the value is 1.
<b>FSA(2B)</b>	Starting angle	The angle data corresponding to the first sample point in the sampled data.
<b>LSA(2B)</b>	End angle	The angle data corresponding to the last sample point in the sampled data.
<b>CS(2B)</b>	Check code	The check code of the current data packet uses a two-byte exclusive OR to check the current data packet.
<b>Si(2B/3B)</b>	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 Intensity, 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:  $SF = (CT \gg 1) / 10.f$ ; 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) =  $S_i / 4$ ; “
- TOF LiDAR: “ Distance(i) =  $S_i$ ; “

$S_i$  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:

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

- Intensity solution formula:
  - Triangle LiDAR: “ Intensity(i) =  $\text{uint16\_t}((S(1) \& 0x03) \ll 8 \mid S(0))$ ; Distance(i) =  $\text{uint16\_t}(S(2) \ll 8 \mid S(1)) \gg 2$ ; “

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

- Triangle LiDAR: “ Intensity =  $\text{uint16\_t}((0xE5 \& 0x03) \ll 8 \mid 0x1F) = 287$ ; Distance =  $\text{uint16\_t}(0x6F \ll 8 \mid 0xE5) \gg 2 = 7161\text{mm}$ ; “

#### Angle analysis:

##### First level analysis:

Starting angle solution formula:  $\text{Angle}_{\{FSA\}} = \{\text{Rshiftbit}(FSA, 1)\} \{64\}$  End angle solution formula:  $\text{Angle}_{\{LSA\}} = \{\text{Rshiftbit}(LSA, 1)\} \{64\}$  Intermediate angle solution formula:  $\text{Angle}_{\{i\}} = \{\text{diff}(\text{Angle})\} \{LSN - 1\} * i + \text{Angle}_{\{FSA\}} (0, 1, \dots, LSN-1)$   $\text{Angle}_{\{0\}} : \text{Angle}_{\{FSA\}}; \text{Angle}_{\{LSN-1\}} : \text{Angle}_{\{LSA\}}$ ;

$\text{Rshiftbit}(\text{data}, 1)$  means shifting the data to the right by one bit.  $\text{diff}(\text{Angle})$  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.

$\text{diff}(\text{Angle})$ :  $(\text{Angle}(\text{LSA}) - \text{Angle}(\text{FSA}))$  If less than zero,  $\text{diff}(\text{Angle}) = (\text{Angle}(\text{LSA}) - \text{Angle}(\text{FSA})) + 360$ , otherwise  $\text{diff}(\text{Angle}) = (\text{Angle}(\text{LSA}) - \text{Angle}(\text{FSA}))$

#### 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:  $\text{Angle}_{\{i\}} = \text{Angle}_{\{i\}} + \text{AngCorrect}_{\{i\}}$ ; (\$1,2,LSN\$) AngCorrect is the angle correction value, and its calculation formula is as follows,  $\text{atan}^{-1}$  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 = 0x6FE5, LSA = 0x79BD, and bring in the first-level solution formula, and get:  $\text{Angle}_{\{FSA\}} = 223.78^\circ$ ,  $\text{Angle}_{\{LSA\}} = 243.47^\circ$ ,  $\text{diff}(\text{Angle}) = \text{Angle}_{\{LSA\}} - \text{Angle}_{\{FSA\}} = 243.47^\circ - 223.78^\circ = 19.69^\circ$ ,  $\text{Angle}_{\{i\}} = \{19.69^\circ\} \times (i - 1) + 223.78^\circ$  (\$1,2,LSN\$) Assume that in the frame data:  $\text{Distance}_{\{1\}} = 1000$ ,  $\text{Distance}_{\{LSN\}} = 8000$ , bring in the second-level solution formula, you get:  $\text{AngCorrect}_{\{1\}} = -6.7622^\circ$ ,  $\text{AngCorrect}_{\{LSN\}} = -7.8374^\circ$ ,  $\text{Angle}_{\{FSA\}} = \text{Angle}_{\{1\}} + \text{AngCorrect}_{\{1\}} = 217.0178^\circ$ ,  $\text{Angle}_{\{LSA\}} = \text{Angle}_{\{LSA\}} + \text{AngCorrect}_{\{LSA\}} = 235.6326^\circ$ . Similarly,  $\text{Angle}_{\{i\}}(2,3, \dots, \text{LSN}-1)$ , 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 need 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:

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

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 uint8_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	Content	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 Angle	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 uint8_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 uint16_t Intensity[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     Intensity[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/](https://github.com/YDLIDAR/ydlidar_tutorials/blob/master/CommunicationProtocol/README.md)↵  
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](#)
  - [Create A System State](#)
  - [Code Example](#)
  - [Python Example](#)
- [ThridParty Project Call Library OR Source](#)
  - [Introduction](#)
  - [Calling a compiled library.](#)
  - [Add source code to the Project.](#)
- [Development Flow](#)
- [C++ API Directory](#)
  - [CYdLidar](#)
  - [YDlidarDriver](#)
  - [ETLidarDriver](#)
  - [Parameter Table](#)

### Samples

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

## Create A System State

In the YDLIDAR SDK, the `ydlidar::os_init()` is optional 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
24                [" ,1.0/scan.config.scan_time,"]Hz");
25            else :
26                print("Failed to get Lidar Data")
27                time.sleep(0.05);
28            laser.turnOff();
29    laser.disconnecting();

```

## ThridParty Project Call Library OR Source

### Introduction

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

- [Calling a compiled 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 compiled 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

|               |              |
|---------------|--------------|
| <i>optval</i> | option value |
|---------------|--------------|

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

#### Parameters

|               |               |
|---------------|---------------|
| <i>optlen</i> | 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

|                |             |
|----------------|-------------|
| <i>optname</i> | 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**

<i>optval</i>	option value
---------------	--------------

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

**Parameters**

<i>optlen</i>	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**

<i>version</i>	Pointer to a version structure for returning the version information. <code>*/ void GetLidarVersion(LidarVersion &amp;version);</code>
----------------	--

- `/**`
  - 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	<i>outscan</i>	LiDAR Scan Data
-----	----------------	-----------------

–

## Parameters

out	<i>hardwareError</i>	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 Stopped, 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**

<i>optval</i>	option value
---------------	--------------

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

**Parameters**

<i>optlen</i>	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**

<i>lidar</i>	a lidar instance
--------------	------------------

- 

**Parameters**

<i>optname</i>	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 lidar;
2 * int lidar_baudrate;
3 * lidar.getlidaropt(LidarPropSerialPort,&lidar_baudrate, sizeof(int));
4 *

```

- **Todo** bool properties

**Note**

## get bool property example

```

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

```

- **Todo** float properties

**Note**

## set float property example

```

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

```

**Parameters**

|               |              |
|---------------|--------------|
| <i>optval</i> | option value |
|---------------|--------------|

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

**Parameters**

|               |               |
|---------------|---------------|
| <i>optlen</i> | 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

|                |                                                                                                                    |
|----------------|--------------------------------------------------------------------------------------------------------------------|
| <i>version</i> | Pointer to a version for returning the version information. */ void <a href="#">GetSdkVersion(char *version)</a> ; |
|----------------|--------------------------------------------------------------------------------------------------------------------|

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

## Parameters

|                |                                                                                                                                                        |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>version</i> | Pointer to a version structure for returning the version information. */ void <a href="#">GetLidarVersion(YDLidar *lidar, LidarVersion *version)</a> ; |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|

- /\*\*
  - 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

|           |              |                |
|-----------|--------------|----------------|
| <i>in</i> | <i>lidar</i> | LiDAR instance |
|-----------|--------------|----------------|

–

## Parameters

|            |                |                 |
|------------|----------------|-----------------|
| <i>out</i> | <i>outscan</i> | 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 Stopped, 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**

|              |                   |
|--------------|-------------------|
| <i>ports</i> | 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↔<br>Rate(K) | Range(m)                   | Frequency<br>HZ) | Intenstiy(b | Single↔<br>Channel | voltage(↔<br>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<br>(PWM)    | false       | false              | 4.8~5.2         |
| S4B   | 4/11  | 153600   | 4                  | 0.10~8.0                   | 5~12(P↔<br>WM)   | true(8)     | false              | 4.8~5.2         |
| S2    | 4/12  | 115200   | 3                  | 0.10~8.0                   | 4~8(P↔<br>WM)    | false       | true               | 4.8~5.2         |
| G4    | 5     | 230400   | 9/8/4              | 0.↔<br>28/0.26/0.↔<br>1~16 | 5~12             | false       | false              | 4.8~5.2         |

| LIDAR  | Model | Baudrate | Sample↔<br>Rate(K) | Range(m)                   | Frequency↔<br>HZ) | Intenstiy(b | Single↔<br>Channel | voltage(↔<br>V) |
|--------|-------|----------|--------------------|----------------------------|-------------------|-------------|--------------------|-----------------|
| X4     | 6     | 128000   | 5                  | 0.12~10                    | 5~12(P↔<br>WM)    | false       | false              | 4.8~5.2         |
| X2/X2L | 6     | 115200   | 3                  | 0.10~8.0                   | 4~8(P↔<br>WM)     | false       | true               | 4.8~5.2         |
| G4PRO  | 7     | 230400   | 9/8/4              | 0.↔<br>28/0.26/0.↔<br>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.↔<br>28/0.26/0.↔<br>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↔<br>WM)     | false       | true               | 4.8~5.2         |
| TX20   | 100   | 115200   | 4                  | 0.1~20                     | 4~8(P↔<br>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 |
|-------------------------------------|----------------|
| <b>F4/S2/X2/X2L/S4/TX8/TX20/G4C</b> | 115200         |
| <b>X4</b>                           | 128000         |
| <b>S4B</b>                          | 153600         |
| <b>G1/G2/R2/G4/G4PRO/F4PRO</b>      | 230400         |
| <b>G2A/G2C</b>                      | 230400         |
| <b>G6/G4B/TG15/TG30/TG50</b>        | 512000         |
| <b>T5/T15(network)</b>              | 8000           |

Table List - SampleRate

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

|                       |          |
|-----------------------|----------|
| <b>G2C</b>            | 4        |
| <b>S2</b>             | 3        |
| <b>TG15/TG30/TG50</b> | 10,18,20 |
| <b>T5/T15</b>         | 20       |

Table List - ScanFrequency

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

Table List - SingleChannel

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

Table List - LidarType

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

Table List - Intensity

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



Table List - SupportMotorDtrCtrl

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



# Chapter 28

## README

### Table of Contents

1. [Introduction](#)
  - [Prerequisites](#)
  - [Supported Languages](#)
2. [YDLidar SDK Communication Protocol](#)
3. [Architecture](#)
4. [Installation](#)
5. [Documents](#)
6. [Support](#)
7. [Contact EAI](#)

### 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 [Y↔DLidar 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](#) Handler.

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](mailto: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

|                |                                                                                             |
|----------------|---------------------------------------------------------------------------------------------|
| <b>Library</b> | ETLidarDriver                                                                               |
| <b>File</b>    | <a href="#">ETLidarDriver.h</a>                                                             |
| <b>Author</b>  | Tony [code at ydlidar com]                                                                  |
| <b>Source</b>  | <a href="https://github.com/ydlidar/YDLidar-SDK">https://github.com/ydlidar/YDLidar-SDK</a> |
| <b>Version</b> | 1.0.0                                                                                       |

This ETLidarDriver support [TYPE\\_TOF\\_NET](#) LiDAR

Copyright

Copyright (c) 2018-2020 EAIBOT Jump to the [::ydlidar::ETLidarDriver](#) interface documentation.





## Chapter 30

# YDlidarDriver

YDlidarDriver API

|                |                                                                                             |
|----------------|---------------------------------------------------------------------------------------------|
| <b>Library</b> | YDlidarDriver                                                                               |
| <b>File</b>    | <a href="#">ydlidar_driver.h</a>                                                            |
| <b>Author</b>  | Tony [code at ydlidar com]                                                                  |
| <b>Source</b>  | <a href="https://github.com/ydlidar/YDLidar-SDK">https://github.com/ydlidar/YDLidar-SDK</a> |
| <b>Version</b> | 1.0.0                                                                                       |

This YDlidarDriver support [TYPE\\_TRIANGLE](#) and [TYPE\\_TOF](#) LiDAR

Copyright

Copyright (c) 2018-2020 EAIBOT Jump to the [::ydlidar::YDlidarDriver](#) interface documentation.



## Chapter 31

# C API

### YDLIDAR C API

|                |                                                                                           |
|----------------|-------------------------------------------------------------------------------------------|
| <b>Library</b> | ydliar_sdk                                                                                |
| <b>File</b>    | <a href="#">ydliar_sdk.h</a>                                                              |
| <b>Author</b>  | Tony [code at ydliar com]                                                                 |
| <b>Source</b>  | <a href="https://github.com/ydliar/YDLidar-SDK">https://github.com/ydliar/YDLidar-SDK</a> |
| <b>Version</b> | 1.0.0                                                                                     |

### Copyright

Copyright (c) 2018-2020 EAIBOT Jump to the [ydliar\\_sdk.h](#) interface documentation.



## 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](#)
- [LidarPropIntensitiy](#)

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](#)
- - [LidarPropIntensity](#)
- 

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:

|                                        |     |
|----------------------------------------|-----|
| <a href="#">etlidar_test</a>           | 115 |
| <a href="#">impl</a>                   | 116 |
| <a href="#">plot_tof_test</a>          | 116 |
| <a href="#">plot_ydlidar_test</a>      | 118 |
| <a href="#">pytest</a>                 | 119 |
| <a href="#">serial</a>                 | 119 |
| <a href="#">serial::Serial</a>         | 119 |
| <a href="#">setup</a>                  | 120 |
| <a href="#">test</a>                   | 120 |
| <a href="#">tof_test</a>               | 121 |
| <a href="#">ydlidar</a>                |     |
| Ydlidar                                | 122 |
| <a href="#">ydlidar::core</a>          |     |
| Ydlidar core                           | 125 |
| <a href="#">ydlidar::core::base</a>    | 125 |
| <a href="#">ydlidar::core::common</a>  |     |
| Ydlidar common                         | 127 |
| <a href="#">ydlidar::core::math</a>    | 138 |
| <a href="#">ydlidar::core::network</a> | 141 |
| <a href="#">ydlidar::core::serial</a>  | 141 |
| <a href="#">ydlidar_test</a>           | 143 |



## Chapter 34

# Hierarchical Index

### 34.1 Class Hierarchy

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

|                                                   |     |
|---------------------------------------------------|-----|
| _dataFrame . . . . .                              | 145 |
| _lidarConfig . . . . .                            | 147 |
| ydlidar::core::common::ChannelDevice . . . . .    | 152 |
| ydlidar::core::network::CSimpleSocket . . . . .   | 166 |
| ydlidar::core::network::CActiveSocket . . . . .   | 150 |
| ydlidar::core::network::CPassiveSocket . . . . .  | 163 |
| ydlidar::core::serial::Serial . . . . .           | 281 |
| cmd_packet . . . . .                              | 162 |
| CStatTimer . . . . .                              | 192 |
| CYdLidar . . . . .                                | 193 |
| dataFrame . . . . .                               | 211 |
| device_health . . . . .                           | 211 |
| device_info . . . . .                             | 212 |
| ydlidar::core::common::DriverInterface . . . . .  | 213 |
| ydlidar::ETLidarDriver . . . . .                  | 233 |
| ydlidar::YDLidarDriver . . . . .                  | 312 |
| ydlidar::core::base::Event . . . . .              | 246 |
| function_state . . . . .                          | 248 |
| LaserConfig . . . . .                             | 248 |
| LaserDebug . . . . .                              | 250 |
| LaserFan . . . . .                                | 253 |
| LaserPoint . . . . .                              | 255 |
| LaserScan . . . . .                               | 256 |
| lidar_ans_header . . . . .                        | 258 |
| lidarConfig . . . . .                             | 259 |
| LidarPort . . . . .                               | 259 |
| LidarVersion . . . . .                            | 261 |
| ydlidar::core::base::Locker . . . . .             | 263 |
| ydlidar::core::serial::MillisecondTimer . . . . . | 264 |
| node_info . . . . .                               | 265 |
| node_package . . . . .                            | 267 |
| node_packages . . . . .                           | 269 |
| offset_angle . . . . .                            | 271 |
| PackageNode . . . . .                             | 271 |
| ydlidar::core::serial::PortInfo . . . . .         | 272 |

|                                             |     |
|---------------------------------------------|-----|
| sampling_rate . . . . .                     | 275 |
| scan_exposure . . . . .                     | 275 |
| scan_frequency . . . . .                    | 276 |
| scan_heart_beat . . . . .                   | 277 |
| scan_points . . . . .                       | 277 |
| scan_rotation . . . . .                     | 278 |
| ydlidar::core::base::ScopedLocker . . . . . | 279 |
| serial::Serial::ScopedReadLock . . . . .    | 280 |
| serial::Serial::ScopedWriteLock . . . . .   | 281 |
| serial::Serial::SerialImpl . . . . .        | 298 |
| string_t . . . . .                          | 305 |
| ydlidar::core::serial::termios2 . . . . .   | 305 |
| Test                                        |     |
| LidarTest . . . . .                         | 260 |
| TestCase                                    |     |
| pytest.PyTestTestCase . . . . .             | 274 |
| ydlidar::core::base::Thread . . . . .       | 306 |
| ydlidar::core::serial::Timeout . . . . .    | 309 |
| YDLidar . . . . .                           | 311 |
| build_ext                                   |     |
| setup.CMakeBuild . . . . .                  | 158 |
| Extension                                   |     |
| setup.CMakeExtension . . . . .              | 160 |

## Chapter 35

# Class Index

### 35.1 Class List

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

|                                                        |     |
|--------------------------------------------------------|-----|
| <a href="#">_dataFrame</a>                             |     |
| UDP Data format                                        | 145 |
| <a href="#">_lidarConfig</a>                           | 147 |
| <a href="#">ydlidar::core::network::CActiveSocket</a>  | 150 |
| <a href="#">ydlidar::core::common::ChannelDevice</a>   | 152 |
| <a href="#">setup.CMakeBuild</a>                       | 158 |
| <a href="#">setup.CMakeExtension</a>                   | 160 |
| <a href="#">cmd_packet</a>                             |     |
| LiDAR request command packet                           | 162 |
| <a href="#">ydlidar::core::network::CPassiveSocket</a> | 163 |
| <a href="#">ydlidar::core::network::CSimpleSocket</a>  | 166 |
| <a href="#">CStatTimer</a>                             | 192 |
| <a href="#">CYdLidar</a>                               |     |
| Set and Get LiDAR Maximum effective range              | 193 |
| <a href="#">dataFrame</a>                              |     |
| Data frame Structure                                   | 211 |
| <a href="#">device_health</a>                          |     |
| LiDAR Health Information                               | 211 |
| <a href="#">device_info</a>                            |     |
| LiDAR Device Information                               | 212 |
| <a href="#">ydlidar::core::common::DriverInterface</a> | 213 |
| <a href="#">ydlidar::ETLidarDriver</a>                 | 233 |
| <a href="#">ydlidar::core::base::Event</a>             | 246 |
| <a href="#">function_state</a>                         | 248 |
| <a href="#">LaserConfig</a>                            |     |
| A struct for returning configuration from the YDLIDAR  | 248 |
| <a href="#">LaserDebug</a>                             |     |
| The Laser Debug struct                                 | 250 |
| <a href="#">LaserFan</a>                               |     |
| The Laser Scan Data struct                             | 253 |
| <a href="#">LaserPoint</a>                             |     |
| The Laser Point struct                                 | 255 |
| <a href="#">LaserScan</a>                              |     |
| The Laser Scan Data struct                             | 256 |
| <a href="#">lidar_ans_header</a>                       |     |
| LiDAR response Header                                  | 258 |

|                                                         |     |
|---------------------------------------------------------|-----|
| <a href="#">lidarConfig</a>                             |     |
| Structure containing scan configuration                 | 259 |
| <a href="#">LidarPort</a>                               |     |
| Lidar ports                                             | 259 |
| <a href="#">LidarTest</a>                               | 260 |
| <a href="#">LidarVersion</a>                            | 261 |
| <a href="#">ydlidar::core::base::Locker</a>             | 263 |
| <a href="#">ydlidar::core::serial::MillisecondTimer</a> | 264 |
| <a href="#">node_info</a>                               |     |
| LiDAR Node info                                         | 265 |
| <a href="#">node_package</a>                            |     |
| LiDAR Intensity Nodes Package                           | 267 |
| <a href="#">node_packages</a>                           |     |
| LiDAR Normal Nodes package                              | 269 |
| <a href="#">offset_angle</a>                            |     |
| LiDAR Zero Offset Angle                                 | 271 |
| <a href="#">PackageNode</a>                             |     |
| Package node info                                       | 271 |
| <a href="#">ydlidar::core::serial::PortInfo</a>         | 272 |
| <a href="#">pytest.PyTestTestCase</a>                   | 274 |
| <a href="#">sampling_rate</a>                           |     |
| LiDAR sampling Rate struct                              | 275 |
| <a href="#">scan_exposure</a>                           |     |
| LiDAR Exposure struct                                   | 275 |
| <a href="#">scan_frequency</a>                          |     |
| LiDAR scan frequency struct                             | 276 |
| <a href="#">scan_heart_beat</a>                         |     |
| LiDAR Heart beat struct                                 | 277 |
| <a href="#">scan_points</a>                             | 277 |
| <a href="#">scan_rotation</a>                           | 278 |
| <a href="#">ydlidar::core::base::ScopedLocker</a>       | 279 |
| <a href="#">serial::Serial::ScopedReadLock</a>          | 280 |
| <a href="#">serial::Serial::ScopedWriteLock</a>         | 281 |
| <a href="#">ydlidar::core::serial::Serial</a>           | 281 |
| <a href="#">serial::Serial::SerialImpl</a>              | 298 |
| <a href="#">string_t</a>                                |     |
| C string                                                | 305 |
| <a href="#">ydlidar::core::serial::termios2</a>         | 305 |
| <a href="#">ydlidar::core::base::Thread</a>             | 306 |
| <a href="#">ydlidar::core::serial::Timeout</a>          | 309 |
| <a href="#">YDLidar</a>                                 |     |
| Lidar instance                                          | 311 |
| <a href="#">ydlidar::YDLidarDriver</a>                  | 312 |



## Chapter 36

# File Index

### 36.1 File List

Here is a list of all files with brief descriptions:

|                                                             |     |
|-------------------------------------------------------------|-----|
| <a href="#">setup.py</a>                                    | 407 |
| <a href="#">core/base/datatype.h</a>                        | 337 |
| <a href="#">core/base/locker.h</a>                          | 341 |
| <a href="#">core/base/thread.h</a>                          | 342 |
| <a href="#">core/base/timer.cpp</a>                         | 343 |
| <a href="#">core/base/timer.h</a>                           | 343 |
| <a href="#">core/base/typedef.h</a>                         | 345 |
| <a href="#">core/base/utils.h</a>                           | 346 |
| <a href="#">core/base/v8stdint.h</a>                        | 346 |
| <a href="#">core/base/ydlidar.h</a>                         | 351 |
| <a href="#">core/common/ChannelDevice.h</a>                 | 353 |
| <a href="#">core/common/DriverInterface.h</a>               | 354 |
| <a href="#">core/common/ydlidar_datatype.h</a>              | 355 |
| <a href="#">core/common/ydlidar_def.cpp</a>                 | 356 |
| <a href="#">core/common/ydlidar_def.h</a>                   | 357 |
| <a href="#">core/common/ydlidar_help.h</a>                  | 360 |
| <a href="#">core/common/ydlidar_protocol.h</a>              | 362 |
| <a href="#">core/math/angles.h</a>                          | 378 |
| <a href="#">core/network/ActiveSocket.cpp</a>               | 380 |
| <a href="#">core/network/ActiveSocket.h</a>                 | 380 |
| <a href="#">core/network/PassiveSocket.cpp</a>              | 381 |
| <a href="#">core/network/PassiveSocket.h</a>                | 381 |
| <a href="#">core/network/SimpleSocket.cpp</a>               | 382 |
| <a href="#">core/network/SimpleSocket.h</a>                 | 383 |
| <a href="#">core/network/StatTimer.h</a>                    | 384 |
| <a href="#">core/serial/common.h</a>                        | 385 |
| <a href="#">core/serial/serial.cpp</a>                      | 394 |
| <a href="#">core/serial/serial.h</a>                        | 395 |
| <a href="#">core/serial/impl/unix/list_ports_linux.cpp</a>  | 386 |
| <a href="#">core/serial/impl/unix/lock.c</a>                | 386 |
| <a href="#">core/serial/impl/unix/lock.h</a>                | 388 |
| <a href="#">core/serial/impl/unix/unix.h</a>                | 390 |
| <a href="#">core/serial/impl/unix/unix_serial.cpp</a>       | 391 |
| <a href="#">core/serial/impl/unix/unix_serial.h</a>         | 393 |
| <a href="#">core/serial/impl/windows/list_ports_win.cpp</a> | 394 |

|                                         |     |
|-----------------------------------------|-----|
| core/serial/impl/windows/win.h          | 394 |
| core/serial/impl/windows/win_serial.cpp | 394 |
| core/serial/impl/windows/win_serial.h   | 394 |
| python/examples/etlidar_test.py         | 398 |
| python/examples/plot_tof_test.py        | 398 |
| python/examples/plot_ydlidar_test.py    | 399 |
| python/examples/test.py                 | 399 |
| python/examples/tof_test.py             | 400 |
| python/examples/ydlidar_test.py         | 400 |
| python/test/pytest.py                   | 400 |
| samples/etlidar_test.cpp                | 401 |
| samples/lidar_c_api_test.c              | 403 |
| samples/tof_test.cpp                    | 404 |
| samples/ydlidar_test.cpp                | 406 |
| src/CYdLidar.cpp                        | 408 |
| src/CYdLidar.h                          | 409 |
| src/ETLidarDriver.cpp                   | 410 |
| src/ETLidarDriver.h                     | 410 |
| src/ydlidar_driver.cpp                  | 411 |
| src/ydlidar_driver.h                    | 412 |
| src/ydlidar_sdk.cpp                     | 413 |
| src/ydlidar_sdk.h                       | 421 |
| test/lidar_test.cpp                     | 429 |
| test/lidar_test.h                       | 430 |

## 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

|                  |                   |
|------------------|-------------------|
| <i>fd</i>        | : The serial port |
| <i>speed</i>     | : The baud rate   |
| <i>data_bits</i> | : The data bits   |
| <i>parity</i>    | : The parity bits |
| <i>stop_bits</i> | : 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                           |
| IOBLK       | Block job control output                                             |
| CNEW_RTSCTS | RTSCTS 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

|                 |  |
|-----------------|--|
| <i>filename</i> |  |
|-----------------|--|

#### Returns

Definition at line 178 of file ydlidar.h.

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

*initialize system state*

#### Parameters

|             |  |
|-------------|--|
| <i>argc</i> |  |
| <i>argv</i> |  |

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

ydliidar common

### 37.14.2 Function Documentation

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

convert LiDAR sampling rate code to User sampling code

#### Parameters

|              |                          |
|--------------|--------------------------|
| <i>model</i> | LiDAR model              |
| <i>rate</i>  | LiDAR sampling rate code |

#### Returns

user sampling code

Definition at line 651 of file ydliidar\_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

|                     |                                  |
|---------------------|----------------------------------|
| <i>model</i>        | LiDAR model                      |
| <i>m_SampleRate</i> | User sampling rate code          |
| <i>defaultRate</i>  | LiDAR Default 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

|              |             |
|--------------|-------------|
| <i>model</i> | 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

|              |             |
|--------------|-------------|
| <i>model</i> | 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

|              |             |
|--------------|-------------|
| <i>model</i> | 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

|              |             |
|--------------|-------------|
| <i>model</i> | 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

|             |            |
|-------------|------------|
| <i>type</i> | 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

|              |             |
|--------------|-------------|
| <i>model</i> | 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**

|              |             |
|--------------|-------------|
| <i>model</i> | 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**

|              |                        |
|--------------|------------------------|
| <i>model</i> | lidar model            |
| <i>Major</i> | firmware Major version |
| <i>Minor</i> | 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**

|             |                                              |
|-------------|----------------------------------------------|
| <i>info</i> | LiDAR <a href="#">LaserDebug</a> 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**

|              |             |
|--------------|-------------|
| <i>model</i> | 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**

|              |             |
|--------------|-------------|
| <i>model</i> | 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**

|                  |                    |
|------------------|--------------------|
| <i>model</i>     | lidar model        |
| <i>frequency</i> | 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

|             |            |
|-------------|------------|
| <i>type</i> | 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

|              |             |
|--------------|-------------|
| <i>model</i> | 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

|             |            |
|-------------|------------|
| <i>type</i> | 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

|                 |                                 |
|-----------------|---------------------------------|
| <i>protocol</i> | 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

|             |                   |
|-------------|-------------------|
| <i>smap</i> | 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

|              |                           |
|--------------|---------------------------|
| <i>value</i> | 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

|             |                                                  |
|-------------|--------------------------------------------------|
| <i>info</i> | the LiDAR <a href="#">LaserDebug</a> 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**

|              |              |
|--------------|--------------|
| <i>model</i> | 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**

|              |             |
|--------------|-------------|
| <i>model</i> | 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**

|              |                                              |
|--------------|----------------------------------------------|
| <i>info</i>  | LiDAR <a href="#">LaserDebug</a> information |
| <i>value</i> | 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**

|             |                              |
|-------------|------------------------------|
| <i>node</i> | LiDAR node_info information  |
| <i>info</i> | 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**

|                 |                                       |
|-----------------|---------------------------------------|
| <i>info</i>     | LiDAR Device information              |
| <i>port</i>     | LiDAR serial port or IP Address       |
| <i>baudrate</i> | 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**

|              |              |
|--------------|--------------|
| <i>s</i>     | string       |
| <i>delim</i> | 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

|                         |                                                                                                                                                                                     |
|-------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>from</i>             | - "from" angle - must lie in $[-M\_PI, M\_PI]$                                                                                                                                      |
| <i>left_limit</i>       | - 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  |
| <i>right_limit</i>      | - 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 |
| <i>result_min_delta</i> | - minimum (delta) angle (in radians) that can be moved from "from" position before hitting the joint stop                                                                           |
| <i>result_max_delta</i> | - maximum (delta) angle (in radians) that can be moved 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 outputs are of course radians.

The result would always be  $-\pi \leq \text{result} \leq \pi$ . Adding the result to "from" will always get you an equivalent 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  $2 \times M\_PI - 0.5$  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  $2 \times M\_PI - 1.0$  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

|                       |                                                                                                                                                    |
|-----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>from</i>           | - "from" angle                                                                                                                                     |
| <i>to</i>             | - "to" angle                                                                                                                                       |
| <i>left_limit</i>     | - 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  |
| <i>right_limit</i>    | - 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 |
| <i>shortest_angle</i> | - 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  $[-2 \times M\_PI, 2 \times M\_PI]$  going the other way along the unit circle.

## Parameters

|              |                                                                                                                                                                                                                             |
|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>angle</i> | The angle to which you want to turn in the range $[-2 \cdot M\_PI, 2 \cdot M\_PI]$ E.g. <code>two_pi_complement(-M_PI/4)</code> returns $7 \cdot M\_PI/4$ <code>two_pi_complement(M_PI/4)</code> returns $-7 \cdot M\_PI/4$ |
|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

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, DHCP 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.

