

# DeltaRobot

## v0.4

Generated by Doxygen 1.8.9.1

Tue Apr 21 2015 21:16:00



# Contents

<b>1</b>	<b>Main Page</b>	<b>1</b>
<b>2</b>	<b>Namespace Documentation</b>	<b>3</b>
2.1	Ui Namespace Reference . . . . .	3
2.1.1	Detailed Description . . . . .	3
<b>3</b>	<b>Class Documentation</b>	<b>5</b>
3.1	AX12 Class Reference . . . . .	5
3.1.1	Detailed Description . . . . .	6
3.1.2	Member Enumeration Documentation . . . . .	7
3.1.2.1	RAM . . . . .	7
3.1.2.2	ROM . . . . .	7
3.1.3	Constructor & Destructor Documentation . . . . .	8
3.1.3.1	AX12 . . . . .	8
3.1.3.2	AX12 . . . . .	8
3.1.3.3	~AX12 . . . . .	8
3.1.4	Member Function Documentation . . . . .	9
3.1.4.1	connectedID . . . . .	9
3.1.4.2	getCurrentLoad . . . . .	9
3.1.4.3	getCurrentPos . . . . .	9
3.1.4.4	getCurrentSpeed . . . . .	9
3.1.4.5	getCurrentTemp . . . . .	9
3.1.4.6	getCurrentVoltage . . . . .	10
3.1.4.7	getID . . . . .	10
3.1.4.8	setGoalPosition . . . . .	10
3.1.4.9	setID . . . . .	10
3.1.4.10	setJointMode . . . . .	10
3.1.4.11	setMinMax . . . . .	11
3.1.4.12	setSpeed . . . . .	11
3.1.5	Member Data Documentation . . . . .	11
3.1.5.1	_ID . . . . .	11
3.1.5.2	_mode . . . . .	11

3.1.5.3	<a href="#">_rads</a>	11
3.1.5.4	<a href="#">dxl</a>	11
3.2	<a href="#">dxl_hal Class Reference</a>	12
3.2.1	<a href="#">Detailed Description</a>	12
3.2.2	<a href="#">Member Function Documentation</a>	12
3.2.2.1	<a href="#">change_baudrate</a>	12
3.2.2.2	<a href="#">clear</a>	12
3.2.2.3	<a href="#">close</a>	12
3.2.2.4	<a href="#">get_curr_time</a>	13
3.2.2.5	<a href="#">isOpen</a>	13
3.2.2.6	<a href="#">open</a>	13
3.2.2.7	<a href="#">read</a>	13
3.2.2.8	<a href="#">write</a>	13
3.2.3	<a href="#">Member Data Documentation</a>	13
3.2.3.1	<a href="#">_open</a>	14
3.2.3.2	<a href="#">_serial</a>	14
3.2.3.3	<a href="#">_time</a>	14
3.2.3.4	<a href="#">_timed</a>	14
3.3	<a href="#">dynamixel Class Reference</a>	14
3.3.1	<a href="#">Detailed Description</a>	16
3.3.2	<a href="#">Constructor &amp; Destructor Documentation</a>	16
3.3.2.1	<a href="#">dynamixel</a>	16
3.3.2.2	<a href="#">dynamixel</a>	16
3.3.3	<a href="#">Member Function Documentation</a>	16
3.3.3.1	<a href="#">change_baudrate</a>	16
3.3.3.2	<a href="#">get_comm_result</a>	17
3.3.3.3	<a href="#">get_packet_time</a>	17
3.3.3.4	<a href="#">get_rxpacket_error</a>	17
3.3.3.5	<a href="#">get_rxpacket_error_byte</a>	17
3.3.3.6	<a href="#">get_rxpacket_length</a>	17
3.3.3.7	<a href="#">get_rxpacket_parameter</a>	17
3.3.3.8	<a href="#">initialize</a>	18
3.3.3.9	<a href="#">is_packet_timeout</a>	18
3.3.3.10	<a href="#">isOpen</a>	18
3.3.3.11	<a href="#">ping</a>	18
3.3.3.12	<a href="#">read_byte</a>	18
3.3.3.13	<a href="#">read_word</a>	19
3.3.3.14	<a href="#">rx_packet</a>	19
3.3.3.15	<a href="#">set_packet_timeout</a>	20
3.3.3.16	<a href="#">set_packet_timeout_ms</a>	21

3.3.3.17	<a href="#">set_txpacket_id</a>	21
3.3.3.18	<a href="#">set_txpacket_instruction</a>	21
3.3.3.19	<a href="#">set_txpacket_length</a>	21
3.3.3.20	<a href="#">set_txpacket_parameter</a>	21
3.3.3.21	<a href="#">terminate</a>	22
3.3.3.22	<a href="#">tx_packet</a>	22
3.3.3.23	<a href="#">txrx_packet</a>	23
3.3.3.24	<a href="#">write_byte</a>	23
3.3.3.25	<a href="#">write_word</a>	23
3.3.4	<a href="#">Member Data Documentation</a>	24
3.3.4.1	<a href="#">dH</a>	24
3.3.4.2	<a href="#">gbCommStatus</a>	24
3.3.4.3	<a href="#">gbInstructionPacket</a>	24
3.3.4.4	<a href="#">gbRxGetLength</a>	24
3.3.4.5	<a href="#">gbRxPacketLength</a>	24
3.3.4.6	<a href="#">gbStatusPacket</a>	24
3.3.4.7	<a href="#">gdByteTransTime</a>	24
3.3.4.8	<a href="#">gdPacketStartTime</a>	24
3.3.4.9	<a href="#">gdRcvWaitTime</a>	24
3.3.4.10	<a href="#">giBusUsing</a>	24
3.4	<a href="#">MainWindow Class Reference</a>	25
3.4.1	<a href="#">Detailed Description</a>	26
3.4.2	<a href="#">Constructor &amp; Destructor Documentation</a>	27
3.4.2.1	<a href="#">MainWindow</a>	27
3.4.2.2	<a href="#">~MainWindow</a>	27
3.4.3	<a href="#">Member Function Documentation</a>	27
3.4.3.1	<a href="#">joyChanged</a>	27
3.4.3.2	<a href="#">joystickChanged</a>	28
3.4.3.3	<a href="#">on_actionOptions_triggered</a>	28
3.4.3.4	<a href="#">on_start_clicked</a>	28
3.4.3.5	<a href="#">update</a>	28
3.4.4	<a href="#">Member Data Documentation</a>	29
3.4.4.1	<a href="#">_axis</a>	29
3.4.4.2	<a href="#">_axisV</a>	29
3.4.4.3	<a href="#">_buts</a>	29
3.4.4.4	<a href="#">_butsV</a>	29
3.4.4.5	<a href="#">_dataP</a>	29
3.4.4.6	<a href="#">_jAxisX</a>	29
3.4.4.7	<a href="#">_jAxisY</a>	29
3.4.4.8	<a href="#">_jAxisZ</a>	29

3.4.4.9	<a href="#">_joy</a>	29
3.4.4.10	<a href="#">_sT</a>	29
3.4.4.11	<a href="#">_timer</a>	30
3.4.4.12	<a href="#">aSCount</a>	30
3.4.4.13	<a href="#">sCount</a>	30
3.4.4.14	<a href="#">ui</a>	30
3.5	<a href="#">OptionsWindow Class Reference</a>	30
3.5.1	<a href="#">Detailed Description</a>	32
3.5.2	<a href="#">Constructor &amp; Destructor Documentation</a>	32
3.5.2.1	<a href="#">OptionsWindow</a>	32
3.5.2.2	<a href="#">~OptionsWindow</a>	32
3.5.3	<a href="#">Member Function Documentation</a>	32
3.5.3.1	<a href="#">events</a>	32
3.5.3.2	<a href="#">joystickChanged</a>	33
3.5.3.3	<a href="#">on_servoRefresh_clicked</a>	33
3.5.3.4	<a href="#">storeData</a>	33
3.5.4	<a href="#">Member Data Documentation</a>	34
3.5.4.1	<a href="#">_joy</a>	34
3.5.4.2	<a href="#">_portSize</a>	34
3.5.4.3	<a href="#">_servo</a>	34
3.5.4.4	<a href="#">_timer</a>	34
3.5.4.5	<a href="#">ui</a>	34
3.6	<a href="#">ServoThread::Servo Struct Reference</a>	34
3.6.1	<a href="#">Detailed Description</a>	35
3.6.2	<a href="#">Constructor &amp; Destructor Documentation</a>	35
3.6.2.1	<a href="#">Servo</a>	35
3.6.2.2	<a href="#">Servo</a>	35
3.6.3	<a href="#">Member Data Documentation</a>	35
3.6.3.1	<a href="#">ID</a>	35
3.6.3.2	<a href="#">load</a>	35
3.6.3.3	<a href="#">pos</a>	35
3.7	<a href="#">ServoThread Class Reference</a>	35
3.7.1	<a href="#">Detailed Description</a>	38
3.7.2	<a href="#">Member Enumeration Documentation</a>	39
3.7.2.1	<a href="#">Mode</a>	39
3.7.2.2	<a href="#">Version</a>	39
3.7.3	<a href="#">Constructor &amp; Destructor Documentation</a>	39
3.7.3.1	<a href="#">ServoThread</a>	39
3.7.3.2	<a href="#">~ServoThread</a>	39
3.7.4	<a href="#">Member Function Documentation</a>	40

3.7.4.1	end	40
3.7.4.2	getServoBaud	40
3.7.4.3	getServoPort	40
3.7.4.4	getServoPortInfo	40
3.7.4.5	getServosInfo	40
3.7.4.6	getServosInfo	41
3.7.4.7	load	41
3.7.4.8	mutex	41
3.7.4.9	pause	41
3.7.4.10	run	41
3.7.4.11	setAngles	42
3.7.4.12	setData	42
3.7.4.13	setServoBaud	42
3.7.4.14	setServoPort	43
3.7.4.15	setServoPortInfo	43
3.7.4.16	setSID	43
3.7.4.17	setStatusBar	43
3.7.4.18	singleAngle	44
3.7.4.19	wakeUp	44
3.7.4.20	write	44
3.7.5	Member Data Documentation	44
3.7.5.1	_axis	44
3.7.5.2	_buts	45
3.7.5.3	_cBaud	45
3.7.5.4	_cond	45
3.7.5.5	_cPort	45
3.7.5.6	_dChanged	45
3.7.5.7	_end	45
3.7.5.8	_mod	45
3.7.5.9	_mutex	45
3.7.5.10	_pause	45
3.7.5.11	_sBaud	45
3.7.5.12	_servos	45
3.7.5.13	_sPort	45
3.7.5.14	_sPortChanged	46
3.7.5.15	_statusBar	46
3.7.5.16	a	46
3.7.5.17	b	46
3.7.5.18	cos60	46
3.7.5.19	L1	46

3.7.5.20	L2	46
3.7.5.21	sin60	46
<b>4</b>	<b>File Documentation</b>	<b>47</b>
4.1	dxl/ax12.cpp File Reference	47
4.1.1	Detailed Description	47
4.2	dxl/ax12.h File Reference	48
4.2.1	Detailed Description	49
4.2.2	Macro Definition Documentation	49
4.2.2.1	M_PI	49
4.3	dxl/dxl_hal.cpp File Reference	49
4.3.1	Detailed Description	49
4.4	dxl/dxl_hal.h File Reference	49
4.4.1	Detailed Description	51
4.4.2	Macro Definition Documentation	51
4.4.2.1	MAXNUM_RXPACKET	51
4.4.2.2	MAXNUM_TXPACKET	51
4.5	dxl/dynamixel.cpp File Reference	51
4.5.1	Detailed Description	52
4.5.2	Macro Definition Documentation	52
4.5.2.1	LATENCY_TIME	52
4.5.2.2	PING_STATUS_LENGTH	52
4.6	dxl/dynamixel.h File Reference	52
4.6.1	Detailed Description	54
4.6.2	Macro Definition Documentation	54
4.6.2.1	BROADCAST_ID	54
4.6.2.2	COMM_RXCORRUPT	54
4.6.2.3	COMM_RXFAIL	54
4.6.2.4	COMM_RXSUCCESS	54
4.6.2.5	COMM_RXTIMEOUT	54
4.6.2.6	COMM_RXWAITING	54
4.6.2.7	COMM_TXERROR	54
4.6.2.8	COMM_TXFAIL	54
4.6.2.9	COMM_TXSUCCESS	54
4.6.2.10	ERR_ACCESS	54
4.6.2.11	ERR_CRC	55
4.6.2.12	ERR_DATA_LENGTH	55
4.6.2.13	ERR_DATA_LIMIT	55
4.6.2.14	ERR_DATA_RANGE	55
4.6.2.15	ERR_INSTRUCTION	55



4.6.2.16	ERR_RESULT_FAIL	55
4.6.2.17	ERRBIT_ALERT	55
4.6.2.18	HIBYTE	55
4.6.2.19	HIWORD	55
4.6.2.20	INST_ACTION	55
4.6.2.21	INST_BULK_READ	55
4.6.2.22	INST_PING	55
4.6.2.23	INST_READ	55
4.6.2.24	INST_REG_WRITE	55
4.6.2.25	INST_RESET	55
4.6.2.26	INST_SYNC_WRITE	55
4.6.2.27	INST_WRITE	55
4.6.2.28	LOBYTE	55
4.6.2.29	LOWORD	55
4.6.2.30	MAKEDWORD	55
4.6.2.31	MAKEWORD	55
4.6.2.32	MAX_ID	55
4.6.2.33	PING_INFO_FIRM_VER	55
4.6.2.34	PING_INFO_MODEL_NUM	55
4.6.2.35	PRT1_PKT_ERRBIT	55
4.6.2.36	PRT1_PKT_ID	55
4.6.2.37	PRT1_PKT_INSTRUCTION	55
4.6.2.38	PRT1_PKT_LENGTH	56
4.6.2.39	PRT1_PKT_PARAMETER0	56
4.7	main.cpp File Reference	56
4.7.1	Detailed Description	56
4.7.2	Function Documentation	56
4.7.2.1	main	56
4.8	mainwindow.cpp File Reference	56
4.8.1	Detailed Description	57
4.9	mainwindow.h File Reference	57
4.9.1	Detailed Description	58
4.10	optionswindow.cpp File Reference	58
4.10.1	Detailed Description	59
4.11	optionswindow.h File Reference	59
4.11.1	Detailed Description	60
4.12	servothread.cpp File Reference	60
4.13	servothread.h File Reference	60
4.13.1	Detailed Description	61

<a href="#">Index</a>	63
-----------------------	----

## Chapter 1

# Main Page

This project is a Delta robot controller using Dynamixel [AX12](#) servos. This type of robot can pick and place objects



## Chapter 2

# Namespace Documentation

### 2.1 Ui Namespace Reference

Namespace to work with a User Interface Qt Form.

#### 2.1.1 Detailed Description

Namespace to work with a User Interface Qt Form.



## Chapter 3

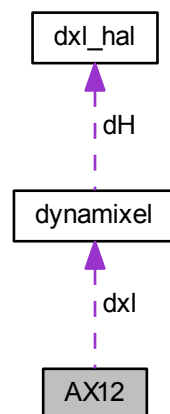
# Class Documentation

### 3.1 AX12 Class Reference

The [AX12](#) class is used to control AX-12 motors from Dynamixel.

```
#include <ax12.h>
```

Collaboration diagram for AX12:



#### Public Member Functions

- [AX12](#) ([dynamixel](#) \*dxl, int ID=-1)  
*Default constructor must pass an initialized dynamixel object if ID == -1 no action is done.*
- [AX12](#) (const [AX12](#) &a)  
*Copy constructor.*
- [~AX12](#) ()  
*Default destructor.*
- QVector< int > [connectedID](#) ()  
*Returns all active servos;.*
- double [getCurrentLoad](#) ()

- Returns the current load from -100% to 100%, 100% is ClockWise and -100% is CounterClockWise.*

  - double [getCurrentPos](#) ()

*Returns the current position from 0° to 300°*
- int [getCurrentTemp](#) ()

*Returns the current Temperature in Celsius.*
- double [getCurrentSpeed](#) ()

*Returns the current speed from -100% to 100%, 100% is ClockWise and -100% is CounterClockWise.*
- double [getCurrentVoltage](#) ()

*Returns the current voltage in Volts.*
- int [getID](#) ()

*To get the current ID.*
- void [setGoalPosition](#) (double goal)

*Sets the Goal's position (in degrees) or speed depending on the mode.*
- void [setID](#) (int ID)

*To set a new ID.*
- void [setJointMode](#) (bool mode)

*To set Joint/Wheel mode, true if Joint.*
- void [setMinMax](#) (double min, double max)

*To set the minimum and maximum angle from 0 to 300°*
- void [setSpeed](#) (double speed)

*To set the maximum speed from 0% to 100% if joint mode or from -100% to 100% if wheel mode.*

## Private Types

- enum [ROM](#) {
  - [ModelNumber](#) = 0, [VersionFirmware](#) = 2, [ID](#) = 3, [BaudRate](#) = 4,
  - [ReturnDelayTime](#) = 5, [CWAngleLimit](#) = 6, [CCWAngleLimit](#) = 8, [HighestLimitTemp](#) = 11,
  - [LowestLimitVoltage](#) = 12, [HighestLimitVoltage](#) = 13, [MaxTorque](#) = 14, [StatusReturnLevel](#) = 16,
  - [AlarmLED](#) = 17, [AlarmShutdown](#) = 18 }

*Contains all the EEPROM directions enumeration.*
- enum [RAM](#) {
  - [TorqueEnable](#) = 24, [LED](#) = 25, [CWComplianceMargin](#) = 26, [CCWComplianceMargin](#) = 27,
  - [CWComplianceSlope](#) = 28, [CCWComplianceSlope](#) = 29, [GoalPosition](#) = 30, [MovingSpeed](#) = 32,
  - [TorqueLimit](#) = 34, [PresentPosition](#) = 36, [PresentSpeed](#) = 38, [PresentLoad](#) = 40,
  - [PresentVoltage](#) = 42, [PresentTemperature](#) = 43, [Registered](#) = 44, [Moving](#) = 46,
  - [Lock](#) = 47, [Punch](#) = 48 }

*Contains all the RAM directions enumerations.*

## Private Attributes

- [dynamixel](#) \* [dxl](#)

*Contains the dynamixel communication.*
- int [\\_ID](#)

*Stores the current ID.*
- bool [\\_mode](#)

*True if we use the joint mode.*
- bool [\\_rads](#)

*True if the angle is returned in radians.*

### 3.1.1 Detailed Description

The [AX12](#) class is used to control AX-12 motors from Dynamixel.