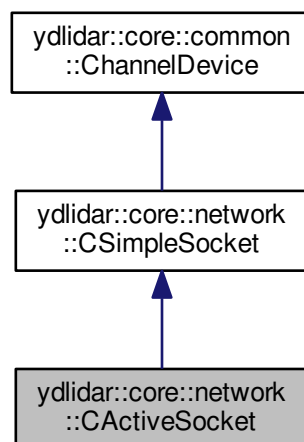
The documentation for this struct was generated from the following file:

- [core/common/ydlidar\\_protocol.h](#)

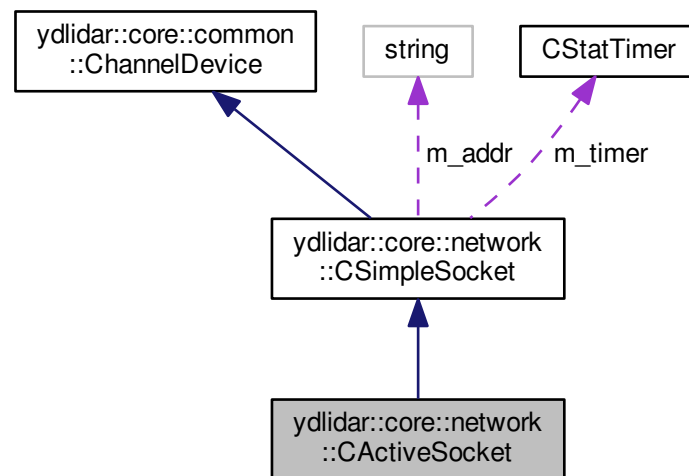
## 38.3 ydlidar::core::network::CActiveSocket Class Reference

```
#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 CActiveSocket.h.

#### 38.3.2 Constructor & Destructor Documentation

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

Definition at line 51 of file CActiveSocket.cpp.

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

Definition at line 63 of file CActiveSocket.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::Socket↔TypeTcp) may successfully call [open\(\)](#) only once, however; connectionless protocol sockets (CSocket::Socket↔TypeUdp) may use [Open\(\)](#) multiple times to change their association.

#### Parameters

|              |                                               |
|--------------|-----------------------------------------------|
| <i>pAddr</i> | specifies the destination address to connect. |
| <i>nPort</i> | specifies the destination port.               |

#### Returns

true if successful connection made, otherwise false.

Reimplemented from [ydlidar::core::network::CSimpleSocket](#).

Definition at line 249 of file CActiveSocket.cpp.

### 38.3.4 Friends And Related Function Documentation

**38.3.4.1** `friend class CPassiveSocket [friend]`

Definition at line 60 of file CActiveSocket.h.

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

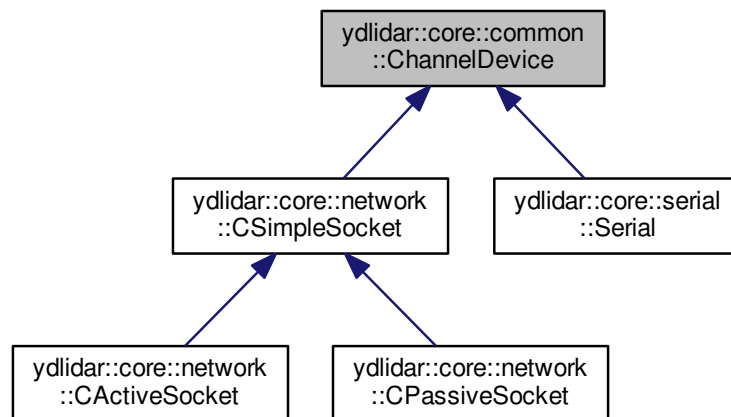
- [core/network/ActiveSocket.h](#)
- [core/network/ActiveSocket.cpp](#)

## 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 `ydliar::core::common::ChannelDevice::ChannelDevice ( )` `[inline]`

Definition at line 9 of file ChannelDevice.h.

38.4.2.2 `virtual ydliar::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 ydliar::core::common::ChannelDevice::available ( )` `[pure virtual]`

Return the number of characters in the buffer.

Returns

Implemented in [ydliar::core::network::CSimpleSocket](#), and [ydliar::core::serial::Serial](#).

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

bind device port

Returns

true if successfully bind, otherwise false

Reimplemented in [ydliar::core::network::CSimpleSocket](#).

Definition at line 15 of file ChannelDevice.h.

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

close serial port or network

Implemented in [ydliar::core::network::CSimpleSocket](#), and [ydliar::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

|               |                                                               |
|---------------|---------------------------------------------------------------|
| <i>buffer</i> | An <code>uint8_t</code> array of at least the requested size. |
| <i>size</i>   | A <code>size_t</code> 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

|             |                                                           |
|-------------|-----------------------------------------------------------|
| <i>size</i> | A <code>size_t</code> 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

|              |                   |
|--------------|-------------------|
| <i>level</i> | 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

|                      |                                                                                                     |
|----------------------|-----------------------------------------------------------------------------------------------------|
| <i>data_count</i>    | A size_t that indicates how many bytes should be wait from the given serial port or network buffer. |
| <i>timeout</i>       | waiting timeout time                                                                                |
| <i>returned_size</i> | 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

|             |                                                                                      |
|-------------|--------------------------------------------------------------------------------------|
| <i>data</i> | A const reference containing the data to be written to the serial port.              |
| <i>size</i> | 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.

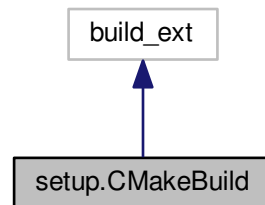
Implemented in [ydlidar::core::network::CSimpleSocket](#), and [ydlidar::core::serial::Serial](#).

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

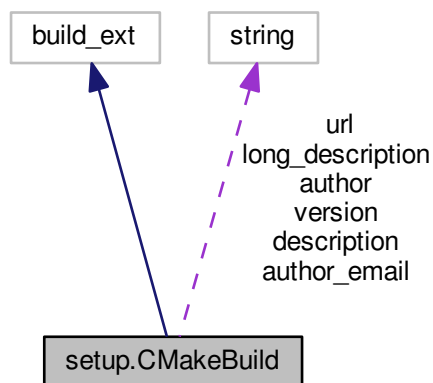
- [core/common/ChannelDevice.h](#)

## 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]`

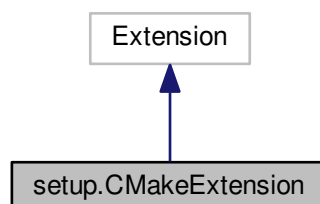
Definition at line 81 of file setup.py.

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

- [setup.py](#)

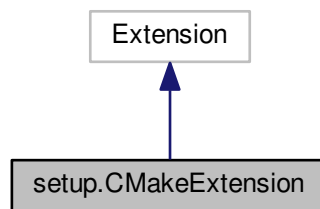
## 38.6 setup.CMakeExtension Class Reference

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`.

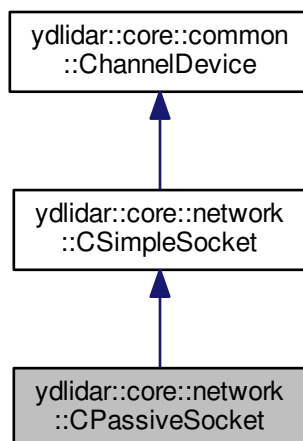
The documentation for this struct was generated from the following file:

- `core/common/ydlidar_protocol.h`

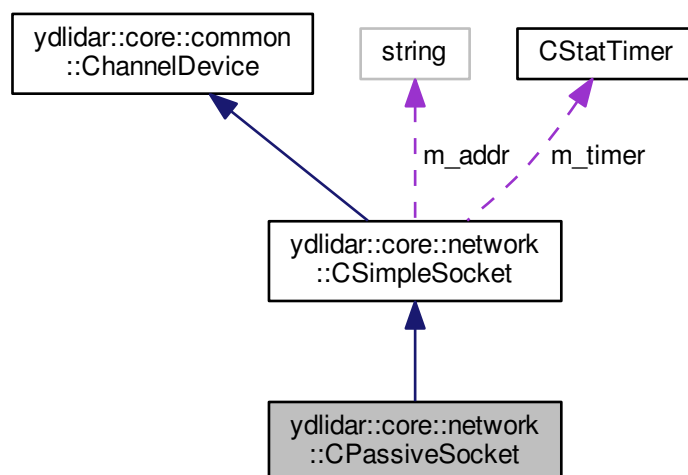
## 38.8 ydlidar::core::network::CPassiveSocket Class Reference

```
#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

|                   |                                    |
|-------------------|------------------------------------|
| <i>pInterface</i> | - interface on which to bind.      |
| <i>pGroup</i>     | - multicast group address to bind. |
| <i>nPort</i>      | - 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 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 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

|                           |                                                     |
|---------------------------|-----------------------------------------------------|
| <i>pAddr</i>              | specifies the IP address on which to listen.        |
| <i>nPort</i>              | specifies the port on which to listen.              |
| <i>nConnectionBacklog</i> | 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

|                    |                                |
|--------------------|--------------------------------|
| <i>pBuf</i>        | block of data to be sent.      |
| <i>bytesToSend</i> | 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](#).

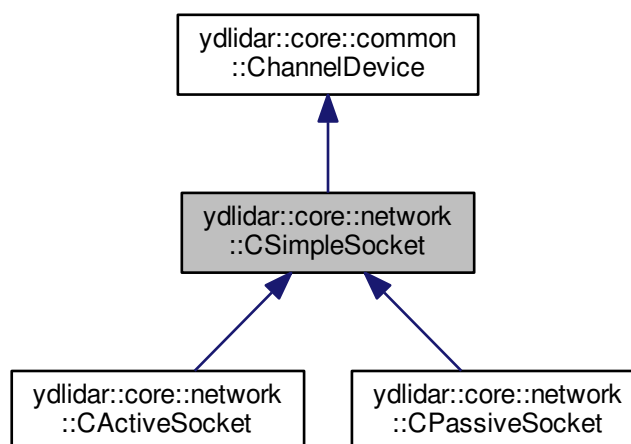
The documentation for this class was generated from the following files:

- [core/network/PassiveSocket.h](#)
- [core/network/PassiveSocket.cpp](#)

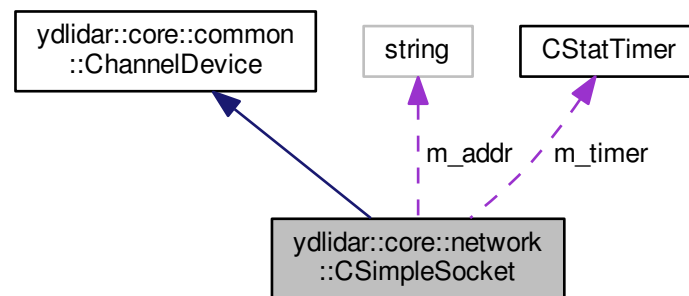
## 38.9 ydlidar::core::network::CSimpleSocket Class Reference

```
#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 shutting 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](#), [SocketTimeout](#), [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 shutting 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](mailto: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 \* *plInterface* )

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 if 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 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 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 occurred for the instance of the [CSimpleSocket](#) instance. This method should be called immediately to retrieve the error code for the failing method call.

**Returns**

last error that occurred.

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

|             |  |
|-------------|--|
| <i>data</i> |  |
| <i>size</i> |  |

## 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

|             |  |
|-------------|--|
| <i>size</i> |  |
|-------------|--|

## 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

|                        |                                                                                                                                                                                               |
|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>nMaxBytes</i>       | maximum number of bytes to receive.                                                                                                                                                           |
| <i>pBuffer, memory</i> | where to receive the data, NULL receives to internal buffer returned with <a href="#">GetData()</a><br>Non-NULL receives directly there, but <a href="#">GetData()</a> 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

|                     |                                      |
|---------------------|--------------------------------------|
| <i>nTimeoutSec</i>  | timeout in seconds for select.       |
| <i>nTimeoutUSec</i> | 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

|                    |                                |
|--------------------|--------------------------------|
| <i>pBuf</i>        | block of data to be sent.      |
| <i>bytesToSend</i> | 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**

|                   |                                                                                                        |
|-------------------|--------------------------------------------------------------------------------------------------------|
| <i>sendVector</i> | pointer to an array of iovec structures                                                                |
| <i>nNumItems</i>  | number of items in the vector to process<br><b>NOTE:</b> 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**

|                |                                                   |
|----------------|---------------------------------------------------|
| <i>nOutFd</i>  | descriptor opened for writing.                    |
| <i>nInFd</i>   | descriptor opened for reading.                    |
| <i>pOffset</i> | from which to start reading data from input file. |
| <i>nCount</i>  | 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

|                            |                             |
|----------------------------|-----------------------------|
| <i>nConnectTimeoutSec</i>  | of timeout in seconds.      |
| <i>nConnectTimeoutUsec</i> | 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 unsent 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

|                |                                                |
|----------------|------------------------------------------------|
| <i>bEnable</i> | true to enable option false to disable option. |
| <i>nTime</i>   | 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

|                         |                             |
|-------------------------|-----------------------------|
| <i>nRecvTimeoutSec</i>  | of timeout in seconds.      |
| <i>nRecvTimeoutUsec</i> | 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

|              |                                                      |
|--------------|------------------------------------------------------|
| <i>nDscp</i> | 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

|              |               |
|--------------|---------------|
| <i>error</i> | 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

|               |                            |
|---------------|----------------------------|
| <i>socket</i> | 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

|                  |                                 |
|------------------|---------------------------------|
| <i>nShutdown</i> | 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

|                      |  |
|----------------------|--|
| <i>data_count</i>    |  |
| <i>timeout</i>       |  |
| <i>returned_size</i> |  |

## 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

|             |  |
|-------------|--|
| <i>data</i> |  |
| <i>size</i> |  |

## 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_bIsBlocking` [protected]

socket flags

Definition at line 663 of file SimpleSocket.h.

**38.9.5.3** `bool ydlidar::core::network::CSimpleSocket::m_bIsMulticast` `[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↔<br>Rate(K) | Range(m)                   | Frequenc↔<br>HZ) | Intenstiy(t↔) | Single↔<br>Channel | voltage(↔<br>V) |
|---------------|-------|----------|--------------------|----------------------------|------------------|---------------|--------------------|-----------------|
| <b>F4</b>     | 1     | 115200   | 4                  | 0.12~12                    | 5~12             | false         | false              | 4.8~5.2         |
| <b>S4</b>     | 4     | 115200   | 4                  | 0.↔<br>10~8.0              | 5~12<br>(PWM)    | false         | false              | 4.8~5.2         |
| <b>S4B</b>    | 4/11  | 153600   | 4                  | 0.↔<br>10~8.0              | 5~12(P↔<br>WM)   | true(8)       | false              | 4.8~5.2         |
| <b>S2</b>     | 4/12  | 115200   | 3                  | 0.↔<br>10~8.0              | 4~8(P↔<br>WM)    | false         | true               | 4.8~5.2         |
| <b>G4</b>     | 5     | 230400   | 9/8/4              | 0.↔<br>28/0.26/0.↔<br>1~16 | 5~12             | false         | false              | 4.8~5.2         |
| <b>X4</b>     | 6     | 128000   | 5                  | 0.12~10                    | 5~12(P↔<br>WM)   | false         | false              | 4.8~5.2         |
| <b>X2/X2L</b> | 6     | 115200   | 3                  | 0.↔<br>10~8.0              | 4~8(P↔<br>WM)    | false         | true               | 4.8~5.2         |
| <b>G4PRO</b>  | 7     | 230400   | 9/8/4              | 0.↔<br>28/0.26/0.↔<br>1~16 | 5~12             | false         | false              | 4.8~5.2         |

| LIDAR | Model | Baudrate | Sample↔<br>Rate(K) | Range(m)                   | Frequency<br>HZ) | Intenstiy(t | Single↔<br>Channel | voltage(↔<br>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.↔<br>28/0.26/0.↔<br>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↔<br>WM)    | false       | true               | 4.8~5.2         |
| TX20  | 100   | 115200   | 4                  | 0.05~20                    | 4~8(P↔<br>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 maximum 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 maximum 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 minimum 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 not 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

|                            |          |
|----------------------------|----------|
| <b>G4/F4</b>               | 4,8,9    |
| <b>F4PRO</b>               | 4,6      |
| <b>G6</b>                  | 8,16,18  |
| <b>G4B</b>                 | 10       |
| <b>G1</b>                  | 9        |
| <b>G2A/G2/R2/X4</b>        | 5        |
| <b>S4/S4B/G4C/TX8/TX20</b> | 4        |
| <b>G2C</b>                 | 4        |
| <b>S2</b>                  | 3        |
| <b>TG15/TG30/TG50</b>      | 10,18,20 |
| <b>T5/T15</b>              | 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 needs to be adjusted by external PWM.  
Set the scan frequency to match the LiDAR.

**Remarks**

unit: Hz

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

**See also**

CYdLidar::lidarSetProp and CYdLidar::lidarGetProp

**LidarPropFixedResolution**

Set and Get LiDAR Fixed angular 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 angular 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                       | 512000 |

## 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 Failed to Start.  
For dual-channel LiDAR, if the settings are reversed.  
the device information cannot be obtained.  
Set the single channel to match the LiDAR.

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

|                                |                               |
|--------------------------------|-------------------------------|
| <b>G1/G2A/G2/G2C</b>           | <a href="#">TYPE_TRIANGLE</a> |
| <b>G4/G4B/G4C/G4PRO</b>        | <a href="#">TYPE_TRIANGLE</a> |
| <b>G6/F4/F4PRO</b>             | <a href="#">TYPE_TRIANGLE</a> |
| <b>S4/S4B/X4/R2/S2/X2/X2L</b>  | <a href="#">TYPE_TRIANGLE</a> |
| <b>TG15/TG30/TG50/TX8/TX20</b> | <a href="#">TYPE_TOF</a>      |
| <b>T5/T15</b>                  | <a href="#">TYPE_TOF_NET</a>  |

## 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.

|                                 |       |
|---------------------------------|-------|
| <b>S4B/G2/G4B</b>               | true  |
| <b>G4/G4C/G4PRO/F4/F4PRO/G6</b> | false |
| <b>G1/G2A/G2C/R2</b>            | false |
| <b>S2/X2/X2L/X4</b>             | false |
| <b>TG15/TG30/TG50</b>           | false |
| <b>TX8/TX20</b>                 | false |
| <b>T5/T15</b>                   | 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.

|                                 |       |
|---------------------------------|-------|
| <b>S4/S4B/S2/X2/X2L/X4</b>      | true  |
| <b>TX8/TX20</b>                 | true  |
| <b>G4/G4C/G4PRO/F4/F4PRO/G6</b> | false |
| <b>G1/G2A/G2C/R2/G2/G4B</b>     | false |
| <b>TG15/TG30/TG50</b>           | false |
| <b>T5/T15</b>                   | 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 | <i>outscan</i>       | LiDAR Scan Data       |
| out | <i>hardwareError</i> | 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

|                |             |
|----------------|-------------|
| <i>optname</i> | 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.setlidaropt(LidarPropSerialPort,&lidar_maxrange, sizeof(float));
```

**Parameters**

|               |                                                                                                                                   |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <i>optval</i> | option value <ul style="list-style-type: none"><li>• std::string(or char*)</li><li>• int</li><li>• bool</li><li>• float</li></ul> |
| <i>optlen</i> | option length <ul style="list-style-type: none"><li>• data type size</li></ul>                                                    |

**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**

|                |                                                                       |
|----------------|-----------------------------------------------------------------------|
| <i>version</i> | 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**

|                |             |
|----------------|-------------|
| <i>optname</i> | 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**

|               |                                                                                                                                   |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <i>optval</i> | option value <ul style="list-style-type: none"><li>• std::string(or char*)</li><li>• int</li><li>• bool</li><li>• float</li></ul> |
| <i>optlen</i> | option length <ul style="list-style-type: none"><li>• data type size</li></ul>                                                    |

**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 Stopped, 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.

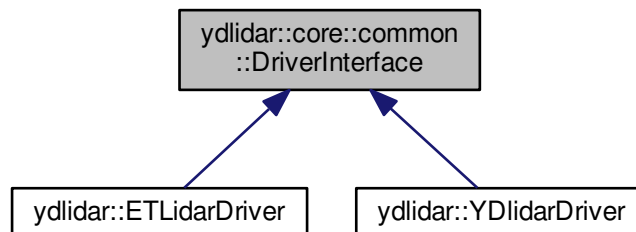
The documentation for this struct was generated from the following file:

- [core/common/ydlidar\\_protocol.h](#)

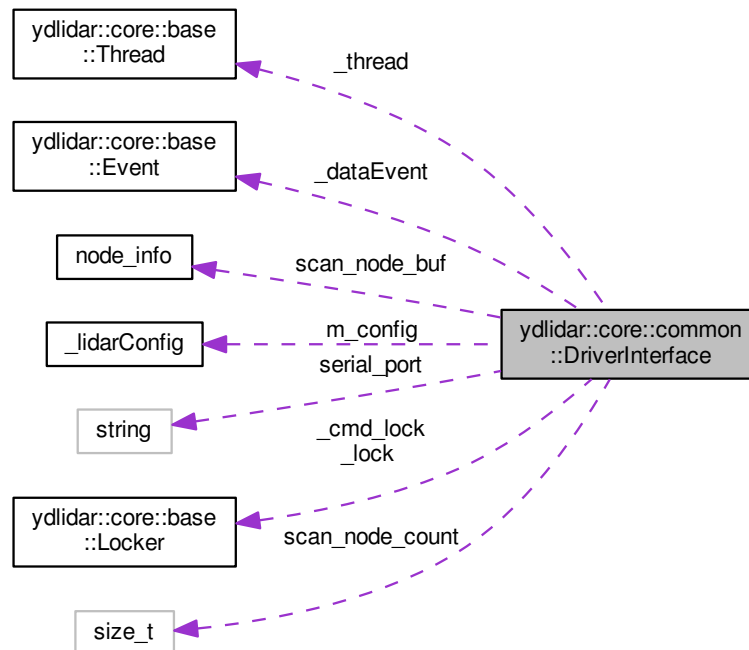
## 38.15 ydlidar::core::common::DriverInterface Class Reference

```
#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 &amp; Destructor Documentation

38.15.3.1 `ydliar::core::common::DriverInterface::DriverInterface ( ) [inline]`

## Constructor

Definition at line 87 of file DriverInterface.h.

38.15.3.2 `virtual ydliar::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 ydliar::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 | <i>port_path</i> | serial port                                         |
| in | <i>baudrate</i>  | 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

|              |            |
|--------------|------------|
| <i>isTCP</i> | 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 | <i>info</i>    | Device information |
| in | <i>timeout</i> | timeout            |

**Returns**

result status

**Return values**

|                     |                          |
|---------------------|--------------------------|
| <i>RESULT_OK</i>    | success                  |
| <i>RESULT_FAILE</i> | 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**

|                     |                          |
|---------------------|--------------------------|
| <i>RESULT_OK</i>    | success                  |
| <i>RESULT_FAILE</i> | 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 | <i>frequency</i> | sampling frequency |
| in | <i>timeout</i>   | timeout            |

#### Returns

return status

#### Return values

|                     |         |
|---------------------|---------|
| <i>RESULT_OK</i>    | success |
| <i>RESULT_FAILE</i> | failed  |

#### Note

Non-scan state, perform current 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 | <i>frequency</i> | scanning frequency |
| in | <i>timeout</i>   | timeout            |

#### Returns

return status

#### Return values

|                     |         |
|---------------------|---------|
| <i>RESULT_OK</i>    | success |
| <i>RESULT_FAILE</i> | failed  |

**Note**

Non-scan state, perform current 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 | <i>angle</i>   | zero offset angle |
| in | <i>timeout</i> | timeout           |

**Returns**

return status

**Return values**

|                     |         |
|---------------------|---------|
| <i>RESULT_OK</i>    | success |
| <i>RESULT_FAILE</i> | failed  |

**Note**

Non-scan state, perform current 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 | <i>nodebuffer</i> | Laser data                 |
| in | <i>count</i>      | one circle of laser points |
| in | <i>timeout</i>    | timeout                    |

**Returns**

return status

**Return values**

|                     |         |
|---------------------|---------|
| <i>RESULT_OK</i>    | success |
| <i>RESULT_FAILE</i> | failed  |

**Note**

Before starting, you must 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**

|              |               |
|--------------|---------------|
| <i>true</i>  | connected     |
| <i>false</i> | 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

|              |              |
|--------------|--------------|
| <i>true</i>  | scanning     |
| <i>false</i> | 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 | <i>enable</i> | 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 | <i>isintensities</i> | 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 | <i>rate</i>    | sampling frequency |
| in | <i>timeout</i> | timeout            |

#### Returns

return status

#### Return values

|                     |         |
|---------------------|---------|
| <i>RESULT_OK</i>    | success |
| <i>RESULT_FAILE</i> | failed  |

#### Note

Non-scan state, perform current 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 | <i>frequency</i> | scanning frequency |
| in | <i>timeout</i>   | timeout            |

#### Returns

return status

#### Return values

|                     |         |
|---------------------|---------|
| <i>RESULT_OK</i>    | success |
| <i>RESULT_FAILE</i> | failed  |

**Note**

Non-scan state, perform current 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 | <i>frequency</i> | scanning frequency |
| in | <i>timeout</i>   | timeout            |

**Returns**

return status

**Return values**

|                     |         |
|---------------------|---------|
| <i>RESULT_OK</i>    | success |
| <i>RESULT_FAILE</i> | failed  |

**Note**

Non-scan state, perform current 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 | <i>frequency</i> | scanning frequency |
| in | <i>timeout</i>   | timeout            |

**Returns**

return status

## Return values

|                     |         |
|---------------------|---------|
| <i>RESULT_OK</i>    | success |
| <i>RESULT_FAILE</i> | failed  |

## Note

Non-scan state, perform current 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 | <i>frequency</i> | scanning frequency |
| in | <i>timeout</i>   | timeout            |

## Returns

return status

## Return values

|                     |         |
|---------------------|---------|
| <i>RESULT_OK</i>    | success |
| <i>RESULT_FAILE</i> | failed  |

## Note

Non-scan state, perform current 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 | <i>force</i>   | Scan mode |
| in | <i>timeout</i> | timeout   |

**Returns**

result status

**Return values**

|                     |         |
|---------------------|---------|
| <i>RESULT_OK</i>    | success |
| <i>RESULT_FAILE</i> | 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**

|                     |         |
|---------------------|---------|
| <i>RESULT_OK</i>    | success |
| <i>RESULT_FAILE</i> | 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.

|                                |                               |
|--------------------------------|-------------------------------|
| <b>G1/G2A/G2/G2C</b>           | <a href="#">TYPE_TRIANGLE</a> |
| <b>G4/G4B/G4C/G4PRO</b>        | <a href="#">TYPE_TRIANGLE</a> |
| <b>G6/F4/F4PRO</b>             | <a href="#">TYPE_TRIANGLE</a> |
| <b>S4/S4B/X4/R2/S2/X2/X2L</b>  | <a href="#">TYPE_TRIANGLE</a> |
| <b>TG15/TG30/TG50/TX8/TX20</b> | <a href="#">TYPE_TOF</a>      |
| <b>T5/T15</b>                  | <a href="#">TYPE_TOF_NET</a>  |

#### 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 = 1 e9 / 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 settlings are reversed.  
the device information cannot be obtained.  
Set the single channel to match the LiDAR.

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

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

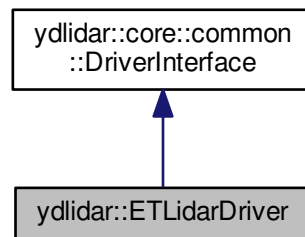
- `core/common/DriverInterface.h`



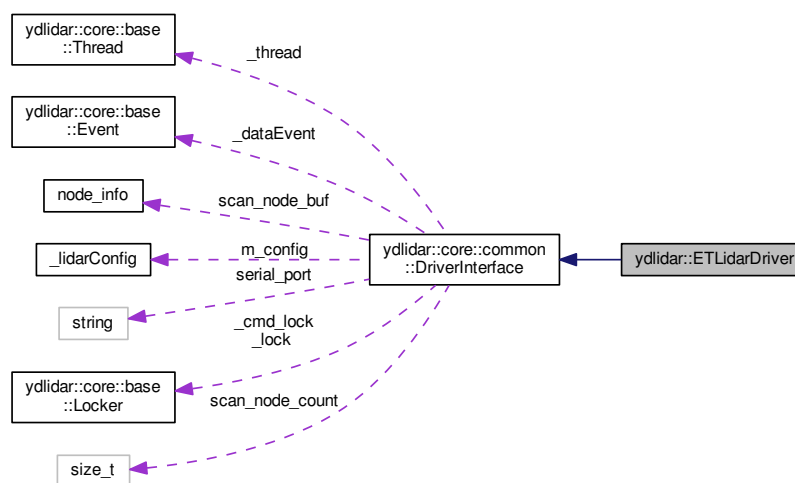
## 38.16 ydlidar::ETLidarDriver Class Reference

```
#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 &intensities)  
*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\_TIMEOUT)  
*Increase the scanning frequency by 0.1 HZ*  
.
  - virtual [result\\_t](#) [setScanFrequencyDisMic](#) ([scan\\_frequency](#) &frequency, uint32\_t timeout=DEFAULT\_TIMEOUT)  
*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

|                |  |
|----------------|--|
| <i>lidar</i> ↔ |  |
| <i>IP</i>      |  |
| <i>port</i>    |  |

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 | <i>port_path</i> | Ip Address   |
| in | <i>baudrate</i>  | 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**

|              |            |
|--------------|------------|
| <i>isTCP</i> | 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 | <i>info</i>    | Device information |
| in | <i>timeout</i> | timeout            |

## Returns

result status

## Return values

|                     |                          |
|---------------------|--------------------------|
| <i>RESULT_OK</i>    | success                  |
| <i>RESULT_FAILE</i> | 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

|                     |                          |
|---------------------|--------------------------|
| <i>RESULT_OK</i>    | success                  |
| <i>RESULT_FAILE</i> | 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 | <i>frequency</i> | sampling frequency |
| in | <i>timeout</i>   | timeout            |

#### Returns

return status

#### Return values

|                     |         |
|---------------------|---------|
| <i>RESULT_OK</i>    | success |
| <i>RESULT_FAILE</i> | failed  |

#### Note

Non-scan state, perform current 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 | <i>frequency</i> | scanning frequency |
| in | <i>timeout</i>   | timeout            |

**Returns**

return status

**Return values**

|                     |         |
|---------------------|---------|
| <i>RESULT_OK</i>    | success |
| <i>RESULT_FAILE</i> | failed  |

**Note**

Non-scan state, perform current 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 | <i>angle</i>   | zero offset angle |
| in | <i>timeout</i> | timeout           |

**Returns**

return status

**Return values**

|                     |         |
|---------------------|---------|
| <i>RESULT_OK</i>    | success |
| <i>RESULT_FAILE</i> | failed  |

**Note**

Non-scan state, perform current 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 | <i>nodebuffer</i> | Laser data                 |
| in | <i>count</i>      | one circle of laser points |
| in | <i>timeout</i>    | timeout                    |

**Returns**

return status

**Return values**

|                     |         |
|---------------------|---------|
| <i>RESULT_OK</i>    | success |
| <i>RESULT_FAILE</i> | 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**

|              |               |
|--------------|---------------|
| <i>true</i>  | connected     |
| <i>false</i> | 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

|              |              |
|--------------|--------------|
| <i>true</i>  | scanning     |
| <i>false</i> | 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 | <i>enable</i> | 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 | <i>isintensities</i> | intensity 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 | <i>rate</i>    | sampling frequency |
| in | <i>timeout</i> | timeout            |

#### Returns

return status

#### Return values

|                     |         |
|---------------------|---------|
| <i>RESULT_OK</i>    | success |
| <i>RESULT_FAILE</i> | failed  |

#### Note

Non-scan state, perform current 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 | <i>frequency</i> | scanning frequency |
| in | <i>timeout</i>   | timeout            |

#### Returns

return status

#### Return values

|                     |         |
|---------------------|---------|
| <i>RESULT_OK</i>    | success |
| <i>RESULT_FAILE</i> | failed  |

**Note**

Non-scan state, perform current 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 | <i>frequency</i> | scanning frequency |
| in | <i>timeout</i>   | timeout            |

**Returns**

return status

**Return values**

|                     |         |
|---------------------|---------|
| <i>RESULT_OK</i>    | success |
| <i>RESULT_FAILE</i> | failed  |

**Note**

Non-scan state, perform current 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 | <i>frequency</i> | scanning frequency |
| in | <i>timeout</i>   | timeout            |

**Returns**

return status

**Return values**

|                     |         |
|---------------------|---------|
| <i>RESULT_OK</i>    | success |
| <i>RESULT_FAILE</i> | failed  |

**Note**

Non-scan state, perform current 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 | <i>frequency</i> | scanning frequency |
| in | <i>timeout</i>   | timeout            |

**Returns**

return status

**Return values**

|                     |         |
|---------------------|---------|
| <i>RESULT_OK</i>    | success |
| <i>RESULT_FAILE</i> | failed  |

**Note**

Non-scan state, perform current 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 | <i>force</i>   | Scan mode |
| in | <i>timeout</i> | timeout   |

## Returns

result status

## Return values

|                     |         |
|---------------------|---------|
| <i>RESULT_OK</i>    | success |
| <i>RESULT_FAILE</i> | 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

|                     |         |
|---------------------|---------|
| <i>RESULT_OK</i>    | success |
| <i>RESULT_FAILE</i> | 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

|               |  |
|---------------|--|
| <i>config</i> |  |
|---------------|--|

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\\_catrr](#)
- 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:

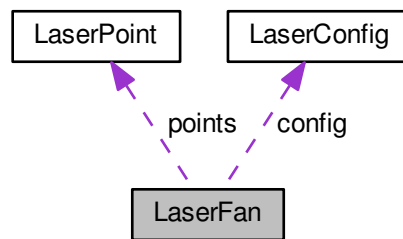
- [core/common/ydlidar\\_datatype.h](#)