### 3.1.2 Member Enumeration Documentation

#### 3.1.2.1 enum AX12::RAM [private]

Contains all the RAM directions enumerations.

Enumerator

***TorqueEnable***  
***LED***  
***CWComplianceMargin***  
***CCWComplianceMargin***  
***CWComplianceSlope***  
***CCWComplianceSlope***  
***GoalPosition***  
***MovingSpeed***  
***TorqueLimit***  
***PresentPosition***  
***PresentSpeed***  
***PresentLoad***  
***PresentVoltage***  
***PresentTemperature***  
***Registered***  
***Moving***  
***Lock***  
***Punch***

```

00044    {
00045        TorqueEnable    = 24,
00046        LED            = 25,
00047        CWComplianceMargin = 26,
00048        CCWComplianceMargin = 27,
00049        CWComplianceSlope = 28,
00050        CCWComplianceSlope = 29,
00051        GoalPosition    = 30,
00052        MovingSpeed     = 32,
00053        TorqueLimit     = 34,
00054        PresentPosition  = 36,
00055        PresentSpeed    = 38,
00056        PresentLoad    = 40,
00057        PresentVoltage  = 42,
00058        PresentTemperature = 43,
00059        Registered     = 44,
00060        Moving         = 46,
00061        Lock          = 47,
00062        Punch         = 48
00063    };
00064

```

#### 3.1.2.2 enum AX12::ROM [private]

Contains all the EEPROM directions enumeration.

Enumerator

***ModelNumber***  
***VersionFirmware***  
***ID***

**BaudRate**  
**ReturnDelayTime**  
**CWAngleLimit**  
**CCWAngleLimit**  
**HighestLimitTemp**  
**LowestLimitVoltage**  
**HighestLimitVoltage**  
**MaxTorque**  
**StatusReturnLevel**  
**AlarmLED**  
**AlarmShutdown**

```

00025     {
00026         ModelNumber      = 0,
00027         VersionFirmware  = 2,
00028         ID                = 3,
00029         BaudRate          = 4,
00030         ReturnDelayTime  = 5,
00031         CWAngleLimit      = 6,
00032         CCWAngleLimit     = 8,
00033         HighestLimitTemp  = 11,
00034         LowestLimitVoltage = 12,
00035         HighestLimitVoltage = 13,
00036         MaxTorque         = 14,
00037         StatusReturnLevel = 16,
00038         AlarmLED          = 17,
00039         AlarmShutdown     = 18
00040     };

```

### 3.1.3 Constructor & Destructor Documentation

#### 3.1.3.1 AX12::AX12 ( dynamixel \* dxl, int ID = -1 )

Default constructor must pass an initialized dynamixel object if ID == -1 no action is done.

```

00005                                     :
00006         dxl (dxl),
00007         _ID (ID),
00008         _mode (true),
00009         _rads (false)
00010 {
00011     if (_ID < 0) return;
00012     dxl->write_byte(_ID, RAM::TorqueEnable, true);
00013 }

```

#### 3.1.3.2 AX12::AX12 ( const AX12 & a )

Copy constructor.

```

00015                                     :
00016         dxl (a.dxl),
00017         _ID (a._ID),
00018         _mode (a._mode),
00019         _rads (a._rads)
00020 {
00021
00022 }

```

#### 3.1.3.3 AX12::~~AX12 ( )

Default destructor.

```

00025 {
00026
00027 }

```

### 3.1.4 Member Function Documentation

#### 3.1.4.1 QVector< int > AX12::connectedID ( )

Returns all active servos;

```
00030 {
00031     QVector <int> res;
00032     for (int i = 0; i < 256; ++i) {
00033         dxl->ping(i);
00034         if (dxl->get_comm_result() == COMM_RXSUCCESS) res.push_back(i);
00035     }
00036
00037     return res;
00038 }
```

#### 3.1.4.2 double AX12::getCurrentLoad ( )

Returns the current load from -100% to 100%, 100% is ClockWise and -100% is CounterClockWise.

```
00041 {
00042     if (_ID < 0) return 0;
00043     int load = dxl->read_word(_ID, RAM::PresentLoad);
00044     load -= 1024;
00045     if (load == -1024) load = 0;
00046     return double((load/1023)*100);
00047 }
```

#### 3.1.4.3 double AX12::getCurrentPos ( )

Returns the current position from 0° to 300°

```
00050 {
00051     if (_ID < 0) return 0;
00052     int pos = dxl->read_word(_ID, RAM::PresentPosition);
00053     if (dxl->get_comm_result() != COMM_RXSUCCESS) return -1;
00054
00055     if (_rads) return double((pos/1023.0)*5*M_PI/3);
00056     return double((pos/1023.0)*300);
00057 }
```

#### 3.1.4.4 double AX12::getCurrentSpeed ( )

Returns the current speed from -100% to 100%, 100% is ClockWise and -100% is CounterClockWise.

```
00068 {
00069     if (_ID < 0) return 0;
00070     int speed = dxl->read_word(_ID, RAM::PresentSpeed);
00071     if (dxl->get_comm_result() != COMM_RXSUCCESS) return -1;
00072     speed -= 1024;
00073     if (speed == -1024) speed = 0;
00074     return double((speed/1023.0)*100);
00075 }
```

#### 3.1.4.5 int AX12::getCurrentTemp ( )

Returns the current Temperature in Celsius.

```
00060 {
00061     if (_ID < 0) return 0;
00062     int temp = dxl->read_byte(_ID, RAM::PresentTemperature);
00063     if (dxl->get_comm_result() != COMM_RXSUCCESS) return -1;
00064     return temp;
00065 }
```

**3.1.4.6 double AX12::getCurrentVoltage ( )**

Returns the current voltage in Volts.

```
00078 {
00079     if (_ID < 0) return 0;
00080     char voltage = dxl->read_byte(_ID, RAM::PresentVoltage);
00081     if (dxl->get_comm_result() != COMM_RXSUCCESS) return -1;
00082     return double(voltage/10.0);
00083 }
```

**3.1.4.7 int AX12::getID ( ) [inline]**

To get the current ID.

```
00111 { return _ID; }
```

**3.1.4.8 void AX12::setGoalPosition ( double goal )**

Sets the Goal's position (in degrees) or speed depending on the mode.

```
00086 {
00087     if (_ID < 0) return;
00088     if (goal > 300.0) goal = 300.0;
00089     else if (goal < 0) goal = 0;
00090     dxl->write_word(_ID, RAM::GoalPosition, int((goal/300.0)*1023));
00091 }
```

**3.1.4.9 void AX12::setID ( int ID )**

To set a new ID.

```
00094 {
00095     _ID = ID;
00096     if (ID < 0) return;
00097     dxl->write_byte(_ID, RAM::TorqueEnable, true);
00098 }
```

**3.1.4.10 void AX12::setJointMode ( bool mode )**

To set Joint/Wheel mode, true if Joint.

```
00101 {
00102     if (_ID < 0) return;
00103     _mode = mode;
00104     if (_mode) {
00105         dxl->write_word(_ID, ROM::CWAngleLimit, 0);
00106         dxl->write_word(_ID, ROM::CCWAngleLimit, 1023);
00107     }
00108     else {
00109         dxl->write_word(_ID, ROM::CWAngleLimit, 0);
00110         dxl->write_word(_ID, ROM::CCWAngleLimit, 0);
00111     }
00112 }
```

#### 3.1.4.11 void AX12::setMinMax ( double *min*, double *max* )

To set the minimum and maximum angle from 0 to 300°

```
00115 {
00116     if (_ID < 0) return;
00117
00118     if (min > max) std::swap(min, max);
00119
00120     if (min < 0.0) min = 0;
00121     if (max > 300.0) max = 300;
00122
00123     min = (min/300)*1023;
00124     max = (max/300)*1023;
00125
00126     dxl->write_word(_ID, ROM::CWAngleLimit, int (min));
00127     dxl->write_word(_ID, ROM::CCWAngleLimit, int (max));
00128 }
```

#### 3.1.4.12 void AX12::setSpeed ( double *speed* )

To set the maximum speed from 0% to 100% if joint mode or from -100% to 100% if wheel mode.

```
00131 {
00132     if (_ID < 0) return;
00133     if (speed > 100.0) speed = 100.0;
00134     if (_mode) {
00135         if (speed < 0.0) speed = 0.0;
00136
00137         int byte = int((speed/100.0) * 1024.0);
00138         if (speed == 100.0) byte = 0;
00139         dxl->write_byte(_ID, RAM::MovingSpeed, byte);
00140     }
00141     else {
00142         if (speed < -100.0) speed = -100.0;
00143
00144         int byte = int(((speed + 100)/100.0) * 1024);
00145         dxl->write_byte(_ID, RAM::MovingSpeed, byte);
00146     }
00147 }
00148 }
```

### 3.1.5 Member Data Documentation

#### 3.1.5.1 int AX12::\_ID [private]

Stores the current ID.

#### 3.1.5.2 bool AX12::\_mode [private]

True if we use the joint mode.

#### 3.1.5.3 bool AX12::\_rads [private]

True if the angle is returned in radians.

#### 3.1.5.4 dynamixel\* AX12::dxl [private]

Contains the dynamixel communication.

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

- [dxl/ax12.h](#)
- [dxl/ax12.cpp](#)

## 3.2 dxl\_hal Class Reference

Dynamixel SDK platform dependent.

```
#include <dxl_hal.h>
```

### Public Member Functions

- bool `open` (QString &devName, int baudrate)
- void `close` (void)
- void `clear` (void)
- int `change_baudrate` (float baudrate)
- int `write` (unsigned char \*pPacket, int numPacket)
- int `read` (unsigned char \*pPacket, int numPacket)
- double `get_curr_time` ()
- bool `isOpen` ()

### Private Attributes

- QSerialPort `_serial`
- int `_time` = 30
- bool `_timed` = false
- bool `_open` = false

### 3.2.1 Detailed Description

Dynamixel SDK platform dependent.

### 3.2.2 Member Function Documentation

#### 3.2.2.1 int dxl\_hal::change\_baudrate ( float *baudrate* )

```
00039 {  
00040     bool res = _serial.setBaudRate(qint32(baudrate));  
00041     return int(res);  
00042 }  
00043 }
```

#### 3.2.2.2 void dxl\_hal::clear ( void )

```
00032 {  
00033     // Clear communication buffer  
00034     _serial.clear();  
00035 }  
00036 }
```

#### 3.2.2.3 void dxl\_hal::close ( void )

```
00025 {  
00026     // Closing device  
00027     _serial.close();  
00028     _open = false;  
00029 }
```

**3.2.2.4 double dxl\_hal::get\_curr\_time ( )**

```

00080 {
00081     return (double)QTime::currentTime().msecsSinceStartOfDay();
00082 }

```

**3.2.2.5 bool dxl\_hal::isOpen ( ) [inline]**

```

00030 { return _open; }

```

**3.2.2.6 bool dxl\_hal::open ( QString & devName, int baudrate )**

```

00007 {
00008     // Opening device
00009     // devIndex: Device index
00010     // baudrate: Real baudrate (ex> 115200, 57600, 38400...)
00011     // Return: 0(Failed), 1(Succeed)
00012
00013     _serial.setPortName(devName);
00014     _serial.setBaudRate(qint32(baudrate));
00015     _serial.setDataBits(QSerialPort::Data8);
00016     _serial.setParity(QSerialPort::NoParity);
00017     _serial.setStopBits(QSerialPort::OneStop);
00018     _serial.setFlowControl(QSerialPort::NoFlowControl);
00019     if(not _serial.open(QIODevice::ReadWrite)) return false;
00020     _open = true;
00021     return true;
00022 }

```

**3.2.2.7 int dxl\_hal::read ( unsigned char \* pPacket, int numPacket )**

```

00063 {
00064     // Recieving date
00065     // *pPacket: data array pointer
00066     // numPacket: number of data array
00067     // Return: number of data recieved. -1 is error.
00068     _timed = false;
00069     if (_serial.isOpen()) {
00070         int n = _serial.read((char*)pPacket, numPacket);
00071         _timed = _serial.waitForReadyRead(_time);
00072         _timed = not _timed;
00073         return n;
00074     }
00075     else return -1;
00076
00077 }

```

**3.2.2.8 int dxl\_hal::write ( unsigned char \* pPacket, int numPacket )**

```

00046 {
00047     // Transmitting date
00048     // *pPacket: data array pointer
00049     // numPacket: number of data array
00050     // Return: number of data transmitted. -1 is error.
00051     _timed = false;
00052     if (_serial.isOpen()) {
00053         int n = _serial.write((char*)pPacket, numPacket);
00054         _timed = _serial.waitForBytesWritten(_time);
00055         _timed = not _timed;
00056         return n;
00057     }
00058     else return -1;
00059
00060 }

```

**3.2.3 Member Data Documentation**

3.2.3.1 `bool dxl_hal::_open = false` [private]

3.2.3.2 `QSerialPort dxl_hal::_serial` [private]

3.2.3.3 `int dxl_hal::_time = 30` [private]

3.2.3.4 `bool dxl_hal::_timed = false` [private]

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

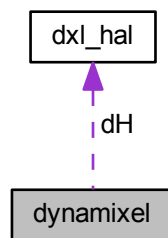
- [dxl/dxl\\_hal.h](#)
- [dxl/dxl\\_hal.cpp](#)

### 3.3 dynamixel Class Reference

Dynamixel 1.0 protocol class.

```
#include <dynamixel.h>
```

Collaboration diagram for dynamixel:



#### Public Member Functions

- [dynamixel](#) ()  
*Default constructor.*
- [dynamixel](#) (QString port\_num, int baud\_rate=1000000)  
*Initialization constructor.*
- `bool` [isOpen](#) ()  
*True if the port is open.*
- `int` [initialize](#) (QString port\_num, int baud\_rate)  
*Initializes the port.*
- `int` [change\\_baudrate](#) (int baud\_rate)  
*Changes the current baud rate.*
- `int` [terminate](#) (void)  
*Closes the communication.*
- `int` [get\\_comm\\_result](#) ()  
*Returns the current com status.*
- `void` [tx\\_packet](#) (void)



- Sends a packet.*
  - void [rx\\_packet](#) (void)
- Receives a packet.*
  - void [txrx\\_packet](#) (void)
- Sends and receives a packet.*
  - void [set\\_txpacket\\_id](#) (int id)
- Sets the sending packet ID.*
  - void [set\\_txpacket\\_instruction](#) (int instruction)
- Sets the sending packet instruction.*
  - void [set\\_txpacket\\_parameter](#) (int index, int value)
- Sets the sending packet parameter.*
  - void [set\\_txpacket\\_length](#) (int length)
- Sets the sending packet length.*
  - bool [get\\_rxpacket\\_error](#) (int error)
- Returns false if no receive error and true if there's an error.*
  - int [get\\_rxpacket\\_error\\_byte](#) (void)
- Returns the error byte.*
  - int [get\\_rxpacket\\_parameter](#) (int index)
- Returns the received parameter.*
  - int [get\\_rxpacket\\_length](#) ()
- Returns the received packet length.*
  - void [ping](#) (int id)
- Ping to the selected id, check com status for the ping result.*
  - int [read\\_byte](#) (int id, int address)
- Reads a byte from the selected ID at the selected address.*
  - void [write\\_byte](#) (int id, int address, int value)
- Writes a byte to the selected ID at the selected address.*
  - int [read\\_word](#) (int id, int address)
- Reads a word to the selected ID at the selected address.*
  - void [write\\_word](#) (int id, int address, int value)
- Writes a word to the selected ID at the selected address.*
  - double [get\\_packet\\_time](#) ()
- Returns the packet time.*
  - void [set\\_packet\\_timeout](#) (int NumRcvByte)
- Sets the timeout in number of received bytes.*
  - void [set\\_packet\\_timeout\\_ms](#) (int msec)
- Sets the timeout in ms.*
  - bool [is\\_packet\\_timeout](#) ()
- Returns true if the packet is timeout.*

### Private Attributes

- [dxl\\_hal](#) dH
  - Conains the serial port comunication.*
- unsigned char [gbInstructionPacket](#) [[MAXNUM\\_TXPACKET](#)] = {0}
  - Contains all the instructions.*
- unsigned char [gbStatusPacket](#) [[MAXNUM\\_RXPACKET](#)] = {0}
  - Contains the status.*
- unsigned int [gbRxPacketLength](#) = 0
  - Received packet length.*

- unsigned int `gbRxGetLength` = 0  
*Temporal length from the received packet.*
- double `gdPacketStartTime` = 0.0  
*Packet start time.*
- double `gdByteTransTime` = 0.0  
*Byte transmission time.*
- double `gdRcvWaitTime` = 0.0  
*Receive wait time.*
- int `gbCommStatus` = `COMM_RXSUCCESS`  
*Current communication status.*
- int `giBusUsing` = 0  
*True if the bus is being used.*

### 3.3.1 Detailed Description

Dynamixel 1.0 protocol class.

### 3.3.2 Constructor & Destructor Documentation

#### 3.3.2.1 `dynamixel::dynamixel ( )`

Default constructor.

```
00014 {
00015
00016 }
```

#### 3.3.2.2 `dynamixel::dynamixel ( QString port_num, int baud_rate = 1000000 )`

Initialization constructor.

```
00019 {
00020     initialize(port_num, baud_rate);
00021 }
```

### 3.3.3 Member Function Documentation

#### 3.3.3.1 `int dynamixel::change_baudrate ( int baud_rate )`

Changes the current baud rate.

```
00039 {
00040     int result = 0;
00041     float baudrate = (float)baud_rate;
00042
00043     result = dH.change_baudrate(baudrate);
00044     if(result == 1)
00045         gdByteTransTime = 1000.0f / baudrate * 10.0; // 1000/baudrate(bit per msec) *
00046         10(start bit + data bit + stop bit)
00047     return result;
00048 }
```

**3.3.3.2** `int dynamixel::get_comm_result( ) [inline]`

Returns the current com status.

```
00115 { return gbCommStatus; }
```

**3.3.3.3** `double dynamixel::get_packet_time( void )`

Returns the packet time.

```
00058 {
00059     double elapsed_time;
00060
00061     elapsed_time = (double) (dH.get_curr_time() -
gdPacketStartTime);
00062
00063     // Overflow
00064     if(elapsed_time < 0) gdPacketStartTime = dH.get_curr_time();
00065
00066     return elapsed_time;
00067 }
```

**3.3.3.4** `bool dynamixel::get_rxpacket_error( int error )`

Returns false if no receive error and true if there's an error.

Parameters

<i>error</i>	Selects the error to check
--------------	----------------------------

```
00279 {
00280     if( gbStatusPacket[PRT1_PKT_ERRBIT] & (unsigned char)error )
00281         return true;
00282
00283     return false;
00284 }
```

**3.3.3.5** `int dynamixel::get_rxpacket_error_byte( void )`

Returns the error byte.

```
00287 {
00288     return gbStatusPacket[PRT1_PKT_ERRBIT];
00289 }
```

**3.3.3.6** `int dynamixel::get_rxpacket_length( )`

Returns the received packet length.

```
00297 {
00298     return (int)gbStatusPacket[PRT1_PKT_LENGTH];
00299 }
```

**3.3.3.7** `int dynamixel::get_rxpacket_parameter( int index )`

Returns the received parameter.

```
00292 {
00293     return (int)gbStatusPacket[PRT1_PKT_PARAMETER0+index];
00294 }
```

### 3.3.3.8 int dynamixel::initialize ( QString port\_num, int baud\_rate )

Initializes the port.

```
00024 {
00025     if( baud_rate < 1900 ) return 0;
00026
00027     if( not dH.open(port_num, baud_rate) ) return false;
00028
00029     // 1000/baudrate(bit per msec) * 10(start bit + data bit + stop bit)
00030     gdByteTransTime = 1000.0 / (double)baud_rate * 10.0;
00031
00032     gbCommStatus = COMM_RXSUCCESS;
00033     giBusUsing = 0;
00034
00035     return true;
00036 }
```

### 3.3.3.9 bool dynamixel::is\_packet\_timeout ( void )

Returns true if the packet is timeout.

#### Returns

True if the packet is timeout

```
00082 {
00083     if(this->get_packet_time() > gdRcvWaitTime)
00084         return true;
00085     return false;
00086 }
```

### 3.3.3.10 bool dynamixel::isOpen ( ) [inline]

True if the port is open.

```
00103 { return dH.isOpen(); }
```

### 3.3.3.11 void dynamixel::ping ( int id )

Ping to the selected id, check com status for the ping result.

#### Parameters

<i>id</i>	ID where the ping is done
-----------	---------------------------

```
00302 {
00303     while(giBusUsing);
00304
00305     gbInstructionPacket[PRT1_PKT_ID] = (unsigned char)id;
00306     gbInstructionPacket[PRT1_PKT_INSTRUCTION] =
INST_PING;
00307     gbInstructionPacket[PRT1_PKT_LENGTH] = 2;
00308
00309     txrx_packet();
00310 }
```

### 3.3.3.12 int dynamixel::read\_byte ( int id, int address )

Reads a byte from the selected ID at the selected address.

## Parameters

<i>id</i>	Selects the ID to read the byte
<i>address</i>	Selects the address to read the byte

```

00313 {
00314     while(giBusUsing);
00315
00316     gbInstructionPacket[PRT1_PKT_ID] = (unsigned char)id;
00317     gbInstructionPacket[PRT1_PKT_INSTRUCTION] =
INST_READ;
00318     gbInstructionPacket[PRT1_PKT_PARAMETER0+0] = (unsigned char)
address;
00319     gbInstructionPacket[PRT1_PKT_PARAMETER0+1] = 1;
00320     gbInstructionPacket[PRT1_PKT_LENGTH] = 4;
00321
00322     txrx_packet();
00323
00324     return (int)gbStatusPacket[PRT1_PKT_PARAMETER0];
00325 }

```

## 3.3.3.13 int dynamixel::read\_word ( int id, int address )

Reads a word to the selected ID at the selected address.

## Parameters

<i>id</i>	Selects the ID to read the word
<i>address</i>	Selects the address to read the word

```

00341 {
00342     while(giBusUsing);
00343
00344     gbInstructionPacket[PRT1_PKT_ID] = (unsigned char)id;
00345     gbInstructionPacket[PRT1_PKT_INSTRUCTION] =
INST_READ;
00346     gbInstructionPacket[PRT1_PKT_PARAMETER0+0] = (unsigned char)
address;
00347     gbInstructionPacket[PRT1_PKT_PARAMETER0+1] = 2;
00348     gbInstructionPacket[PRT1_PKT_LENGTH] = 4;
00349
00350     txrx_packet();
00351
00352     return MAKEWORD((int)gbStatusPacket[
PRT1_PKT_PARAMETER0+0], (int)gbStatusPacket[
PRT1_PKT_PARAMETER0+1]);
00353 }

```

## 3.3.3.14 void dynamixel::rx\_packet ( void )

Receives a packet.

```

00152 {
00153     unsigned char i = 0, j = 0, nRead = 0;
00154     unsigned char checksum = 0;
00155
00156     if( giBusUsing == 0 )
00157         return;
00158
00159     if( gbInstructionPacket[PRT1_PKT_ID] ==
BROADCAST_ID )
00160     {
00161         gbCommStatus = COMM_RXSUCCESS;
00162         giBusUsing = 0;
00163         return;
00164     }
00165
00166     if( gbCommStatus == COMM_TXSUCCESS )
00167     {
00168         gbRxGetLength = 0;
00169         //gbRxPacketLength = 6; //minimum wait length
00170     }
00171

```

```

00172     while(1)
00173     {
00174         nRead = dh.read( &gbStatusPacket[gbRxGetLength],
gbRxPacketLength - gbRxGetLength );
00175         gbRxGetLength += nRead;
00176
00177         if(gbRxGetLength > 4)
00178             gbRxPacketLength = gbStatusPacket[
PRT1_PKT_LENGTH] + 4;
00179
00180         if( gbRxGetLength < gbRxPacketLength )
00181         {
00182             if( is_packet_timeout() == 1 )
00183             {
00184                 if(gbRxGetLength == 0)
00185                     gbCommStatus = COMM_RXTIMEOUT;
00186                 else
00187                     gbCommStatus = COMM_RXCORRUPT;
00188                 giBusUsing = 0;
00189                 return;
00190             }
00191             gbCommStatus = COMM_RXWAITING;
00192             //return;
00193         }
00194         else
00195         {
00196             break;
00197         }
00198     }
00199
00200     // Find packet header
00201     for( i=0; i<(gbRxGetLength-1); i++ )
00202     {
00203         if( gbStatusPacket[i] == 0xff && gbStatusPacket[i+1] == 0xff )
00204             break;
00205         else if( i == gbRxGetLength-2 && gbStatusPacket[gbRxGetLength-1] == 0xff )
00206             break;
00207         else {
00208             gbCommStatus = COMM_RXCORRUPT;
00209             return;
00210         }
00211     }
00212
00213     if( i > 0 )
00214     {
00215         for( j=0; j<(gbRxGetLength-i); j++ )
00216             gbStatusPacket[j] = gbStatusPacket[j + i];
00217
00218         gbRxGetLength -= i;
00219     }
00220
00221     // Check id pairing
00222     if( gbInstructionPacket[PRT1_PKT_ID] !=
gbStatusPacket[PRT1_PKT_ID])
00223     {
00224         gbCommStatus = COMM_RXCORRUPT;
00225         giBusUsing = 0;
00226         return;
00227     }
00228
00229     // Check checksum
00230     for( i=0; i<(gbStatusPacket[PRT1_PKT_LENGTH]+1); i++ )
00231         checksum += gbStatusPacket[i+2];
00232     checksum = ~checksum;
00233
00234     if( gbStatusPacket[gbStatusPacket[
PRT1_PKT_LENGTH]+3] != checksum )
00235     {
00236         gbCommStatus = COMM_RXCORRUPT;
00237         giBusUsing = 0;
00238         return;
00239     }
00240
00241     gbCommStatus = COMM_RXSUCCESS;
00242     giBusUsing = 0;
00243 }

```

### 3.3.3.15 void dynamixel::set\_packet\_timeout ( int NumRcvByte )

Sets the timeout in number of received bytes.

## Parameters

<i>NumRcvByte</i>	Number of received bytes to do a timeout
-------------------	--

```

00070 {
00071     gdPacketStartTime = dH.get_curr_time();
00072     gdRcvWaitTime = (gdByteTransTime*(double)NumRcvByte + 2.0*
    LATENCY_TIME + 2.0);
00073 }

```

**3.3.3.16 void dynamixel::set\_packet\_timeout\_ms ( int msec )**

Sets the timeout in ms.

## Parameters

<i>msec</i>	Miliseconds for the timeout
-------------	-----------------------------

```

00076 {
00077     gdPacketStartTime = dH.get_curr_time();
00078     gdRcvWaitTime = (double)msec;
00079 }

```

**3.3.3.17 void dynamixel::set\_txpacket\_id ( int id )**

Sets the sending packet ID.

```

00258 {
00259     gbInstructionPacket[PRT1_PKT_ID] = (unsigned char)id;
00260 }

```

**3.3.3.18 void dynamixel::set\_txpacket\_instruction ( int instruction )**

Sets the sending packet instruction.

```

00263 {
00264     gbInstructionPacket[PRT1_PKT_INSTRUCTION] = (unsigned char)
    instruction;
00265 }

```

**3.3.3.19 void dynamixel::set\_txpacket\_length ( int length )**

Sets the sending packet length.

```

00274 {
00275     gbInstructionPacket[PRT1_PKT_LENGTH] = (unsigned char)length;
00276 }

```

**3.3.3.20 void dynamixel::set\_txpacket\_parameter ( int index, int value )**

Sets the sending packet parameter.

```

00268 {
00269     gbInstructionPacket[PRT1_PKT_PARAMETER0+index] = (unsigned char)
    value;
00270 }
00271 }

```

### 3.3.3.21 int dynamixel::terminate ( void )

Closes the communication.

```
00051 {
00052     dH.close();
00053     return 0;
00054 }
```

### 3.3.3.22 void dynamixel::tx\_packet ( void )

Sends a packet.

```
00090 {
00091     unsigned char pkt_idx = 0;
00092     unsigned char TxNumByte, RealTxNumByte;
00093     unsigned char checksum = 0;
00094
00095     if( giBusUsing == 1 )
00096     {
00097         gbCommStatus = COMM_TXFAIL;
00098         return;
00099     }
00100
00101     giBusUsing = 1;
00102
00103     if( gbInstructionPacket[PRT1_PKT_INSTRUCTION] !=
INST_READ
00104         && gbInstructionPacket[PRT1_PKT_INSTRUCTION] !=
INST_WRITE
00105         && gbInstructionPacket[PRT1_PKT_INSTRUCTION] !=
INST_REG_WRITE
00106         && gbInstructionPacket[PRT1_PKT_INSTRUCTION] !=
INST_ACTION
00107         && gbInstructionPacket[PRT1_PKT_INSTRUCTION] !=
INST_RESET
00108         && gbInstructionPacket[PRT1_PKT_INSTRUCTION] !=
INST_SYNC_WRITE )
00109     {
00110         gbCommStatus = COMM_TXERROR;
00111         giBusUsing = 0;
00112         return;
00113     }
00114
00115     gbInstructionPacket[0] = 0xff;
00116     gbInstructionPacket[1] = 0xff;
00117     for( pkt_idx = 0; pkt_idx < (gbInstructionPacket[
PRT1_PKT_LENGTH]+1); pkt_idx++ )
00118         checksum += gbInstructionPacket[pkt_idx+2];
00119     gbInstructionPacket[gbInstructionPacket[
PRT1_PKT_LENGTH]+3] = ~checksum;
00120
00121     //if( gbCommStatus == COMM_RXTIMEOUT || gbCommStatus == COMM_RXCORRUPT )
00122     //    dH.clear();
00123
00124     dH.clear();
00125
00126     TxNumByte = gbInstructionPacket[PRT1_PKT_LENGTH] + 4;
00127     RealTxNumByte = dH.write( gbInstructionPacket, TxNumByte );
00128
00129     if( TxNumByte != RealTxNumByte )
00130     {
00131         gbCommStatus = COMM_TXFAIL;
00132         giBusUsing = 0;
00133         return;
00134     }
00135
00136     if( gbInstructionPacket[PRT1_PKT_INSTRUCTION] ==
INST_READ )
00137     {
00138         gbRxPacketLength = gbInstructionPacket[
PRT1_PKT_PARAMETER0+1] + 6;
00139         set_packet_timeout( gbInstructionPacket[
PRT1_PKT_PARAMETER0+1] + 6 );
00140     }
00141     else
00142     {
00143         gbRxPacketLength = 6;
00144     }
```



```

00145         set_packet_timeout( 6 );
00146     }
00147
00148     gbCommStatus = COMM_TXSUCCESS;
00149 }

```

### 3.3.3.23 void dynamixel::txrx\_packet ( void )

Sends and receives a packet.

```

00246 {
00247     tx_packet ();
00248
00249     if ( gbCommStatus != COMM_TXSUCCESS )
00250         return;
00251
00252
00253     rx_packet ();
00254 }

```

### 3.3.3.24 void dynamixel::write\_byte ( int id, int address, int value )

Writes a byte to the selected ID at the selected address.

#### Parameters

<i>id</i>	Selects the ID to write the byte
<i>address</i>	Selects the address to write the byte
<i>value</i>	Value to set at the selected location

```

00328 {
00329     while (giBusUsing);
00330
00331     gbInstructionPacket[PRT1_PKT_ID] = (unsigned char)id;
00332     gbInstructionPacket[PRT1_PKT_INSTRUCTION] =
INST_WRITE;
00333     gbInstructionPacket[PRT1_PKT_PARAMETER0+0] = (unsigned char)
address;
00334     gbInstructionPacket[PRT1_PKT_PARAMETER0+1] = (unsigned char)value
;
00335     gbInstructionPacket[PRT1_PKT_LENGTH] = 4;
00336
00337     txrx_packet ();
00338 }

```

### 3.3.3.25 void dynamixel::write\_word ( int id, int address, int value )

Writes a word to the selected ID at the selected address.