## 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:

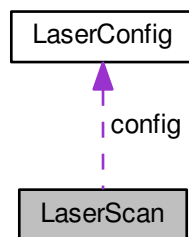
- [core/common/ydlidar\\_def.h](#)

## 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:

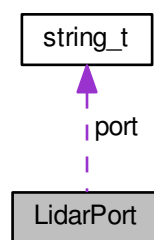
- [core/common/ydlidar\\_protocol.h](#)

## 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 `ydliar_def.h`.

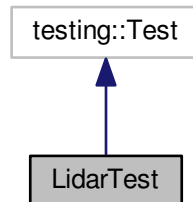
The documentation for this struct was generated from the following file:

- [core/common/ydliar\\_def.h](#)

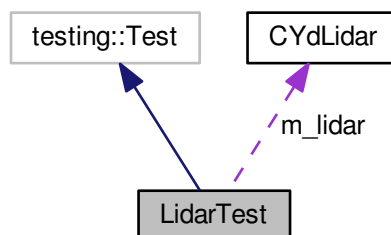
### 38.27 LidarTest Class Reference

```
#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 `ydliar_def.h`.

### 38.28.2 Member Data Documentation

#### 38.28.2.1 `uint8_t LidarVersion::hardware`

Hardware version

Definition at line 200 of file `ydliar_def.h`.

#### 38.28.2.2 `uint8_t LidarVersion::sn[16]`

serial number

Definition at line 204 of file `ydliar_def.h`.

#### 38.28.2.3 `uint8_t LidarVersion::soft_major`

major number

Definition at line 201 of file `ydliar_def.h`.

#### 38.28.2.4 `uint8_t LidarVersion::soft_minor`

minor number

Definition at line 202 of file `ydliar_def.h`.

#### 38.28.2.5 `uint8_t LidarVersion::soft_patch`

patch number

Definition at line 203 of file `ydliar_def.h`.

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

- `core/common/ydliar_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\\_frequency](#)  
*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:

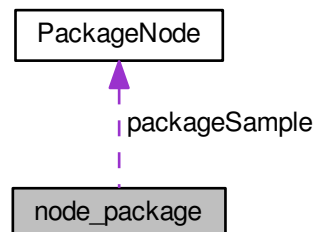
- [core/common/ydlidar\\_protocol.h](#)

## 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 [PackageSampleQuality](#)  
*intensity*
- uint16\_t [PackageSampleDistance](#)  
*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::PackageSampleDistance

range

Definition at line 168 of file ydlidar\_protocol.h.

#### 38.35.2.2 uint8\_t PackageNode::PackageSampleQuality

intensity

Definition at line 167 of file ydlidar\_protocol.h.

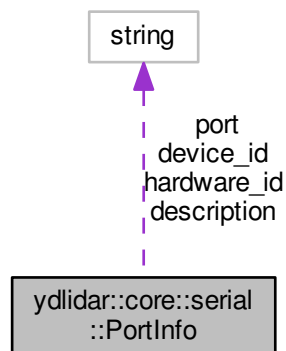
The documentation for this struct was generated from the following file:

- [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](#)).

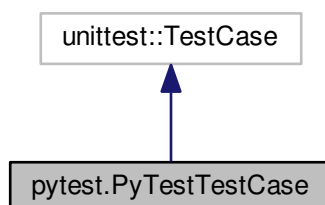
Definition at line 684 of file serial.h.

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

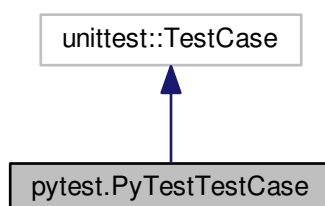
- `core/serial/serial.h`

## 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.testParametersIsWrappedCorrectly ( 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 `ydliidar_protocol.h`.

### 38.42.2 Member Data Documentation

#### 38.42.2.1 `uint8_t scan_points::flag`

Definition at line 232 of file `ydliidar_protocol.h`.

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

- `core/common/ydliidar_protocol.h`

## 38.43 `scan_rotation` Struct Reference

```
#include <ydliidar_protocol.h>
```

## Public Attributes

- `uint8_t rotation`

### 38.43.1 Detailed Description

Definition at line 217 of file `ydliidar_protocol.h`.

### 38.43.2 Member Data Documentation

#### 38.43.2.1 `uint8_t scan_rotation::rotation`

Definition at line 218 of file `ydliidar_protocol.h`.

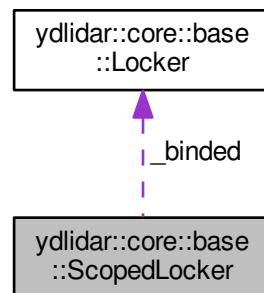
The documentation for this struct was generated from the following file:

- `core/common/ydliidar_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.

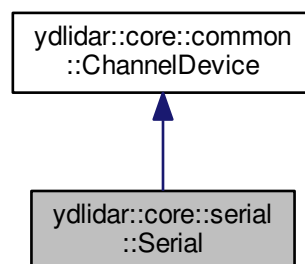
The documentation for this class was generated from the following file:

- core/serial/[serial.cpp](#)

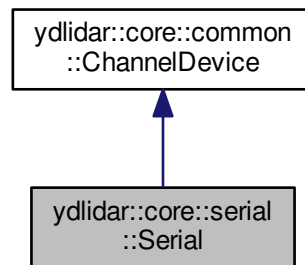
## 38.47 ydlidar::core::serial::Serial Class Reference

```
#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 setTimeout` () 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

|                 |                                                                                                                                     |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------|
| <i>port</i>     | A std::string containing the address of the serial port, which would be something like 'COM1' on Windows and '/dev/ttyS0' on Linux. |
| <i>baudrate</i> | An unsigned 32-bit integer that represents the baudrate                                                                             |
| <i>timeout</i>  | A <a href="#">serial::Timeout</a> struct that defines the timeout conditions for the serial port.                                   |

See also

[serial::Timeout](#)

## Parameters

|                    |                                                                                                                                          |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------|
| <i>bytesize</i>    | Size of each byte in the serial transmission of data, default is eightbits, possible values are: fivebits, sixbits, sevenbits, eightbits |
| <i>parity</i>      | Method of parity, default is parity_none, possible values are: parity_none, parity_odd, parity_even                                      |
| <i>stopbits</i>    | Number of stop bits used, default is stopbits_one, possible values are: stopbits_one, stopbits_one_point_five, stopbits_two              |
| <i>flowcontrol</i> | Type of flowcontrol used, default is flowcontrol_none, possible values are: flowcontrol_none, flowcontrol_software, flowcontrol_hardware |

## Exceptions

|                                       |  |
|---------------------------------------|--|
| <i>serial::PortNotOpenedException</i> |  |
| <i>serial::IOException</i>            |  |
| <i>std::invalid_argument</i>          |  |

## 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

|                                    |  |
|------------------------------------|--|
| <code>std::invalid_argument</code> |  |
|------------------------------------|--|

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

|                              |  |
|------------------------------|--|
| <code>systemErrorCode</code> |  |
|------------------------------|--|

## 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

|               |                                                  |
|---------------|--------------------------------------------------|
| <i>buffer</i> | An uint8_t array of at least the requested size. |
| <i>size</i>   | 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

|               |                                              |
|---------------|----------------------------------------------|
| <i>buffer</i> | A reference to a std::vector of uint8_t.     |
| <i>size</i>   | 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

|               |                                              |
|---------------|----------------------------------------------|
| <i>buffer</i> | A reference to a std::string.                |
| <i>size</i>   | 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

|               |                                                  |
|---------------|--------------------------------------------------|
| <i>buffer</i> | An uint8_t array of at least the requested size. |
| <i>size</i>   | 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**

|               |                                                            |
|---------------|------------------------------------------------------------|
| <i>buffer</i> | A std::string reference used to store the data.            |
| <i>size</i>   | A maximum length of a line, defaults to 65536 ( $2^{16}$ ) |
| <i>eol</i>    | 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**

|             |                                                            |
|-------------|------------------------------------------------------------|
| <i>size</i> | A maximum length of a line, defaults to 65536 ( $2^{16}$ ) |
| <i>eol</i>  | 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**

|             |                                                                    |
|-------------|--------------------------------------------------------------------|
| <i>size</i> | A maximum length of combined lines, defaults to 65536 ( $2^{16}$ ) |
| <i>eol</i>  | 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

|             |                                                           |
|-------------|-----------------------------------------------------------|
| <i>size</i> | A <code>size_t</code> 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

|                 |                                                         |
|-----------------|---------------------------------------------------------|
| <i>baudrate</i> | 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

|                 |                                                                                                                                          |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------|
| <i>bytesize</i> | 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

|                    |                                                                                                                                          |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------|
| <i>flowcontrol</i> | 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

|               |                                                                                                     |
|---------------|-----------------------------------------------------------------------------------------------------|
| <i>parity</i> | 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

|             |                                                                                                                                                     |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>port</i> | 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

|                                    |  |
|------------------------------------|--|
| <code>std::invalid_argument</code> |  |
|------------------------------------|--|

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

|                 |                                                                                                                             |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------|
| <i>stopbits</i> | 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

|                |                                                                                                                                       |
|----------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <i>timeout</i> | A <a href="#">serial::Timeout</a> 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 conjunction 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 TIOCMWAIT 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

|                      |                                                                                                     |
|----------------------|-----------------------------------------------------------------------------------------------------|
| <i>data_count</i>    | A size_t that indicates how many bytes should be wait from the given serial port or network buffer. |
| <i>timeout</i>       | waiting timeout time                                                                                |
| <i>returned_size</i> | 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

|             |                                                                                      |
|-------------|--------------------------------------------------------------------------------------|
| <i>data</i> | A const reference containing the data to be written to the serial port.              |
| <i>size</i> | 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

|                                       |  |
|---------------------------------------|--|
| <i>serial::PortNotOpenedException</i> |  |
| <i>serial::SerialException</i>        |  |
| <i>serial::IOException</i>            |  |

Definition at line 275 of file serial.cpp.

## 38.47.4.48 size\_t ydlidar::core::serial::Serial::write ( const std::vector&lt; uint8\_t &gt; &amp; data )

Write a string to the serial port.

## Parameters

|             |                                                                         |
|-------------|-------------------------------------------------------------------------|
| <i>data</i> | 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

|             |                                                                         |
|-------------|-------------------------------------------------------------------------|
| <i>data</i> | 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

|             |                                                                                                   |
|-------------|---------------------------------------------------------------------------------------------------|
| <i>data</i> | A const reference containing the data to be written to the serial port.                           |
| <i>size</i> | A <code>size_t</code> 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::ThreadCreate↵`  
`ObjectFunctor ( 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** `ydliar::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 ydliar::core::serial::Timeout::max ( ) [inline], [static]`

Definition at line 67 of file serial.h.

**38.52.3.2** `static Timeout ydliar::core::serial::Timeout::simpleTimeout ( uint32_t timeout ) [inline], [static]`

Convenience function to generate [Timeout](#) structs using a single absolute timeout.

#### Parameters

|                |                                                                                                            |
|----------------|------------------------------------------------------------------------------------------------------------|
| <i>timeout</i> | 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 ydliar::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 ydliar::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 ydliar::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.

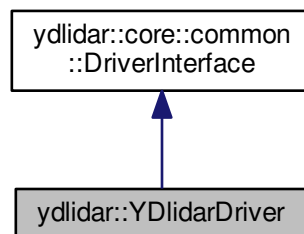
The documentation for this struct was generated from the following file:

- [core/common/ydlidar\\_def.h](#)

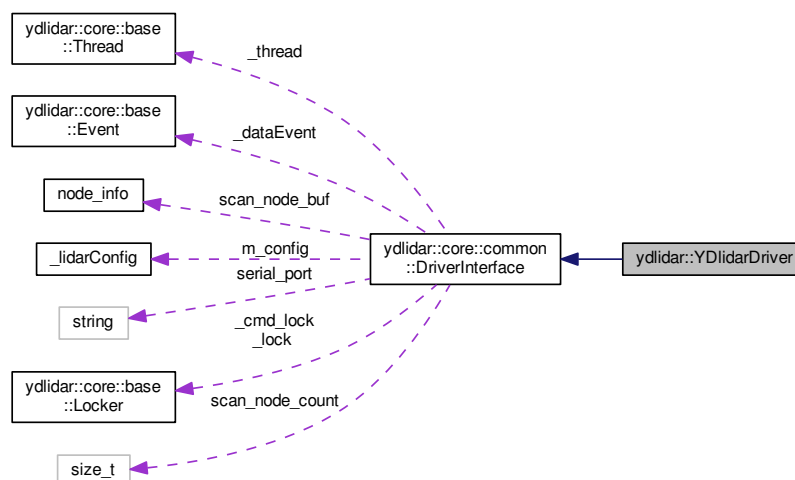
## 38.54 ydlidar::YDlidarDriver Class Reference

```
#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 &intensities)  
*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_TIMEOUT)`  
*Increase the scanning frequency by 0.1 HZ*
- virtual `result_t setScanFrequencyDisMic (scan_frequency &frequency, uint32_t timeout=DEFAULT_TIMEOUT)`  
*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 | <i>nodebuffer</i> | Laser data                 |
| in | <i>count</i>      | one circle of laser points |

## Returns

return status

## Return values

|                     |         |
|---------------------|---------|
| <i>RESULT_OK</i>    | success |
| <i>RESULT_FAILE</i> | 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

|                   |                     |
|-------------------|---------------------|
| <i>recvBuffer</i> | LiDAR Data buffer   |
| <i>byte</i>       | current byte        |
| <i>recvPos</i>    | current recived pos |
| <i>recvSize</i>   | Buffer size         |
| <i>pos</i>        | Device Buffer pos   |

**Returns**

status

**Return values**

|                     |         |
|---------------------|---------|
| <i>RESULT_OK</i>    | success |
| <i>RESULT_FAILE</i> | 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 | <i>port_path</i> | serial port                                         |
| in | <i>baudrate</i>  | 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 data 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

|                    |            |
|--------------------|------------|
| <code>isTCP</code> | 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 | <i>angle</i>   | zero offset angle |
| in | <i>timeout</i> | timeout           |

#### Returns

return status

#### Return values

|                     |         |
|---------------------|---------|
| <i>RESULT_OK</i>    | success |
| <i>RESULT_FAILE</i> | failed  |

#### Note

scanning state,perform current 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 | <i>data</i> | data      |
| in | <i>size</i> | date size |

#### Returns

return status



## Return values

|                     |         |
|---------------------|---------|
| <i>RESULT_OK</i>    | success |
| <i>RESULT_FAILE</i> | 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 | <i>info</i>    | Device information |
| in | <i>timeout</i> | timeout            |

## Returns

result status

## Return values

|                     |                          |
|---------------------|--------------------------|
| <i>RESULT_OK</i>    | success                  |
| <i>RESULT_FAILE</i> | 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

|                     |                          |
|---------------------|--------------------------|
| <i>RESULT_OK</i>    | success                  |
| <i>RESULT_FAILE</i> | 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 | <i>frequency</i> | sampling frequency |
| in | <i>timeout</i>   | timeout            |

#### Returns

return status

#### Return values

|                     |         |
|---------------------|---------|
| <i>RESULT_OK</i>    | success |
| <i>RESULT_FAILE</i> | failed  |

#### Note

Non-scan state, perform current 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 | <i>frequency</i> | scanning frequency |
| in | <i>timeout</i>   | timeout            |

#### Returns

return status

#### Return values

|                     |         |
|---------------------|---------|
| <i>RESULT_OK</i>    | success |
| <i>RESULT_FAILE</i> | failed  |

**Note**

Non-scan state, perform current 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 | <i>angle</i>   | zero offset angle |
| in | <i>timeout</i> | timeout           |

**Returns**

return status

**Return values**

|                     |         |
|---------------------|---------|
| <i>RESULT_OK</i>    | success |
| <i>RESULT_FAILE</i> | failed  |

**Note**

Non-scan state, perform current 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 | <i>nodebuffer</i> | Laser data                 |
| in | <i>count</i>      | one circle of laser points |
| in | <i>timeout</i>    | timeout                    |

#### Returns

return status

#### Return values

|                     |         |
|---------------------|---------|
| <i>RESULT_OK</i>    | success |
| <i>RESULT_FAILE</i> | failed  |

#### Note

Before starting, you must 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

|              |               |
|--------------|---------------|
| <i>true</i>  | connected     |
| <i>false</i> | 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

|              |              |
|--------------|--------------|
| <i>true</i>  | scanning     |
| <i>false</i> | 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 | <i>timeout</i> | timeout |
|----|----------------|---------|

#### Returns

return status

#### Return values

|                     |         |
|---------------------|---------|
| <i>RESULT_OK</i>    | success |
| <i>RESULT_FAILE</i> | failed  |

**Note**

Non-scan state, perform current 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 | <i>cmd</i>         | command code |
| in | <i>payload</i>     | payload      |
| in | <i>payloadsize</i> | payloadsize  |

**Returns**

result status

**Return values**

|                     |         |
|---------------------|---------|
| <i>RESULT_OK</i>    | success |
| <i>RESULT_FAILE</i> | 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 | <i>data</i> | data      |
| in | <i>size</i> | data size |

**Returns**

return status

**Return values**

|                     |         |
|---------------------|---------|
| <i>RESULT_OK</i>    | success |
| <i>RESULT_FAILE</i> | 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 | <i>enable</i> | hot plug : true support false no support |
| in | <i>enable</i> | 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 | <i>isintensities</i> | intensity 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 | <i>rate</i>    | sampling frequency |
| in | <i>timeout</i> | timeout            |

**Returns**

return status

**Return values**

|                     |         |
|---------------------|---------|
| <i>RESULT_OK</i>    | success |
| <i>RESULT_FAILE</i> | failed  |

**Note**

Non-scan state, perform current 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 | <i>frequency</i> | scanning frequency |
| in | <i>timeout</i>   | timeout            |

**Returns**

return status

**Return values**

|                     |         |
|---------------------|---------|
| <i>RESULT_OK</i>    | success |
| <i>RESULT_FAILE</i> | failed  |

**Note**

Non-scan state, perform current 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 | <i>frequency</i> | scanning frequency |
| in | <i>timeout</i>   | timeout            |

## Returns

return status

## Return values

|                     |         |
|---------------------|---------|
| <i>RESULT_OK</i>    | success |
| <i>RESULT_FAILE</i> | failed  |

## Note

Non-scan state, perform current 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 | <i>frequency</i> | scanning frequency |
| in | <i>timeout</i>   | timeout            |

## Returns

return status

## Return values

|                     |         |
|---------------------|---------|
| <i>RESULT_OK</i>    | success |
| <i>RESULT_FAILE</i> | failed  |

## Note

Non-scan state, perform current 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 | <i>frequency</i> | scanning frequency |
| in | <i>timeout</i>   | timeout            |

#### Returns

return status

#### Return values

|                     |         |
|---------------------|---------|
| <i>RESULT_OK</i>    | success |
| <i>RESULT_FAILE</i> | failed  |

#### Note

Non-scan state, perform current 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 | <i>force</i>   | scan model |
| in | <i>timeout</i> | timeout    |

#### Returns

return status

#### Return values

|                     |         |
|---------------------|---------|
| <i>RESULT_OK</i>    | success |
| <i>RESULT_FAILE</i> | 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**

|                     |         |
|---------------------|---------|
| <i>RESULT_OK</i>    | success |
| <i>RESULT_FAILE</i> | 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 | <i>force</i>   | Scan mode |
| in | <i>timeout</i> | timeout   |

**Returns**

result status

**Return values**

|                     |         |
|---------------------|---------|
| <i>RESULT_OK</i>    | success |
| <i>RESULT_FAILE</i> | 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

|                     |         |
|---------------------|---------|
| <i>RESULT_OK</i>    | success |
| <i>RESULT_FAILE</i> | 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

|                     |         |
|---------------------|---------|
| <i>RESULT_OK</i>    | success |
| <i>RESULT_FAILE</i> | 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

|                |         |
|----------------|---------|
| <i>timeout</i> | timeout |
|----------------|---------|

#### Returns

status

## Return values

|                     |         |
|---------------------|---------|
| <i>RESULT_OK</i>    | success |
| <i>RESULT_FAILE</i> | 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

|                |         |
|----------------|---------|
| <i>timeout</i> | timeout |
|----------------|---------|

## Returns

status

## Return values

|                     |         |
|---------------------|---------|
| <i>RESULT_OK</i>    | success |
| <i>RESULT_FAILE</i> | 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 | <i>data_count</i>    | wait max data size |
| in | <i>timeout</i>       | timeout            |
| in | <i>returned_size</i> | really data size   |

## Returns

return status

## Return values

|                       |              |
|-----------------------|--------------|
| <i>RESULT_OK</i>      | success      |
| <i>RESULT_TIMEOUT</i> | wait timeout |
| <i>RESULT_FAILE</i>   | 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 | <i>node</i>    | lidar point information |
| in | <i>timeout</i> | 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 | <i>header</i>  | package header |
| in | <i>timeout</i> | timeout        |

**Returns**

return status

**Return values**

|                       |         |
|-----------------------|---------|
| <i>RESULT_OK</i>      | success |
| <i>RESULT_TIMEOUT</i> | timeout |
| <i>RESULT_FAILE</i>   | 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 | <i>nodebuffer</i> | laser node        |
| in | <i>count</i>      | lidar points size |
| in | <i>timeout</i>    | timeout           |

#### Returns

result status

#### Return values

|                       |         |
|-----------------------|---------|
| <i>RESULT_OK</i>      | success |
| <i>RESULT_TIMEOUT</i> | timeout |
| <i>RESULT_FAILE</i>   | 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](#)





## File Documentation

```
#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>
```

```

graph TD
    core[core/base/datatype.h] --> string_h[string.h]
    core --> string[string]
    core --> signal_h[signal.h]
    core --> errno[cerrno]
    core --> stdexcept[stdexcept]
    core --> csignal[csignal]
    core --> sys_stat_h[sys/stat.h]
    core --> typedef_h[typedef.h]
    core --> unistd_h[unistd.h]
    string_h --> stddef_h[stddef.h]
    string_h --> stdio_h[stdio.h]
    string --> string_h
    signal_h --> signal_h
    errno --> errno
    stdexcept --> stdexcept
    csignal --> csignal
    sys_stat_h --> sys_stat_h
    typedef_h --> typedef_h
    unistd_h --> unistd_h
    typedef_h --> stdint_h[stdint.h]
    unistd_h --> stdint_h
  
```

[illegible]

## 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.

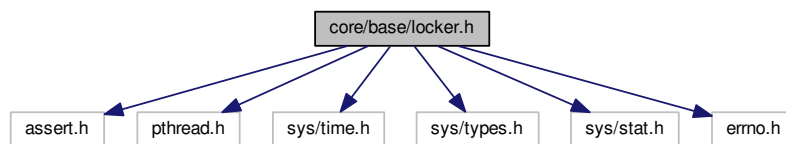
**39.1.2 Typedef Documentation****39.1.2.1 typedef int32\_t result\_t**

Definition at line 135 of file datatype.h.

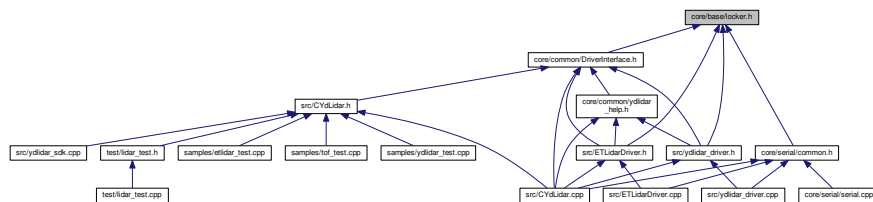
## 39.2 core/base/locker.h File Reference

```
#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

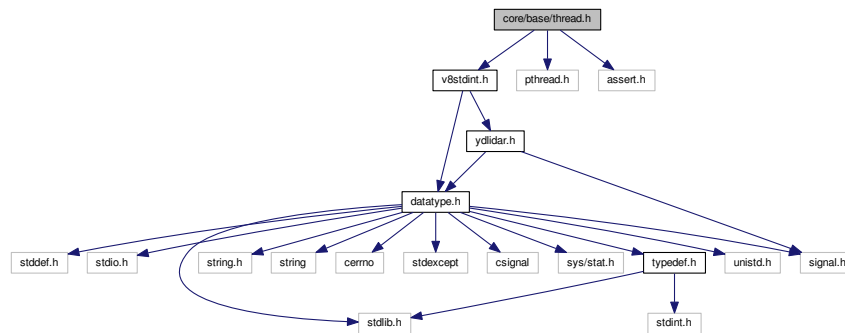
- class [ydlidar::core::base::Locker](#)
- class [ydlidar::core::base::Event](#)
- class [ydlidar::core::base::ScopedLocker](#)

## Namespaces

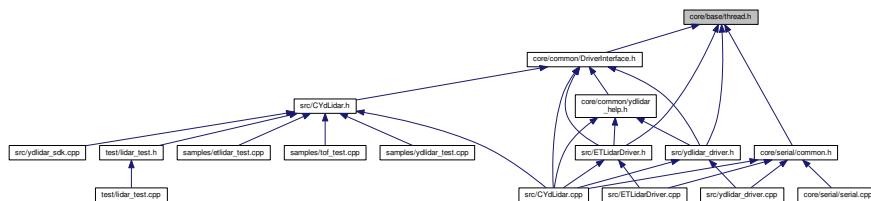
- [ydlidar](#)  
*ydlidar*
- [ydlidar::core](#)  
*ydlidar core*
- [ydlidar::core::base](#)

### 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

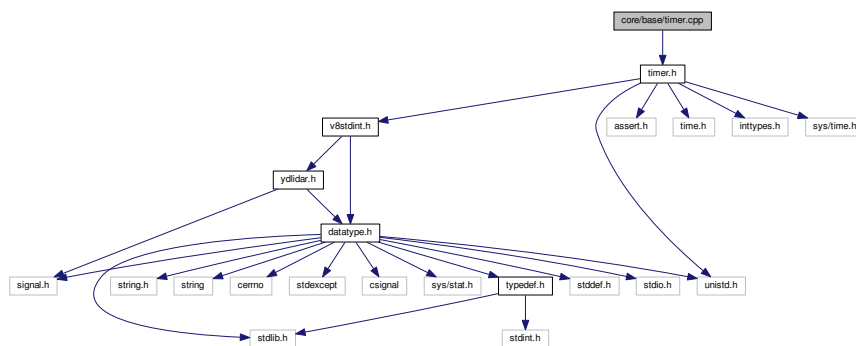
#### 39.3.1.1 `#define CLASS_THREAD( c, x ) Thread::ThreadCreateObjectFunctor<c, &c::x>(this)`

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](#)

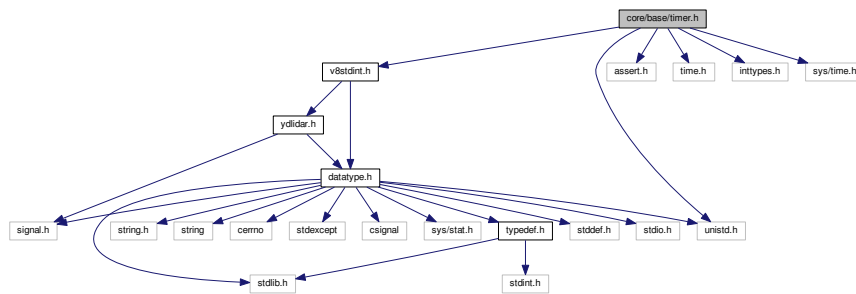
### Functions

- `uint32_t impl::getHDTimer ()`
- `uint64_t impl::getCurrentTime ()`

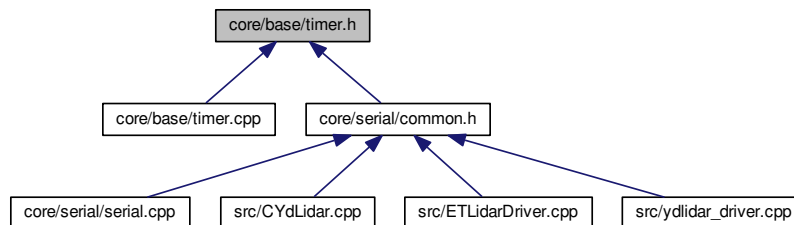
## 39.5 core/base/timer.h File Reference

```
#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]`

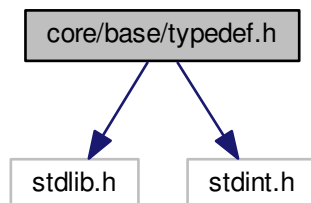
Definition at line 25 of file timer.h.

## 39.6 core/base/typedef.h File Reference

```
#include <stdlib.h>
```

```
#include <stdint.h>
```

Include dependency graph for typedef.h:



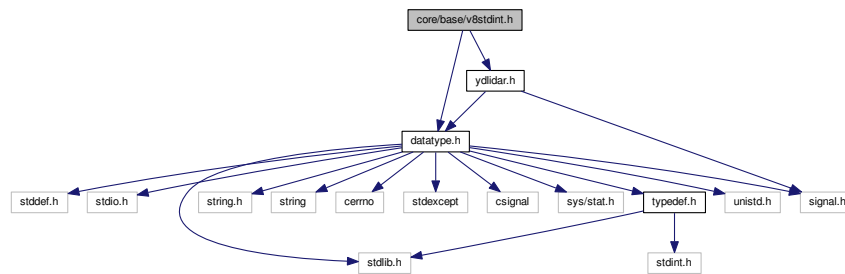
```
graph BT; core_base_utlis_h[core/base/utlis.h] --> src_CYdLidar_h[src/CYdLidar.h]; samples_etlidar_test_cpp[samples/etlidar_test.cpp] --> src_CYdLidar_h; samples_tof_test_cpp[samples/tof_test.cpp] --> src_CYdLidar_h; samples_ydlidar_test_cpp[samples/ydlidar_test.cpp] --> src_CYdLidar_h; src_CYdLidar_cpp[src/CYdLidar.cpp] --> src_CYdLidar_h; src_ydlidar_sdk_cpp[src/ydlidar_sdk.cpp] --> src_CYdLidar_h; test_lidar_test_h[test/lidar_test.h] --> src_CYdLidar_h; test_lidar_test_cpp[test/lidar_test.cpp] --> test_lidar_test_h;
```

- `#define YDLIDAR_API`

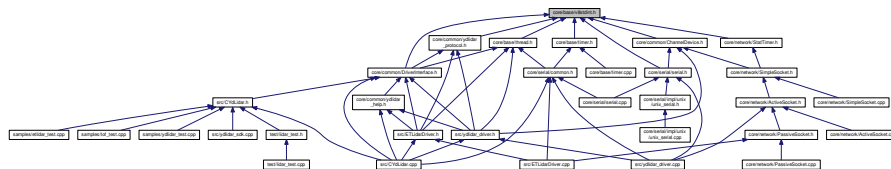
- ### 39.7.1 Macro Definition Documentation

Generated by Doxygen

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 htonl(x) (((uint64)htonl(x)) << 32 + htonl(x) >> 32)`
- `#define ntohl(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 NANOSECONDS_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 NANOSECONDS_CONVERSION 1000000000`

Definition at line 140 of file v8stdint.h.

39.8.1.21 `#define ntohl( 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.



## 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, private, 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.

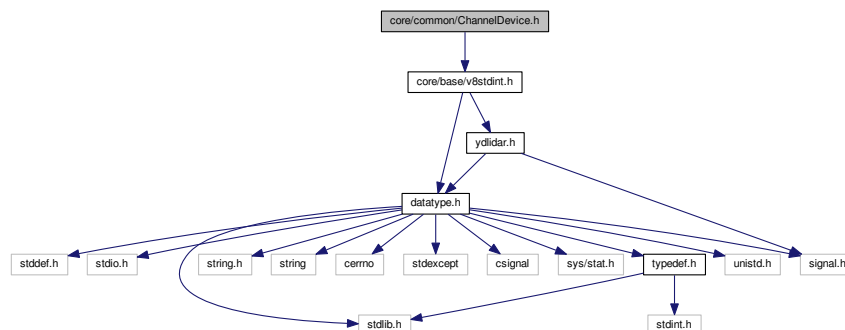
39.9.4.2 `signal_handler_t old_signal_handler = 0` `[static]`

Definition at line 33 of file ydlidar.h.

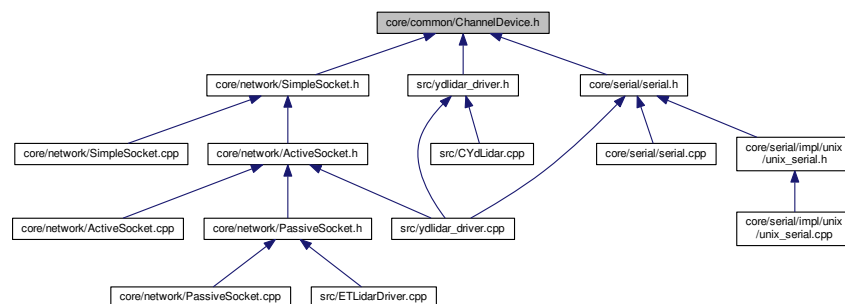
## 39.10 core/common/ChannelDevice.h File Reference

```
#include <core/base/v8stdint.h>
```

Include dependency graph for ChannelDevice.h:



This graph shows which files directly or indirectly include this file:



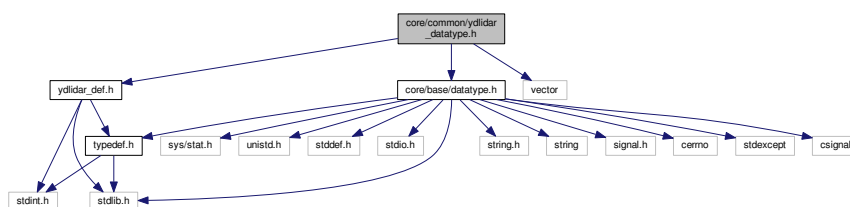


## Namespaces

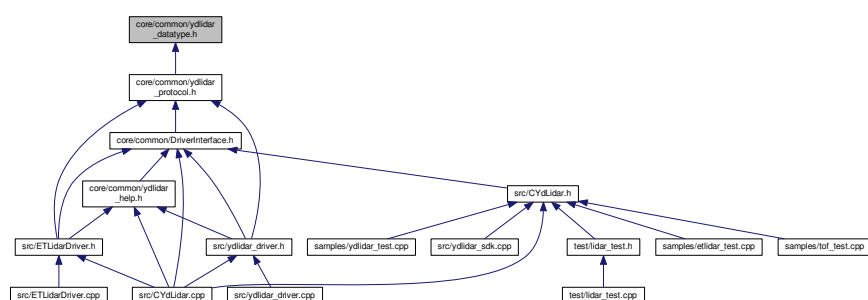
- `ydlidar`  
*ydlidar*
- `ydlidar::core`  
*ydlidar core*
- `ydlidar::core::common`  
*ydlidar common*

## 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

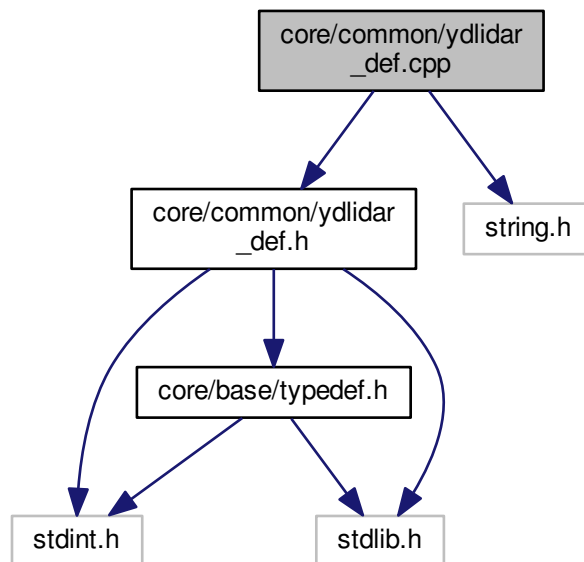
- struct `LaserDebug`  
*The Laser Debug struct.*
- struct `LaserScan`  
*The Laser Scan Data struct.*

### 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

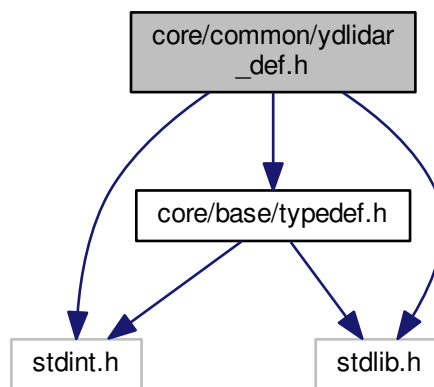
`to_init`

Definition at line 28 of file ydlidar\_def.cpp.

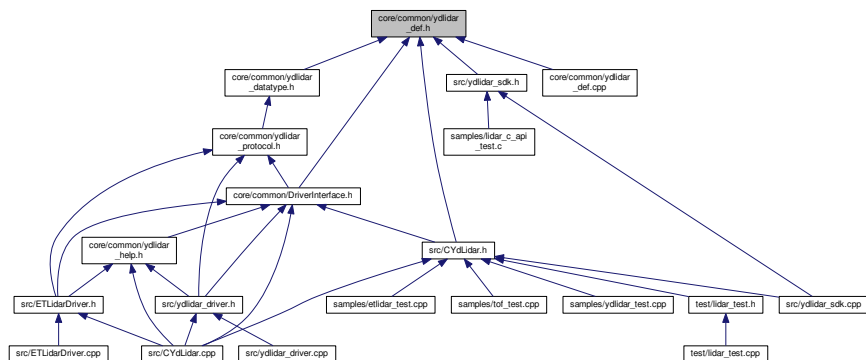
## 39.14 core/common/ydlidar\_def.h File Reference

```
#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](#), [LidarPropIntensitiy](#), [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](#)

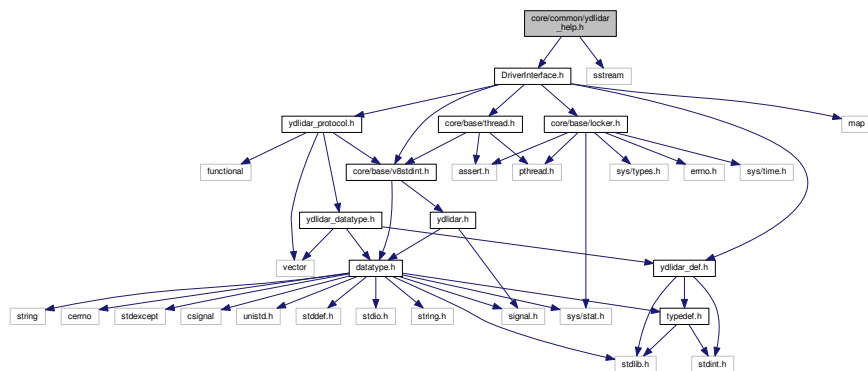
## Parameters

|                      |
|----------------------|
| <code>to_init</code> |
|----------------------|

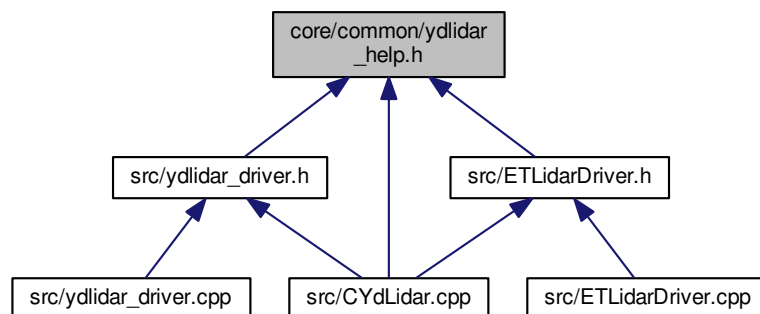
Definition at line 28 of file `ydliar_def.cpp`.

### 39.15 core/common/ydliar\_help.h File Reference

```
#include "DriverInterface.h"
#include <sstream>
Include dependency graph for ydliar_help.h:
```



This graph shows which files directly or indirectly include this file:



### Namespaces

- [ydliar](#)  
*ydliar*
- [ydliar::core](#)  
*ydliar core*
- [ydliar::core::common](#)  
*ydliar common*



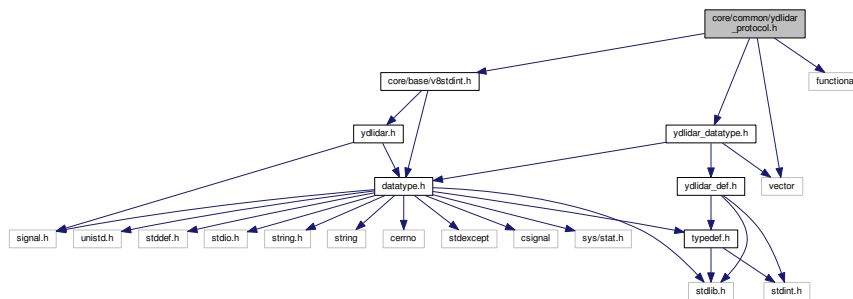
## Functions

- `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`)

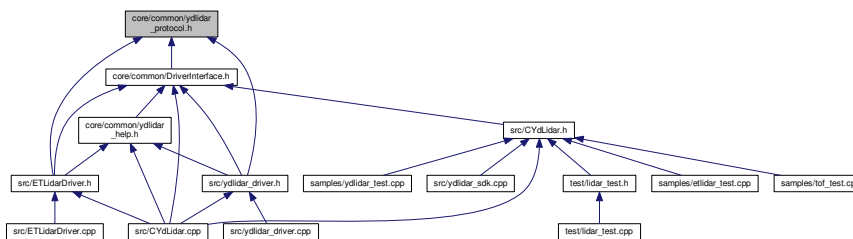
- print LiDAR version information*
- `std::vector< float > ydlidar::core::common::split` (const std::string &s, char delim)  
*split string to vector by delim format*
- `bool ydlidar::core::common::isV1Protocol` (uint8\_t protocol)  
*Whether the ET LiDAR Protocol type is V1.*

## 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  
*Maximum 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](#) [PackageSampleQuality](#)  
*intensity*
- [uint16\\_t](#) [PackageSampleDistance](#)  
*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 `ydliar_protocol.h`.

39.16.1.16 `#define LIDAR_CMD_GET_AIMSPEED 0x0D`

Definition at line 99 of file `ydliar_protocol.h`.

39.16.1.17 `#define LIDAR_CMD_GET_DEVICE_HEALTH 0x92`

Definition at line 78 of file `ydliar_protocol.h`.

39.16.1.18 `#define LIDAR_CMD_GET_DEVICE_INFO 0x90`

Definition at line 77 of file `ydliar_protocol.h`.

39.16.1.19 `#define LIDAR_CMD_GET_EAI 0x55`

Definition at line 76 of file `ydliar_protocol.h`.

39.16.1.20 `#define LIDAR_CMD_GET_OFFSET_ANGLE 0x93`

Definition at line 113 of file `ydliar_protocol.h`.

39.16.1.21 `#define LIDAR_CMD_GET_SAMPLING_RATE 0xD1`

Definition at line 102 of file `ydliar_protocol.h`.

39.16.1.22 `#define LIDAR_CMD_RESET 0x80`

Definition at line 74 of file `ydliar_protocol.h`.

39.16.1.23 `#define LIDAR_CMD_RUN_INVERSION 0x07`

Definition at line 94 of file `ydliar_protocol.h`.

39.16.1.24 `#define LIDAR_CMD_RUN_POSITIVE 0x06`

Definition at line 93 of file `ydliar_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 `ydliar_protocol.h`.

39.16.1.36 `#define LIDAR_CMDFLAG_HAS_PAYLOAD 0x80`

Definition at line 82 of file `ydliar_protocol.h`.

39.16.1.37 `#define LIDAR_RESP_MEASUREMENT_ANGLE_SAMPLE_SHIFT 8`

Definition at line 91 of file `ydliar_protocol.h`.

39.16.1.38 `#define LIDAR_RESP_MEASUREMENT_ANGLE_SHIFT 1`

Definition at line 89 of file `ydliar_protocol.h`.

39.16.1.39 `#define LIDAR_RESP_MEASUREMENT_CHECKBIT (0x1<<0)`

Definition at line 88 of file `ydliar_protocol.h`.

39.16.1.40 `#define LIDAR_RESP_MEASUREMENT_DISTANCE_SHIFT 2`

Definition at line 90 of file `ydliar_protocol.h`.

39.16.1.41 `#define LIDAR_RESP_MEASUREMENT_QUALITY_SHIFT 2`

Definition at line 87 of file `ydliar_protocol.h`.

39.16.1.42 `#define LIDAR_RESP_MEASUREMENT_SYNCBIT (0x1<<0)`

Definition at line 86 of file `ydliar_protocol.h`.

39.16.1.43 `#define LIDAR_STATUS_ERROR 0x2`

Definition at line 105 of file `ydliar_protocol.h`.

39.16.1.44 `#define LIDAR_STATUS_OK 0x0`

Definition at line 103 of file `ydliar_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`

Maximum 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 PackageSampleDistance**

range

Definition at line 943 of file ydlidar\_protocol.h.

**39.16.5.26    uint8\_t PackageSampleQuality**

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\_frequency**

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

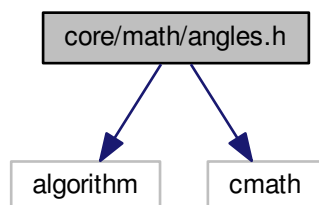
Definition at line 943 of file ydlidar\_protocol.h.

#### 39.16.5.41 uint8\_t type

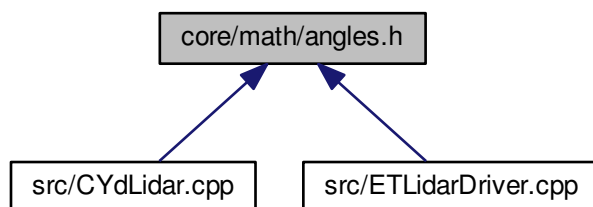
Definition at line 946 of file ydlidar\_protocol.h.

## 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 `ydliar::core::math::from_degrees` (double degrees)  
*Convert degrees to radians.*
- static double `ydliar::core::math::to_degrees` (double radians)  
*Convert radians to degrees.*
- static double `ydliar::core::math::normalize_angle_positive` (double angle)  
*normalize\_angle\_positive*
- static double `ydliar::core::math::normalize_angle_positive_from_degree` (double angle)  
*normalize\_angle\_positive\_from\_degree*
- static double `ydliar::core::math::normalize_angle` (double angle)  
*normalize*
- static double `ydliar::core::math::shortest_angular_distance` (double from, double to)  
*shortest\_angular\_distance*
- static double `ydliar::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 `ydliar::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 `ydliar::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

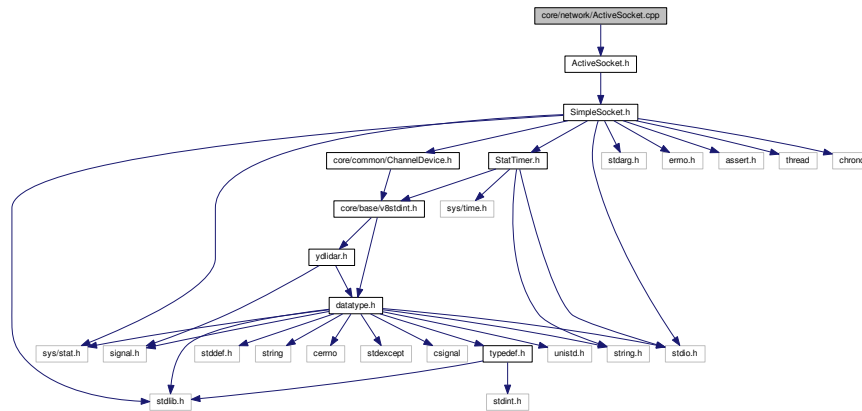
#### 39.17.1.1 `#define M_PI 3.1415926`

Definition at line 39 of file angles.h.

### 39.18 core/network/ActiveSocket.cpp File Reference

```
#include "ActiveSocket.h"
```

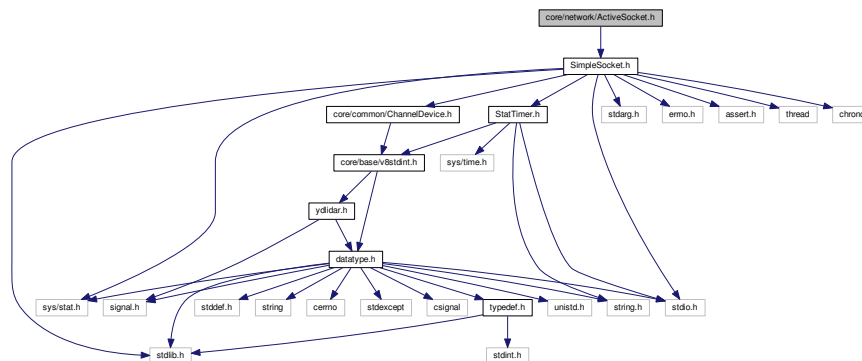
Include dependency graph for ActiveSocket.cpp:



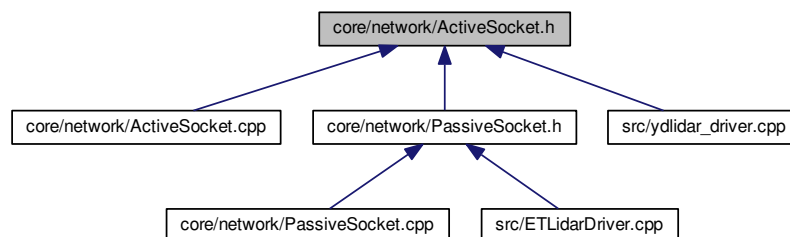
### 39.19 core/network/ActiveSocket.h File Reference

```
#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](#)

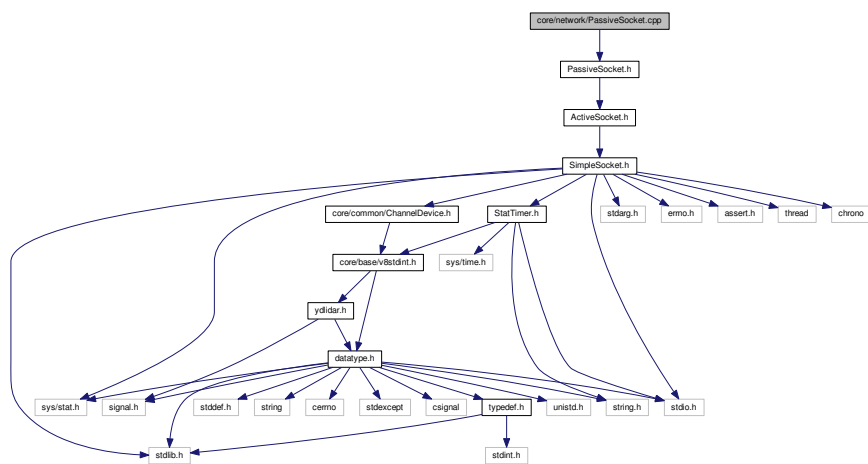
## Namespaces

- [ydlidar](#)
  - ydlidar*
- [ydlidar::core](#)
  - ydlidar core*
- [ydlidar::core::network](#)

## 39.20 core/network/PassiveSocket.cpp File Reference

```
#include "PassiveSocket.h"
```

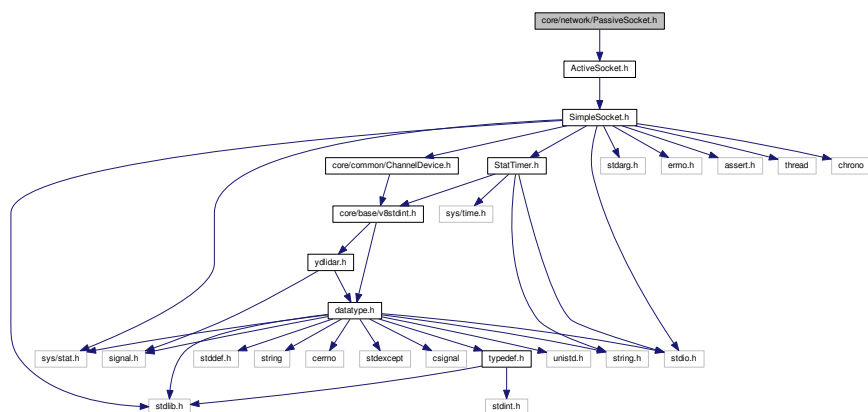
Include dependency graph for PassiveSocket.cpp:



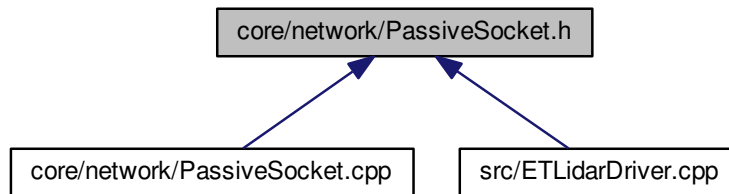
## 39.21 core/network/PassiveSocket.h File Reference

```
#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](#)

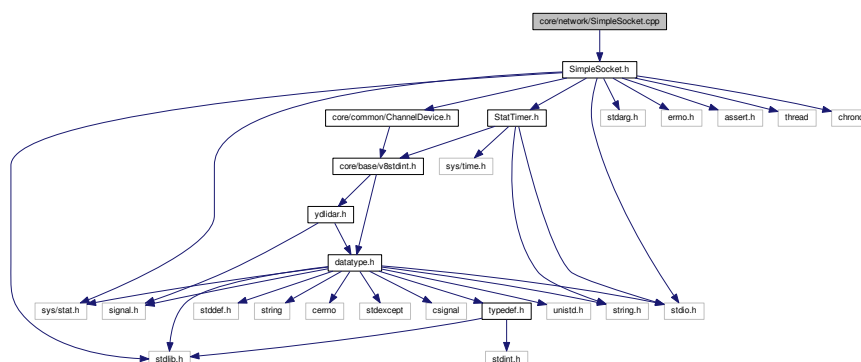
## Namespaces

- [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

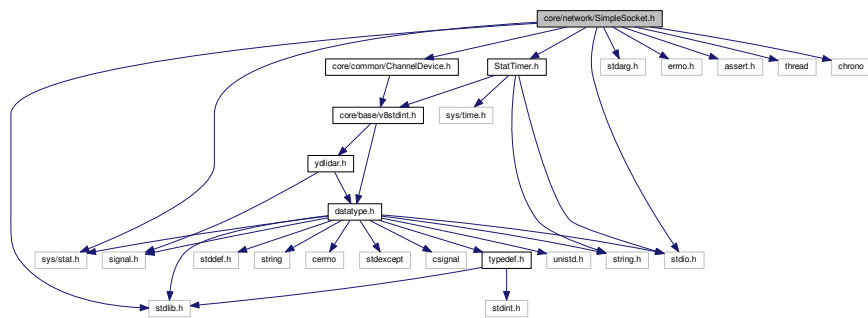
39.22.1.1 volatile bool m\_WSAStartup = false [static]

Definition at line 49 of file SimpleSocket.cpp.

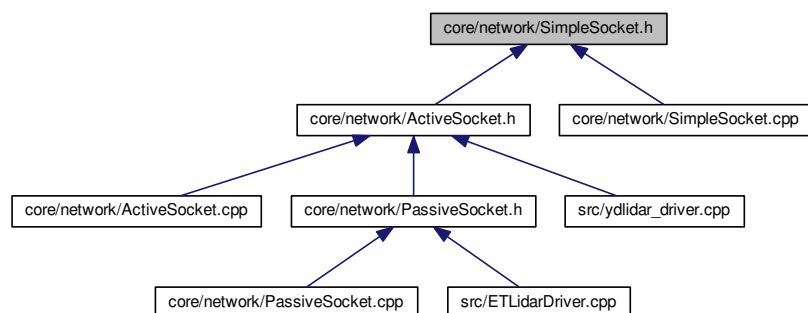
## 39.23 core/network/SimpleSocket.h File Reference

```
#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.

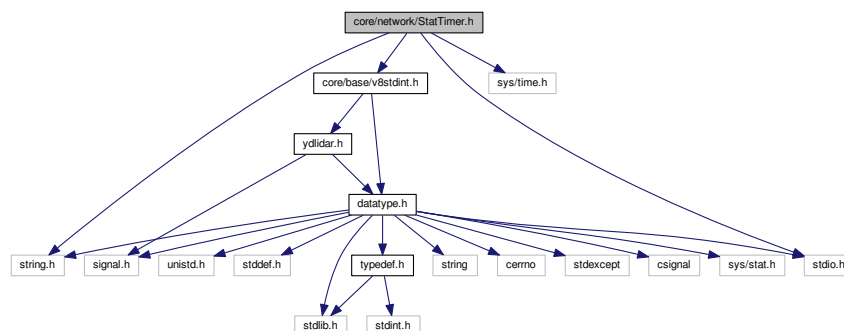
#### 39.23.1.2 `#define SOCKET_SENDFILE_BLOCKSIZE 8192`

Definition at line 102 of file SimpleSocket.h.

## 39.24 core/network/StatTimer.h File Reference

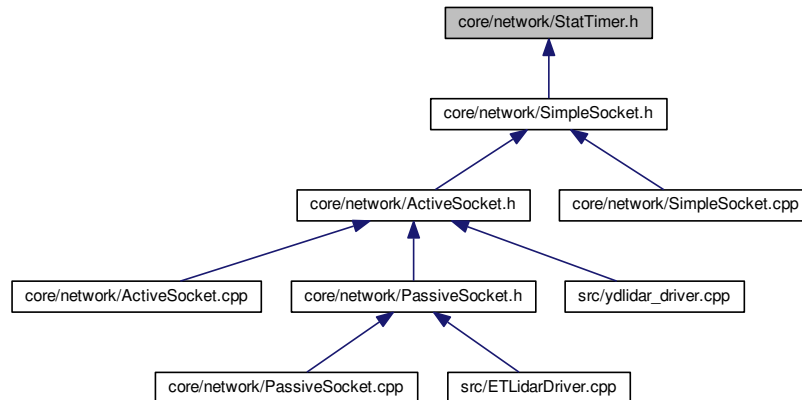
```
#include <string.h>
#include <stdio.h>
#include <sys/time.h>
#include <core/base/v8stdint.h>
```

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.

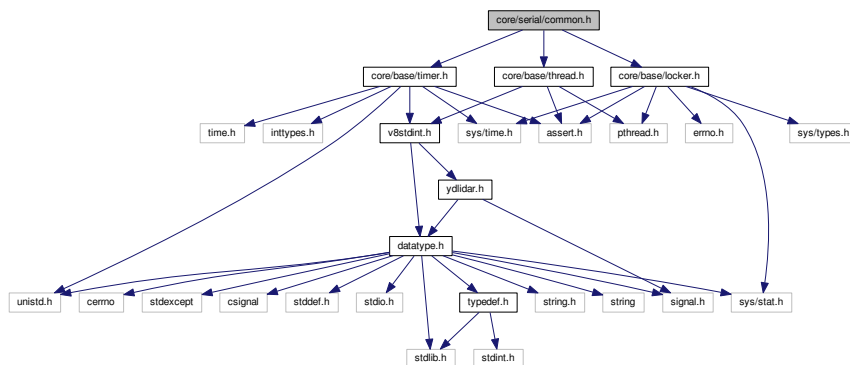
## 39.25 core/serial/common.h File Reference

```
#include <core/base/thread.h>
```

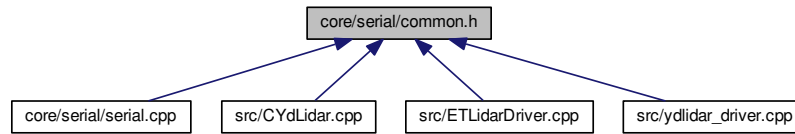
```
#include <core/base/locker.h>
```

```
#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

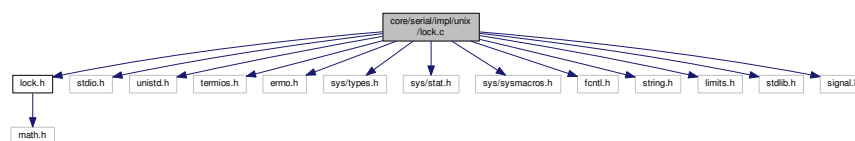
## 39.27 core/serial/impl/unix/lock.c File Reference

```

#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.

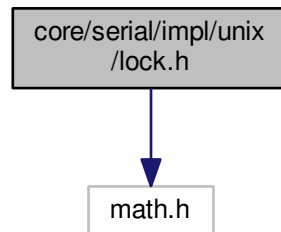
### 39.27.1.8 void uucp\_unlock ( const char \* *filename*, int *openpid* )

Definition at line 408 of file lock.c.

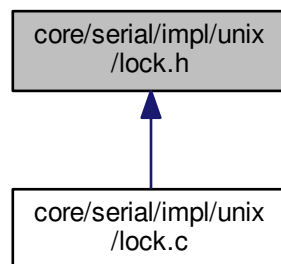
## 39.28 core/serial/impl/unix/lock.h File Reference

```
#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>
```

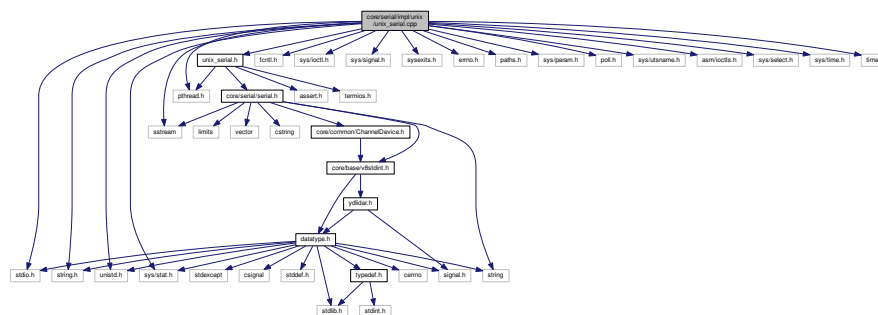
Include dependency graph for unix.h:



## 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 <sys/types.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::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`

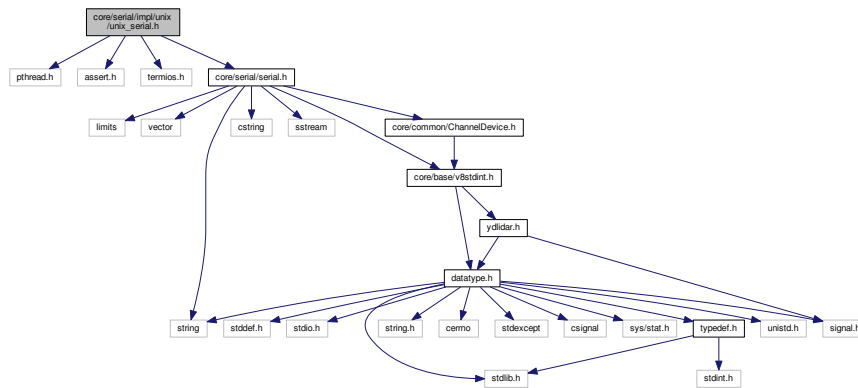
Definition at line 49 of file `unix_serial.cpp`.



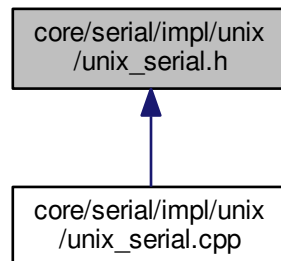
## 39.31 core/serial/impl/unix/unix\_serial.h File Reference

```
#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](#)

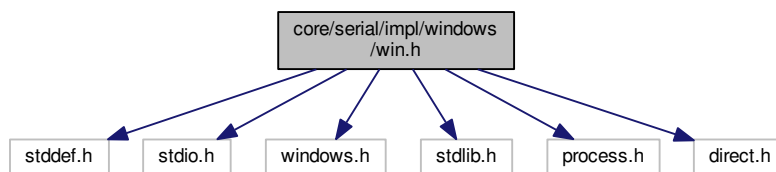
### Namespaces

- [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:
```

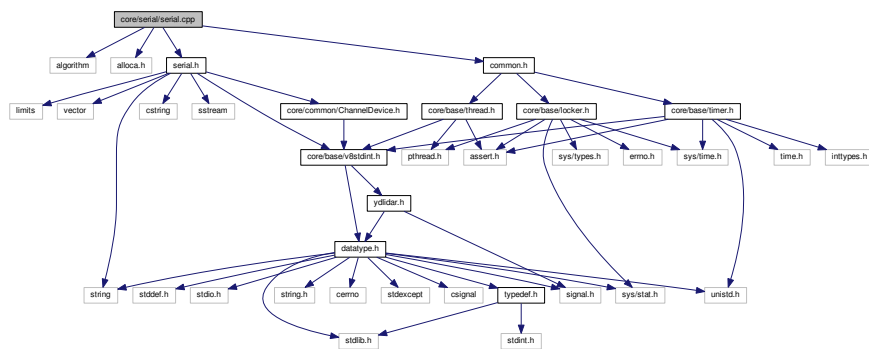


### 39.34 core/serial/impl/windows/win\_serial.cpp File Reference

### 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](#)

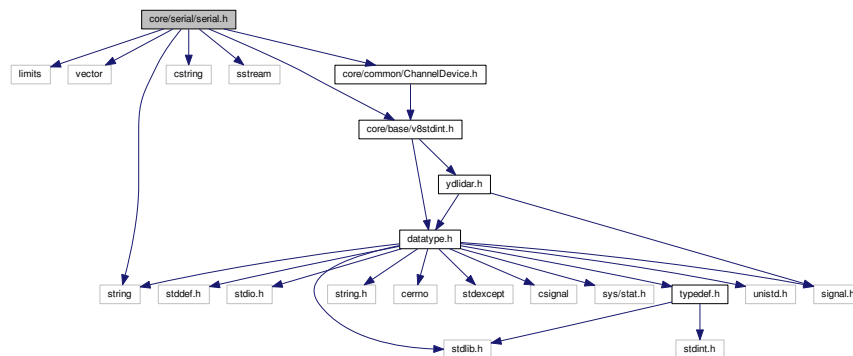
## Namespaces

- [ydlidar](#)
  - ydlidar*
- [ydlidar::core](#)
  - ydlidar core*
- [ydlidar::core::serial](#)

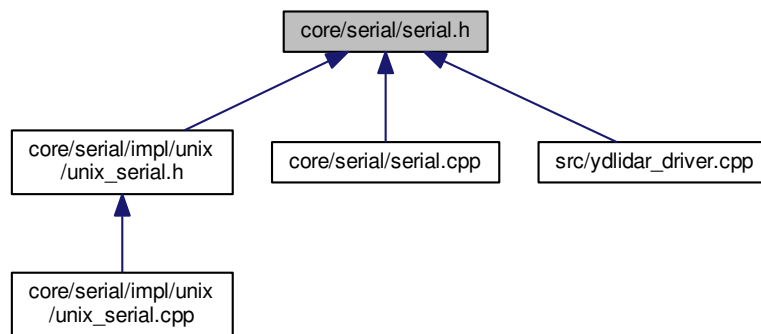
## 39.37 core/serial/serial.h File Reference