#### Parameters

<i>id</i>	Selects the ID to write the word
<i>address</i>	Selects the address to write the word
<i>value</i>	Value to set at the selected location

```

00356 {
00357     while (giBusUsing);
00358
00359     gbInstructionPacket[PRT1_PKT_ID] = (unsigned char)id;
00360     gbInstructionPacket[PRT1_PKT_INSTRUCTION] =
INST_WRITE;
00361     gbInstructionPacket[PRT1_PKT_PARAMETER0+0] = (unsigned char)
address;
00362     gbInstructionPacket[PRT1_PKT_PARAMETER0+1] = (unsigned char)
LOBYTE (value);
00363     gbInstructionPacket[PRT1_PKT_PARAMETER0+2] = (unsigned char)
HIBYTE (value);

```

```

00364     gbInstructionPacket[PRT1_PKT_LENGTH] = 5;
00365
00366     txrx_packet();
00367 }

```

### 3.3.4 Member Data Documentation

#### 3.3.4.1 dxl\_hal dynamixel::dH [private]

Contains the serial port communication.

#### 3.3.4.2 int dynamixel::gbCommStatus = COMM\_RXSUCCESS [private]

Current communication status.

#### 3.3.4.3 unsigned char dynamixel::gbInstructionPacket[MAXNUM\_TXPACKET] = {0} [private]

Contains all the instructions.

#### 3.3.4.4 unsigned int dynamixel::gbRxGetLength = 0 [private]

Temporal length from the received packet.

#### 3.3.4.5 unsigned int dynamixel::gbRxPacketLength = 0 [private]

Received packet length.

#### 3.3.4.6 unsigned char dynamixel::gbStatusPacket[MAXNUM\_RXPACKET] = {0} [private]

Contains the status.

#### 3.3.4.7 double dynamixel::gdByteTransTime = 0.0 [private]

Byte transmission time.

#### 3.3.4.8 double dynamixel::gdPacketStartTime = 0.0 [private]

Packet start time.

#### 3.3.4.9 double dynamixel::gdRcvWaitTime = 0.0 [private]

Receive wait time.

#### 3.3.4.10 int dynamixel::giBusUsing = 0 [private]

True if the bus is being used.

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

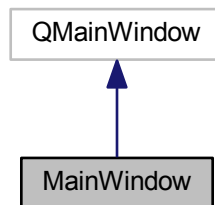
- [dxl/dynamixel.h](#)
- [dxl/dynamixel.cpp](#)

## 3.4 MainWindow Class Reference

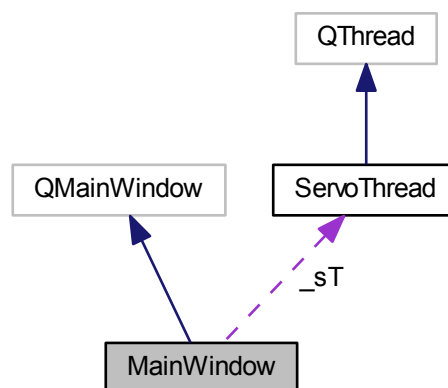
Contains all the windows and other classes.

```
#include <mainwindow.h>
```

Inheritance diagram for MainWindow:



Collaboration diagram for MainWindow:



### Signals

- `void joystickChanged ()`  
*Emitted when a joystick changes.*

### Public Member Functions

- `MainWindow (QWidget *parent=0)`  
*Default constructor.*
- `~MainWindow ()`  
*Default destructor.*

## Private Slots

- void [joyChanged](#) ()  
*Handles a joystick update.*
- void [on\\_actionOptions\\_triggered](#) ()  
*To select the options.*
- void [update](#) ()  
*Updates all data to the servo thread.*
- void [on\\_start\\_clicked](#) ()

## Private Attributes

- QVector< QLabel \* > [\\_axis](#)  
*Handles all the axis labels.*
- QVector< float > [\\_axisV](#)  
*Contains the axis value;.*
- QVector< QLabel \* > [\\_buts](#)  
*Handles all the button labels.*
- QVector< bool > [\\_butsV](#)  
*Handles all buttons values.*
- QString [\\_dataP](#)  
*Contains the path to the data location.*
- int [\\_jAxisX](#) = -1  
*Axis for the X value.*
- int [\\_jAxisY](#) = -1  
*Axis for the Y value.*
- int [\\_jAxisZ](#) = -1  
*Axis for the Z value.*
- XJoystick [\\_joy](#)  
*To handle the joystick.*
- ServoThread [\\_sT](#)  
*Contains the thread controlling all the servos and external hardware.*
- QTimer [\\_timer](#)  
*To update the joystick value.*
- Ui::MainWindow \* [ui](#)  
*Contains the user interface.*

## Static Private Attributes

- static const int [sCount](#) = 3  
*Contains the number of minimum servos to work.*
- static const int [aSCount](#) = 0  
*Contains the number of additional servos used.*

### 3.4.1 Detailed Description

Contains all the windows and other classes.

### 3.4.2 Constructor & Destructor Documentation

#### 3.4.2.1 MainWindow::MainWindow ( QWidget \* parent = 0 ) [explicit]

Default constructor.

```

00005                                     :
00006     QMainWindow(parent),
00007     _axis(XJoystick::AxisCount),
00008     _axisV(XJoystick::AxisCount),
00009     _buts(XJoystick::ButtonCount),
00010     _butsV(XJoystick::ButtonCount),
00011     ui(new Ui::MainWindow)
00012 {
00013     ui->setupUi(this);
00014
00015     _sT.setStatusBar(ui->statusbar);
00016     _sT.start();
00017
00018     connect(&_joy, SIGNAL(changed()), this, SLOT(joyChanged()));
00019     connect(&_timer, SIGNAL(timeout()), this, SLOT(update()));
00020
00021
00022     _timer.setInterval(10);
00023     _timer.start();
00024
00025     // JOYSTICK
00026     QVector< QString > V(_joy.getAllAxis());
00027     // Adding axis
00028     QGridLayout *wL = new QGridLayout;
00029     for (int i = 0; i < XJoystick::AxisCount; ++i) {
00030         QHBoxLayout *L = new QHBoxLayout;
00031         L->addWidget(new QLabel(V[i].append(":"), this));
00032         _axis[i] = new QLabel("#");
00033         L->addWidget(_axis[i]);
00034         L->addStretch();
00035         wL->addLayout(L, i%3, i/3);
00036     }
00037     ui->joyAxis->setLayout(wL);
00038
00039     // Adding buttons
00040     wL = new QGridLayout;
00041     for (int i = 0; i < XJoystick::ButtonCount; ++i) {
00042         _buts[i] = new QLabel(QString::number(i + 1));
00043         wL->addWidget(_buts[i], i/8, i%8);
00044         _buts[i]->setEnabled(false);
00045         _buts[i]->hide();
00046     }
00047     ui->joyButs->setLayout(wL);
00048     ui->joyAxis->hide();
00049     ui->joyButs->hide();
00050     ui->line->hide();
00051
00052
00053     // Creating data Path
00054     _dataP = QStandardPaths::writableLocation(QStandardPaths::AppDataLocation);
00055     QDir dir(_dataP);
00056     if (!dir.exists()) dir.mkpath(_dataP);
00057 }

```

#### 3.4.2.2 MainWindow::~MainWindow ( )

Default destructor.

```

00060 {
00061     delete ui;
00062 }

```

### 3.4.3 Member Function Documentation

#### 3.4.3.1 void MainWindow::joyChanged ( ) [private],[slot]

Handles a joystick update.

```

00065 {
00066     int sel = _joy.current();
00067
00068     QVector< XJoystick::Info > V(_joy.available());
00069     bool found = false;
00070     int i = 0;
00071     while (i < V.size() and not found) { found = V[i].ID == sel; ++i; }
00072     if (not found) {
00073         if (V.size() > 0) {
00074             _joy.select(V[0].ID);
00075             ui->line->hide();
00076
00077             // Showing axis
00078             ui->joyAxis->show();
00079
00080             // Showing buttons
00081             for (QLabel *l : _buts) l->hide();
00082             ui->joyButs->show();
00083             int n = _joy.buttonCount();
00084             for (int i = 0; i < n; ++i) _buts[i]->show();
00085         }
00086         else {
00087             _joy.select(-1);
00088             ui->joyAxis->hide();
00089             ui->joyButs->hide();
00090             ui->line->hide();
00091         }
00092     }
00093     emit joystickChanged();
00094 }

```

#### 3.4.3.2 void MainWindow::joystickChanged ( ) [signal]

Emmitted when a joystick changes.

#### 3.4.3.3 void MainWindow::on\_actionOptions\_triggered ( ) [private],[slot]

To select the options.

```

00098 {
00099     OptionsWindow o(_joy, &_sT, this);
00100     o.exec();
00101
00102     connect(this, SIGNAL(joystickChanged()), &o, SLOT(
00103         joystickChanged()));
00104     if (o.result()) o.storeData();
00105 }

```

#### 3.4.3.4 void MainWindow::on\_start\_clicked ( ) [private],[slot]

```

00120 {
00121     QString text = ui->start->text();
00122
00123     if (text == "Start") {
00124         _sT.wakeUp();
00125         ui->start->setText("Stop");
00126     }
00127     else if (text == "Stop") {
00128         _sT.pause();
00129         ui->start->setText("Start");
00130     }
00131 }

```

#### 3.4.3.5 void MainWindow::update ( ) [private],[slot]

Updates all data to the servo thread.

```

00108 {
00109     _joy.update();

```

```

00110     for (int i = 0; i < XJoystick::AxisCount; ++i) _axisV[i] = _joy[i];
00111     for (int i = 0; i < XJoystick::ButtonCount; ++i) _butsV[i] = _joy.button(i);
00112
00113     _sT.setData(_axisV, _butsV);
00114     QVector<ServoThread::Servo> servo(_sT.getServosInfo());
00115
00116     // TODO: Finish update function
00117 }

```

### 3.4.4 Member Data Documentation

#### 3.4.4.1 QVector< QLabel \*> MainWindow::\_axis [private]

Handles all the axis labels.

#### 3.4.4.2 QVector< float > MainWindow::\_axisV [private]

Contains the axis value;.

#### 3.4.4.3 QVector< QLabel \*> MainWindow::\_buts [private]

Handles all the button labels.

#### 3.4.4.4 QVector< bool > MainWindow::\_butsV [private]

Handles all buttons values.

#### 3.4.4.5 QString MainWindow::\_dataP [private]

Contains the path to the data location.

#### 3.4.4.6 int MainWindow::\_jAxisX = -1 [private]

Axis for the X value.

#### 3.4.4.7 int MainWindow::\_jAxisY = -1 [private]

Axis for the Y value.

#### 3.4.4.8 int MainWindow::\_jAxisZ = -1 [private]

Axis for the Z value.

#### 3.4.4.9 XJoystick MainWindow::\_joy [private]

To handle the joystick.

#### 3.4.4.10 ServoThread MainWindow::\_sT [private]

Contains the thread controlling all the servos and external hardware.

#### 3.4.4.11 QTimer MainWindow::\_timer [private]

To update the joystick value.

#### 3.4.4.12 const int MainWindow::aSCount = 0 [static], [private]

Contains the number of additional servos used.

#### 3.4.4.13 const int MainWindow::sCount = 3 [static], [private]

Contains the number of minimum servos to work.

#### 3.4.4.14 Ui::MainWindow\* MainWindow::ui [private]

Contains the user interface.

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

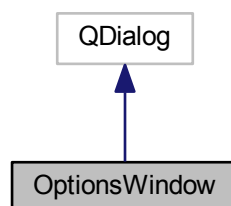
- [mainwindow.h](#)
- [mainwindow.cpp](#)

## 3.5 OptionsWindow Class Reference

Class used to handle a Window to set the options.

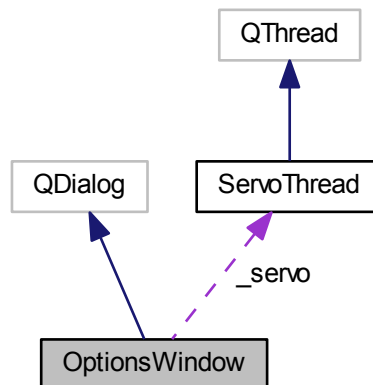
```
#include <optionswindow.h>
```

Inheritance diagram for OptionsWindow:





Collaboration diagram for OptionsWindow:



### Public Slots

- void `joystickChanged ()`  
*To handle the change of a joystick.*

### Public Member Functions

- `OptionsWindow (XJoystick &J, ServoThread *servo, QWidget *parent=0)`  
*Default constructor.*
- `~OptionsWindow ()`  
*Destructor.*
- void `storeData ()`  
*Stores all data.*

### Private Slots

- void `events ()`  
*Handles events that need to be updated continously.*
- void `on_servoRefresh_clicked ()`  
*Refreshes all the servos connected to the port.*

### Private Attributes

- XJoystick & `_joy`  
*Contains the Joystick to handle options.*
- int `_portSize`  
*Contains the size of the ports.*
- `ServoThread * _servo`  
*Pointer to the servo thread class.*
- QTimer `_timer`

*Waits for a new COM port.*

- `Ui::OptionsWindow * ui`

*Containsh the GUI.*

### 3.5.1 Detailed Description

Class used to handle a Window to set the options.

### 3.5.2 Constructor & Destructor Documentation

#### 3.5.2.1 OptionsWindow::OptionsWindow ( XJoystick & J, ServoThread \* servo, QWidget \* parent = 0 ) [explicit]

Default constructor.

```

00005                                     :
00006     QDialog(parent),
00007     _joy(J),
00008     _portSize(-1),
00009     _servo(servo),
00010     _timer(this),
00011     ui(new Ui::OptionsWindow)
00012 {
00013     ui->setupUi(this);
00014     this->setWindowTitle("Options");
00015
00016     QVector< QString > A(_joy.getAllAxis());
00017
00018     ui->joyMX->addItem("None", -1);
00019     ui->joyMY->addItem("None", -1);
00020     ui->joyMZ->addItem("None", -1);
00021
00022     for (int i = 0; i < A.size(); ++i) ui->joyMX->addItem(A[i], i);
00023     for (int i = 0; i < A.size(); ++i) ui->joyMY->addItem(A[i], i);
00024     for (int i = 0; i < A.size(); ++i) ui->joyMZ->addItem(A[i], i);
00025
00026     joystickChanged();
00027
00028     _timer.setInterval(500);
00029     _timer.setSingleShot(false);
00030     _timer.start();
00031     connect(&_amp;_timer, SIGNAL(timeout()), this, SLOT(events()));
00032
00033 }
```

#### 3.5.2.2 OptionsWindow::~OptionsWindow ( )

Destructor.

```

00036 {
00037     delete ui;
00038 }
```

### 3.5.3 Member Function Documentation

#### 3.5.3.1 void OptionsWindow::events ( ) [private],[slot]

Handles events that need to be updated continuously.

```

00069 {
00070     auto ports = QSerialPortInfo::availablePorts();
00071
00072     if (ports.size() != _portSize) {
00073         _portSize = ports.size();
00074
00075         QString portC(ui->portC->currentData().toString());
00076         QString portS(ui->portS->currentData().toString());
```

```

00077
00078     int selC = 0, selS = 0;
00079
00080     ui->portC->clear();
00081     ui->portS->clear();
00082
00083     ui->portC->addItem("None", "");
00084     ui->portS->addItem("None", "");
00085
00086     for (int i = 0; i < ports.size(); ++i) {
00087         QString text(ports[i].portName());
00088         text += ": " + ports[i].description();
00089         ui->portC->addItem(text, ports[i].portName());
00090         ui->portS->addItem(text, ports[i].portName());
00091
00092         if (ports[i].portName() == portC) selC = i + 1;
00093         if (ports[i].portName() == portS) selS = i + 1;
00094     }
00095
00096     ui->portC->setCurrentIndex(selC);
00097     ui->portS->setCurrentIndex(selS);
00098 }
00099 }

```

### 3.5.3.2 void OptionsWindow::joystickChanged ( ) [slot]

To handle the change of a joystick.

```

00049 {
00050     // Clear all the items and write the new items
00051     ui->joySel->clear();
00052     ui->joySel->addItem("None", -1);
00053
00054     // Adding items and searching the current
00055     int pos = 0;
00056     QVector<XJoystick::Info> V(_joy.available());
00057     for (int i = 0; i < V.size(); ++i) {
00058         QString text(V[i].name);
00059         text += ": " + QString::number(V[i].ID);
00060         if (V[i].ID == _joy.current()) pos = i;
00061         ui->joySel->addItem(text, V[i].ID);
00062     }
00063     ui->joySel->setCurrentIndex(pos);
00064
00065     ui->joyN->setText(QString::number(V.size()));
00066 }

```

### 3.5.3.3 void OptionsWindow::on\_servoRefresh\_clicked ( ) [private],[slot]

Refreshes all the servos connected to the port.

```

00102 {
00103     dynamixel dxl;
00104     QString port;
00105     int baud;
00106     _servo->getServoPortInfo(port, baud);
00107
00108     dxl.initialize(port, baud);
00109 }
00110 }

```

### 3.5.3.4 void OptionsWindow::storeData ( )

Stores all data.

```

00041 {
00042     // Storing joystick data
00043     _joy.select(ui->joySel->currentData().toInt());
00044
00045 }
00046 }

```

### 3.5.4 Member Data Documentation

#### 3.5.4.1 XJoystick& OptionsWindow::\_joy [private]

Contains the Joystick to handle options.

#### 3.5.4.2 int OptionsWindow::\_portSize [private]

Contains the size of the ports.

#### 3.5.4.3 ServoThread\* OptionsWindow::\_servo [private]

Pointer to the servo thread class.

#### 3.5.4.4 QTimer OptionsWindow::\_timer [private]

Waits for a new COM port.

#### 3.5.4.5 Ui::OptionsWindow\* OptionsWindow::ui [private]

Contains the GUI.

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

- [optionswindow.h](#)
- [optionswindow.cpp](#)

## 3.6 ServoThread::Servo Struct Reference

Struct for the [AX12](#) servos.

```
#include <servothread.h>
```

### Public Member Functions

- [Servo](#) (int [ID](#)==1, double [load](#)==1, double [pos](#)==1)  
*Default constructor.*
- [Servo](#) (const [Servo](#) &s)  
*Copy constructor.*

### Public Attributes

- int [ID](#)  
*Contains the servo ID.*
- double [load](#)  
*Contains the servo load.*
- double [pos](#)  
*Contains the servo position.*

### 3.6.1 Detailed Description

Struct for the [AX12](#) servos.

### 3.6.2 Constructor & Destructor Documentation

#### 3.6.2.1 ServoThread::Servo::Servo ( int *ID* = -1, double *load* = -1, double *pos* = -1 ) [inline]

Default constructor.

```
00043         : ID(ID), load(load), pos(pos) {}
```

#### 3.6.2.2 ServoThread::Servo::Servo ( const Servo & s ) [inline]

Copy constructor.

```
00046 : ID(s.ID), load(s.load), pos(s.pos) {}
```

### 3.6.3 Member Data Documentation

#### 3.6.3.1 int ServoThread::Servo::ID

Contains the servo ID.

#### 3.6.3.2 double ServoThread::Servo::load

Contains the servo load.

#### 3.6.3.3 double ServoThread::Servo::pos

Contains the servo position.

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

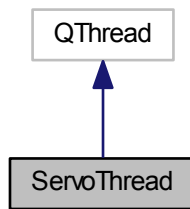
- [servothread.h](#)

## 3.7 ServoThread Class Reference

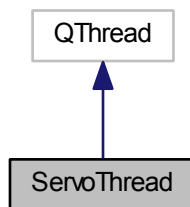
The [ServoThread](#)'s class handles the communication between the delta robot servos and the PC.

```
#include <servothread.h>
```

Inheritance diagram for ServoThread:



Collaboration diagram for ServoThread:



## Classes

- struct [Servo](#)  
*Struct for the [AX12](#) servos.*

## Public Types

- enum [Mode](#) { [controlled](#), [manual](#) }  
*Contains the working mode.*

## Public Member Functions

- [ServoThread](#) ()  
*Default constructor.*
- [~ServoThread](#) ()  
*Default destructor.*
- void [end](#) ()  
*Ends the execution.*
- void [load](#) (QString &file)  
*Loads the data from the selected file.*

- void `pause` ()  
*Pauses the execution.*
- int `getServoBaud` ()  
*Returns the current servo Baud rate.*
- QString `getServoPort` ()  
*Returns the current servo Port.*
- void `getServoPortInfo` (QString &port, int &baud)  
*Returns both servo Port and baud Rate.*
- void `getServosInfo` (QVector< Servo > &V)  
*Returns the servos info, with all its load and current position.*
- QVector< Servo > `getServosInfo` ()  
*Overloaded function to get the servo info.*
- QMutex \* `mutex` ()  
*Returns the mutex used in the thread.*
- void `setData` (QVector< float > &aV, QVector< bool > &butts)  
*Adds the loaded data.*
- void `setServoBaud` (unsigned int baud)  
*Sets the servos port baud rate.*
- void `setServoPort` (QString &port)  
*Sets the servos port.*
- void `setServoPortInfo` (QString &port, unsigned int baud)  
*Sets the servos port info, data and selected port.*
- void `setSID` (QVector< int > &V)  
*Sets the servos ID.*
- void `setStatusBar` (QStatusBar \*status)  
*Sets the status bar.*
- void `wakeUp` ()  
*Continues program's execution.*
- void `write` (QString &file)  
*Writes data to the selected directory.*

## Private Types

- enum `Version` { `v_1_0` }  
*Enum containing all the save file versions.*

## Private Member Functions

- void `run` ()  
*Used to create another thread.*
- void `setAngles` (double x0, double y0, double z0, double &theta1, double &theta2, double &theta3)  
*Used to calculate the servos angles.*
- double `singleAngle` (double x0, double y0, double z0)  
*Calculates the angle of one servo in the selected position.*

## Private Attributes

- const double [cos60](#) = 0.5  
*Contains the cosinus of 60.*
- const double [sin60](#) = sqrt(3)/2  
*Contains the sinus of 60.*
- const double [a](#) = 17.233  
*The arm length.*
- const double [b](#) = 22.648  
*The forearm length.*
- const double [L1](#) = 5.000  
*The base center lenght.*
- const double [L2](#) = 6.000  
*The platform center length.*
- QVector< float > [\\_axis](#)  
*Contains the axis value.*
- QVector< bool > [\\_buts](#)  
*Contains the buttons value.*
- int [\\_cBaud](#)  
*Contains the baud rate used to communicate with the clamp.*
- QWaitCondition [\\_cond](#)  
*To start and pause the thread.*
- QString [\\_cPort](#)  
*Contains the selected com port used to comunitate with the clamp.*
- bool [\\_dChanged](#)  
*True if the data changes.*
- bool [\\_end](#)  
*True when we must end executino.*
- [Mode \\_mod](#)  
*Contains the working mode.*
- QMutex [\\_mutex](#)  
*To prevent memory errors.*
- bool [\\_pause](#)  
*Pauses the execution of the thread.*
- int [\\_sBaud](#)  
*Contains the used baud rate to communicate with the servos.*
- QVector< [Servo](#) > [\\_servos](#)  
*Contains the servos information.*
- QString [\\_sPort](#)  
*Contains the selected com port used in the communication with servos.*
- bool [\\_sPortChanged](#)  
*True if the servos port changes.*
- QStatusBar \* [\\_statusBar](#)  
*Pointer to the window status Bar.*

### 3.7.1 Detailed Description

The [ServoThread](#)'s class handles the communication between the delta robot servos and the PC.



## 3.7.2 Member Enumeration Documentation

### 3.7.2.1 enum ServoThread::Mode

Contains the working mode.

Enumerator

***controlled***  
***manual***

```
00051     {
00052         controlled,
00053         manual
00054     };
```

### 3.7.2.2 enum ServoThread::Version [private]

Enum containing all the save file versions.

Enumerator

***v\_1\_0***

```
00028     {
00029         v_1_0
00030     };
```

## 3.7.3 Constructor & Destructor Documentation

### 3.7.3.1 ServoThread::ServoThread ( )

Default constructor.

```
00004         :
00005         _axis(XJoystick::AxisCount),
00006         _buts(XJoystick::ButtonCount),
00007         _cBaud(9600),
00008         _cPort("COM3"),
00009         _dChanged(false),
00010         _end(false),
00011         _mod(Mode::manual),
00012         _pause(true),
00013         _sBaud(1000000),
00014         _servos(3),
00015         _sPort("COM9"),
00016         _sPortChanged(false),
00017         _statusBar(NULL)
00018     {
00019
00020     }
```

### 3.7.3.2 ServoThread::~ServoThread ( )

Default destructor.

```
00023     {
00024         _mutex.lock();
00025         _end = true;
00026         _cond.wakeOne();
00027         _mutex.unlock();
00028
00029         wait();
00030     }
```

### 3.7.4 Member Function Documentation

#### 3.7.4.1 void ServoThread::end ( ) [inline]

Ends the execution.

```
00064     {
00065         _mutex.lock();
00066         _end = true;
00067         _cond.wakeOne();
00068         _mutex.unlock();
00069
00070         wait();
00071     }
```

#### 3.7.4.2 int ServoThread::getServoBaud ( ) [inline]

Returns the current servo Baud rate.

```
00086     {
00087         QMutexLocker mL(&_mutex);
00088         return _sBaud;
00089     }
```

#### 3.7.4.3 QString ServoThread::getServoPort ( ) [inline]

Returns the current servo Port.

```
00093     {
00094         QMutexLocker mL(&_mutex);
00095         return _sPort;
00096     }
```

#### 3.7.4.4 void ServoThread::getServoPortInfo ( QString & port, int & baud ) [inline]

Returns both servo Port and baud Rate.

```
00100     {
00101         _mutex.lock();
00102         baud = _sBaud;
00103         port = _sPort;
00104         _mutex.unlock();
00105     }
```

#### 3.7.4.5 void ServoThread::getServosInfo ( QVector< Servo > & V ) [inline]

Returns the servos info, with all its load and current position.

Parameters

V	Servo vector to store information
---	-----------------------------------

```
00111     {
00112         _mutex.lock();
00113         V = _servos;
00114         _mutex.unlock();
00115     }
```

**3.7.4.6** `QVector<Servo> ServoThread::getServosInfo ( )` [inline]

Overloaded function to get the servo info.

```
00119     {
00120         QMutexLocker mL(&_mutex);
00121         return _servos;
00122     }
```

**3.7.4.7** `void ServoThread::load ( QString & file )`

Loads the data from the selected file.

```
00033 {
00034     _mutex.lock();
00035     QFile f(file);
00036     f.open(QIODevice::ReadOnly);
00037     QDataStream df(&f);
00038
00039     int ver;
00040     df >> ver;
00041     if (ver == Version::v_l_0) {
00042         int n;
00043         df >> _cBaud >> _cPort >> _sBaud >> _sPort >> n;
00044
00045         _servos.resize(n);
00046         for (Servo &s : _servos) df >> s.ID;
00047         _dChanged = true;
00048     }
00049     else qWarning() << "Not a valid file";
00050     _mutex.unlock();
00051 }
```

**3.7.4.8** `QMutex* ServoThread::mutex ( )` [inline]

Returns the mutex used in the thread.

```
00125 { return &_mutex; }
```

**3.7.4.9** `void ServoThread::pause ( )` [inline]

Pauses the execution.

```
00078     {
00079         _mutex.lock();
00080         _pause = true;
00081         _mutex.unlock();
00082     }
```

**3.7.4.10** `void ServoThread::run ( )` [private]

Used to create another thread.

```
00079 {
00080     _mutex.lock();
00081     int sBaud = _sBaud;
00082     QString sPort = _sPort;
00083
00084     _mutex.unlock();
00085     dynamixel dxl(sPort, sBaud);
00086     QVector< AX12 > (_servos.size(), &dxl);
00087
00088     while (not _end) {
00089
```

```

00090         msleep(10);
00091         _mutex.lock();
00092         if (not _end and _pause) {
00093             dxl.terminate();
00094             _cond.wait(&_mutex);
00095             dxl.initialize(sPort, sBaud);
00096         }
00097         if (_dChanged) {
00098             if (sPort != _sPort) {
00099                 sPort = _sPort;
00100                 sBaud = _sBaud;
00101                 dxl.terminate();
00102                 dxl.initialize(sPort, sBaud);
00103             }
00104         }
00105         _dChanged = false;
00106         _mutex.unlock();
00107     }
00108
00109     dxl.terminate();
00110     exit(0);
00111 }

```

#### 3.7.4.11 void ServoThread::setAngles ( double x0, double y0, double z0, double & theta1, double & theta2, double & theta3 ) [private]

Used to calculate the servos angles.

```

00115 {
00116     double x1 = x0 + L2 - L1;
00117     double y1 = y0;
00118     double z1 = z0;
00119     theta1 = singleAngle(x1,y1,z1);
00120
00121     double x2 = z0*sin60 - x0*cos60 + L2 - L1;
00122     double y2 = y0;
00123     double z2 = -z0*cos60 - x0*sin60;
00124     theta2 = singleAngle(x2,y2,z2);
00125
00126     double x3 = -z0*sin60 - x0*cos60 + L2 - L1;
00127     double y3 = y0;
00128     double z3 = -z0*cos60 + x0*sin60;
00129     theta3 = singleAngle(x3,y3,z3);
00130 }

```

#### 3.7.4.12 void ServoThread::setData ( QVector< float > & aV, QVector< bool > & buts )

Adds the loaded data.

Parameters

<i>aV</i>	Contains the axis values
<i>buts</i>	Contains the buttons values

```

00054 {
00055     _mutex.lock();
00056     // Copying the joystick values
00057     _axis = aV;
00058     _buts = buts;
00059     _dChanged = true;
00060
00061     _mutex.unlock();
00062 }

```

#### 3.7.4.13 void ServoThread::setServoBaud ( unsigned int baud ) [inline]

Sets the servos port baud rate.

## Parameters

<i>baud</i>	Positive number containing the baud rate
-------------	--

```

00135     {
00136         _mutex.lock();
00137         _sBaud = baud;
00138         _mutex.unlock();
00139     }

```

**3.7.4.14** void ServoThread::setServoPort ( QString & *port* ) [inline]

Sets the servos port.

## Parameters

<i>port</i>	String containing the port name
-------------	---------------------------------

```

00144     {
00145         _mutex.lock();
00146         _sPort = port;
00147         _mutex.unlock();
00148     }

```

**3.7.4.15** void ServoThread::setServoPortInfo ( QString & *port*, unsigned int *baud* ) [inline]

Sets the servos port info, data and selected port.

## Parameters

<i>port</i>	String containing the selected port
<i>baud</i>	Contains the selected baud rate

```

00154     {
00155         _mutex.lock();
00156         _sPort = port;
00157         _sBaud = baud;
00158         _mutex.unlock();
00159     }

```

**3.7.4.16** void ServoThread::setSID ( QVector< int > & *V* ) [inline]

Sets the servos ID.

## Parameters

<i>V</i>	Vector containing all the servos ID
----------	-------------------------------------

```

00164     {
00165         _mutex.lock();
00166         if (V.size() != _servos.size()) _servos.resize(V.size());
00167         for (int i = 0; i < V.size(); ++i) _servos[i].ID = V[i];
00168         _dChanged = true;
00169         _mutex.unlock();
00170     }
00171 }

```

**3.7.4.17** void ServoThread::setStatusbar ( QStatusBar \* *status* ) [inline]

Sets the status bar.

## Parameters

<i>status</i>	Pointer to the status bar
---------------	---------------------------

```

00176     {
00177         _statusBar = status;
00178     }

```

### 3.7.4.18 double ServoThread::singleAngle ( double x0, double y0, double z0 ) [private]

Calculates the angle of one servo in the selected position.

```

00133 {
00134     double n = b * b - a * a - z0 * z0 - x0 * x0 - y0 * y0;
00135     double raiz = sqrt (n*n*y0*y0 - 4*(x0*x0 + y0*y0)*(-x0*x0*a*a + n*n/4));
00136
00137     if (x0 < 0) raiz *= -1;
00138     double y = (-n*y0 + raiz ) / (2*(x0*x0 + y0*y0));
00139
00140     int signe = 1;
00141     if ((b*b - (y0 + a)*(y0 + a)) < (x0*x0 + z0*z0) && x0 < 0) signe *= -1;
00142     double x = sqrt(a*a - y*y)*signe;
00143     return atan2 (y,x);
00144 }

```

### 3.7.4.19 void ServoThread::wakeUp ( ) [inline]

Continues program's execution.

```

00182     {
00183         _mutex.lock();
00184         _pause = false;
00185         _cond.wakeOne();
00186         _mutex.unlock();
00187     }

```

### 3.7.4.20 void ServoThread::write ( QString & file )

Writes data to the selected directory.

## Parameters

<i>file</i>	Path to the file
-------------	------------------

```

00065 {
00066     _mutex.lock();
00067     QFile f(file);
00068     f.open(QIODevice::WriteOnly);
00069     QDataStream df(&f);
00070
00071     df << int (Version::v_1_0) << _cBaud << _cPort << _sBaud <<
    _sPort
00072     << _servos.size();
00073     for (const Servo &s : _servos) df << s.ID;
00074
00075     _mutex.unlock();
00076 }

```

## 3.7.5 Member Data Documentation

### 3.7.5.1 QVector< float > ServoThread::\_axis [private]

Contains the axis value.

**3.7.5.2 QVector< bool > ServoThread::\_buts** [private]

Contains the buttons value.

**3.7.5.3 int ServoThread::\_cBaud** [private]

Contains the baud rate used to communicate with the clamp.

**3.7.5.4 QWaitCondition ServoThread::\_cond** [private]

To start and pause the thread.