```
#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`



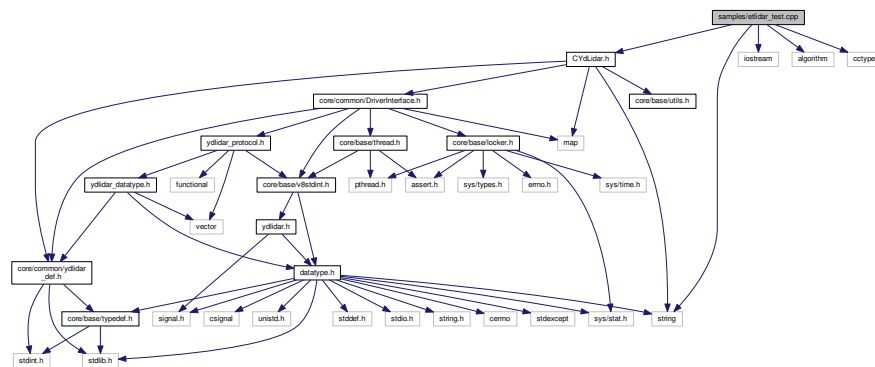
## Namespaces

- [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

|             |  |
|-------------|--|
| <i>argc</i> |  |
| <i>argv</i> |  |

#### 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: °

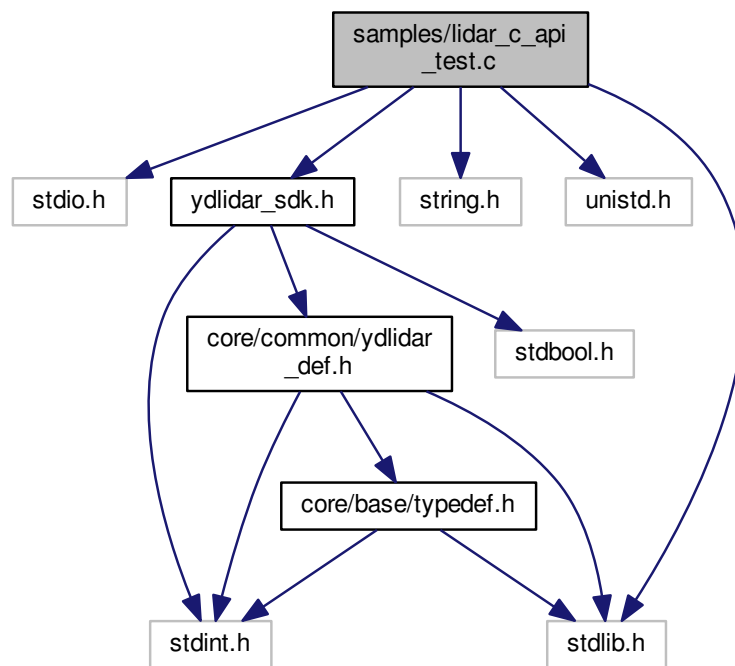
unit: m

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[] )`

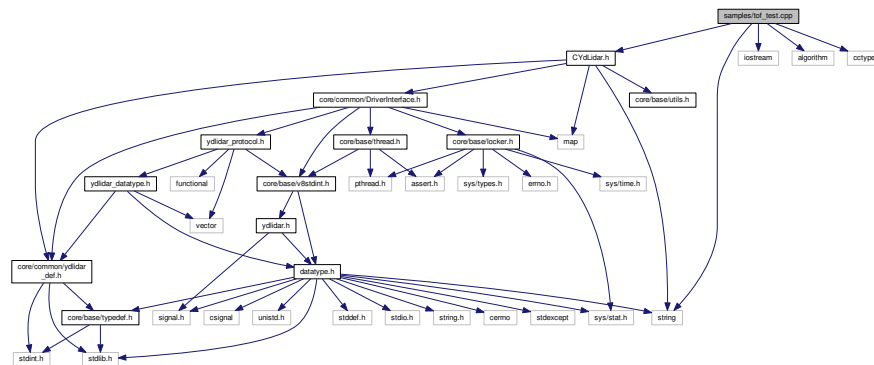
last port

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

|             |  |
|-------------|--|
| <i>argc</i> |  |
| <i>argv</i> |  |

#### 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.

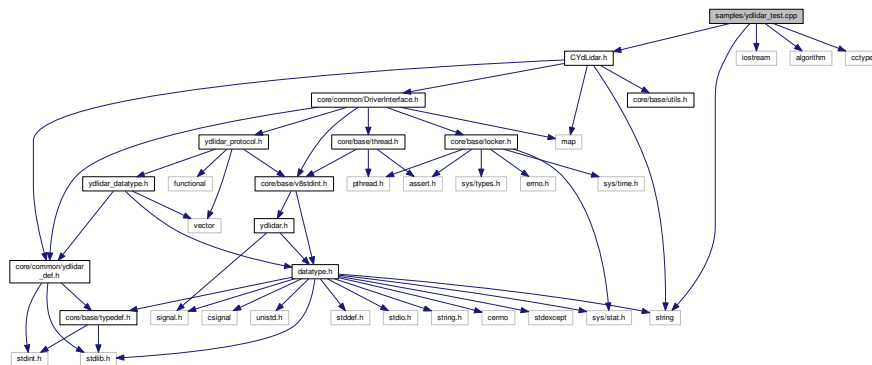
Uninitialize the SDK and Disconnect the LiDAR.

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

|             |  |
|-------------|--|
| <i>argc</i> |  |
| <i>argv</i> |  |

#### 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"



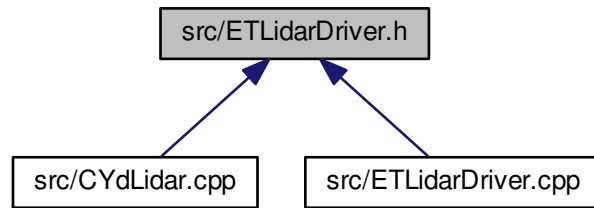


```
#include <core/base/utlis.h>
#include <core/common/ydlidar_def.h>
#include <core/common/DriverInterface.h>
#include <string>
#include <map>
```





This graph shows which files directly or indirectly include this file:



## Classes

- class `ydlidar::ETLidarDriver`

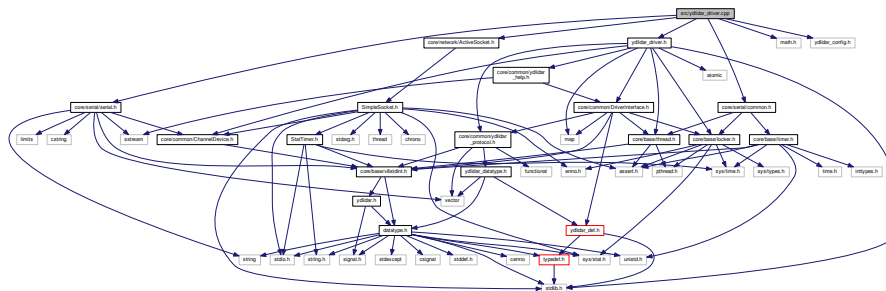
## Namespaces

- `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 "ydliidar_driver.h"
#include <core/serial/common.h>
#include <math.h>
#include <ydlidar_config.h>
```

Include dependency graph for ydlidar\_driver.cpp:



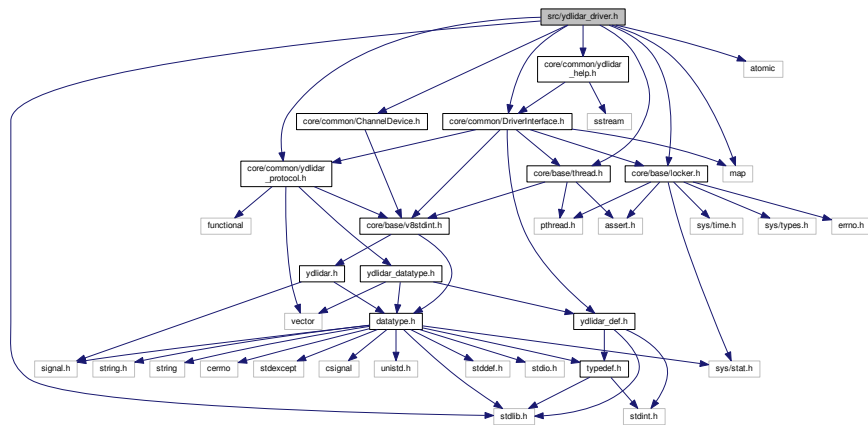
## Namespaces

- [ydlidar](#)  
*ydlidar*

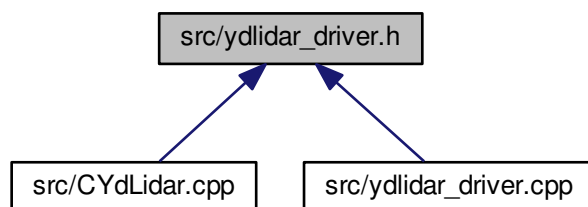
## 39.82 src/ydlidar\_driver.h File Reference

```
#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](#)



- 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 `ydliar_sdk.cpp`.

#### 39.83.1.2 `void disconnecting ( YDLidar * lidar )`

Uninitialize the SDK and Disconnect the LiDAR.

Definition at line 167 of file `ydliar_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  | <i>lidar</i>   | LiDAR instance  |
| out | <i>outscan</i> | LiDAR Scan Data |

##### Returns

true if successfully started, otherwise false.

Definition at line 129 of file `ydliar_sdk.cpp`.

#### 39.83.1.4 `bool getlidaropt ( YDLidar * lidar, int optname, void * optval, int optlen )`

get lidar property

## Parameters

|                |                  |
|----------------|------------------|
| <i>lidar</i>   | a lidar instance |
| <i>optname</i> | 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**

|               |                                                                                                                                        |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------|
| <i>optval</i> | option value <ul style="list-style-type: none"> <li>• std::string(or char*)</li> <li>• int</li> <li>• bool</li> <li>• float</li> </ul> |
| <i>optlen</i> | option length <ul style="list-style-type: none"> <li>• data type size</li> </ul>                                                       |

**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**

|                |                                                                       |
|----------------|-----------------------------------------------------------------------|
| <i>version</i> | 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**

|                |                                                             |
|----------------|-------------------------------------------------------------|
| <i>version</i> | 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**

|              |                   |
|--------------|-------------------|
| <i>lidar</i> | CYdLidar instance |
|--------------|-------------------|

Definition at line 38 of file ydlidar\_sdk.cpp.

**39.83.1.10 int lidarPortList ( LidarPort \* ports )**

get lidar serial port

**Parameters**

|              |                   |
|--------------|-------------------|
| <i>ports</i> | 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**

|                |                  |
|----------------|------------------|
| <i>lidar</i>   | a lidar instance |
| <i>optname</i> | 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**

|               |                                                                                                                                   |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <i>optval</i> | option value <ul style="list-style-type: none"><li>• std::string(or char*)</li><li>• int</li><li>• bool</li><li>• float</li></ul> |
| <i>optlen</i> | option length <ul style="list-style-type: none"><li>• data type size</li></ul>                                                    |

**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 Stopped, 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**

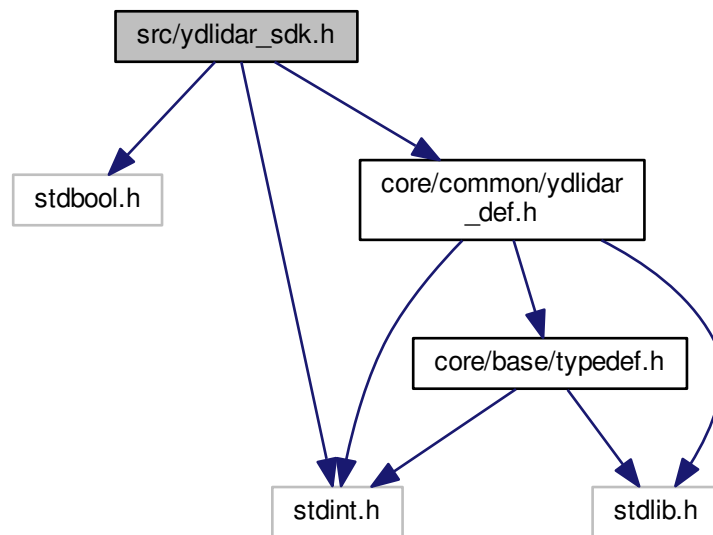
true if successfully started, otherwise false.

Definition at line 115 of file ydlidar\_sdk.cpp.

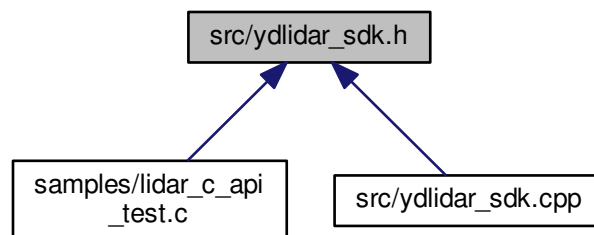
## 39.84 src/ydlidar\_sdk.h File Reference

```
#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  | <i>lidar</i>   | LiDAR instance  |
| out | <i>outscan</i> | 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

|                |                  |
|----------------|------------------|
| <i>lidar</i>   | a lidar instance |
| <i>optname</i> | 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

|               |                                                                                                                                   |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <i>optval</i> | option value <ul style="list-style-type: none"><li>• std::string(or char*)</li><li>• int</li><li>• bool</li><li>• float</li></ul> |
| <i>optlen</i> | option length <ul style="list-style-type: none"><li>• data type size</li></ul>                                                    |

#### 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**

|                |                                                                       |
|----------------|-----------------------------------------------------------------------|
| <i>version</i> | 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**

|                |                                                             |
|----------------|-------------------------------------------------------------|
| <i>version</i> | 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

|              |                   |
|--------------|-------------------|
| <i>lidar</i> | CYdLidar instance |
|--------------|-------------------|

Definition at line 38 of file ydlidar\_sdk.cpp.

39.84.1.10 int lidarPortList ( LidarPort \* *ports* )

get lidar serial port

## Parameters

|              |                   |
|--------------|-------------------|
| <i>ports</i> | 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

|                |                  |
|----------------|------------------|
| <i>lidar</i>   | a lidar instance |
| <i>optname</i> | 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**

|               |                                                                                                                                   |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <i>optval</i> | option value <ul style="list-style-type: none"><li>• std::string(or char*)</li><li>• int</li><li>• bool</li><li>• float</li></ul> |
| <i>optlen</i> | option length <ul style="list-style-type: none"><li>• data type size</li></ul>                                                    |

**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 Stopped, 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**

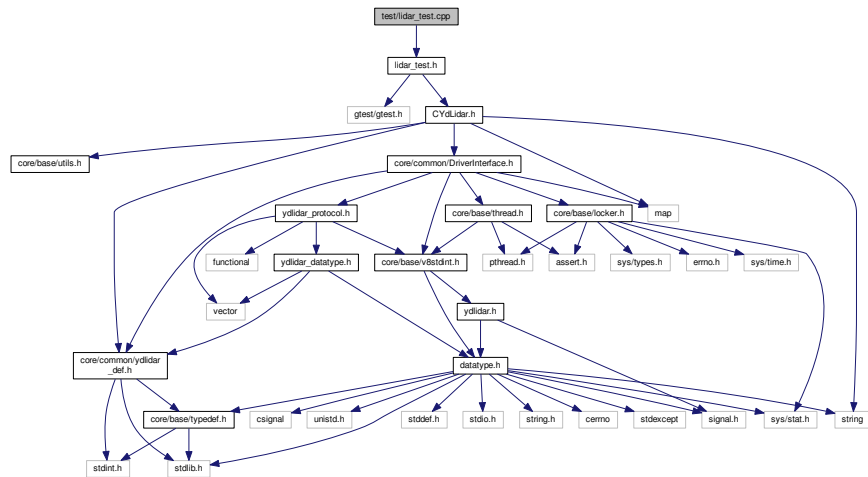
true if successfully started, otherwise false.

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 \(LidarTest, ScanFrequency\)](#)
- [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 , ScanFrequency )

Definition at line 49 of file lidar\_test.cpp.

#### 39.85.1.7 TEST\_F( LidarTest , TurnOn )

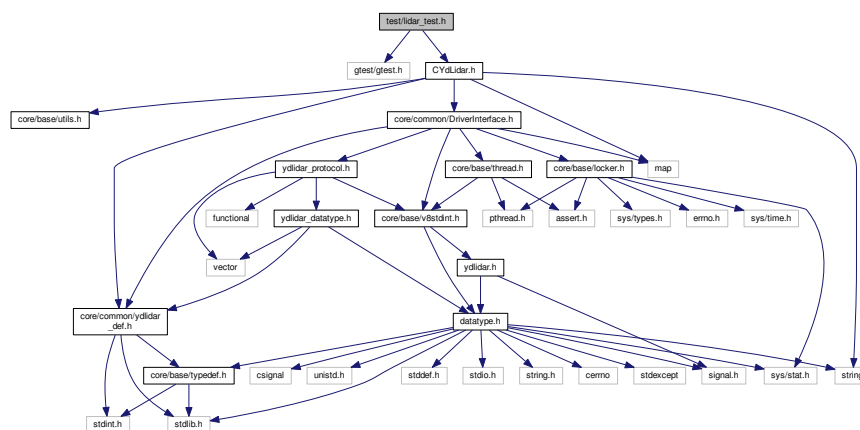
Definition at line 56 of file lidar\_test.cpp.

### 39.86 test/lidar\_test.h File Reference

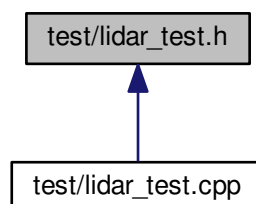
```
#include "gtest/gtest.h"
```

```
#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

- `__WORDSIZE`
  - `datatype.h`, 338
- `__attribute__`
  - `datatype.h`, 338
  - `ydlidar_protocol.h`, 372
- `__init__`
  - `setup::CMakeExtension`, 161
- `__small_endian`
  - `datatype.h`, 338
- `_access`
  - `datatype.h`, 338
- `_binded`
  - `ydlidar::core::base::ScopedLocker`, 280
- `_cmd_lock`
  - `ydlidar::core::common::DriverInterface`, 228
- `_cond_cattr`
  - `ydlidar::core::base::Event`, 247
- `_cond_locker`
  - `ydlidar::core::base::Event`, 247
- `_cond_var`
  - `ydlidar::core::base::Event`, 247
- `_countof`
  - `ydlidar_protocol.h`, 366
- `_dataEvent`
  - `ydlidar::core::common::DriverInterface`, 228
- `_dataFrame`, 145
  - `dataFormat`, 145
  - `dataIndex`, 145
  - `dataNum`, 146
  - `deviceType`, 146
  - `disScale`, 146
  - `frameBuf`, 146
  - `frameCrc`, 146
  - `frameHead`, 146
  - `frameIndex`, 146
  - `frameType`, 146
  - `headFrameFlag`, 146
  - `startAngle`, 146
  - `timestamp`, 147
- `_func`
  - `ydlidar::core::base::Thread`, 308
- `_handle`
  - `ydlidar::core::base::Thread`, 308
- `_isAutoReset`
  - `ydlidar::core::base::Event`, 247
- `_is_signalled`
  - `ydlidar::core::base::Event`, 247
- `_itoa`
  - `datatype.h`, 338
- `_lidarConfig`, 147
  - `correction_angle`, 148
  - `correction_distance`, 148
  - `dataRecvIp`, 148
  - `dataRecvPort`, 148
  - `deviceGatewayIp`, 148
  - `deviceIp`, 148
  - `deviceNetmask`, 148
  - `dhcp_en`, 149
  - `fov_end`, 149
  - `fov_start`, 149
  - `laser_en`, 149
  - `laserScanFrequency`, 149
  - `motor_en`, 149
  - `motor_rpm`, 149
  - `trans_sel`, 149
- `_lock`
  - `ydlidar::core::base::Locker`, 264
  - `ydlidar::core::common::DriverInterface`, 228
- `_param`
  - `ydlidar::core::base::Thread`, 309
- `_thread`
  - `ydlidar::core::common::DriverInterface`, 229
- `~CActiveSocket`
  - `ydlidar::core::network::CActiveSocket`, 151
- `~CPassiveSocket`
  - `ydlidar::core::network::CPassiveSocket`, 164
- `~CSimpleSocket`
  - `ydlidar::core::network::CSimpleSocket`, 172
- `~CStatTimer`
  - `CStatTimer`, 192
- `~CYdLidar`
  - `CYdLidar`, 205
- `~ChannelDevice`
  - `ydlidar::core::common::ChannelDevice`, 154
- `~DriverInterface`
  - `ydlidar::core::common::DriverInterface`, 218
- `~ETLidarDriver`
  - `ydlidar::ETLidarDriver`, 235
- `~Event`
  - `ydlidar::core::base::Event`, 247
- `~LidarTest`
  - `LidarTest`, 261
- `~Locker`
  - `ydlidar::core::base::Locker`, 263
- `~ScopedLocker`
  - `ydlidar::core::base::ScopedLocker`, 279
- `~ScopedReadLock`
  - `ydlidar::core::serial::Serial::ScopedReadLock`, 280

- ~ScopedWriteLock
  - ydlidar::core::serial::Serial::ScopedWriteLock, 281
- ~Serial
  - ydlidar::core::serial::Serial, 285
- ~SerialImpl
  - ydlidar::core::serial::Serial::SerialImpl, 300
- ~Thread
  - ydlidar::core::base::Thread, 307
- ~YDLidarDriver
  - ydlidar::YDLidarDriver, 315
- Accept
  - ydlidar::core::network::CPassiveSocket, 164
- angle
  - LaserPoint, 255
  - offset\_angle, 271
  - ydlidar\_protocol.h, 372
- angle\_increment
  - LaserConfig, 249
- angle\_q6\_checkbit
  - node\_info, 266
  - ydlidar\_protocol.h, 372
- angles.h
  - M\_PI, 379
- ani
  - plot\_tof\_test, 117
  - plot\_ydlidar\_test, 118
- animate
  - plot\_tof\_test, 116
  - plot\_ydlidar\_test, 118
- ascendScanData
  - ydlidar::YDLidarDriver, 316
- author
  - setup::CMakeBuild, 159
- author\_email
  - setup::CMakeBuild, 159
- available
  - ydlidar::core::common::ChannelDevice, 154
  - ydlidar::core::network::CSimpleSocket, 172
  - ydlidar::core::serial::Serial, 285
  - ydlidar::core::serial::Serial::SerialImpl, 300
- BEGIN\_STATIC\_CODE
  - timer.h, 345
- BOTHER
  - unix\_serial.cpp, 392
- BindInterface
  - ydlidar::core::network::CSimpleSocket, 172
- BindMulticast
  - ydlidar::core::network::CPassiveSocket, 164
- bindport
  - ydlidar::core::common::ChannelDevice, 154
  - ydlidar::core::network::CSimpleSocket, 172
- Both
  - ydlidar::core::network::CSimpleSocket, 170
- BreakConditionError
  - ydlidar::core::serial::Serial, 284
- build\_extension
  - setup::CMakeBuild, 159
- bytesize\_t
  - ydlidar::core::serial, 142
- c\_cc
  - ydlidar::core::serial::termios2, 306
- c\_cflag
  - ydlidar::core::serial::termios2, 306
- c\_iflag
  - ydlidar::core::serial::termios2, 306
- c\_ispeed
  - ydlidar::core::serial::termios2, 306
- c\_lflag
  - ydlidar::core::serial::termios2, 306
- c\_line
  - ydlidar::core::serial::termios2, 306
- c\_oflag
  - ydlidar::core::serial::termios2, 306
- c\_ospeed
  - ydlidar::core::serial::termios2, 306
- CActiveSocket
  - ydlidar::core::network::CActiveSocket, 151
- CLASS\_THREAD
  - thread.h, 343
- CLEARERR
  - v8stdint.h, 348
- CPassiveSocket
  - ydlidar::core::network::CActiveSocket, 152
  - ydlidar::core::network::CPassiveSocket, 164
- CShutdownMode
  - ydlidar::core::network::CSimpleSocket, 170
- CSimpleSocket
  - ydlidar::core::network::CSimpleSocket, 171
- CSocketError
  - ydlidar::core::network::CSimpleSocket, 170
- CSocketType
  - ydlidar::core::network::CSimpleSocket, 171
- CStatTimer, 192
  - ~CStatTimer, 192
  - CStatTimer, 192
  - GetCurrentTime, 192
  - GetEndTime, 192
  - GetMicroSeconds, 193
  - GetMilliSeconds, 193
  - GetSeconds, 193
  - GetStartTime, 193
  - Initialize, 193
  - SetEndTime, 193
  - SetStartTime, 193
- CT\_Normal
  - ydlidar\_protocol.h, 372
- CT\_RingStart
  - ydlidar\_protocol.h, 372
- CT\_Tail
  - ydlidar\_protocol.h, 372
- CYdLidar, 193
  - ~CYdLidar, 205
  - CYdLidar, 205
  - DescribeError, 205
  - disconnecting, 205

- doProcessSimple, [206](#)
- GetLidarVersion, [208](#)
- getlidaropt, [206](#)
- initialize, [208](#)
- setlidaropt, [208](#)
- turnOff, [210](#)
- turnOn, [210](#)
- CYdLidar.cpp
  - removeExceptionSample, [408](#)
- cacheScanData
  - ydlidar::YDLidarDriver, [317](#)
- ChannelDevice
  - ydlidar::core::common::ChannelDevice, [154](#)
- check\_group\_uucp
  - lock.c, [387](#)
  - lock.h, [389](#)
- check\_lock\_pid
  - lock.c, [387](#)
  - lock.h, [389](#)
- check\_lock\_status
  - lock.c, [387](#)
  - lock.h, [389](#)
- checkAutoConnecting
  - ydlidar::YDLidarDriver, [317](#)
- checkDeviceInfo
  - ydlidar::YDLidarDriver, [317](#)
- checksum
  - node\_package, [268](#)
  - node\_packages, [270](#)
  - ydlidar\_protocol.h, [373](#)
- checkTransDelay
  - ydlidar::YDLidarDriver, [318](#)
- clearDTR
  - ydlidar::YDLidarDriver, [318](#)
- clone
  - setup::CMakeBuild, [159](#)
- Close
  - ydlidar::core::network::CSimpleSocket, [172](#)
- close
  - ydlidar::core::serial::Serial::SerialImpl, [300](#)
- closePort
  - ydlidar::core::common::ChannelDevice, [154](#)
  - ydlidar::core::network::CSimpleSocket, [173](#)
  - ydlidar::core::serial::Serial, [285](#)
- cmd\_flag
  - cmd\_packet, [162](#)
  - ydlidar\_protocol.h, [373](#)
- cmd\_packet, [162](#)
  - cmd\_flag, [162](#)
  - data, [162](#)
  - size, [162](#)
  - syncByte, [162](#)
- cmdclass
  - setup::CMakeBuild, [159](#)
- config
  - LaserFan, [254](#)
  - LaserScan, [257](#)
- connect
  - ydlidar::ETLidarDriver, [235](#)
  - ydlidar::YDLidarDriver, [318](#)
  - ydlidar::core::common::DriverInterface, [218](#)
- ConvertLidarToUserSmaple
  - ydlidar::core::common, [128](#)
- ConvertUserToLidarSmaple
  - ydlidar::core::common, [128](#)
- core/base/datatype.h, [337](#)
- core/base/locker.h, [341](#)
- core/base/thread.h, [342](#)
- core/base/timer.cpp, [343](#)
- core/base/timer.h, [343](#)
- core/base/typedef.h, [345](#)
- core/base/utils.h, [346](#)
- core/base/v8stdint.h, [346](#)
- core/base/ydlidar.h, [351](#)
- core/common/ChannelDevice.h, [353](#)
- core/common/DriverInterface.h, [354](#)
- core/common/ydlidar\_datatype.h, [355](#)
- core/common/ydlidar\_def.cpp, [356](#)
- core/common/ydlidar\_def.h, [357](#)
- core/common/ydlidar\_help.h, [360](#)
- core/common/ydlidar\_protocol.h, [362](#)
- core/math/angles.h, [378](#)
- core/network/ActiveSocket.cpp, [380](#)
- core/network/ActiveSocket.h, [380](#)
- core/network/PassiveSocket.cpp, [381](#)
- core/network/PassiveSocket.h, [381](#)
- core/network/SimpleSocket.cpp, [382](#)
- core/network/SimpleSocket.h, [383](#)
- core/network/StatTimer.h, [384](#)
- core/serial/common.h, [385](#)
- core/serial/impl/unix/list\_ports\_linux.cpp, [386](#)
- core/serial/impl/unix/lock.c, [386](#)
- core/serial/impl/unix/lock.h, [388](#)
- core/serial/impl/unix/unix.h, [390](#)
- core/serial/impl/unix/unix\_serial.cpp, [391](#)
- core/serial/impl/unix/unix\_serial.h, [393](#)
- core/serial/impl/windows/list\_ports\_win.cpp, [394](#)
- core/serial/impl/windows/win.h, [394](#)
- core/serial/impl/windows/win\_serial.cpp, [394](#)
- core/serial/impl/windows/win\_serial.h, [394](#)
- core/serial/serial.cpp, [394](#)
- core/serial/serial.h, [395](#)
- correction\_angle
  - \_lidarConfig, [148](#)
- correction\_distance
  - \_lidarConfig, [148](#)
- createThread
  - ydlidar::YDLidarDriver, [319](#)
  - ydlidar::core::base::Thread, [308](#)
- createThreadAux
  - ydlidar::core::base::Thread, [308](#)
- CT
  - ydlidar\_protocol.h, [372](#)
- DATA\_FRAME
  - datatype.h, [339](#)
- DEFAULT\_CONNECTION\_TIMEOUT\_SEC