**3.7.5.5 QString ServoThread::\_cPort** [private]

Contains the selected com port used to comunicate with the clamp.

**3.7.5.6 bool ServoThread::\_dChanged** [private]

True if the data changes.

**3.7.5.7 bool ServoThread::\_end** [private]

True when we must end executino.

**3.7.5.8 Mode ServoThread::\_mod** [private]

Contains the working mode.

**3.7.5.9 QMutex ServoThread::\_mutex** [private]

To prevent memory errors.

**3.7.5.10 bool ServoThread::\_pause** [private]

Pauses the execution of the thread.

**3.7.5.11 int ServoThread::\_sBaud** [private]

Contains the used baud rate to communicate with the servos.

**3.7.5.12 QVector< Servo > ServoThread::\_servos** [private]

Contains the servos information.

**3.7.5.13 QString ServoThread::\_sPort** [private]

Contains the selected com port used in the communication with servos.

**3.7.5.14** `bool ServoThread::_sPortChanged` [private]

True if the servos port changes.

**3.7.5.15** `QStatusBar* ServoThread::_statusBar` [private]

Pointer to the window status Bar.

**3.7.5.16** `const double ServoThread::a = 17.233` [private]

The arm length.

**3.7.5.17** `const double ServoThread::b = 22.648` [private]

The forearm length.

**3.7.5.18** `const double ServoThread::cos60 = 0.5` [private]

Contains the cosinus of 60.

**3.7.5.19** `const double ServoThread::L1 = 5.000` [private]

The base center lenght.

**3.7.5.20** `const double ServoThread::L2 = 6.000` [private]

The platform center length.

**3.7.5.21** `const double ServoThread::sin60 = sqrt(3)/2` [private]

Contains the sinus of 60.

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

- [servothread.h](#)
- [servothread.cpp](#)



## Chapter 4

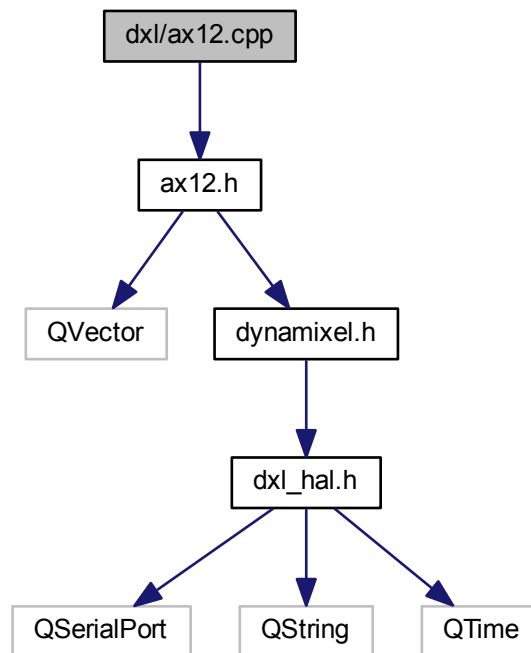
# File Documentation

### 4.1 dxl/ax12.cpp File Reference

Contains the [AX12](#) class implementation.

```
#include "ax12.h"
```

Include dependency graph for ax12.cpp:



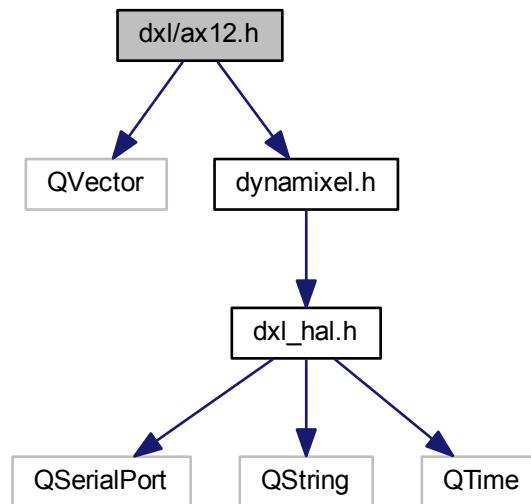
#### 4.1.1 Detailed Description

Contains the [AX12](#) class implementation.

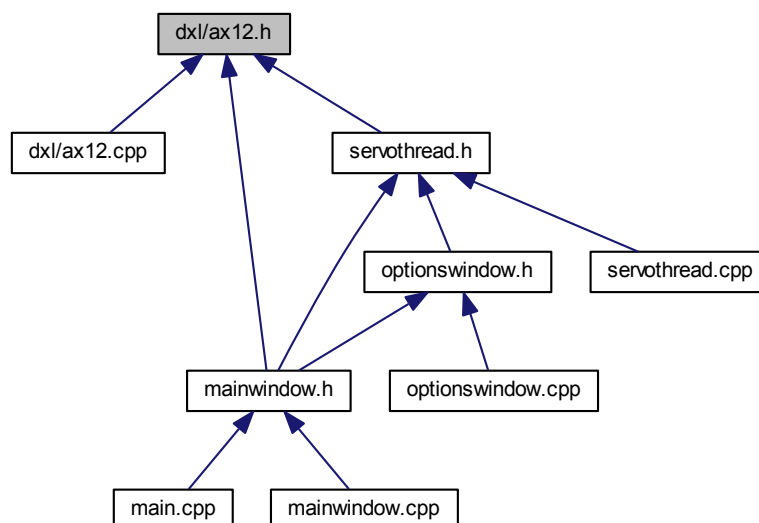
## 4.2 dxl/ax12.h File Reference

Contains the [AX12](#) class declaration.

```
#include <QVector>
#include "dynamixel.h"
Include dependency graph for ax12.h:
```



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



## Classes

- class [AX12](#)

The [AX12](#) class is used to control AX-12 motors from Dynamixel.

## Macros

- `#define M\_PI 3.14159265358979323846`

### 4.2.1 Detailed Description

Contains the [AX12](#) class declaration.

### 4.2.2 Macro Definition Documentation

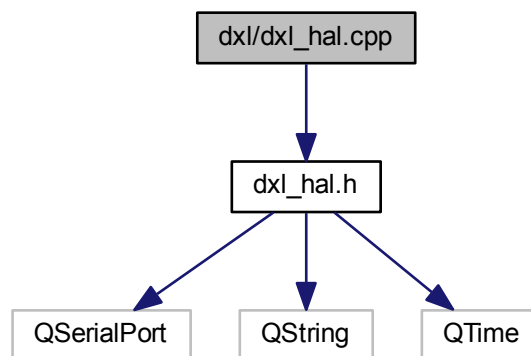
4.2.2.1 `#define M\_PI 3.14159265358979323846`

## 4.3 dxl/dxl\_hal.cpp File Reference

Contains the Dynamixel SDK platform dependent header source.

```
#include "dxl_hal.h"
```

Include dependency graph for dxl\_hal.cpp:



### 4.3.1 Detailed Description

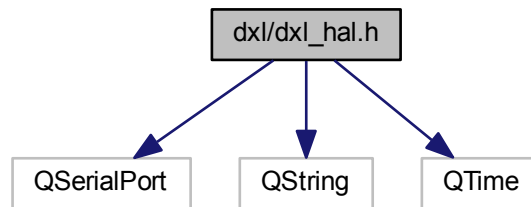
Contains the Dynamixel SDK platform dependent header source.

## 4.4 dxl/dxl\_hal.h File Reference

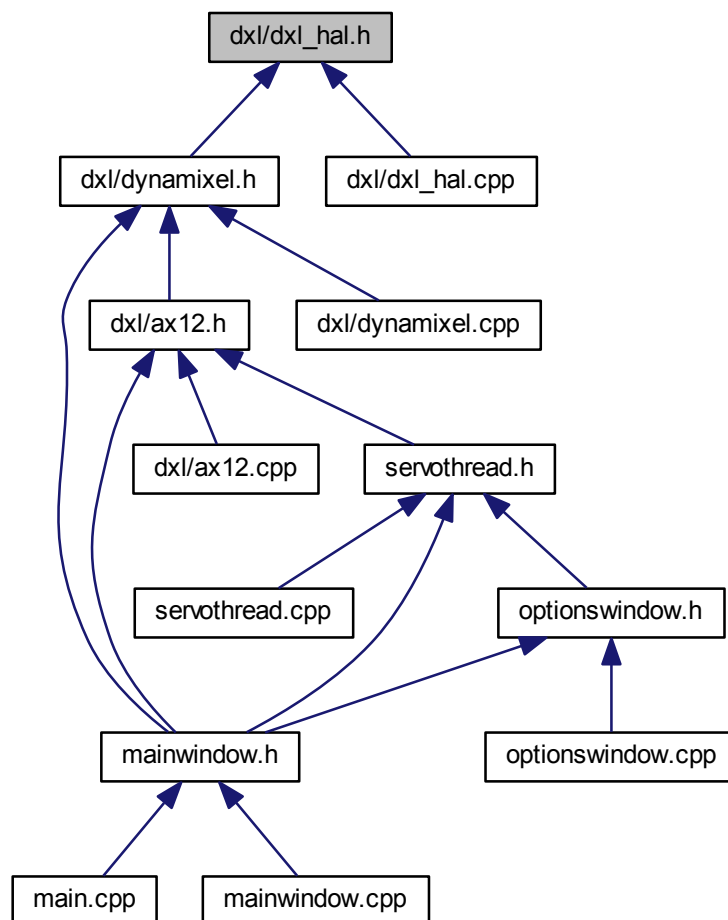
Contains the Dynamixel SDK platform dependent header declaration.

```
#include <QSerialPort>
#include <QString>
#include <QTime>
```

Include dependency graph for dxl\_hal.h:



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



## Classes

- class [dxl\\_hal](#)  
*Dynamixel SDK platform dependent.*

## Macros

- `#define MAXNUM_TXPACKET (10000)`
- `#define MAXNUM_RXPACKET (10000)`

### 4.4.1 Detailed Description

Contains the Dynamixel SDK platform dependent header declaration.

### 4.4.2 Macro Definition Documentation

4.4.2.1 `#define MAXNUM_RXPACKET (10000)`

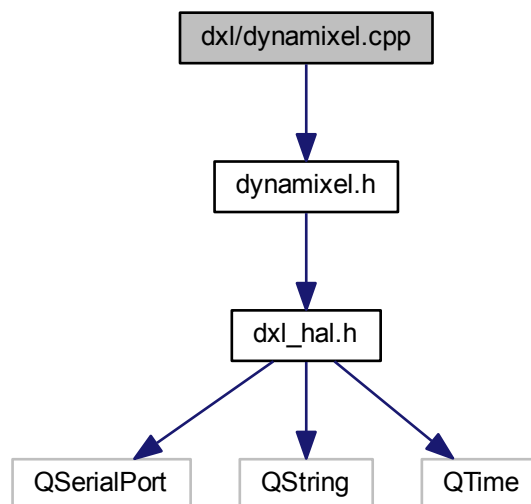
4.4.2.2 `#define MAXNUM_TXPACKET (10000)`

## 4.5 dxl/dynamixel.cpp File Reference

Contains the dynamixel and dynamixel2 classes implementation.

```
#include "dynamixel.h"
```

Include dependency graph for dynamixel.cpp:



## Macros

- `#define LATENCY_TIME (16)`

- `#define PING_STATUS_LENGTH (14)`

#### 4.5.1 Detailed Description

Contains the dynamixel and dynamixel2 classes implementation.

#### 4.5.2 Macro Definition Documentation

4.5.2.1 `#define LATENCY_TIME (16)`

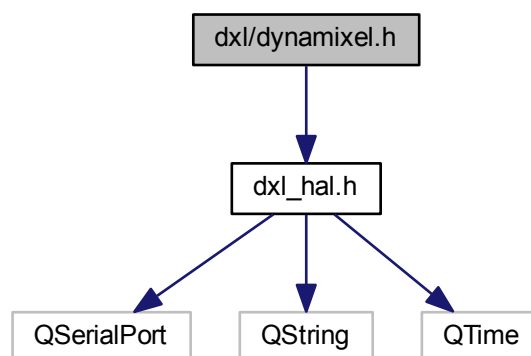
4.5.2.2 `#define PING_STATUS_LENGTH (14)`

### 4.6 dxl/dynamixel.h File Reference

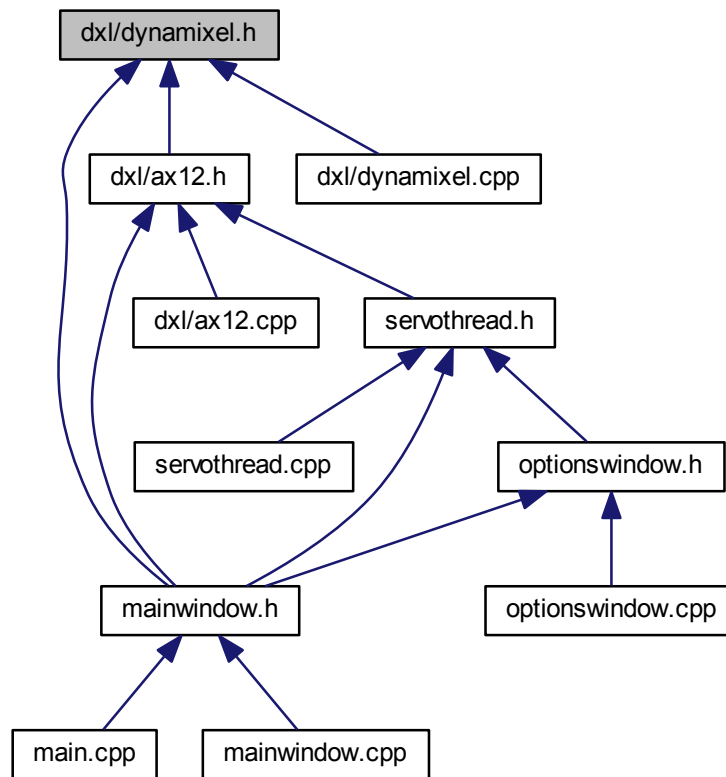
Contains the dynamixel and dynamixel2 classes declaration.

```
#include "dxl_hal.h"
```

Include dependency graph for dynamixel.h:



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



## Classes

- class [dynamixel](#)  
*Dynamixel 1.0 protocol class.*

## Macros

- #define [MAX\\_ID](#) (252)
- #define [BROADCAST\\_ID](#) (254)
- #define [COMM\\_TXSUCCESS](#) (0)
- #define [COMM\\_RXSUCCESS](#) (1)
- #define [COMM\\_TXFAIL](#) (2)
- #define [COMM\\_RXFAIL](#) (3)
- #define [COMM\\_TXERROR](#) (4)
- #define [COMM\\_RXWAITING](#) (5)
- #define [COMM\\_RXTIMEOUT](#) (6)
- #define [COMM\\_RXCORRUPT](#) (7)
- #define [ERRBIT\\_ALERT](#) (128)
- #define [ERR\\_RESULT\\_FAIL](#) (1)
- #define [ERR\\_INSTRUCTION](#) (2)
- #define [ERR\\_CRC](#) (3)

- `#define ERR_DATA_RANGE` (4)
- `#define ERR_DATA_LENGTH` (5)
- `#define ERR_DATA_LIMIT` (6)
- `#define ERR_ACCESS` (7)
- `#define PRT1_PKT_ID` (2)
- `#define PRT1_PKT_LENGTH` (3)
- `#define PRT1_PKT_INSTRUCTION` (4)
- `#define PRT1_PKT_ERRBIT` (4)
- `#define PRT1_PKT_PARAMETER0` (5)
- `#define INST_PING` (1)
- `#define INST_READ` (2)
- `#define INST_WRITE` (3)
- `#define INST_REG_WRITE` (4)
- `#define INST_ACTION` (5)
- `#define INST_RESET` (6)
- `#define INST_SYNC_WRITE` (131)
- `#define INST_BULK_READ` (146)
- `#define PING_INFO_MODEL_NUM` (1)
- `#define PING_INFO_FIRM_VER` (2)
- `#define MAKEWORD(a, b) (((unsigned short)(((unsigned char)(((unsigned long)(a)) & 0xff)) | ((unsigned short)((unsigned char)(((unsigned long)(b)) & 0xff))) << 8)))`
- `#define MAKEDWORD(a, b) (((unsigned int)(((unsigned short)(((unsigned long)(a)) & 0xffff)) | ((unsigned int)((unsigned short)(((unsigned long)(b)) & 0xffff))) << 16)))`
- `#define LOWORD(l) (((unsigned short)(((unsigned long)(l)) & 0xffff)))`
- `#define HIWORD(l) (((unsigned short)(((unsigned long)(l)) >> 16) & 0xffff))`
- `#define LOBYTE(w) (((unsigned char)(((unsigned long)(w)) & 0xff)))`
- `#define HIBYTE(w) (((unsigned char)(((unsigned long)(w)) >> 8) & 0xff))`

#### 4.6.1 Detailed Description

Contains the dynamixel and dynamixel2 classes declaration.

#### 4.6.2 Macro Definition Documentation

4.6.2.1 `#define BROADCAST_ID` (254)

4.6.2.2 `#define COMM_RXCORRUPT` (7)

4.6.2.3 `#define COMM_RXFAIL` (3)

4.6.2.4 `#define COMM_RXSUCCESS` (1)

4.6.2.5 `#define COMM_RXTIMEOUT` (6)

4.6.2.6 `#define COMM_RXWAITING` (5)

4.6.2.7 `#define COMM_TXERROR` (4)

4.6.2.8 `#define COMM_TXFAIL` (2)

4.6.2.9 `#define COMM_TXSUCCESS` (0)

4.6.2.10 `#define ERR_ACCESS` (7)



- 4.6.2.11 `#define ERR_CRC (3)`
- 4.6.2.12 `#define ERR_DATA_LENGTH (5)`
- 4.6.2.13 `#define ERR_DATA_LIMIT (6)`
- 4.6.2.14 `#define ERR_DATA_RANGE (4)`
- 4.6.2.15 `#define ERR_INSTRUCTION (2)`
- 4.6.2.16 `#define ERR_RESULT_FAIL (1)`
- 4.6.2.17 `#define ERRBIT_ALERT (128)`
- 4.6.2.18 `#define HIBYTE( w ) (((unsigned char)((((unsigned long)(w)) >> 8) & 0xff))`
- 4.6.2.19 `#define HIWORD( l ) (((unsigned short)((((unsigned long)(l)) >> 16) & 0xffff))`
- 4.6.2.20 `#define INST_ACTION (5)`
- 4.6.2.21 `#define INST_BULK_READ (146)`
- 4.6.2.22 `#define INST_PING (1)`
- 4.6.2.23 `#define INST_READ (2)`
- 4.6.2.24 `#define INST_REG_WRITE (4)`
- 4.6.2.25 `#define INST_RESET (6)`
- 4.6.2.26 `#define INST_SYNC_WRITE (131)`
- 4.6.2.27 `#define INST_WRITE (3)`
- 4.6.2.28 `#define LOBYTE( w ) ((unsigned char)((((unsigned long)(w)) & 0xff))`
- 4.6.2.29 `#define LOWORD( l ) ((unsigned short)((((unsigned long)(l)) & 0xffff))`
- 4.6.2.30 `#define MAKEDWORD( a, b ) (((unsigned int)((((unsigned short)((((unsigned long)(a)) & 0xffff)) | ((unsigned int)((unsigned short)((((unsigned long)(b)) & 0xffff))) << 16))`
- 4.6.2.31 `#define MAKEWORD( a, b ) (((unsigned short)((((unsigned char)((((unsigned long)(a)) & 0xff)) | ((unsigned short)((unsigned char)((((unsigned long)(b)) & 0xff))) << 8))`
- 4.6.2.32 `#define MAX_ID (252)`
- 4.6.2.33 `#define PING_INFO_FIRM_VER (2)`
- 4.6.2.34 `#define PING_INFO_MODEL_NUM (1)`
- 4.6.2.35 `#define PRT1_PKT_ERRBIT (4)`
- 4.6.2.36 `#define PRT1_PKT_ID (2)`
- 4.6.2.37 `#define PRT1_PKT_INSTRUCTION (4)`

4.6.2.38 `#define PRT1_PKT_LENGTH (3)`

4.6.2.39 `#define PRT1_PKT_PARAMETER0 (5)`

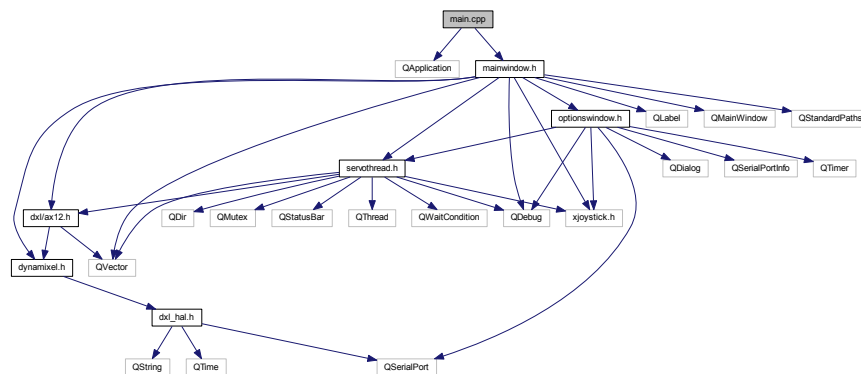
## 4.7 main.cpp File Reference

Contains the Main of the program.

```
#include <QApplication>
```

```
#include "mainwindow.h"
```

Include dependency graph for main.cpp:



## Functions

- `int main (int argc, char *argv[])`

### 4.7.1 Detailed Description

Contains the Main of the program.

### 4.7.2 Function Documentation

#### 4.7.2.1 `int main ( int argc, char * argv[] )`

```

00009 {
00010     QApplication a(argc, argv);
00011     MainWindow w;
00012     w.show();
00013     return a.exec();
00014 }
```

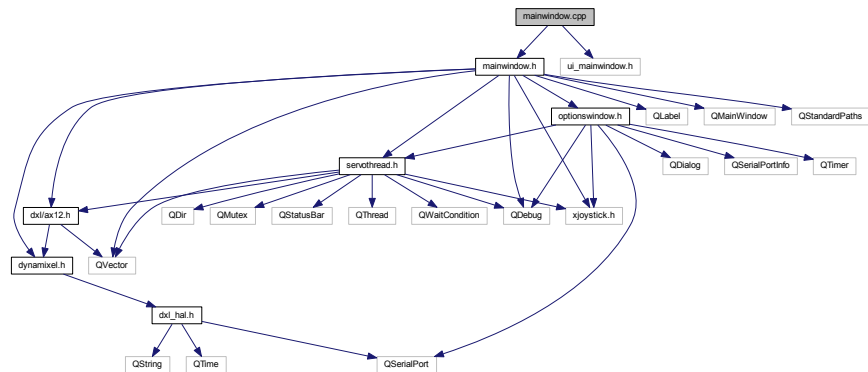
## 4.8 mainwindow.cpp File Reference

Contains the `MainWindow` class implementation.

```
#include "mainwindow.h"
```

```
#include "ui_mainwindow.h"
```

Include dependency graph for mainwindow.cpp:



#### 4.8.1 Detailed Description

Contains the [MainWindow](#) class implementation.

## 4.9 mainwindow.h File Reference

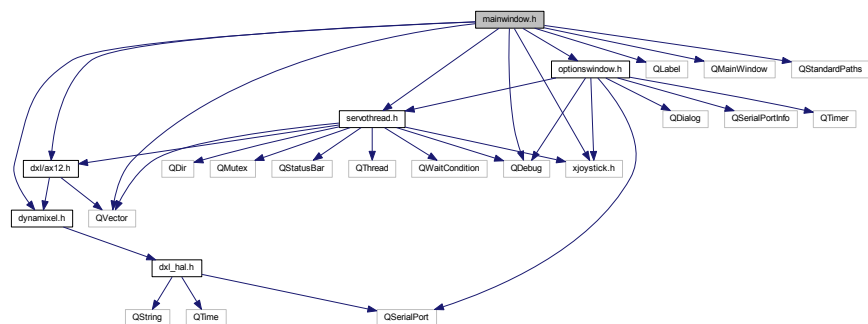
Contains the [MainWindow](#) class declaration.

```

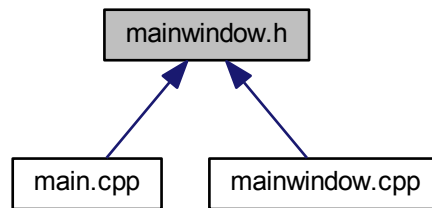
#include <QDebug>
#include <QLabel>
#include <QMainWindow>
#include <QVector>
#include <QStandardPaths>
#include <xjoystick.h>
#include "dxl/ax12.h"
#include "dxl/dynamixel.h"
#include "optionswindow.h"
#include "servothread.h"

```

Include dependency graph for mainwindow.h:



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



## Classes

- class [MainWindow](#)  
*Contains all the windows and other classes.*

## Namespaces

- [Ui](#)  
*Namespace to work with a User Interface Qt Form.*

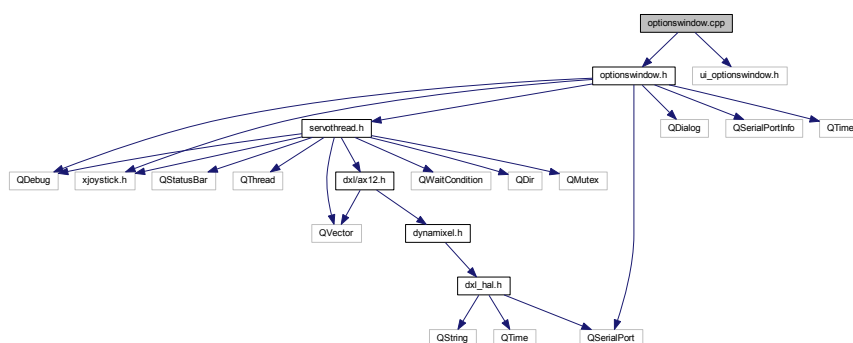
### 4.9.1 Detailed Description

Contains the [MainWindow](#) class declaration.

## 4.10 optionswindow.cpp File Reference

Contains the [OptionsWindow](#) class implementation.

```
#include "optionswindow.h"
#include "ui_optionswindow.h"
Include dependency graph for optionswindow.cpp:
```



#### 4.10.1 Detailed Description

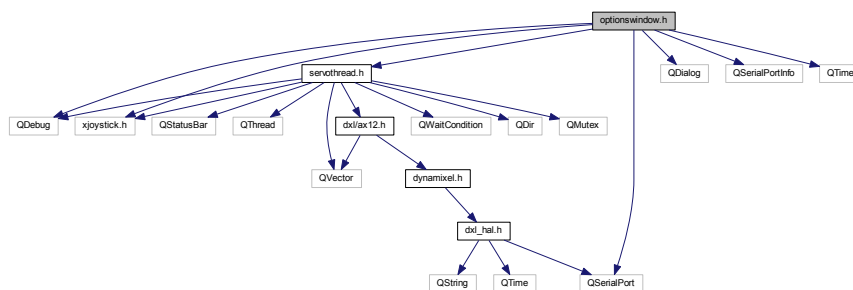
Contains the `OptionsWindow` class implementation.

## 4.11 optionswindow.h File Reference

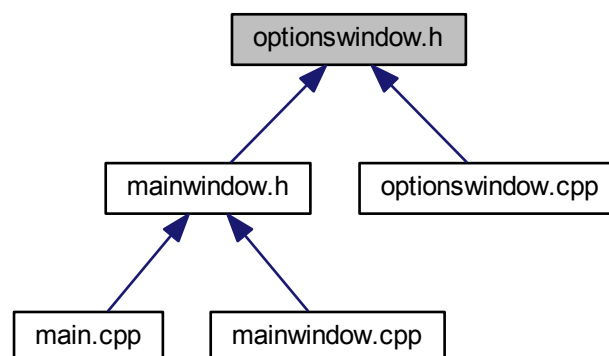
Contains the [OptionsWindow](#) class declaration.

```
#include <QDebug>
#include <QDialog>
#include <QSerialPort>
#include <QSerialPortInfo>
#include <QTimer>
#include <xjoystick.h>
#include "servothread.h"
```

Include dependency graph for optionswindow.h:



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



## Classes

- class `OptionsWindow`

*Class used to handle a Window to set the options.*

## Namespaces

- [Ui](#)

*Namespace to work with a User Interface Qt Form.*

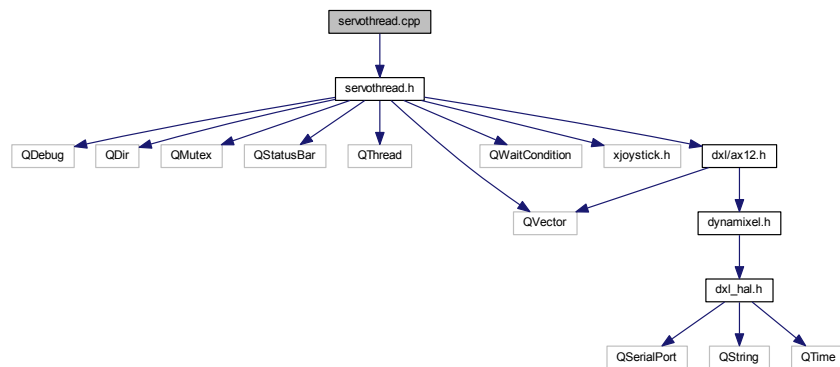
### 4.11.1 Detailed Description

Contains the [OptionsWindow](#) class declaration.

## 4.12 servothread.cpp File Reference

```
#include "servothread.h"
```

Include dependency graph for servothread.cpp:



### 4.13 servothread.h File Reference

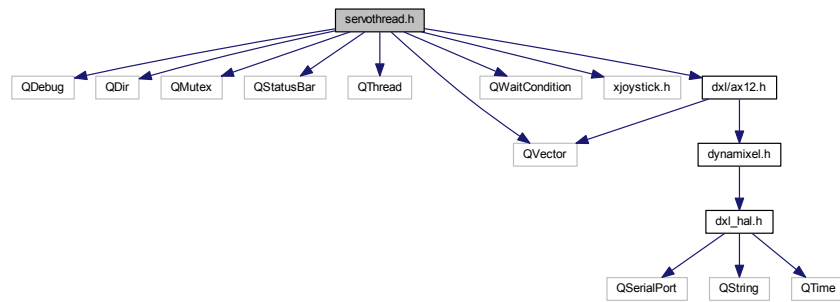
Contains the [ServoThread](#) class implementation.

```

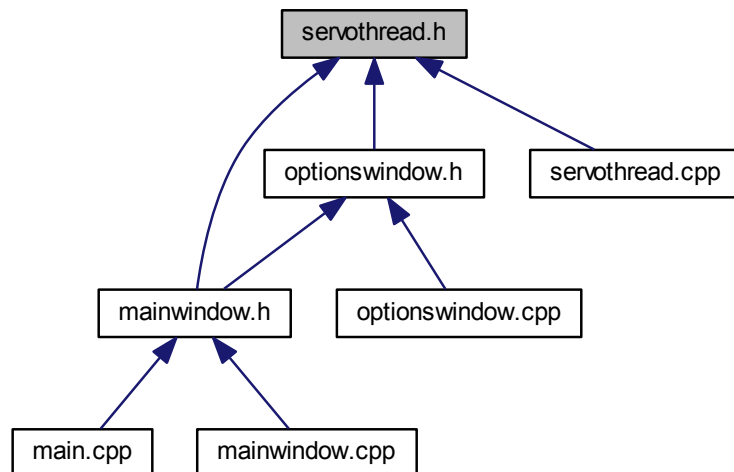
#include <QDebug>
#include <QDir>
#include <QMutex>
#include <QStatusBar>
#include <QThread>
#include <QVector>
#include <QWaitCondition>
#include <xjoystick.h>
#include "dxl/ax12.h"

```

Include dependency graph for servothread.h:



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



## Classes

- class [ServoThread](#)  
The *ServoThread*'s class handles the communication between the delta robot servos and the PC.
- struct [ServoThread::Servo](#)  
Struct for the *AX12* servos.

### 4.13.1 Detailed Description

Contains the [ServoThread](#) class implementation.

Contains the [ServoThread](#) class declaration.