- v8stdint.h, 348
- DEFAULT\_CONNECTION\_TIMEOUT\_USEC
  - v8stdint.h, 348
- DEFAULT\_HEART\_BEAT
  - ydlidar::core::common::DriverInterface, 218
- DEFAULT\_INTENSITY
  - datatype.h, 339
- DEFAULT\_REV\_TIMEOUT\_SEC
  - v8stdint.h, 348
- DEFAULT\_REV\_TIMEOUT\_USEC
  - v8stdint.h, 348
- DEFAULT\_TIMEOUT\_COUNT
  - ydlidar::core::common::DriverInterface, 218
- DEFAULT\_TIMEOUT
  - ydlidar::core::common::DriverInterface, 218
- DSL
  - datatype.h, 339
- data
  - cmd\_packet, 162
  - string\_t, 305
  - ydlidar\_protocol.h, 373
- dataFormat
  - \_dataFrame, 145
- dataFrame, 211
  - ydlidar\_protocol.h, 372
- dataIndex
  - \_dataFrame, 145
- dataNum
  - \_dataFrame, 146
- dataRecvIp
  - \_lidarConfig, 148
- dataRecvPort
  - \_lidarConfig, 148
- datatype.h
  - \_\_WORDSIZE, 338
  - \_\_attribute\_\_, 338
  - \_\_small\_endian, 338
  - \_access, 338
  - \_itoa, 338
  - DATA\_FRAME, 339
  - DEFAULT\_INTENSITY, 339
  - DSL, 339
  - FRAME\_PREAMBLE, 339
  - INVALID\_TIMESTAMP, 339
  - IS\_FAIL, 339
  - IS\_OK, 339
  - IS\_TIMEOUT, 339
  - LIDAR\_2D, 339
  - RESULT\_FAIL, 339
  - RESULT\_OK, 340
  - RESULT\_TIMEOUT, 340
  - result\_t, 340
  - UNUSED, 340
  - valLastName, 340
  - valName, 340
- debugInfo
  - node\_info, 266
  - ydlidar\_protocol.h, 373
- delay
  - timer.h, 345
- DescribeError
  - CYdLidar, 205
  - ydlidar::ETLidarDriver, 236
  - ydlidar::YDLidarDriver, 319
  - ydlidar::core::common::ChannelDevice, 154
  - ydlidar::core::common::DriverInterface, 219
  - ydlidar::core::network::CSimpleSocket, 173
  - ydlidar::core::serial::Serial, 285
  - ydlidar\_sdk.cpp, 414
  - ydlidar\_sdk.h, 422
- description
  - setup::CMakeBuild, 159
  - ydlidar::core::serial::PortInfo, 273
- device\_health, 211
  - error\_code, 211
  - status, 211
- device\_id
  - ydlidar::core::serial::PortInfo, 273
- device\_info, 212
  - firmware\_version, 212
  - hardware\_version, 212
  - model, 212
  - serialnum, 213
- deviceGatewayIp
  - \_lidarConfig, 148
- deviceIp
  - \_lidarConfig, 148
- deviceNetmask
  - \_lidarConfig, 148
- DeviceNotFoundError
  - ydlidar::core::serial::Serial, 284
- deviceType
  - \_dataFrame, 146
- DeviceTypeID
  - ydlidar\_def.h, 358
- dhcp\_en
  - \_lidarConfig, 149
- disScale
  - \_dataFrame, 146
- disableDataGrabbing
  - ydlidar::YDLidarDriver, 319
- DisableNagleAlgoritm
  - ydlidar::core::network::CSimpleSocket, 173
- disconnect
  - ydlidar::ETLidarDriver, 236
  - ydlidar::YDLidarDriver, 319
  - ydlidar::core::common::DriverInterface, 219
- disconnecting
  - CYdLidar, 205
  - ydlidar\_sdk.cpp, 414
  - ydlidar\_sdk.h, 422
- distance\_q2
  - node\_info, 266
  - ydlidar\_protocol.h, 373
- doProcessSimple
  - CYdLidar, 206

- ydlidar\_sdk.cpp, [414](#)
  - ydlidar\_sdk.h, [422](#)
- doc/Dataset.md, [397](#)
- doc/Diagram.md, [397](#)
- doc/FAQs/General\_FAQs.md, [397](#)
- doc/FAQs/General\_FAQs\_cn.md, [397](#)
- doc/FAQs/Hardware\_FAQs.md, [397](#)
- doc/FAQs/Hardware\_FAQs\_cn.md, [397](#)
- doc/FAQs/README.md, [397](#)
- doc/FAQs/Software\_FAQs.md, [397](#)
- doc/FAQs/Software\_FAQs\_cn.md, [397](#)
- doc/README.md, [397](#)
- doc/Tutorials.md, [398](#)
- doc/YDLIDAR\_SDK\_API\_for\_Developers.md, [398](#)
- doc/YDLidar-SDK-Communication-Protocol.md, [398](#)
- doc/howto/README.md, [397](#)
- doc/howto/how\_to\_build\_and\_debug\_using\_vscode.↔  
md, [397](#)
- doc/howto/how\_to\_build\_and\_install.md, [397](#)
- doc/howto/how\_to\_create\_a\_pull.md, [397](#)
- doc/howto/how\_to\_create\_a\_udev\_rules.md, [397](#)
- doc/howto/how\_to\_gerenrate\_vs\_project\_by\_cmake.↔  
md, [398](#)
- doc/howto/how\_to\_solve\_slow\_pull\_from\_cn.md, [398](#)
- doc/quickstart/README.md, [397](#)
- doc/quickstart/ydlidar\_sdk\_software\_installation\_↔  
guide.md, [398](#)
- doc/tutorials/examine\_the\_simple\_lidar\_tutorial.md, [398](#)
- doc/tutorials/writing\_lidar\_tutorial\_c++.md, [398](#)
- doc/tutorials/writing\_lidar\_tutorial\_c.md, [398](#)
- doc/tutorials/writing\_lidar\_tutorial\_python.md, [398](#)
- DriverInterface
  - ydlidar::core::common::DriverInterface, [218](#)
- END\_STATIC\_CODE
  - timer.h, [345](#)
- ETLidarDriver
  - ydlidar::ETLidarDriver, [235](#)
- EVENT\_FAILED
  - ydlidar::core::base::Event, [246](#)
- EVENT\_OK
  - ydlidar::core::base::Event, [246](#)
- EVENT\_TIMEOUT
  - ydlidar::core::base::Event, [246](#)
- eightbits
  - ydlidar::core::serial, [142](#)
- enable
  - scan\_heart\_beat, [277](#)
  - ydlidar\_protocol.h, [373](#)
- EnableNagleAlgorithn
  - ydlidar::core::network::CSimpleSocket, [173](#)
- error\_code
  - device\_health, [211](#)
  - ydlidar\_protocol.h, [373](#)
- error\_package
  - node\_info, [266](#)
  - ydlidar\_protocol.h, [373](#)
- etlidar\_test, [115](#)
  - laser, [115](#)
  - r, [115](#)
  - ret, [115](#)
  - scan, [115](#)
- etlidar\_test.cpp
  - main, [401](#)
- Event
  - ydlidar::core::base::Event, [247](#)
- exposure
  - scan\_exposure, [276](#)
  - ydlidar\_protocol.h, [374](#)
- ext\_modules
  - setup::CMakeBuild, [159](#)
- FALSE
  - v8stdint.h, [348](#)
- FCLOSE
  - v8stdint.h, [348](#)
- FEOF
  - v8stdint.h, [348](#)
- FERROR
  - v8stdint.h, [348](#)
- FFLUSH
  - v8stdint.h, [348](#)
- FILE\_HANDLE
  - v8stdint.h, [349](#)
- FILENO
  - v8stdint.h, [349](#)
- FOPEN
  - v8stdint.h, [349](#)
- FPRINTF
  - v8stdint.h, [349](#)
- FRAME\_PREAMBLE
  - datatype.h, [339](#)
- FSTAT
  - v8stdint.h, [349](#)
- fhs\_lock
  - lock.c, [387](#)
  - lock.h, [389](#)
- fhs\_unlock
  - lock.c, [387](#)
  - lock.h, [389](#)
- fig
  - plot\_tof\_test, [117](#)
  - plot\_ydlidar\_test, [118](#)
- fileExists
  - ydlidar::core::base, [126](#)
- find\_min\_max\_delta
  - ydlidar::core::math, [138](#)
- firmware\_version
  - device\_info, [212](#)
  - ydlidar\_protocol.h, [374](#)
- fivebits
  - ydlidar::core::serial, [142](#)
- flag
  - scan\_points, [278](#)
  - ydlidar\_protocol.h, [374](#)
- flowcontrol\_hardware
  - ydlidar::core::serial, [142](#)
- flowcontrol\_none

- ydlidar::core::serial, 142
- flowcontrol\_software
  - ydlidar::core::serial, 142
- flowcontrol\_t
  - ydlidar::core::serial, 142
- Flush
  - ydlidar::core::network::CSimpleSocket, 174
- flush
  - ydlidar::core::common::ChannelDevice, 155
  - ydlidar::core::network::CSimpleSocket, 174
  - ydlidar::core::serial::Serial, 286
  - ydlidar::core::serial::Serial::SerialImpl, 300
- flushInput
  - ydlidar::core::serial::Serial, 286
  - ydlidar::core::serial::Serial::SerialImpl, 300
- flushOutput
  - ydlidar::core::serial::Serial, 286
  - ydlidar::core::serial::Serial::SerialImpl, 300
- flushSerial
  - ydlidar::YDlidarDriver, 319
- forceUnlock
  - ydlidar::core::base::ScopedLocker, 280
- fov\_end
  - \_lidarConfig, 149
- fov\_start
  - \_lidarConfig, 149
- frameBuf
  - \_dataFrame, 146
- frameCrc
  - \_dataFrame, 146
- frameHead
  - \_dataFrame, 146
- frameIndex
  - \_dataFrame, 146
- frameType
  - \_dataFrame, 146
- FramingError
  - ydlidar::core::serial::Serial, 284
- frequency
  - scan\_frequency, 277
  - ydlidar\_protocol.h, 374
- from\_degrees
  - ydlidar::core::math, 139
- function\_state, 248
  - state, 248
- g\_signal\_status
  - ydlidar.h, 353
- GET\_CLOCK\_COUNT
  - StatTimer.h, 385
- GLASSNOISEINTENSITY
  - ydlidar\_protocol.h, 366
- getAutoZeroOffsetAngle
  - ydlidar::YDlidarDriver, 320
- getBaudrate
  - ydlidar::core::serial::Serial, 286
  - ydlidar::core::serial::Serial::SerialImpl, 300
- getBytesTime
  - ydlidar::core::common::ChannelDevice, 155
- ydlidar::core::serial::Serial, 287
  - ydlidar::core::serial::Serial::SerialImpl, 300
- GetBytesReceived
  - ydlidar::core::network::CSimpleSocket, 174
- GetBytesSent
  - ydlidar::core::network::CSimpleSocket, 174
- getBytesize
  - ydlidar::core::serial::Serial, 286
  - ydlidar::core::serial::Serial::SerialImpl, 300
- getCTS
  - ydlidar::core::serial::Serial, 287
  - ydlidar::core::serial::Serial::SerialImpl, 301
- getCD
  - ydlidar::core::serial::Serial, 287
  - ydlidar::core::serial::Serial::SerialImpl, 301
- GetClientAddr
  - ydlidar::core::network::CSimpleSocket, 174
- GetClientPort
  - ydlidar::core::network::CSimpleSocket, 175
- GetConnectTimeoutSec
  - ydlidar::core::network::CSimpleSocket, 175
- GetConnectTimeoutUSec
  - ydlidar::core::network::CSimpleSocket, 175
- GetCurrentTime
  - CStatTimer, 192
- getCurrentTime
  - impl, 116
- getDSR
  - ydlidar::core::serial::Serial, 287
  - ydlidar::core::serial::Serial::SerialImpl, 301
- GetData
  - ydlidar::core::network::CSimpleSocket, 175
- getData
  - ydlidar::YDlidarDriver, 320
- getDeviceInfo
  - ydlidar::ETLidarDriver, 236
  - ydlidar::YDlidarDriver, 321
  - ydlidar::core::common::DriverInterface, 219
- GetEndTime
  - CStatTimer, 192
- getFinishedScanCfg
  - ydlidar::ETLidarDriver, 237
  - ydlidar::core::common::DriverInterface, 220
- getFlowcontrol
  - ydlidar::core::serial::Serial, 287
  - ydlidar::core::serial::Serial::SerialImpl, 301
- getHDTimer
  - impl, 116
- getHandle
  - ydlidar::core::base::Thread, 308
- getHealth
  - ydlidar::ETLidarDriver, 237
  - ydlidar::YDlidarDriver, 321
  - ydlidar::core::common::DriverInterface, 220
- getLidarType
  - ydlidar::core::common::DriverInterface, 220
- GetLidarVersion
  - CYdlidar, 208

- ydlidar\_sdk.cpp, 416
  - ydlidar\_sdk.h, 424
- getLockHandle
  - ydlidar::core::base::Locker, 264
- GetMicroSeconds
  - CStatTimer, 193
- GetMilliseconds
  - CStatTimer, 193
- GetMulticast
  - ydlidar::core::network::CSimpleSocket, 175
- getParam
  - ydlidar::core::base::Thread, 308
- getParity
  - ydlidar::core::serial::Serial, 287
  - ydlidar::core::serial::Serial::SerialImpl, 301
- getPointTime
  - ydlidar::core::common::DriverInterface, 220
- getPort
  - ydlidar::core::serial::Serial, 288
  - ydlidar::core::serial::Serial::SerialImpl, 301
- GetReceiveTimeoutSec
  - ydlidar::core::network::CSimpleSocket, 176
- GetReceiveTimeoutUsec
  - ydlidar::core::network::CSimpleSocket, 176
- GetReceiveWindowSize
  - ydlidar::core::network::CSimpleSocket, 176
- getRI
  - ydlidar::core::serial::Serial, 288
  - ydlidar::core::serial::Serial::SerialImpl, 301
- getSDKVersion
  - ydlidar::ETLidarDriver, 239
  - ydlidar::YDLidarDriver, 323
  - ydlidar::core::common::DriverInterface, 222
- getSamplingRate
  - ydlidar::ETLidarDriver, 237
  - ydlidar::YDLidarDriver, 321
  - ydlidar::core::common::DriverInterface, 221
- getScanCfg
  - ydlidar::ETLidarDriver, 238
- getScanFrequency
  - ydlidar::ETLidarDriver, 238
  - ydlidar::YDLidarDriver, 322
  - ydlidar::core::common::DriverInterface, 221
- GetSdkVersion
  - ydlidar\_sdk.cpp, 416
  - ydlidar\_sdk.h, 425
- GetSeconds
  - CStatTimer, 193
- GetSendTimeoutSec
  - ydlidar::core::network::CSimpleSocket, 176
- GetSendTimeoutUsec
  - ydlidar::core::network::CSimpleSocket, 177
- GetSendWindowSize
  - ydlidar::core::network::CSimpleSocket, 177
- GetServerAddr
  - ydlidar::core::network::CSimpleSocket, 177
- GetServerPort
  - ydlidar::core::network::CSimpleSocket, 177
- getSingleChannel
  - ydlidar::core::common::DriverInterface, 222
- GetSocketDescriptor
  - ydlidar::core::network::CSimpleSocket, 178
- GetSocketDscp
  - ydlidar::core::network::CSimpleSocket, 178
- GetSocketError
  - ydlidar::core::network::CSimpleSocket, 178
- GetSocketType
  - ydlidar::core::network::CSimpleSocket, 178
- GetStartTime
  - CStatTimer, 193
- getStopbits
  - ydlidar::core::serial::Serial, 288
  - ydlidar::core::serial::Serial::SerialImpl, 301
- getSupportMotorDtrCtrl
  - ydlidar::core::common::DriverInterface, 222
- getSystemError
  - ydlidar::core::serial::Serial, 288
  - ydlidar::core::serial::Serial::SerialImpl, 301
- getTermios
  - ydlidar::core::serial::Serial::SerialImpl, 301
- getTime
  - timer.h, 345
- getTimeout
  - ydlidar::core::serial::Serial, 289
  - ydlidar::core::serial::Serial::SerialImpl, 302
- GetTotalTimeMs
  - ydlidar::core::network::CSimpleSocket, 179
- GetTotalTimeUsec
  - ydlidar::core::network::CSimpleSocket, 179
- getZeroOffsetAngle
  - ydlidar::ETLidarDriver, 239
  - ydlidar::YDLidarDriver, 323
  - ydlidar::core::common::DriverInterface, 222
- getlidaropt
  - CYdLidar, 206
  - ydlidar\_sdk.cpp, 414
  - ydlidar\_sdk.h, 423
- getms
  - timer.h, 345
- grabScanData
  - ydlidar::ETLidarDriver, 240
  - ydlidar::YDLidarDriver, 323
  - ydlidar::core::common::DriverInterface, 223
- hardware
  - LidarVersion, 262
- hardware\_id
  - ydlidar::core::serial::PortInfo, 273
- hardware\_version
  - device\_info, 212
  - ydlidar\_protocol.h, 374
- hasIntensity
  - ydlidar::core::common, 130
- hasSampleRate
  - ydlidar::core::common, 130
- hasScanFrequencyCtrl
  - ydlidar::core::common, 130



- hasZeroAngle
  - ydlidar::core::common, 131
- headFrameFlag
  - \_dataFrame, 146
- htonll
  - v8stdint.h, 349
- INVALID\_SOCKET
  - SimpleSocket.h, 384
- INVALID\_TIMESTAMP
  - datatype.h, 339
- IS\_FAIL
  - datatype.h, 339
- IS\_OK
  - datatype.h, 339
- IS\_TIMEOUT
  - datatype.h, 339
- impl, 116
  - getCurrentTime, 116
  - getHDTimer, 116
- index
  - node\_info, 266
  - ydlidar\_protocol.h, 374
- init
  - ydlidar::core::base, 126
  - ydlidar::core::base::Locker, 264
- Initialize
  - CStatTimer, 193
  - ydlidar::core::network::CSimpleSocket, 179
- initialize
  - CYdLidar, 208
  - ydlidar\_sdk.cpp, 416
  - ydlidar\_sdk.h, 425
- intensity
  - LaserPoint, 255
- inter\_byte\_timeout
  - ydlidar::core::serial::Timeout, 310
- is\_device\_locked
  - lock.c, 387
  - lock.h, 389
- is\_standardbaudrate
  - ydlidar::core::serial, 143
- isAutoReconnect
  - ydlidar::core::common::DriverInterface, 229
- isAutoconnting
  - ydlidar::core::common::DriverInterface, 229
- isNetTOFLidar
  - ydlidar::core::common, 131
- isNetTOFLidarByModel
  - ydlidar::core::common, 131
- IsNonblocking
  - ydlidar::core::network::CSimpleSocket, 179
- isOctaveLidar
  - ydlidar::core::common, 132
- isOldVersionTOFLidar
  - ydlidar::core::common, 132
- isOpen
  - ydlidar::core::common::ChannelDevice, 155
  - ydlidar::core::network::CSimpleSocket, 179
- ydlidar::core::serial::Serial, 289
- ydlidar::core::serial::Serial::SerialImpl, 302
- isSerialNumbValid
  - ydlidar::core::common, 132
- IsSocketValid
  - ydlidar::core::network::CSimpleSocket, 180
- isSupportLidar
  - ydlidar::core::common, 133
- isSupportMotorCtrl
  - ydlidar::core::common, 133
- isSupportScanFrequency
  - ydlidar::core::common, 133
- isTOFLidar
  - ydlidar::core::common, 133
- isTOFLidarByModel
  - ydlidar::core::common, 134
- isTriangleLidar
  - ydlidar::core::common, 134
- isV1Protocol
  - ydlidar::core::common, 134
- isValidSampleRate
  - ydlidar::core::common, 135
- isValidValue
  - ydlidar::core::common, 135
- isVersionValid
  - ydlidar::core::common, 135
- isconnected
  - ydlidar::ETLidarDriver, 240
  - ydlidar::YDLidarDriver, 324
  - ydlidar::core::common::DriverInterface, 223
- isscanning
  - ydlidar::ETLidarDriver, 241
  - ydlidar::YDLidarDriver, 324
  - ydlidar::core::common::DriverInterface, 223
- join
  - ydlidar::core::base::Thread, 308
- LIDAR\_2D
  - datatype.h, 339
- LIDAR\_ANS\_SYNC\_BYTE1
  - ydlidar\_protocol.h, 366
- LIDAR\_ANS\_SYNC\_BYTE2
  - ydlidar\_protocol.h, 366
- LIDAR\_ANS\_TYPE\_DEVHEALTH
  - ydlidar\_protocol.h, 366
- LIDAR\_ANS\_TYPE\_DEVINFO
  - ydlidar\_protocol.h, 367
- LIDAR\_ANS\_TYPE\_MEASUREMENT
  - ydlidar\_protocol.h, 367
- LIDAR\_CMD\_ADD\_EXPOSURE
  - ydlidar\_protocol.h, 367
- LIDAR\_CMD\_DIS\_EXPOSURE
  - ydlidar\_protocol.h, 367
- LIDAR\_CMD\_DISABLE\_CONST\_FREQ
  - ydlidar\_protocol.h, 367
- LIDAR\_CMD\_DISABLE\_LOW\_POWER
  - ydlidar\_protocol.h, 367
- LIDAR\_CMD\_ENABLE\_CONST\_FREQ



- ydliar\_protocol.h, [367](#)
- LIDAR\_CMD\_ENABLE\_LOW\_POWER
  - ydliar\_protocol.h, [367](#)
- LIDAR\_CMD\_FORCE\_SCAN
  - ydliar\_protocol.h, [367](#)
- LIDAR\_CMD\_FORCE\_STOP
  - ydliar\_protocol.h, [367](#)
- LIDAR\_CMD\_GET\_AIMSPEED
  - ydliar\_protocol.h, [368](#)
- LIDAR\_CMD\_GET\_DEVICE\_HEALTH
  - ydliar\_protocol.h, [368](#)
- LIDAR\_CMD\_GET\_DEVICE\_INFO
  - ydliar\_protocol.h, [368](#)
- LIDAR\_CMD\_GET\_EAI
  - ydliar\_protocol.h, [368](#)
- LIDAR\_CMD\_GET\_OFFSET\_ANGLE
  - ydliar\_protocol.h, [368](#)
- LIDAR\_CMD\_GET\_SAMPLING\_RATE
  - ydliar\_protocol.h, [368](#)
- LIDAR\_CMD\_RESET
  - ydliar\_protocol.h, [368](#)
- LIDAR\_CMD\_RUN\_INVERSION
  - ydliar\_protocol.h, [368](#)
- LIDAR\_CMD\_RUN\_POSITIVE
  - ydliar\_protocol.h, [368](#)
- LIDAR\_CMD\_SAVE\_SET\_EXPOSURE
  - ydliar\_protocol.h, [368](#)
- LIDAR\_CMD\_SCAN
  - ydliar\_protocol.h, [369](#)
- LIDAR\_CMD\_SET\_AIMSPEED\_ADDMIC
  - ydliar\_protocol.h, [369](#)
- LIDAR\_CMD\_SET\_AIMSPEED\_ADD
  - ydliar\_protocol.h, [369](#)
- LIDAR\_CMD\_SET\_AIMSPEED\_DISMIC
  - ydliar\_protocol.h, [369](#)
- LIDAR\_CMD\_SET\_AIMSPEED\_DIS
  - ydliar\_protocol.h, [369](#)
- LIDAR\_CMD\_SET\_LOW\_EXPOSURE
  - ydliar\_protocol.h, [369](#)
- LIDAR\_CMD\_SET\_SAMPLING\_RATE
  - ydliar\_protocol.h, [369](#)
- LIDAR\_CMD\_STATE\_MODEL\_MOTOR
  - ydliar\_protocol.h, [369](#)
- LIDAR\_CMD\_STOP
  - ydliar\_protocol.h, [369](#)
- LIDAR\_CMD\_SYNC\_BYTE
  - ydliar\_protocol.h, [369](#)
- LIDAR\_CMDFLAG\_HAS\_PAYLOAD
  - ydliar\_protocol.h, [370](#)
- LIDAR\_RESP\_MEASUREMENT\_ANGLE\_SAMPLE↔SHIFT
  - ydliar\_protocol.h, [370](#)
- LIDAR\_RESP\_MEASUREMENT\_ANGLE\_SHIFT
  - ydliar\_protocol.h, [370](#)
- LIDAR\_RESP\_MEASUREMENT\_CHECKBIT
  - ydliar\_protocol.h, [370](#)
- LIDAR\_RESP\_MEASUREMENT\_DISTANCE\_SHIFT
  - ydliar\_protocol.h, [370](#)
- LIDAR\_RESP\_MEASUREMENT\_QUALITY\_SHIFT
  - ydliar\_protocol.h, [370](#)
- LIDAR\_RESP\_MEASUREMENT\_SYNCBIT
  - ydliar\_protocol.h, [370](#)
- LIDAR\_STATUS\_ERROR
  - ydliar\_protocol.h, [370](#)
- LIDAR\_STATUS\_OK
  - ydliar\_protocol.h, [370](#)
- LIDAR\_STATUS\_WARNING
  - ydliar\_protocol.h, [370](#)
- LOCK\_FAILED
  - ydliar::core::base::Locker, [263](#)
- LOCK\_OK
  - ydliar::core::base::Locker, [263](#)
- LOCK\_STATUS
  - ydliar::core::base::Locker, [263](#)
- LOCK\_TIMEOUT
  - ydliar::core::base::Locker, [263](#)
- LOCK
  - lock.h, [389](#)
- LSTAT
  - v8stdint.h, [349](#)
- laser
  - etlidar\_test, [115](#)
  - plot\_tof\_test, [117](#)
  - plot\_ydliar\_test, [118](#)
  - test, [120](#)
  - tof\_test, [121](#)
  - ydliar\_test, [144](#)
- laser\_en
  - \_lidarConfig, [149](#)
- LaserConfig, [248](#)
  - angle\_increment, [249](#)
  - max\_angle, [249](#)
  - max\_range, [249](#)
  - min\_angle, [250](#)
  - min\_range, [250](#)
  - scan\_time, [250](#)
  - time\_increment, [250](#)
- LaserDebug, [250](#)
  - MaxDebugIndex, [251](#)
  - W1F6GNoise\_W1F5SNoise\_W1F4MotorCtl\_W4↔F0SnYear, [251](#)
  - W2F5Output2K4K5K\_W5F0Date, [251](#)
  - W3F4BoradHardVer\_W4F0Moth, [251](#)
  - W3F4CusHardVer\_W4F0CusSoftVer, [251](#)
  - W3F4CusMajor\_W4F0CusMinor, [251](#)
  - W3F4HardwareVer\_W4F0FirewareMajor, [252](#)
  - W4F3Model\_W3F0DebugInfTranVer, [252](#)
  - W7F0FirewareMinor, [252](#)
  - W7F0Health, [252](#)
  - W7F0LaserCurrent, [252](#)
  - W7F0SnNumH, [252](#)
  - W7F0SnNumL, [252](#)
- LaserFan, [253](#)
  - config, [254](#)
  - npoints, [254](#)
  - points, [254](#)

- stamp, 254
- LaserFanDestroy
  - ydlidar\_def.cpp, 356
  - ydlidar\_def.h, 359
- LaserFanInit
  - ydlidar\_def.cpp, 356
  - ydlidar\_def.h, 359
- LaserPoint, 255
  - angle, 255
  - intensity, 255
  - range, 255
- LaserScan, 256
  - config, 257
  - points, 257
  - stamp, 257
- laserScanFrequency
  - \_lidarConfig, 149
- lfs\_lock
  - lock.h, 390
- lfs\_unlock
  - lock.h, 390
- lib\_lock\_dev\_lock
  - lock.h, 390
- lib\_lock\_dev\_unlock
  - lock.h, 390
- lidar
  - YDLidar, 311
- lidar\_ans\_header, 258
  - size, 258
  - subType, 258
  - syncByte1, 258
  - syncByte2, 258
  - type, 258
- lidar\_c\_api\_test.c
  - main, 403
- lidar\_polar
  - plot\_tof\_test, 117
  - plot\_ydlidar\_test, 118
- lidar\_test.cpp
  - main, 429
  - TEST\_F, 429, 430
- lidarConfig, 259
  - ydlidar\_protocol.h, 372
- lidarCreate
  - ydlidar\_sdk.cpp, 417
  - ydlidar\_sdk.h, 425
- lidarDestroy
  - ydlidar\_sdk.cpp, 417
  - ydlidar\_sdk.h, 425
- lidarModelDefaultSampleRate
  - ydlidar::core::common, 136
- lidarModelToString
  - ydlidar::core::common, 136
- LidarPort, 259
  - port, 260
- lidarPortList
  - ydlidar, 124
  - ydlidar::YDLidarDriver, 325
  - ydlidar\_sdk.cpp, 417
  - ydlidar\_sdk.h, 426
- LidarPropAbnormalCheckCount
  - ydlidar\_def.h, 359
- LidarPropAutoReconnect
  - ydlidar\_def.h, 359
- LidarPropDeviceType
  - ydlidar\_def.h, 359
- LidarPropFixedResolution
  - ydlidar\_def.h, 359
- LidarPropIgnoreArray
  - ydlidar\_def.h, 359
- LidarPropIntenstiy
  - ydlidar\_def.h, 359
- LidarPropInverted
  - ydlidar\_def.h, 359
- LidarPropLidarType
  - ydlidar\_def.h, 359
- LidarPropMaxAngle
  - ydlidar\_def.h, 359
- LidarPropMaxRange
  - ydlidar\_def.h, 359
- LidarPropMinAngle
  - ydlidar\_def.h, 359
- LidarPropMinRange
  - ydlidar\_def.h, 359
- LidarPropReversion
  - ydlidar\_def.h, 359
- LidarPropSampleRate
  - ydlidar\_def.h, 359
- LidarPropScanFrequency
  - ydlidar\_def.h, 359
- LidarPropSerialBaudrate
  - ydlidar\_def.h, 359
- LidarPropSerialPort
  - ydlidar\_def.h, 359
- LidarPropSingleChannel
  - ydlidar\_def.h, 359
- LidarPropSupportMotorDtrCtrl
  - ydlidar\_def.h, 359
- LidarProperty
  - ydlidar\_def.h, 358
- LidarTest, 260
  - ~LidarTest, 261
  - LidarTest, 261
  - m\_lidar, 261
  - SetUp, 261
  - TearDown, 261
- LidarTypeID
  - ydlidar\_def.h, 359
- LidarVersion, 261
  - hardware, 262
  - sn, 262
  - soft\_major, 262
  - soft\_minor, 262
  - soft\_patch, 262
- list\_ports
  - ydlidar::core::serial, 143

- Listen
  - ydlidar::core::network::CPassiveSocket, 165
- lock
  - ydlidar::core::base::Locker, 264
- lock.c
  - check\_group\_uucp, 387
  - check\_lock\_pid, 387
  - check\_lock\_status, 387
  - fhs\_lock, 387
  - fhs\_unlock, 387
  - is\_device\_locked, 387
  - uucp\_lock, 387
  - uucp\_unlock, 387
- lock.h
  - check\_group\_uucp, 389
  - check\_lock\_pid, 389
  - check\_lock\_status, 389
  - fhs\_lock, 389
  - fhs\_unlock, 389
  - is\_device\_locked, 389
  - LOCK, 389
  - lfs\_lock, 390
  - lfs\_unlock, 390
  - lib\_lock\_dev\_lock, 390
  - lib\_lock\_dev\_unlock, 390
  - lock\_device, 390
  - UNLOCK, 389
  - unlock\_device, 390
  - uucp\_lock, 390
  - uucp\_unlock, 390
- lock\_device
  - lock.h, 390
- Locker
  - ydlidar::core::base::Locker, 263
- long\_description
  - setup::CMakeBuild, 160
- m\_LidarType
  - ydlidar::core::common::DriverInterface, 230
- M\_PI
  - angles.h, 379
  - ydlidar\_protocol.h, 371
- m\_PointTime
  - ydlidar::core::common::DriverInterface, 230
- m\_SingleChannel
  - ydlidar::core::common::DriverInterface, 230
- m\_SupportMotorDtrCtrl
  - ydlidar::core::common::DriverInterface, 231
- m\_WSASStartup
  - SimpleSocket.cpp, 383
- m\_addr
  - ydlidar::core::network::CSimpleSocket, 188
- m\_blsBlocking
  - ydlidar::core::network::CSimpleSocket, 188
- m\_blsMulticast
  - ydlidar::core::network::CSimpleSocket, 188
- m\_baudrate
  - ydlidar::core::common::DriverInterface, 229
- m\_config
  - ydlidar::core::common::DriverInterface, 229
- m\_errorFds
  - ydlidar::core::network::CSimpleSocket, 189
- m\_intensities
  - ydlidar::core::common::DriverInterface, 229
- m\_isConnected
  - ydlidar::core::common::DriverInterface, 229
- m\_isScanning
  - ydlidar::core::common::DriverInterface, 229
- m\_lidar
  - LidarTest, 261
- m\_nBufferSize
  - ydlidar::core::network::CSimpleSocket, 189
- m\_nBytesReceived
  - ydlidar::core::network::CSimpleSocket, 189
- m\_nBytesSent
  - ydlidar::core::network::CSimpleSocket, 189
- m\_nFlags
  - ydlidar::core::network::CSimpleSocket, 189
- m\_nSocketDomain
  - ydlidar::core::network::CSimpleSocket, 189
- m\_nSocketType
  - ydlidar::core::network::CSimpleSocket, 189
- m\_open
  - ydlidar::core::network::CSimpleSocket, 189
- m\_pBuffer
  - ydlidar::core::network::CSimpleSocket, 190
- m\_port
  - ydlidar::core::network::CSimpleSocket, 190
- m\_readFds
  - ydlidar::core::network::CSimpleSocket, 190
- m\_socket
  - ydlidar::core::network::CSimpleSocket, 190
- m\_socketErrno
  - ydlidar::core::network::CSimpleSocket, 190
- m\_stClientSockaddr
  - ydlidar::core::network::CSimpleSocket, 190
- m\_stConnectTimeout
  - ydlidar::core::network::CSimpleSocket, 190
- m\_stLinger
  - ydlidar::core::network::CSimpleSocket, 190
- m\_stMulticastGroup
  - ydlidar::core::network::CSimpleSocket, 191
- m\_stRecvTimeout
  - ydlidar::core::network::CSimpleSocket, 191
- m\_stSendTimeout
  - ydlidar::core::network::CSimpleSocket, 191
- m\_stServerSockaddr
  - ydlidar::core::network::CSimpleSocket, 191
- m\_timer
  - ydlidar::core::network::CSimpleSocket, 191
- m\_writeFds
  - ydlidar::core::network::CSimpleSocket, 191
- MAX\_SCAN\_NODES
  - ydlidar::core::common::DriverInterface, 218
- MICROSECONDS\_CONVERSION
  - v8stdint.h, 349
- MILLISECONDS\_CONVERSION