# Index

- AX12, [11](#)
- MainWindow, [29](#)
  - ServoThread, [44](#)
- MainWindow, [29](#)
- MainWindow, [29](#)
  - ServoThread, [44](#)
- MainWindow, [29](#)
- ServoThread, [45](#)
- ServoThread, [45](#)
- ServoThread, [45](#)
- MainWindow, [29](#)
- ServoThread, [45](#)
- MainWindow, [29](#)
- MainWindow, [29](#)
- MainWindow, [29](#)
  - OptionsWindow, [34](#)
- ServoThread, [45](#)
- AX12, [11](#)
- ServoThread, [45](#)
- dxl\_hal, [13](#)
- ServoThread, [45](#)
- OptionsWindow, [34](#)
- AX12, [11](#)
- AX12, [11](#)
- ServoThread, [45](#)
- dxl\_hal, [14](#)
- ServoThread, [45](#)
- OptionsWindow, [34](#)
- ServoThread, [45](#)
- ServoThread, [46](#)
- dxl\_hal, [14](#)
- dxl\_hal, [14](#)
- MainWindow, [29](#)
  - OptionsWindow, [34](#)
- ~AX12
  - AX12, [8](#)
- ~MainWindow
  - MainWindow, [27](#)
- ~OptionsWindow
  - OptionsWindow, [32](#)
- ~ServoThread
  - ServoThread, [39](#)
- a
  - ServoThread, [46](#)
- aSCount
  - MainWindow, [30](#)
- AX12, [5](#)
  - \_ID, [11](#)
  - \_mode, [11](#)
  - \_rads, [11](#)
  - ~AX12, [8](#)
  - AX12, [8](#)
  - AlarmLED, [8](#)
  - AlarmShutdown, [8](#)
  - BaudRate, [7](#)
  - CCWAngleLimit, [8](#)
  - CCWComplianceMargin, [7](#)
  - CCWComplianceSlope, [7](#)
  - CWAngleLimit, [8](#)
  - CWComplianceMargin, [7](#)
  - CWComplianceSlope, [7](#)
  - connectedID, [9](#)

- dxl, 11
- getCurrentLoad, 9
- getCurrentPos, 9
- getCurrentSpeed, 9
- getCurrentTemp, 9
- getCurrentVoltage, 9
- getID, 10
- GoalPosition, 7
- HighestLimitTemp, 8
- HighestLimitVoltage, 8
- ID, 7
- LED, 7
- Lock, 7
- LowestLimitVoltage, 8
- MaxTorque, 8
- ModelNumber, 7
- Moving, 7
- MovingSpeed, 7
- PresentLoad, 7
- PresentPosition, 7
- PresentSpeed, 7
- PresentTemperature, 7
- PresentVoltage, 7
- Punch, 7
- RAM, 7
- ROM, 7
- Registered, 7
- ReturnDelayTime, 8
- setGoalPosition, 10
- setID, 10
- setJointMode, 10
- setMinMax, 10
- setSpeed, 11
- StatusReturnLevel, 8
- TorqueEnable, 7
- TorqueLimit, 7
- VersionFirmware, 7
- AlarmLED
  - AX12, 8
- AlarmShutdown
  - AX12, 8
- ax12.h
  - M\_PI, 49
- b
  - ServoThread, 46
- BROADCAST\_ID
  - dynamixel.h, 54
- BaudRate
  - AX12, 7
- CCWAngleLimit
  - AX12, 8
- CCWComplianceMargin
  - AX12, 7
- CCWComplianceSlope
  - AX12, 7
- COMM\_RXCORRUPT
  - dynamixel.h, 54
- COMM\_RXFAIL
  - dynamixel.h, 54
- COMM\_RXSUCCESS
  - dynamixel.h, 54
- COMM\_RXTIMEOUT
  - dynamixel.h, 54
- COMM\_RXWAITING
  - dynamixel.h, 54
- COMM\_TXERROR
  - dynamixel.h, 54
- COMM\_TXFAIL
  - dynamixel.h, 54
- COMM\_TXSUCCESS
  - dynamixel.h, 54
- CWAngleLimit
  - AX12, 8
- CWComplianceMargin
  - AX12, 7
- CWComplianceSlope
  - AX12, 7
- change\_baudrate
  - dxl\_hal, 12
  - dynamixel, 16
- clear
  - dxl\_hal, 12
- close
  - dxl\_hal, 12
- connectedID
  - AX12, 9
- controlled
  - ServoThread, 39
- cos60
  - ServoThread, 46
- dH
  - dynamixel, 24
- dxl
  - AX12, 11
  - dxl/ax12.cpp, 47
  - dxl/ax12.h, 48
  - dxl/dxl\_hal.cpp, 49
  - dxl/dxl\_hal.h, 49
  - dxl/dynamixel.cpp, 51
  - dxl/dynamixel.h, 52
  - dxl\_hal, 12
    - \_open, 13
    - \_serial, 14
    - \_time, 14
    - \_timed, 14
    - change\_baudrate, 12
    - clear, 12
    - close, 12
    - get\_curr\_time, 12
    - isOpen, 13
    - open, 13
    - read, 13
    - write, 13
  - dxl\_hal.h
    - MAXNUM\_RXPACKET, 51

- MAXNUM\_TXPACKET, 51
- dynamixel, 14
  - change\_baudrate, 16
  - dH, 24
  - dynamixel, 16
  - gbCommStatus, 24
  - gbInstructionPacket, 24
  - gbRxGetLength, 24
  - gbRxPacketLength, 24
  - gbStatusPacket, 24
  - gdByteTransTime, 24
  - gdPacketStartTime, 24
  - gdRcvWaitTime, 24
  - get\_comm\_result, 16
  - get\_packet\_time, 17
  - get\_rxpacket\_error, 17
  - get\_rxpacket\_error\_byte, 17
  - get\_rxpacket\_length, 17
  - get\_rxpacket\_parameter, 17
  - giBusUsing, 24
  - initialize, 17
  - is\_packet\_timeout, 18
  - isOpen, 18
  - ping, 18
  - read\_byte, 18
  - read\_word, 19
  - rx\_packet, 19
  - set\_packet\_timeout, 20
  - set\_packet\_timeout\_ms, 21
  - set\_txpacket\_id, 21
  - set\_txpacket\_instruction, 21
  - set\_txpacket\_length, 21
  - set\_txpacket\_parameter, 21
  - terminate, 21
  - tx\_packet, 22
  - txrx\_packet, 23
  - write\_byte, 23
  - write\_word, 23
- dynamixel.cpp
  - LATENCY\_TIME, 52
  - PING\_STATUS\_LENGTH, 52
- dynamixel.h
  - BROADCAST\_ID, 54
  - COMM\_RXCORRUPT, 54
  - COMM\_RXFAIL, 54
  - COMM\_RXSUCCESS, 54
  - COMM\_RXTIMEOUT, 54
  - COMM\_RXWAITING, 54
  - COMM\_TXERROR, 54
  - COMM\_TXFAIL, 54
  - COMM\_TXSUCCESS, 54
  - ERR\_ACCESS, 54
  - ERR\_CRC, 54
  - ERR\_DATA\_LENGTH, 55
  - ERR\_DATA\_LIMIT, 55
  - ERR\_DATA\_RANGE, 55
  - ERR\_INSTRUCTION, 55
  - ERR\_RESULT\_FAIL, 55
  - ERRBIT\_ALERT, 55
  - HIBYTE, 55
  - HIWORD, 55
  - INST\_ACTION, 55
  - INST\_BULK\_READ, 55
  - INST\_PING, 55
  - INST\_READ, 55
  - INST\_REG\_WRITE, 55
  - INST\_RESET, 55
  - INST\_SYNC\_WRITE, 55
  - INST\_WRITE, 55
  - LOBYTE, 55
  - LOWORD, 55
  - MAKEDWORD, 55
  - MAKEWORD, 55
  - MAX\_ID, 55
  - PING\_INFO\_FIRM\_VER, 55
  - PING\_INFO\_MODEL\_NUM, 55
  - PRT1\_PKT\_ERRBIT, 55
  - PRT1\_PKT\_ID, 55
  - PRT1\_PKT\_INSTRUCTION, 55
  - PRT1\_PKT\_LENGTH, 55
  - PRT1\_PKT\_PARAMETER0, 56
- ERR\_ACCESS
  - dynamixel.h, 54
- ERR\_CRC
  - dynamixel.h, 54
- ERR\_DATA\_LENGTH
  - dynamixel.h, 55
- ERR\_DATA\_LIMIT
  - dynamixel.h, 55
- ERR\_DATA\_RANGE
  - dynamixel.h, 55
- ERR\_INSTRUCTION
  - dynamixel.h, 55
- ERR\_RESULT\_FAIL
  - dynamixel.h, 55
- ERRBIT\_ALERT
  - dynamixel.h, 55
- end
  - ServoThread, 40
- events
  - OptionsWindow, 32
- gbCommStatus
  - dynamixel, 24
- gbInstructionPacket
  - dynamixel, 24
- gbRxGetLength
  - dynamixel, 24
- gbRxPacketLength
  - dynamixel, 24
- gbStatusPacket
  - dynamixel, 24
- gdByteTransTime
  - dynamixel, 24
- gdPacketStartTime
  - dynamixel, 24

- gdRcvWaitTime
  - dynamixel, [24](#)
- get\_comm\_result
  - dynamixel, [16](#)
- get\_curr\_time
  - dxl\_hal, [12](#)
- get\_packet\_time
  - dynamixel, [17](#)
- get\_rxpacket\_error
  - dynamixel, [17](#)
- get\_rxpacket\_error\_byte
  - dynamixel, [17](#)
- get\_rxpacket\_length
  - dynamixel, [17](#)
- get\_rxpacket\_parameter
  - dynamixel, [17](#)
- getCurrentLoad
  - AX12, [9](#)
- getCurrentPos
  - AX12, [9](#)
- getCurrentSpeed
  - AX12, [9](#)
- getCurrentTemp
  - AX12, [9](#)
- getCurrentVoltage
  - AX12, [9](#)
- getID
  - AX12, [10](#)
- getServoBaud
  - ServoThread, [40](#)
- getServoPort
  - ServoThread, [40](#)
- getServoPortInfo
  - ServoThread, [40](#)
- getServosInfo
  - ServoThread, [40](#)
- giBusUsing
  - dynamixel, [24](#)
- GoalPosition
  - AX12, [7](#)
- HIBYTE
  - dynamixel.h, [55](#)
- HIWORD
  - dynamixel.h, [55](#)
- HighestLimitTemp
  - AX12, [8](#)
- HighestLimitVoltage
  - AX12, [8](#)
- ID
  - AX12, [7](#)
  - ServoThread::Servo, [35](#)
- INST\_ACTION
  - dynamixel.h, [55](#)
- INST\_BULK\_READ
  - dynamixel.h, [55](#)
- INST\_PING
  - dynamixel.h, [55](#)
- INST\_READ
  - dynamixel.h, [55](#)
- INST\_REG\_WRITE
  - dynamixel.h, [55](#)
- INST\_RESET
  - dynamixel.h, [55](#)
- INST\_SYNC\_WRITE
  - dynamixel.h, [55](#)
- INST\_WRITE
  - dynamixel.h, [55](#)
- initialize
  - dynamixel, [17](#)
- is\_packet\_timeout
  - dynamixel, [18](#)
- isOpen
  - dxl\_hal, [13](#)
  - dynamixel, [18](#)
- joyChanged
  - MainWindow, [27](#)
- joystickChanged
  - MainWindow, [28](#)
  - OptionsWindow, [33](#)
- L1
  - ServoThread, [46](#)
- L2
  - ServoThread, [46](#)
- LATENCY\_TIME
  - dynamixel.cpp, [52](#)
- LED
  - AX12, [7](#)
- LOBYTE
  - dynamixel.h, [55](#)
- LOWORD
  - dynamixel.h, [55](#)
- load
  - ServoThread, [41](#)
  - ServoThread::Servo, [35](#)
- Lock
  - AX12, [7](#)
- LowestLimitVoltage
  - AX12, [8](#)
- M\_PI
  - ax12.h, [49](#)
- MAKEDWORD
  - dynamixel.h, [55](#)
- MAKEWORD
  - dynamixel.h, [55](#)
- MAX\_ID
  - dynamixel.h, [55](#)
- MAXNUM\_RXPACKET
  - dxl\_hal.h, [51](#)
- MAXNUM\_TXPACKET
  - dxl\_hal.h, [51](#)
- main
  - main.cpp, [56](#)
  - main.cpp, [56](#)

- main, 56
- MainWindow, 25
  - \_axis, 29
  - \_axisV, 29
  - \_buts, 29
  - \_butsV, 29
  - \_dataP, 29
  - \_jAxisX, 29
  - \_jAxisY, 29
  - \_jAxisZ, 29
  - \_joy, 29
  - \_sT, 29
  - \_timer, 29
  - ~MainWindow, 27
  - aSCount, 30
  - joyChanged, 27
  - joystickChanged, 28
  - MainWindow, 27
  - on\_actionOptions\_triggered, 28
  - on\_start\_clicked, 28
  - sCount, 30
  - ui, 30
  - update, 28
- mainwindow.cpp, 56
- mainwindow.h, 57
- manual
  - ServoThread, 39
- MaxTorque
  - AX12, 8
- Mode
  - ServoThread, 39
- ModelNumber
  - AX12, 7
- Moving
  - AX12, 7
- MovingSpeed
  - AX12, 7
- mutex
  - ServoThread, 41
- on\_actionOptions\_triggered
  - MainWindow, 28
- on\_servoRefresh\_clicked
  - OptionsWindow, 33
- on\_start\_clicked
  - MainWindow, 28
- open
  - dxl\_hal, 13
- OptionsWindow, 30
  - \_joy, 34
  - \_portSize, 34
  - \_servo, 34
  - \_timer, 34
  - ~OptionsWindow, 32
  - events, 32
  - joystickChanged, 33
  - on\_servoRefresh\_clicked, 33
  - OptionsWindow, 32
  - storeData, 33
- ui, 34
- optionswindow.cpp, 58
- optionswindow.h, 59
- PING\_INFO\_FIRM\_VER
  - dynamixel.h, 55
- PING\_INFO\_MODEL\_NUM
  - dynamixel.h, 55
- PING\_STATUS\_LENGTH
  - dynamixel.cpp, 52
- PRT1\_PKT\_ERRBIT
  - dynamixel.h, 55
- PRT1\_PKT\_ID
  - dynamixel.h, 55
- PRT1\_PKT\_INSTRUCTION
  - dynamixel.h, 55
- PRT1\_PKT\_LENGTH
  - dynamixel.h, 55
- PRT1\_PKT\_PARAMETER0
  - dynamixel.h, 56
- pause
  - ServoThread, 41
- ping
  - dynamixel, 18
- pos
  - ServoThread::Servo, 35
- PresentLoad
  - AX12, 7
- PresentPosition
  - AX12, 7
- PresentSpeed
  - AX12, 7
- PresentTemperature
  - AX12, 7
- PresentVoltage
  - AX12, 7
- Punch
  - AX12, 7
- RAM
  - AX12, 7
- ROM
  - AX12, 7
- read
  - dxl\_hal, 13
- read\_byte
  - dynamixel, 18
- read\_word
  - dynamixel, 19
- Registered
  - AX12, 7
- ReturnDelayTime
  - AX12, 8
- run
  - ServoThread, 41
- rx\_packet
  - dynamixel, 19
- sCount

- MainWindow, 30
- Servo
  - ServoThread::Servo, 35
- ServoThread, 35
  - \_axis, 44
  - \_buts, 44
  - \_cBaud, 45
  - \_cPort, 45
  - \_cond, 45
  - \_dChanged, 45
  - \_end, 45
  - \_mod, 45
  - \_mutex, 45
  - \_pause, 45
  - \_sBaud, 45
  - \_sPort, 45
  - \_sPortChanged, 45
  - \_servos, 45
  - \_statusBar, 46
  - ~ServoThread, 39
  - a, 46
  - b, 46
  - controlled, 39
  - cos60, 46
  - end, 40
  - getServoBaud, 40
  - getServoPort, 40
  - getServoPortInfo, 40
  - getServosInfo, 40
  - L1, 46
  - L2, 46
  - load, 41
  - manual, 39
  - Mode, 39
  - mutex, 41
  - pause, 41
  - run, 41
  - ServoThread, 39
  - setAngles, 42
  - setData, 42
  - setSID, 43
  - setServoBaud, 42
  - setServoPort, 43
  - setServoPortInfo, 43
  - setStatusBar, 43
  - sin60, 46
  - singleAngle, 44
  - v\_1\_0, 39
  - Version, 39
  - wakeUp, 44
  - write, 44
- ServoThread::Servo, 34
  - ID, 35
  - load, 35
  - pos, 35
  - Servo, 35
- servothread.cpp, 60
- servothread.h, 60
- set\_packet\_timeout
  - dynamixel, 20
- set\_packet\_timeout\_ms
  - dynamixel, 21
- set\_txpacket\_id
  - dynamixel, 21
- set\_txpacket\_instruction
  - dynamixel, 21
- set\_txpacket\_length
  - dynamixel, 21
- set\_txpacket\_parameter
  - dynamixel, 21
- setAngles
  - ServoThread, 42
- setData
  - ServoThread, 42
- setGoalPosition
  - AX12, 10
- setID
  - AX12, 10
- setJointMode
  - AX12, 10
- setMinMax
  - AX12, 10
- setSID
  - ServoThread, 43
- setServoBaud
  - ServoThread, 42
- setServoPort
  - ServoThread, 43
- setServoPortInfo
  - ServoThread, 43
- setSpeed
  - AX12, 11
- setStatusBar
  - ServoThread, 43
- sin60
  - ServoThread, 46
- singleAngle
  - ServoThread, 44
- StatusReturnLevel
  - AX12, 8
- storeData
  - OptionsWindow, 33
- terminate
  - dynamixel, 21
- TorqueEnable
  - AX12, 7
- TorqueLimit
  - AX12, 7
- tx\_packet
  - dynamixel, 22
- txrx\_packet
  - dynamixel, 23
- Ui, 3
- ui
  - MainWindow, 30

- OptionsWindow, [34](#)
- update
  - MainWindow, [28](#)
- v\_1\_0
  - ServoThread, [39](#)
- Version
  - ServoThread, [39](#)
- VersionFirmware
  - AX12, [7](#)
- wakeUp
  - ServoThread, [44](#)
- write
  - dxl\_hal, [13](#)
  - ServoThread, [44](#)
- write\_byte
  - dynamixel, [23](#)
- write\_word
  - dynamixel, [23](#)