- v8stdint.h, 349
- main
  - etlidar\_test.cpp, 401
  - lidar\_c\_api\_test.c, 403
  - lidar\_test.cpp, 429
  - tof\_test.cpp, 404
  - ydlidar\_test.cpp, 406
- max
  - ydlidar::core::serial::Timeout, 310
- max\_angle
  - LaserConfig, 249
- max\_range
  - LaserConfig, 249
- MaxDebugIndex
  - LaserDebug, 251
- MillisecondTimer
  - ydlidar::core::serial::MillisecondTimer, 265
- min\_angle
  - LaserConfig, 250
- min\_range
  - LaserConfig, 250
- model
  - device\_info, 212
  - ydlidar\_protocol.h, 374
- motor\_en
  - \_lidarConfig, 149
- motor\_rpm
  - \_lidarConfig, 149
- NANOSECONDS\_CONVERSION
  - v8stdint.h, 349
- name
  - setup::CMakeBuild, 160
- NoError
  - ydlidar::core::serial::Serial, 284
- Node\_Default\_Quality
  - ydlidar\_protocol.h, 371
- Node\_NotSync
  - ydlidar\_protocol.h, 371
- Node\_Sync
  - ydlidar\_protocol.h, 371
- node\_info, 265
  - angle\_q6\_checkbit, 266
  - debugInfo, 266
  - distance\_q2, 266
  - error\_package, 266
  - index, 266
  - scan\_frequency, 266
  - stamp, 266
  - sync\_flag, 267
  - sync\_quality, 267
- node\_package, 267
  - checksum, 268
  - nowPackageNum, 268
  - package\_CT, 268
  - package\_Head, 268
  - packageFirstSampleAngle, 268
  - packageLastSampleAngle, 269
  - packageSample, 269
- node\_packages, 269
  - checksum, 270
  - nowPackageNum, 270
  - package\_CT, 270
  - package\_Head, 270
  - packageFirstSampleAngle, 270
  - packageLastSampleAngle, 270
  - packageSampleDistance, 270
- normalize\_angle
  - ydlidar::core::math, 139
- normalize\_angle\_positive
  - ydlidar::core::math, 139
- normalize\_angle\_positive\_from\_degree
  - ydlidar::core::math, 139
- NotOpenError
  - ydlidar::core::serial::Serial, 284
- nowPackageNum
  - node\_package, 268
  - node\_packages, 270
  - ydlidar\_protocol.h, 374
- npoints
  - LaserFan, 254
- ntohl
  - v8stdint.h, 350
- offset\_angle, 271
  - angle, 271
- ok
  - ydlidar::core::base, 126
- old\_signal\_handler
  - ydlidar.h, 353
- Open
  - ydlidar::core::network::CActiveSocket, 152
  - ydlidar::core::network::CSimpleSocket, 180
- open
  - ydlidar::core::common::ChannelDevice, 155
  - ydlidar::core::network::CSimpleSocket, 180
  - ydlidar::core::serial::Serial, 289
  - ydlidar::core::serial::Serial::SerialImpl, 302
- OpenError
  - ydlidar::core::serial::Serial, 284
- operator==
  - ydlidar::core::base::Thread, 308
- os\_init
  - ydlidar, 124
  - ydlidar\_sdk.cpp, 418
  - ydlidar\_sdk.h, 426
- os\_isOk
  - ydlidar, 124
  - ydlidar\_sdk.cpp, 418
  - ydlidar\_sdk.h, 426
- os\_shutdown
  - ydlidar, 125
  - ydlidar\_sdk.cpp, 418
  - ydlidar\_sdk.h, 426
- PRINTF
  - v8stdint.h, 350
- package\_CT

- node\_package, 268
- node\_packages, 270
- ydlidar\_protocol.h, 375
- package\_Head
  - node\_package, 268
  - node\_packages, 270
  - ydlidar\_protocol.h, 375
- package\_Sample\_Index
  - ydlidar::core::common::DriverInterface, 232
- packageFirstSampleAngle
  - node\_package, 268
  - node\_packages, 270
  - ydlidar\_protocol.h, 375
- packageLastSampleAngle
  - node\_package, 269
  - node\_packages, 270
  - ydlidar\_protocol.h, 375
- PackageNode, 271
  - PackageSampleDistance, 272
  - PackageSampleQuality, 272
- PackagePaidBytes
  - ydlidar\_protocol.h, 371
- packageSample
  - node\_package, 269
  - ydlidar\_protocol.h, 375
- packageSampleDistance
  - node\_packages, 270
  - ydlidar\_protocol.h, 375
- PackageSampleMaxLngth
  - ydlidar\_protocol.h, 371
- PackageSampleDistance
  - PackageNode, 272
  - ydlidar\_protocol.h, 375
- PackageSampleQuality
  - PackageNode, 272
  - ydlidar\_protocol.h, 375
- parity\_even
  - ydlidar::core::serial, 142
- parity\_mark
  - ydlidar::core::serial, 142
- parity\_none
  - ydlidar::core::serial, 142
- parity\_odd
  - ydlidar::core::serial, 142
- parity\_space
  - ydlidar::core::serial, 142
- parity\_t
  - ydlidar::core::serial, 142
- ParityError
  - ydlidar::core::serial::Serial, 284
- ParseLaserDebugInfo
  - ydlidar::core::common, 136
- parsePackageNode
  - ydlidar::core::common, 137
- PermissionError
  - ydlidar::core::serial::Serial, 284
- PH
  - ydlidar\_protocol.h, 371
- plot\_tof\_test, 116
  - ani, 117
  - animate, 116
  - fig, 117
  - laser, 117
  - lidar\_polar, 117
  - port, 117
  - ports, 117
  - RMAX, 117
  - ret, 117
  - scan, 117
- plot\_ydlidar\_test, 118
  - ani, 118
  - animate, 118
  - fig, 118
  - laser, 118
  - lidar\_polar, 118
  - port, 118
  - ports, 119
  - RMAX, 119
  - ret, 119
  - scan, 119
- points
  - LaserFan, 254
  - LaserScan, 257
- port
  - LidarPort, 260
  - plot\_tof\_test, 117
  - plot\_ydlidar\_test, 118
  - test, 120
  - tof\_test, 121
  - ydlidar::core::serial::PortInfo, 273
  - ydlidar\_test, 144
- ports
  - plot\_tof\_test, 117
  - plot\_ydlidar\_test, 119
  - test, 120
  - tof\_test, 121
  - ydlidar\_test, 144
- printfVersionInfo
  - ydlidar::core::common, 137
- PropertyBuilderByName
  - ydlidar.h, 352
- Protocol\_V1
  - ydlidar\_protocol.h, 372
- Protocol\_V2
  - ydlidar\_protocol.h, 372
- ProtocolVer
  - ydlidar\_protocol.h, 372
- pytest, 119
- pytest.PyTestTestCase, 274
- pytest::PyTestTestCase
  - testOSInitsWrappedCorrectly, 274
  - testParametersIsWrappedCorrectly, 274
- python/examples/etlidar\_test.py, 398
- python/examples/plot\_tof\_test.py, 398
- python/examples/plot\_ydlidar\_test.py, 399
- python/examples/test.py, 399

- python/examples/tof\_test.py, [400](#)
- python/examples/ydlidar\_test.py, [400](#)
- python/test/pytest.py, [400](#)
- r
  - etlidar\_test, [115](#)
  - test, [121](#)
  - tof\_test, [121](#)
  - ydlidar\_test, [144](#)
- README.md, [397](#)
- RESULT\_FAIL
  - datatype.h, [339](#)
- RESULT\_OK
  - datatype.h, [340](#)
- RESULT\_TIMEOUT
  - datatype.h, [340](#)
- RMAX
  - plot\_tof\_test, [117](#)
  - plot\_ydlidar\_test, [119](#)
- range
  - LaserPoint, [255](#)
- rate
  - sampling\_rate, [275](#)
  - ydlidar\_protocol.h, [376](#)
- read
  - ydlidar::core::serial::Serial, [289](#), [290](#)
  - ydlidar::core::serial::Serial::SerialImpl, [302](#)
- read\_timeout\_constant
  - ydlidar::core::serial::Timeout, [310](#)
- read\_timeout\_multiplier
  - ydlidar::core::serial::Timeout, [310](#)
- readData
  - ydlidar::core::common::ChannelDevice, [155](#)
  - ydlidar::core::network::CSimpleSocket, [180](#)
  - ydlidar::core::serial::Serial, [291](#)
- ReadError
  - ydlidar::core::serial::Serial, [284](#)
- readLock
  - ydlidar::core::serial::Serial::SerialImpl, [302](#)
- readSize
  - ydlidar::core::common::ChannelDevice, [156](#)
  - ydlidar::core::network::CSimpleSocket, [181](#)
  - ydlidar::core::serial::Serial, [292](#)
- readUnlock
  - ydlidar::core::serial::Serial::SerialImpl, [302](#)
- readline
  - ydlidar::core::serial::Serial, [291](#), [292](#)
- readlines
  - ydlidar::core::serial::Serial, [292](#)
- Receive
  - ydlidar::core::network::CSimpleSocket, [181](#)
- Receives
  - ydlidar::core::network::CSimpleSocket, [170](#)
- release
  - ydlidar::core::base::Event, [247](#)
  - ydlidar::core::base::Locker, [264](#)
- remaining
  - ydlidar::core::serial::MillisecondTimer, [265](#)
- removeExceptionSample
  - CYdLidar.cpp, [408](#)
- reset
  - ydlidar::YDlidarDriver, [325](#)
- ResourceError
  - ydlidar::core::serial::Serial, [284](#)
- result\_t
  - datatype.h, [340](#)
- ret
  - etlidar\_test, [115](#)
  - plot\_tof\_test, [117](#)
  - plot\_ydlidar\_test, [119](#)
  - test, [121](#)
  - tof\_test, [122](#)
  - ydlidar\_test, [144](#)
- retryCount
  - ydlidar::core::common::DriverInterface, [232](#)
- rotation
  - scan\_rotation, [278](#)
  - ydlidar\_protocol.h, [376](#)
- run
  - setup::CMakeBuild, [159](#)
- SNCCS
  - unix\_serial.cpp, [392](#)
- SNPRINTF
  - v8stdint.h, [350](#)
- SOCKET\_SENDFILE\_BLOCKSIZE
  - SimpleSocket.h, [384](#)
- STAT\_BLK\_SIZE
  - v8stdint.h, [350](#)
- STRTOUULL
  - v8stdint.h, [350](#)
- STRUCT\_STAT
  - v8stdint.h, [350](#)
- SUNNOISEINTENSITY
  - ydlidar\_protocol.h, [371](#)
- samples/etlidar\_test.cpp, [401](#)
- samples/lidar\_c\_api\_test.c, [403](#)
- samples/tof\_test.cpp, [404](#)
- samples/ydlidar\_test.cpp, [406](#)
- sampling\_rate, [275](#)
  - rate, [275](#)
- scan
  - etlidar\_test, [115](#)
  - plot\_tof\_test, [117](#)
  - plot\_ydlidar\_test, [119](#)
  - test, [121](#)
  - tof\_test, [122](#)
  - ydlidar\_test, [144](#)
- scan\_exposure, [275](#)
  - exposure, [276](#)
- scan\_frequence
  - node\_info, [266](#)
  - ydlidar\_protocol.h, [376](#)
- scan\_frequency, [276](#)
  - frequency, [277](#)
- scan\_heart\_beat, [277](#)
  - enable, [277](#)
- scan\_node\_buf

- ydlidar::core::common::DriverInterface, 232
- scan\_node\_count
  - ydlidar::core::common::DriverInterface, 232
- scan\_points, 277
  - flag, 278
- scan\_rotation, 278
  - rotation, 278
- scan\_time
  - LaserConfig, 250
- ScopedLocker
  - ydlidar::core::base::ScopedLocker, 279
- ScopedReadLock
  - ydlidar::core::serial::Serial::ScopedReadLock, 280
- ScopedWriteLock
  - ydlidar::core::serial::Serial::ScopedWriteLock, 281
- Select
  - ydlidar::core::network::CSimpleSocket, 181, 182
- Send
  - ydlidar::core::network::CPassiveSocket, 165
  - ydlidar::core::network::CSimpleSocket, 182
- sendBreak
  - ydlidar::core::serial::Serial, 293
  - ydlidar::core::serial::Serial::SerialImpl, 302
- sendCommand
  - ydlidar::YDLidarDriver, 326
- sendData
  - ydlidar::YDLidarDriver, 326
- SendFile
  - ydlidar::core::network::CSimpleSocket, 183
- Sends
  - ydlidar::core::network::CSimpleSocket, 170
- Serial
  - ydlidar::core::serial::Serial, 284
- serial, 119
- serial::Serial, 119
- serial::Serial::ScopedReadLock, 280
- serial::Serial::ScopedWriteLock, 281
- serial::Serial::SerialImpl, 298
- serial\_port
  - ydlidar::core::common::DriverInterface, 232
- SerialImpl
  - ydlidar::core::serial::Serial::SerialImpl, 300
- SerialPortError
  - ydlidar::core::serial::Serial, 284
- serialnum
  - device\_info, 213
  - ydlidar\_protocol.h, 376
- set
  - ydlidar::core::base::Event, 247
- set\_common\_props
  - ydlidar::core::serial, 143
- set\_databits
  - ydlidar::core::serial, 143
- set\_flowcontrol
  - ydlidar::core::serial, 143
- set\_parity
  - ydlidar::core::serial, 143
- set\_signal\_handler
  - ydlidar.h, 352
- set\_stopbits
  - ydlidar::core::serial, 143
- setAutoReconnect
  - ydlidar::ETLidarDriver, 241
  - ydlidar::YDLidarDriver, 327
  - ydlidar::core::common::DriverInterface, 224
- setBaudrate
  - ydlidar::core::serial::Serial, 293
  - ydlidar::core::serial::Serial::SerialImpl, 302
- SetBlocking
  - ydlidar::core::network::CSimpleSocket, 183
- setBreak
  - ydlidar::core::serial::Serial, 293
  - ydlidar::core::serial::Serial::SerialImpl, 302
- setBytesize
  - ydlidar::core::serial::Serial, 293
  - ydlidar::core::serial::Serial::SerialImpl, 302
- SetConnectTimeout
  - ydlidar::core::network::CSimpleSocket, 183
- setCustomBaudRate
  - ydlidar::core::serial::Serial::SerialImpl, 303
- setDTR
  - ydlidar::YDLidarDriver, 327
  - ydlidar::core::common::ChannelDevice, 156
  - ydlidar::core::serial::Serial, 294
  - ydlidar::core::serial::Serial::SerialImpl, 303
- SetEndTime
  - CStatTimer, 193
- setFlowcontrol
  - ydlidar::core::serial::Serial, 294
  - ydlidar::core::serial::Serial::SerialImpl, 303
- setIntensities
  - ydlidar::ETLidarDriver, 241
  - ydlidar::YDLidarDriver, 327
  - ydlidar::core::common::DriverInterface, 224
- setLidarType
  - ydlidar::core::common::DriverInterface, 224
- SetMulticast
  - ydlidar::core::network::CSimpleSocket, 184
- SetNonblocking
  - ydlidar::core::network::CSimpleSocket, 184
- SetOptionLinger
  - ydlidar::core::network::CSimpleSocket, 184
- SetOptionReuseAddr
  - ydlidar::core::network::CSimpleSocket, 185
- setParity
  - ydlidar::core::serial::Serial, 294
  - ydlidar::core::serial::Serial::SerialImpl, 303
- setPointTime
  - ydlidar::core::common::DriverInterface, 224
- setPort
  - ydlidar::core::serial::Serial, 294
  - ydlidar::core::serial::Serial::SerialImpl, 303
- setRTS
  - ydlidar::core::serial::Serial, 295
  - ydlidar::core::serial::Serial::SerialImpl, 303
- SetReceiveTimeout



- ydlidar::core::network::CSimpleSocket, 185
- SetReceiveWindowSize
  - ydlidar::core::network::CSimpleSocket, 185
- setSamplingRate
  - ydlidar::ETLidarDriver, 241
  - ydlidar::YDLidarDriver, 327
  - ydlidar::core::common::DriverInterface, 224
- setScanFrequencyAdd
  - ydlidar::ETLidarDriver, 242
  - ydlidar::YDLidarDriver, 328
  - ydlidar::core::common::DriverInterface, 225
- setScanFrequencyAddMic
  - ydlidar::ETLidarDriver, 243
  - ydlidar::YDLidarDriver, 328
  - ydlidar::core::common::DriverInterface, 226
- setScanFrequencyDis
  - ydlidar::ETLidarDriver, 243
  - ydlidar::YDLidarDriver, 329
  - ydlidar::core::common::DriverInterface, 226
- setScanFrequencyDisMic
  - ydlidar::ETLidarDriver, 244
  - ydlidar::YDLidarDriver, 329
  - ydlidar::core::common::DriverInterface, 227
- SetSendTimeout
  - ydlidar::core::network::CSimpleSocket, 185
- SetSendWindowSize
  - ydlidar::core::network::CSimpleSocket, 186
- setSingleChannel
  - ydlidar::core::common::DriverInterface, 227
- SetSocketDscp
  - ydlidar::core::network::CSimpleSocket, 186
- SetSocketError
  - ydlidar::core::network::CSimpleSocket, 186
- SetSocketHandle
  - ydlidar::core::network::CSimpleSocket, 187
- SetSocketType
  - ydlidar::core::network::CSimpleSocket, 187
- setStandardBaudRate
  - ydlidar::core::serial::Serial::SerialImpl, 303
- SetStartTime
  - CStatTimer, 193
- setStopbits
  - ydlidar::core::serial::Serial, 295
  - ydlidar::core::serial::Serial::SerialImpl, 303
- setSupportMotorDtrCtrl
  - ydlidar::core::common::DriverInterface, 227
- setTermios
  - ydlidar::core::serial::Serial::SerialImpl, 303
- setTimeout
  - ydlidar::core::serial::Serial, 295, 296
  - ydlidar::core::serial::Serial::SerialImpl, 303
- SetUp
  - LidarTest, 261
- setlidaropt
  - CYdLidar, 208
  - ydlidar\_sdk.cpp, 418
  - ydlidar\_sdk.h, 426
- setup, 120
  - YDLIDAR\_SDK\_BRANCH, 120
  - YDLIDAR\_SDK\_REPO, 120
- setup.CMakeBuild, 158
- setup.CMakeExtension, 160
- setup.py, 407
- setup::CMakeBuild
  - author, 159
  - author\_email, 159
  - build\_extension, 159
  - clone, 159
  - cmdclass, 159
  - description, 159
  - ext\_modules, 159
  - long\_description, 160
  - name, 160
  - run, 159
  - url, 160
  - version, 160
  - zip\_safe, 160
- setup::CMakeExtension
  - \_\_init\_\_, 161
  - sourcedir, 161
- sevenbits
  - ydlidar::core::serial, 142
- shortest\_angular\_distance
  - ydlidar::core::math, 139
- shortest\_angular\_distance\_with\_limits
  - ydlidar::core::math, 140
- Shutdown
  - ydlidar::core::network::CSimpleSocket, 187
- shutdown
  - ydlidar::core::base, 126
- signal\_handler
  - ydlidar.h, 352
- signal\_handler\_t
  - ydlidar.h, 352
- SimpleSocket.cpp
  - m\_WSAStartup, 383
- SimpleSocket.h
  - INVALID\_SOCKET, 384
  - SOCKET\_SENDFILE\_BLOCKSIZE, 384
- simpleTimeout
  - ydlidar::core::serial::Timeout, 310
- sixbits
  - ydlidar::core::serial, 142
- size
  - cmd\_packet, 162
  - lidar\_ans\_header, 258
  - ydlidar\_protocol.h, 376
- sn
  - LidarVersion, 262
- SocketAddressInUse
  - ydlidar::core::network::CSimpleSocket, 171
- SocketConnectionAborted
  - ydlidar::core::network::CSimpleSocket, 171
- SocketConnectionRefused
  - ydlidar::core::network::CSimpleSocket, 171
- SocketConnectionReset



- ydlidar::core::network::CSimpleSocket, 171
- SocketEinprogress
  - ydlidar::core::network::CSimpleSocket, 171
- SocketError
  - ydlidar::core::network::CSimpleSocket, 171
- SocketEunknown
  - ydlidar::core::network::CSimpleSocket, 171
- SocketEwouldblock
  - ydlidar::core::network::CSimpleSocket, 171
- SocketFirewallError
  - ydlidar::core::network::CSimpleSocket, 171
- SocketInterrupted
  - ydlidar::core::network::CSimpleSocket, 171
- SocketInvalidAddress
  - ydlidar::core::network::CSimpleSocket, 171
- SocketInvalidPointer
  - ydlidar::core::network::CSimpleSocket, 171
- SocketInvalidPort
  - ydlidar::core::network::CSimpleSocket, 171
- SocketInvalidSocket
  - ydlidar::core::network::CSimpleSocket, 171
- SocketInvalidSocketBuffer
  - ydlidar::core::network::CSimpleSocket, 171
- SocketNotconnected
  - ydlidar::core::network::CSimpleSocket, 171
- SocketProtocolError
  - ydlidar::core::network::CSimpleSocket, 171
- SocketSuccess
  - ydlidar::core::network::CSimpleSocket, 171
- SocketTimedout
  - ydlidar::core::network::CSimpleSocket, 171
- SocketTypeInvalid
  - ydlidar::core::network::CSimpleSocket, 171
- SocketTypeRaw
  - ydlidar::core::network::CSimpleSocket, 171
- SocketTypeTcp
  - ydlidar::core::network::CSimpleSocket, 171
- SocketTypeTcp6
  - ydlidar::core::network::CSimpleSocket, 171
- SocketTypeUdp
  - ydlidar::core::network::CSimpleSocket, 171
- SocketTypeUdp6
  - ydlidar::core::network::CSimpleSocket, 171
- soft\_major
  - LidarVersion, 262
- soft\_minor
  - LidarVersion, 262
- soft\_patch
  - LidarVersion, 262
- sourcedir
  - setup::CMakeExtension, 161
- split
  - ydlidar::core::common, 137
- src/CYdLidar.cpp, 408
- src/CYdLidar.h, 409
- src/ETLidarDriver.cpp, 410
- src/ETLidarDriver.h, 410
- src/ydlidar\_driver.cpp, 411
- src/ydlidar\_driver.h, 412
- src/ydlidar\_sdk.cpp, 413
- src/ydlidar\_sdk.h, 421
- stamp
  - LaserFan, 254
  - LaserScan, 257
  - node\_info, 266
  - ydlidar\_protocol.h, 376
- startAngle
  - \_dataFrame, 146
- startAutoScan
  - ydlidar::YDLidarDriver, 330
- startMotor
  - ydlidar::YDLidarDriver, 331
- startScan
  - ydlidar::ETLidarDriver, 244
  - ydlidar::YDLidarDriver, 331
  - ydlidar::core::common::DriverInterface, 227
- StatTimer.h
  - GET\_CLOCK\_COUNT, 385
- state
  - function\_state, 248
  - ydlidar\_protocol.h, 376
- status
  - device\_health, 211
  - ydlidar\_protocol.h, 376
- stop
  - ydlidar::ETLidarDriver, 245
  - ydlidar::YDLidarDriver, 331
  - ydlidar::core::common::DriverInterface, 228
- stopMotor
  - ydlidar::YDLidarDriver, 332
- stopScan
  - ydlidar::YDLidarDriver, 332
- stopbits\_one
  - ydlidar::core::serial, 142
- stopbits\_one\_point\_five
  - ydlidar::core::serial, 142
- stopbits\_t
  - ydlidar::core::serial, 142
- stopbits\_two
  - ydlidar::core::serial, 142
- string\_t, 305
  - data, 305
- subType
  - lidar\_ans\_header, 258
  - ydlidar\_protocol.h, 377
- sync\_flag
  - node\_info, 267
  - ydlidar\_protocol.h, 377
- sync\_quality
  - node\_info, 267
  - ydlidar\_protocol.h, 377
- syncByte
  - cmd\_packet, 162
  - ydlidar\_protocol.h, 377
- syncByte1
  - lidar\_ans\_header, 258

- ydlidar\_protocol.h, 377
- syncByte2
  - lidar\_ans\_header, 258
  - ydlidar\_protocol.h, 377
- TCGETS2
  - unix\_serial.cpp, 392
- TCSETS2
  - unix\_serial.cpp, 392
- TEST\_F
  - lidar\_test.cpp, 429, 430
- TIOCINQ
  - unix\_serial.cpp, 392
- TRUE
  - v8stdint.h, 350
- TYPE\_TOF\_NET
  - ydlidar\_def.h, 359
- TYPE\_TOF
  - ydlidar\_def.h, 359
- TYPE\_TRIANGLE
  - ydlidar\_def.h, 359
- TYPE\_Tail
  - ydlidar\_def.h, 359
- TearDown
  - LidarTest, 261
- terminate
  - ydlidar::core::base::Thread, 308
- test, 120
  - laser, 120
  - port, 120
  - ports, 120
  - r, 121
  - ret, 121
  - scan, 121
- test/lidar\_test.cpp, 429
- test/lidar\_test.h, 430
- testOSInitIsWrappedCorrectly
  - pytest::PyTestTestCase, 274
- testParametersIsWrappedCorrectly
  - pytest::PyTestTestCase, 274
- Thread
  - ydlidar::core::base::Thread, 307
- thread.h
  - CLASS\_THREAD, 343
- thread\_proc\_t
  - v8stdint.h, 350
- ThreadCreateObjectFunctor
  - ydlidar::core::base::Thread, 308
- time\_increment
  - LaserConfig, 250
- Timeout
  - ydlidar::core::serial::Timeout, 310
- TimeoutError
  - ydlidar::core::serial::Serial, 284
- timer.h
  - BEGIN\_STATIC\_CODE, 345
  - delay, 345
  - END\_STATIC\_CODE, 345
  - getTime, 345
  - getms, 345
- timespec\_from\_ms
  - ydlidar::core::serial, 143
- timestamp
  - \_dataFrame, 147
- to\_degrees
  - ydlidar::core::math, 140
- tof\_test, 121
  - laser, 121
  - port, 121
  - ports, 121
  - r, 121
  - ret, 122
  - scan, 122
- tof\_test.cpp
  - main, 404
- trans\_sel
  - \_lidarConfig, 149
- TranslateSocketError
  - ydlidar::core::network::CSimpleSocket, 187
- trigger\_interrupt\_guard\_condition
  - ydlidar.h, 353
- turnOff
  - CYdLidar, 210
  - ydlidar\_sdk.cpp, 420
  - ydlidar\_sdk.h, 428
- turnOn
  - CYdLidar, 210
  - ydlidar\_sdk.cpp, 420
  - ydlidar\_sdk.h, 428
- two\_pi\_complement
  - ydlidar::core::math, 140
- type
  - lidar\_ans\_header, 258
  - ydlidar\_protocol.h, 377
- UNLOCK
  - lock.h, 389
- UNUSED
  - datatype.h, 340
- unix\_serial.cpp
  - BOTHER, 392
  - SNCCS, 392
  - TCGETS2, 392
  - TCSETS2, 392
  - TIOCINQ, 392
- UnknownError
  - ydlidar::core::serial::Serial, 284
- unlock
  - ydlidar::core::base::Locker, 264
- unlock\_device
  - lock.h, 390
- UnsupportedOperationError
  - ydlidar::core::serial::Serial, 284
- updateScanCfg
  - ydlidar::ETLidarDriver, 245
- url
  - setup::CMakeBuild, 160
- utils.h

- YDLIDAR\_API, 346
- uucp\_lock
  - lock.c, 387
  - lock.h, 390
- uucp\_unlock
  - lock.c, 387
  - lock.h, 390
- v8stdint.h
  - CLEARERR, 348
  - DEFAULT\_CONNECTION\_TIMEOUT\_SEC, 348
  - DEFAULT\_CONNECTION\_TIMEOUT\_USEC, 348
  - DEFAULT\_REV\_TIMEOUT\_SEC, 348
  - DEFAULT\_REV\_TIMEOUT\_USEC, 348
  - FALSE, 348
  - FCLOSE, 348
  - FEOF, 348
  - FERROR, 348
  - FFLUSH, 348
  - FILE\_HANDLE, 349
  - FILENO, 349
  - FOPEN, 349
  - FPRINTF, 349
  - FSTAT, 349
  - htonll, 349
  - LSTAT, 349
  - MICROSECONDS\_CONVERSION, 349
  - MILLISECONDS\_CONVERSION, 349
  - NANOSECONDS\_CONVERSION, 349
  - ntohll, 350
  - PRINTF, 350
  - SNPRINTF, 350
  - STAT\_BLK\_SIZE, 350
  - STRTOU, 350
  - STRUCT\_STAT, 350
  - TRUE, 350
  - thread\_proc\_t, 350
  - VPRINTF, 350
- VPRINTF
  - v8stdint.h, 350
- valLastName
  - datatype.h, 340
- valName
  - datatype.h, 340
- version
  - setup::CMakeBuild, 160
- W1F6GNoise\_W1F5SNoise\_W1F4MotorCtl\_W4F0↵
  - SnYear
    - LaserDebug, 251
- W2F5Output2K4K5K\_W5F0Date
  - LaserDebug, 251
- W3F4BoradHardVer\_W4F0Moth
  - LaserDebug, 251
- W3F4CusHardVer\_W4F0CusSoftVer
  - LaserDebug, 251
- W3F4CusMajor\_W4F0CusMinor
  - LaserDebug, 251
- W3F4HardwareVer\_W4F0FirewareMajor
  - LaserDebug, 252
- W4F3Model\_W3F0DebugInfTranVer
  - LaserDebug, 252
- W7F0FirewareMinor
  - LaserDebug, 252
- W7F0Health
  - LaserDebug, 252
- W7F0LaserCurrent
  - LaserDebug, 252
- W7F0SnNumH
  - LaserDebug, 252
- W7F0SnNumL
  - LaserDebug, 252
- WSACleanUp
  - ydlidar::core::network::CSimpleSocket, 188
- wait
  - ydlidar::core::base::Event, 247
- waitByteTimes
  - ydlidar::core::serial::Serial, 296
  - ydlidar::core::serial::Serial::SerialImpl, 304
- waitDevicePackage
  - ydlidar::YDlidarDriver, 333
- waitForChange
  - ydlidar::core::serial::Serial, 296
  - ydlidar::core::serial::Serial::SerialImpl, 304
- WaitForData
  - ydlidar::core::network::CSimpleSocket, 187
- waitForData
  - ydlidar::YDlidarDriver, 333
- waitPackage
  - ydlidar::YDlidarDriver, 334
- waitReadable
  - ydlidar::core::serial::Serial, 297
  - ydlidar::core::serial::Serial::SerialImpl, 304
- waitResponseHeader
  - ydlidar::YDlidarDriver, 334
- waitScanData
  - ydlidar::YDlidarDriver, 334
- waitfordata
  - ydlidar::core::common::ChannelDevice, 157
  - ydlidar::core::network::CSimpleSocket, 187
  - ydlidar::core::serial::Serial, 296
  - ydlidar::core::serial::Serial::SerialImpl, 304
- write
  - ydlidar::core::serial::Serial, 297, 298
  - ydlidar::core::serial::Serial::SerialImpl, 304
- write\_timeout\_constant
  - ydlidar::core::serial::Timeout, 310
- write\_timeout\_multiplier
  - ydlidar::core::serial::Timeout, 311
- writeData
  - ydlidar::core::common::ChannelDevice, 157
  - ydlidar::core::network::CSimpleSocket, 188
  - ydlidar::core::serial::Serial, 298
- WriteError
  - ydlidar::core::serial::Serial, 284
- writeLock
  - ydlidar::core::serial::Serial::SerialImpl, 304

- writeUnlock
  - ydlidar::core::serial::Serial::SerialImpl, 304
- YDLIDAR\_API
  - utils.h, 346
- YDLIDAR\_F2
  - ydlidar::core::common::DriverInterface, 217
- YDLIDAR\_F4
  - ydlidar::core::common::DriverInterface, 217
- YDLIDAR\_F4PRO
  - ydlidar::core::common::DriverInterface, 217
- YDLIDAR\_G1
  - ydlidar::core::common::DriverInterface, 217
- YDLIDAR\_G10
  - ydlidar::core::common::DriverInterface, 217
- YDLIDAR\_G2A
  - ydlidar::core::common::DriverInterface, 217
- YDLIDAR\_G2B
  - ydlidar::core::common::DriverInterface, 217
- YDLIDAR\_G2C
  - ydlidar::core::common::DriverInterface, 217
- YDLIDAR\_G4
  - ydlidar::core::common::DriverInterface, 217
- YDLIDAR\_G4PRO
  - ydlidar::core::common::DriverInterface, 217
- YDLIDAR\_G4B
  - ydlidar::core::common::DriverInterface, 217
- YDLIDAR\_G4C
  - ydlidar::core::common::DriverInterface, 217
- YDLIDAR\_G6
  - ydlidar::core::common::DriverInterface, 217
- YDLIDAR\_R2
  - ydlidar::core::common::DriverInterface, 217
- YDLIDAR\_RATE\_10K
  - ydlidar::core::common::DriverInterface, 218
- YDLIDAR\_RATE\_4K
  - ydlidar::core::common::DriverInterface, 218
- YDLIDAR\_RATE\_8K
  - ydlidar::core::common::DriverInterface, 218
- YDLIDAR\_RATE\_9K
  - ydlidar::core::common::DriverInterface, 218
- YDLIDAR\_S2
  - ydlidar::core::common::DriverInterface, 217
- YDLIDAR\_S4
  - ydlidar::core::common::DriverInterface, 217
- YDLIDAR\_S4B
  - ydlidar::core::common::DriverInterface, 217
- YDLIDAR\_SDK\_BRANCH
  - setup, 120
- YDLIDAR\_SDK\_REPO
  - setup, 120
- YDLIDAR\_T1
  - ydlidar::core::common::DriverInterface, 217
- YDLIDAR\_T15
  - ydlidar::core::common::DriverInterface, 217
- YDLIDAR\_TG15
  - ydlidar::core::common::DriverInterface, 217
- YDLIDAR\_TG30
  - ydlidar::core::common::DriverInterface, 217
- YDLIDAR\_TG50
  - ydlidar::core::common::DriverInterface, 217
- YDLIDAR\_TYPC\_UDP
  - ydlidar\_def.h, 358
- YDLIDAR\_TYPE\_SERIAL
  - ydlidar\_def.h, 358
- YDLIDAR\_TYPE\_TCP
  - ydlidar\_def.h, 358
- YDLIDAR\_Tail
  - ydlidar::core::common::DriverInterface, 217
- YDLIDAR\_X4
  - ydlidar::core::common::DriverInterface, 217
- YDLidar, 311
  - lidar, 311
- YDLidarDriver
  - ydlidar::YDLidarDriver, 315
- ydlidar, 122
  - lidarPortList, 124
  - os\_init, 124
  - os\_isOk, 124
  - os\_shutdown, 125
- ydlidar.h
  - g\_signal\_status, 353
  - old\_signal\_handler, 353
  - PropertyBuilderByName, 352
  - set\_signal\_handler, 352
  - signal\_handler, 352
  - signal\_handler\_t, 352
  - trigger\_interrupt\_guard\_condition, 353
- ydlidar::ETLidarDriver, 233
  - ~ETLidarDriver, 235
  - connect, 235
  - DescribeError, 236
  - disconnect, 236
  - ETLidarDriver, 235
  - getDeviceInfo, 236
  - getFinishedScanCfg, 237
  - getHealth, 237
  - getSDKVersion, 239
  - getSamplingRate, 237
  - getScanCfg, 238
  - getScanFrequency, 238
  - getZeroOffsetAngle, 239
  - grabScanData, 240
  - isconnected, 240
  - isscanning, 241
  - setAutoReconnect, 241
  - setIntensities, 241
  - setSamplingRate, 241
  - setScanFrequencyAdd, 242
  - setScanFrequencyAddMic, 243
  - setScanFrequencyDis, 243
  - setScanFrequencyDisMic, 244
  - startScan, 244
  - stop, 245
  - updateScanCfg, 245
- ydlidar::YDLidarDriver, 312
  - ~YDLidarDriver, 315

- ascendScanData, 316
- cacheScanData, 317
- checkAutoConnecting, 317
- checkDeviceInfo, 317
- checkTransDelay, 318
- clearDTR, 318
- connect, 318
- createThread, 319
- DescribeError, 319
- disableDataGrabbing, 319
- disconnect, 319
- flushSerial, 319
- getAutoZeroOffsetAngle, 320
- getData, 320
- getDeviceInfo, 321
- getHealth, 321
- getSDKVersion, 323
- getSamplingRate, 321
- getScanFrequency, 322
- getZeroOffsetAngle, 323
- grabScanData, 323
- isconnected, 324
- isscanning, 324
- lidarPortList, 325
- reset, 325
- sendCommand, 326
- sendData, 326
- setAutoReconnect, 327
- setDTR, 327
- setIntensities, 327
- setSamplingRate, 327
- setScanFrequencyAdd, 328
- setScanFrequencyAddMic, 328
- setScanFrequencyDis, 329
- setScanFrequencyDisMic, 329
- startAutoScan, 330
- startMotor, 331
- startScan, 331
- stop, 331
- stopMotor, 332
- stopScan, 332
- waitDevicePackage, 333
- waitForData, 333
- waitPackage, 334
- waitResponseHeader, 334
- waitScanData, 334
- YDlidarDriver, 315
- ydliar::core, 125
- ydliar::core::base, 125
  - fileExists, 126
  - init, 126
  - ok, 126
  - shutdown, 126
- ydliar::core::base::Event, 246
  - \_cond\_catr, 247
  - \_cond\_locker, 247
  - \_cond\_var, 247
  - \_isAutoReset, 247
  - \_is\_signalled, 247
  - ~Event, 247
  - EVENT\_FAILED, 246
  - EVENT\_OK, 246
  - EVENT\_TIMEOUT, 246
  - Event, 247
  - release, 247
  - set, 247
  - wait, 247
- ydliar::core::base::Locker, 263
  - \_lock, 264
  - ~Locker, 263
  - getLockHandle, 264
  - init, 264
  - LOCK\_FAILED, 263
  - LOCK\_OK, 263
  - LOCK\_STATUS, 263
  - LOCK\_TIMEOUT, 263
  - lock, 264
  - Locker, 263
  - release, 264
  - unlock, 264
- ydliar::core::base::ScopedLocker, 279
  - \_binded, 280
  - ~ScopedLocker, 279
  - forceUnlock, 280
  - ScopedLocker, 279
- ydliar::core::base::Thread, 306
  - \_func, 308
  - \_handle, 308
  - \_param, 309
  - ~Thread, 307
  - createThread, 308
  - createThreadAux, 308
  - getHandle, 308
  - getParam, 308
  - join, 308
  - operator==, 308
  - terminate, 308
  - Thread, 307
  - ThreadCreateObjectFunctor, 308
- ydliar::core::common, 127
  - ConvertLidarToUserSmample, 128
  - ConvertUserToLidarSmample, 128
  - hasIntensity, 130
  - hasSampleRate, 130
  - hasScanFrequencyCtrl, 130
  - hasZeroAngle, 131
  - isNetTOFLidar, 131
  - isNetTOFLidarByModel, 131
  - isOctaveLidar, 132
  - isOldVersionTOFLidar, 132
  - isSerialNumbValid, 132
  - isSupportLidar, 133
  - isSupportMotorCtrl, 133
  - isSupportScanFrequency, 133
  - isTOFLidar, 133
  - isTOFLidarByModel, 134

- isTriangleLidar, [134](#)
- isV1Protocol, [134](#)
- isValidSampleRate, [135](#)
- isValidValue, [135](#)
- isVersionValid, [135](#)
- lidarModelDefaultSampleRate, [136](#)
- lidarModelToString, [136](#)
- ParseLaserDebugInfo, [136](#)
- parsePackageNode, [137](#)
- printfVersionInfo, [137](#)
- split, [137](#)
- ydlidar::core::common::ChannelDevice, [152](#)
  - ~ChannelDevice, [154](#)
  - available, [154](#)
  - bindport, [154](#)
  - ChannelDevice, [154](#)
  - closePort, [154](#)
  - DescribeError, [154](#)
  - flush, [155](#)
  - getByteTime, [155](#)
  - isOpen, [155](#)
  - open, [155](#)
  - readData, [155](#)
  - readSize, [156](#)
  - setDTR, [156](#)
  - waitfordata, [157](#)
  - writeData, [157](#)
- ydlidar::core::common::DriverInterface, [213](#)
  - \_cmd\_lock, [228](#)
  - \_dataEvent, [228](#)
  - \_lock, [228](#)
  - \_thread, [229](#)
  - ~DriverInterface, [218](#)
  - connect, [218](#)
  - DEFAULT\_HEART\_BEAT, [218](#)
  - DEFAULT\_TIMEOUT\_COUNT, [218](#)
  - DEFAULT\_TIMEOUT, [218](#)
  - DescribeError, [219](#)
  - disconnect, [219](#)
  - DriverInterface, [218](#)
  - getDeviceInfo, [219](#)
  - getFinishedScanCfg, [220](#)
  - getHealth, [220](#)
  - getLidarType, [220](#)
  - getPointTime, [220](#)
  - getSDKVersion, [222](#)
  - getSamplingRate, [221](#)
  - getScanFrequency, [221](#)
  - getSingleChannel, [222](#)
  - getSupportMotorDtrCtrl, [222](#)
  - getZeroOffsetAngle, [222](#)
  - grabScanData, [223](#)
  - isAutoReconnect, [229](#)
  - isAutoconnting, [229](#)
  - isconnected, [223](#)
  - isscanning, [223](#)
  - m\_LidarType, [230](#)
  - m\_PointTime, [230](#)
  - m\_SingleChannel, [230](#)
  - m\_SupportMotorDtrCtrl, [231](#)
  - m\_baudrate, [229](#)
  - m\_config, [229](#)
  - m\_intensities, [229](#)
  - m\_isConnected, [229](#)
  - m\_isScanning, [229](#)
  - MAX\_SCAN\_NODES, [218](#)
  - package\_Sample\_Index, [232](#)
  - retryCount, [232](#)
  - scan\_node\_buf, [232](#)
  - scan\_node\_count, [232](#)
  - serial\_port, [232](#)
  - setAutoReconnect, [224](#)
  - setIntensities, [224](#)
  - setLidarType, [224](#)
  - setPointTime, [224](#)
  - setSamplingRate, [224](#)
  - setScanFrequencyAdd, [225](#)
  - setScanFrequencyAddMic, [226](#)
  - setScanFrequencyDis, [226](#)
  - setScanFrequencyDisMic, [227](#)
  - setSingleChannel, [227](#)
  - setSupportMotorDtrCtrl, [227](#)
  - startScan, [227](#)
  - stop, [228](#)
  - YDLIDAR\_F2, [217](#)
  - YDLIDAR\_F4, [217](#)
  - YDLIDAR\_F4PRO, [217](#)
  - YDLIDAR\_G1, [217](#)
  - YDLIDAR\_G10, [217](#)
  - YDLIDAR\_G2A, [217](#)
  - YDLIDAR\_G2B, [217](#)
  - YDLIDAR\_G2C, [217](#)
  - YDLIDAR\_G4, [217](#)
  - YDLIDAR\_G4PRO, [217](#)
  - YDLIDAR\_G4B, [217](#)
  - YDLIDAR\_G4C, [217](#)
  - YDLIDAR\_G6, [217](#)
  - YDLIDAR\_R2, [217](#)
  - YDLIDAR\_RATE\_10K, [218](#)
  - YDLIDAR\_RATE\_4K, [218](#)
  - YDLIDAR\_RATE\_8K, [218](#)
  - YDLIDAR\_RATE\_9K, [218](#)
  - YDLIDAR\_S2, [217](#)
  - YDLIDAR\_S4, [217](#)
  - YDLIDAR\_S4B, [217](#)
  - YDLIDAR\_T1, [217](#)
  - YDLIDAR\_T15, [217](#)
  - YDLIDAR\_TG15, [217](#)
  - YDLIDAR\_TG30, [217](#)
  - YDLIDAR\_TG50, [217](#)
  - YDLIDAR\_Tail, [217](#)
  - YDLIDAR\_X4, [217](#)
- ydlidar::core::math, [138](#)
  - find\_min\_max\_delta, [138](#)
  - from\_degrees, [139](#)
  - normalize\_angle, [139](#)

- normalize\_angle\_positive, 139
- normalize\_angle\_positive\_from\_degree, 139
- shortest\_angular\_distance, 139
- shortest\_angular\_distance\_with\_limits, 140
- to\_degrees, 140
- two\_pi\_complement, 140
- ydliar::core::network, 141
- ydliar::core::network::CActiveSocket, 150
  - ~CActiveSocket, 151
  - CActiveSocket, 151
  - CPassiveSocket, 152
  - Open, 152
- ydliar::core::network::CPassiveSocket, 163
  - ~CPassiveSocket, 164
  - Accept, 164
  - BindMulticast, 164
  - CPassiveSocket, 164
  - Listen, 165
  - Send, 165
- ydliar::core::network::CSimpleSocket, 166
  - ~CSimpleSocket, 172
  - available, 172
  - BindInterface, 172
  - bindport, 172
  - Both, 170
  - CShutdownMode, 170
  - CSimpleSocket, 171
  - CSocketError, 170
  - CSocketType, 171
  - Close, 172
  - closePort, 173
  - DescribeError, 173
  - DisableNagleAlgoritm, 173
  - EnableNagleAlgoritm, 173
  - Flush, 174
  - flush, 174
  - GetBytesReceived, 174
  - GetBytesSent, 174
  - GetClientAddr, 174
  - GetClientPort, 175
  - GetConnectTimeoutSec, 175
  - GetConnectTimeoutUsec, 175
  - GetData, 175
  - GetMulticast, 175
  - GetReceiveTimeoutSec, 176
  - GetReceiveTimeoutUsec, 176
  - GetReceiveWindowSize, 176
  - GetSendTimeoutSec, 176
  - GetSendTimeoutUsec, 177
  - GetSendWindowSize, 177
  - GetServerAddr, 177
  - GetServerPort, 177
  - GetSocketDescriptor, 178
  - GetSocketDscp, 178
  - GetSocketError, 178
  - GetSocketType, 178
  - GetTotalTimeMs, 179
  - GetTotalTimeUsec, 179
  - Initialize, 179
  - IsNonblocking, 179
  - isOpen, 179
  - IsSocketValid, 180
  - m\_addr, 188
  - m\_bIsBlocking, 188
  - m\_bIsMulticast, 188
  - m\_errorFds, 189
  - m\_nBufferSize, 189
  - m\_nBytesReceived, 189
  - m\_nBytesSent, 189
  - m\_nFlags, 189
  - m\_nSocketDomain, 189
  - m\_nSocketType, 189
  - m\_open, 189
  - m\_pBuffer, 190
  - m\_port, 190
  - m\_readFds, 190
  - m\_socket, 190
  - m\_socketErrno, 190
  - m\_stClientSockaddr, 190
  - m\_stConnectTimeout, 190
  - m\_stLinger, 190
  - m\_stMulticastGroup, 191
  - m\_stRecvTimeout, 191
  - m\_stSendTimeout, 191
  - m\_stServerSockaddr, 191
  - m\_timer, 191
  - m\_writeFds, 191
  - Open, 180
  - open, 180
  - readData, 180
  - readSize, 181
  - Receive, 181
  - Receives, 170
  - Select, 181, 182
  - Send, 182
  - SendFile, 183
  - Sends, 170
  - SetBlocking, 183
  - SetConnectTimeout, 183
  - SetMulticast, 184
  - SetNonblocking, 184
  - SetOptionLinger, 184
  - SetOptionReuseAddr, 185
  - SetReceiveTimeout, 185
  - SetReceiveWindowSize, 185
  - SetSendTimeout, 185
  - SetSendWindowSize, 186
  - SetSocketDscp, 186
  - SetSocketError, 186
  - SetSocketHandle, 187
  - SetSocketType, 187
  - Shutdown, 187
  - SocketAddressInUse, 171
  - SocketConnectionAborted, 171
  - SocketConnectionRefused, 171
  - SocketConnectionReset, 171



- SocketEinprogress, [171](#)
- SocketError, [171](#)
- SocketEunknown, [171](#)
- SocketEwouldblock, [171](#)
- SocketFirewallError, [171](#)
- SocketInterrupted, [171](#)
- SocketInvalidAddress, [171](#)
- SocketInvalidPointer, [171](#)
- SocketInvalidPort, [171](#)
- SocketInvalidSocket, [171](#)
- SocketInvalidSocketBuffer, [171](#)
- SocketNotconnected, [171](#)
- SocketProtocolError, [171](#)
- SocketSuccess, [171](#)
- SocketTimeout, [171](#)
- SocketTypeInvalid, [171](#)
- SocketTypeRaw, [171](#)
- SocketTypeTcp, [171](#)
- SocketTypeTcp6, [171](#)
- SocketTypeUdp, [171](#)
- SocketTypeUdp6, [171](#)
- TranslateSocketError, [187](#)
- WSACleanUp, [188](#)
- WaitForData, [187](#)
- waitfordata, [187](#)
- writeData, [188](#)
- ydliar::core::serial, [141](#)
  - bytesize\_t, [142](#)
  - eightbits, [142](#)
  - fivebits, [142](#)
  - flowcontrol\_hardware, [142](#)
  - flowcontrol\_none, [142](#)
  - flowcontrol\_software, [142](#)
  - flowcontrol\_t, [142](#)
  - is\_standardbaudrate, [143](#)
  - list\_ports, [143](#)
  - parity\_even, [142](#)
  - parity\_mark, [142](#)
  - parity\_none, [142](#)
  - parity\_odd, [142](#)
  - parity\_space, [142](#)
  - parity\_t, [142](#)
  - set\_common\_props, [143](#)
  - set\_databits, [143](#)
  - set\_flowcontrol, [143](#)
  - set\_parity, [143](#)
  - set\_stopbits, [143](#)
  - sevenbits, [142](#)
  - sixbits, [142](#)
  - stopbits\_one, [142](#)
  - stopbits\_one\_point\_five, [142](#)
  - stopbits\_t, [142](#)
  - stopbits\_two, [142](#)
  - timespec\_from\_ms, [143](#)
- ydliar::core::serial::MillisecondTimer, [264](#)
  - MillisecondTimer, [265](#)
  - remaining, [265](#)
- ydliar::core::serial::PortInfo, [272](#)
  - description, [273](#)
  - device\_id, [273](#)
  - hardware\_id, [273](#)
  - port, [273](#)
- ydliar::core::serial::Serial, [281](#)
  - ~Serial, [285](#)
  - available, [285](#)
  - BreakConditionError, [284](#)
  - closePort, [285](#)
  - DescribeError, [285](#)
  - DeviceNotFoundError, [284](#)
  - flush, [286](#)
  - flushInput, [286](#)
  - flushOutput, [286](#)
  - FramingError, [284](#)
  - getBaudrate, [286](#)
  - getByteTime, [287](#)
  - getBytesize, [286](#)
  - getCTS, [287](#)
  - getCD, [287](#)
  - getDSR, [287](#)
  - getFlowcontrol, [287](#)
  - getParity, [287](#)
  - getPort, [288](#)
  - getRI, [288](#)
  - getStopbits, [288](#)
  - getSystemError, [288](#)
  - getTimeout, [289](#)
  - isOpen, [289](#)
  - NoError, [284](#)
  - NotOpenError, [284](#)
  - open, [289](#)
  - OpenError, [284](#)
  - ParityError, [284](#)
  - PermissionError, [284](#)
  - read, [289](#), [290](#)
  - readData, [291](#)
  - ReadError, [284](#)
  - readSize, [292](#)
  - readline, [291](#), [292](#)
  - readlines, [292](#)
  - ResourceError, [284](#)
  - sendBreak, [293](#)
  - Serial, [284](#)
  - SerialPortError, [284](#)
  - setBaudrate, [293](#)
  - setBreak, [293](#)
  - setBytesize, [293](#)
  - setDTR, [294](#)
  - setFlowcontrol, [294](#)
  - setParity, [294](#)
  - setPort, [294](#)
  - setRTS, [295](#)
  - setStopbits, [295](#)
  - setTimeout, [295](#), [296](#)
  - TimeoutError, [284](#)
  - UnknownError, [284](#)
  - UnsupportedOperationError, [284](#)



- waitByteTimes, [296](#)
- waitForChange, [296](#)
- waitReadable, [297](#)
- waitfordata, [296](#)
- write, [297](#), [298](#)
- writeData, [298](#)
- WriteError, [284](#)
- ydliidar::core::serial::Serial::ScopedReadLock
  - ~ScopedReadLock, [280](#)
  - ScopedReadLock, [280](#)
- ydliidar::core::serial::Serial::ScopedWriteLock
  - ~ScopedWriteLock, [281](#)
  - ScopedWriteLock, [281](#)
- ydliidar::core::serial::Serial::SerialImpl
  - ~SerialImpl, [300](#)
  - available, [300](#)
  - close, [300](#)
  - flush, [300](#)
  - flushInput, [300](#)
  - flushOutput, [300](#)
  - getBaudrate, [300](#)
  - getByteTime, [300](#)
  - getBytesize, [300](#)
  - getCTS, [301](#)
  - getCD, [301](#)
  - getDSR, [301](#)
  - getFlowcontrol, [301](#)
  - getParity, [301](#)
  - getPort, [301](#)
  - getRI, [301](#)
  - getStopbits, [301](#)
  - getSystemError, [301](#)
  - getTermios, [301](#)
  - getTimeout, [302](#)
  - isOpen, [302](#)
  - open, [302](#)
  - read, [302](#)
  - readLock, [302](#)
  - readUnlock, [302](#)
  - sendBreak, [302](#)
  - SerialImpl, [300](#)
  - setBaudrate, [302](#)
  - setBreak, [302](#)
  - setBytesize, [302](#)
  - setCustomBaudRate, [303](#)
  - setDTR, [303](#)
  - setFlowcontrol, [303](#)
  - setParity, [303](#)
  - setPort, [303](#)
  - setRTS, [303](#)
  - setStandardBaudRate, [303](#)
  - setStopbits, [303](#)
  - setTermios, [303](#)
  - setTimeout, [303](#)
  - waitByteTimes, [304](#)
  - waitForChange, [304](#)
  - waitReadable, [304](#)
  - waitfordata, [304](#)
  - write, [304](#)
  - writeLock, [304](#)
  - writeUnlock, [304](#)
- ydliidar::core::serial::Timeout, [309](#)
  - inter\_byte\_timeout, [310](#)
  - max, [310](#)
  - read\_timeout\_constant, [310](#)
  - read\_timeout\_multiplier, [310](#)
  - simpleTimeout, [310](#)
  - Timeout, [310](#)
  - write\_timeout\_constant, [310](#)
  - write\_timeout\_multiplier, [311](#)
- ydliidar::core::serial::termios2, [305](#)
  - c\_cc, [306](#)
  - c\_cflag, [306](#)
  - c\_iflag, [306](#)
  - c\_ispeed, [306](#)
  - c\_lflag, [306](#)
  - c\_line, [306](#)
  - c\_oflag, [306](#)
  - c\_ospeed, [306](#)
- ydliidar\_def.cpp
  - LaserFanDestroy, [356](#)
  - LaserFanInit, [356](#)
- ydliidar\_def.h
  - DeviceTypeID, [358](#)
  - LaserFanDestroy, [359](#)
  - LaserFanInit, [359](#)
  - LidarPropAbnormalCheckCount, [359](#)
  - LidarPropAutoReconnect, [359](#)
  - LidarPropDeviceType, [359](#)
  - LidarPropFixedResolution, [359](#)
  - LidarPropIgnoreArray, [359](#)
  - LidarPropIntenstiy, [359](#)
  - LidarPropInverted, [359](#)
  - LidarPropLidarType, [359](#)
  - LidarPropMaxAngle, [359](#)
  - LidarPropMaxRange, [359](#)
  - LidarPropMinAngle, [359](#)
  - LidarPropMinRange, [359](#)
  - LidarPropReversion, [359](#)
  - LidarPropSampleRate, [359](#)
  - LidarPropScanFrequency, [359](#)
  - LidarPropSerialBaudrate, [359](#)
  - LidarPropSerialPort, [359](#)
  - LidarPropSingleChannel, [359](#)
  - LidarPropSupportMotorDtrCtrl, [359](#)
  - LidarProperty, [358](#)
  - LidarTypeID, [359](#)
  - TYPE\_TOF\_NET, [359](#)
  - TYPE\_TOF, [359](#)
  - TYPE\_TRIANGLE, [359](#)
  - TYPE\_Tail, [359](#)
  - YDLIDAR\_TYPC\_UDP, [358](#)
  - YDLIDAR\_TYPE\_SERIAL, [358](#)
  - YDLIDAR\_TYPE\_TCP, [358](#)
- ydliidar\_protocol.h
  - \_\_attribute\_\_, [372](#)

- [\\_countof, 366](#)
- [angle, 372](#)
- [angle\\_q6\\_checkbit, 372](#)
- [CT\\_Normal, 372](#)
- [CT\\_RingStart, 372](#)
- [CT\\_Tail, 372](#)
- [checkSum, 373](#)
- [cmd\\_flag, 373](#)
- [CT, 372](#)
- [data, 373](#)
- [dataFrame, 372](#)
- [debugInfo, 373](#)
- [distance\\_q2, 373](#)
- [enable, 373](#)
- [error\\_code, 373](#)
- [error\\_package, 373](#)
- [exposure, 374](#)
- [firmware\\_version, 374](#)
- [flag, 374](#)
- [frequency, 374](#)
- [GLASSNOISEINTENSITY, 366](#)
- [hardware\\_version, 374](#)
- [index, 374](#)
- [LIDAR\\_ANS\\_SYNC\\_BYTE1, 366](#)
- [LIDAR\\_ANS\\_SYNC\\_BYTE2, 366](#)
- [LIDAR\\_ANS\\_TYPE\\_DEVHEALTH, 366](#)
- [LIDAR\\_ANS\\_TYPE\\_DEVINFO, 367](#)
- [LIDAR\\_ANS\\_TYPE\\_MEASUREMENT, 367](#)
- [LIDAR\\_CMD\\_ADD\\_EXPOSURE, 367](#)
- [LIDAR\\_CMD\\_DIS\\_EXPOSURE, 367](#)
- [LIDAR\\_CMD\\_DISABLE\\_CONST\\_FREQ, 367](#)
- [LIDAR\\_CMD\\_DISABLE\\_LOW\\_POWER, 367](#)
- [LIDAR\\_CMD\\_ENABLE\\_CONST\\_FREQ, 367](#)
- [LIDAR\\_CMD\\_ENABLE\\_LOW\\_POWER, 367](#)
- [LIDAR\\_CMD\\_FORCE\\_SCAN, 367](#)
- [LIDAR\\_CMD\\_FORCE\\_STOP, 367](#)
- [LIDAR\\_CMD\\_GET\\_AIMSPEED, 368](#)
- [LIDAR\\_CMD\\_GET\\_DEVICE\\_HEALTH, 368](#)
- [LIDAR\\_CMD\\_GET\\_DEVICE\\_INFO, 368](#)
- [LIDAR\\_CMD\\_GET\\_EAI, 368](#)
- [LIDAR\\_CMD\\_GET\\_OFFSET\\_ANGLE, 368](#)
- [LIDAR\\_CMD\\_GET\\_SAMPLING\\_RATE, 368](#)
- [LIDAR\\_CMD\\_RESET, 368](#)
- [LIDAR\\_CMD\\_RUN\\_INVERSION, 368](#)
- [LIDAR\\_CMD\\_RUN\\_POSITIVE, 368](#)
- [LIDAR\\_CMD\\_SAVE\\_SET\\_EXPOSURE, 368](#)
- [LIDAR\\_CMD\\_SCAN, 369](#)
- [LIDAR\\_CMD\\_SET\\_AIMSPEED\\_ADDMIC, 369](#)
- [LIDAR\\_CMD\\_SET\\_AIMSPEED\\_ADD, 369](#)
- [LIDAR\\_CMD\\_SET\\_AIMSPEED\\_DISMIC, 369](#)
- [LIDAR\\_CMD\\_SET\\_AIMSPEED\\_DIS, 369](#)
- [LIDAR\\_CMD\\_SET\\_LOW\\_EXPOSURE, 369](#)
- [LIDAR\\_CMD\\_SET\\_SAMPLING\\_RATE, 369](#)
- [LIDAR\\_CMD\\_STATE\\_MODEL\\_MOTOR, 369](#)
- [LIDAR\\_CMD\\_STOP, 369](#)
- [LIDAR\\_CMD\\_SYNC\\_BYTE, 369](#)
- [LIDAR\\_CMDFLAG\\_HAS\\_PAYLOAD, 370](#)
- [LIDAR\\_RESP\\_MEASUREMENT\\_ANGLE\\_SAMPLE\\_SHIFT, 370](#)
- [LIDAR\\_RESP\\_MEASUREMENT\\_ANGLE\\_SHIFT, 370](#)
- [LIDAR\\_RESP\\_MEASUREMENT\\_CHECKBIT, 370](#)
- [LIDAR\\_RESP\\_MEASUREMENT\\_DISTANCE\\_SHIFT, 370](#)
- [LIDAR\\_RESP\\_MEASUREMENT\\_QUALITY\\_SHIFT, 370](#)
- [LIDAR\\_RESP\\_MEASUREMENT\\_SYNCBIT, 370](#)
- [LIDAR\\_STATUS\\_ERROR, 370](#)
- [LIDAR\\_STATUS\\_OK, 370](#)
- [LIDAR\\_STATUS\\_WARNING, 370](#)
- [lidarConfig, 372](#)
- [M\\_PI, 371](#)
- [model, 374](#)
- [Node\\_Default\\_Quality, 371](#)
- [Node\\_NotSync, 371](#)
- [Node\\_Sync, 371](#)
- [nowPackageNum, 374](#)
- [package\\_CT, 375](#)
- [package\\_Head, 375](#)
- [packageFirstSampleAngle, 375](#)
- [packageLastSampleAngle, 375](#)
- [PackagePaidBytes, 371](#)
- [packageSample, 375](#)
- [packageSampleDistance, 375](#)
- [PackageSampleMaxLngth, 371](#)
- [PackageSampleDistance, 375](#)
- [PackageSampleQuality, 375](#)
- [PH, 371](#)
- [Protocol\\_V1, 372](#)
- [Protocol\\_V2, 372](#)
- [ProtocolVer, 372](#)
- [rate, 376](#)
- [rotation, 376](#)
- [SUNNOISEINTENSITY, 371](#)
- [scan\\_frequency, 376](#)
- [serialnum, 376](#)
- [size, 376](#)
- [stamp, 376](#)
- [state, 376](#)
- [status, 376](#)
- [subType, 377](#)
- [sync\\_flag, 377](#)
- [sync\\_quality, 377](#)
- [syncByte, 377](#)
- [syncByte1, 377](#)
- [syncByte2, 377](#)
- [type, 377](#)
- [ydlidar\\_sdk.cpp](#)
  - [DescribeError, 414](#)
  - [disconnecting, 414](#)
  - [doProcessSimple, 414](#)
  - [GetLidarVersion, 416](#)
  - [GetSdkVersion, 416](#)
  - [getlidaropt, 414](#)
  - [initialize, 416](#)

- lidarCreate, [417](#)
- lidarDestroy, [417](#)
- lidarPortList, [417](#)
- os\_init, [418](#)
- os\_isOk, [418](#)
- os\_shutdown, [418](#)
- setlidaropt, [418](#)
- turnOff, [420](#)
- turnOn, [420](#)
- ydliar\_sdk.h
  - DescribeError, [422](#)
  - disconnecting, [422](#)
  - doProcessSimple, [422](#)
  - GetLidarVersion, [424](#)
  - GetSdkVersion, [425](#)
  - getlidaropt, [423](#)
  - initialize, [425](#)
  - lidarCreate, [425](#)
  - lidarDestroy, [425](#)
  - lidarPortList, [426](#)
  - os\_init, [426](#)
  - os\_isOk, [426](#)
  - os\_shutdown, [426](#)
  - setlidaropt, [426](#)
  - turnOff, [428](#)
  - turnOn, [428](#)
- ydliar\_test, [143](#)
  - laser, [144](#)
  - port, [144](#)
  - ports, [144](#)
  - r, [144](#)
  - ret, [144](#)
  - scan, [144](#)
- ydliar\_test.cpp
  - main, [406](#)
- zip\_safe
  - setup::CMakeBuild, [160](#)