

# DeltaRobot

## v0.4

Generated by Doxygen 1.8.9.1

Sun Apr 26 2015 20:14:01



# 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 . . . . .	7
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 . . . . .	9
3.1.3.4	~AX12 . . . . .	9
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 . . . . .	10
3.1.4.5	getCurrentTemp . . . . .	10
3.1.4.6	getCurrentVoltage . . . . .	10
3.1.4.7	getID . . . . .	10
3.1.4.8	setDxl . . . . .	10
3.1.4.9	setGoalPosition . . . . .	10
3.1.4.10	setID . . . . .	11
3.1.4.11	setJointMode . . . . .	11
3.1.4.12	setMinMax . . . . .	11
3.1.4.13	setSpeed . . . . .	12
3.1.5	Member Data Documentation . . . . .	12

3.1.5.1	<a href="#">_dxl</a>	12
3.1.5.2	<a href="#">_ID</a>	12
3.1.5.3	<a href="#">_mode</a>	12
3.1.5.4	<a href="#">_rads</a>	12
3.2	<a href="#">ServoThread::Dominoe Struct Reference</a>	12
3.2.1	<a href="#">Detailed Description</a>	13
3.2.2	<a href="#">Member Data Documentation</a>	13
3.2.2.1	<a href="#">ori</a>	13
3.2.2.2	<a href="#">X</a>	13
3.2.2.3	<a href="#">Y</a>	13
3.3	<a href="#">dxl_hal Class Reference</a>	13
3.3.1	<a href="#">Detailed Description</a>	13
3.3.2	<a href="#">Member Function Documentation</a>	14
3.3.2.1	<a href="#">change_baudrate</a>	14
3.3.2.2	<a href="#">clear</a>	14
3.3.2.3	<a href="#">close</a>	14
3.3.2.4	<a href="#">get_curr_time</a>	14
3.3.2.5	<a href="#">isOpen</a>	14
3.3.2.6	<a href="#">open</a>	14
3.3.2.7	<a href="#">read</a>	14
3.3.2.8	<a href="#">write</a>	15
3.3.3	<a href="#">Member Data Documentation</a>	15
3.3.3.1	<a href="#">_open</a>	15
3.3.3.2	<a href="#">_serial</a>	15
3.3.3.3	<a href="#">_time</a>	15
3.3.3.4	<a href="#">_timed</a>	15
3.4	<a href="#">dynamixel Class Reference</a>	15
3.4.1	<a href="#">Detailed Description</a>	17
3.4.2	<a href="#">Constructor &amp; Destructor Documentation</a>	18
3.4.2.1	<a href="#">dynamixel</a>	18
3.4.2.2	<a href="#">dynamixel</a>	18
3.4.3	<a href="#">Member Function Documentation</a>	18
3.4.3.1	<a href="#">change_baudrate</a>	18
3.4.3.2	<a href="#">get_comm_result</a>	18
3.4.3.3	<a href="#">get_packet_time</a>	18
3.4.3.4	<a href="#">get_rxpacket_error</a>	18
3.4.3.5	<a href="#">get_rxpacket_error_byte</a>	19
3.4.3.6	<a href="#">get_rxpacket_length</a>	19
3.4.3.7	<a href="#">get_rxpacket_parameter</a>	19
3.4.3.8	<a href="#">initialize</a>	19

3.4.3.9	<a href="#">is_packet_timeout</a>	19
3.4.3.10	<a href="#">isOpen</a>	20
3.4.3.11	<a href="#">ping</a>	20
3.4.3.12	<a href="#">read_byte</a>	20
3.4.3.13	<a href="#">read_word</a>	20
3.4.3.14	<a href="#">rx_packet</a>	21
3.4.3.15	<a href="#">set_packet_timeout</a>	22
3.4.3.16	<a href="#">set_packet_timeout_ms</a>	22
3.4.3.17	<a href="#">set_txpacket_id</a>	22
3.4.3.18	<a href="#">set_txpacket_instruction</a>	23
3.4.3.19	<a href="#">set_txpacket_length</a>	23
3.4.3.20	<a href="#">set_txpacket_parameter</a>	23
3.4.3.21	<a href="#">terminate</a>	23
3.4.3.22	<a href="#">tx_packet</a>	23
3.4.3.23	<a href="#">txrx_packet</a>	24
3.4.3.24	<a href="#">write_byte</a>	24
3.4.3.25	<a href="#">write_word</a>	25
3.4.4	<a href="#">Member Data Documentation</a>	26
3.4.4.1	<a href="#">dH</a>	26
3.4.4.2	<a href="#">gbCommStatus</a>	26
3.4.4.3	<a href="#">gbInstructionPacket</a>	26
3.4.4.4	<a href="#">gbRxGetLength</a>	26
3.4.4.5	<a href="#">gbRxPacketLength</a>	26
3.4.4.6	<a href="#">gbStatusPacket</a>	26
3.4.4.7	<a href="#">gdByteTransTime</a>	26
3.4.4.8	<a href="#">gdPacketStartTime</a>	26
3.4.4.9	<a href="#">gdRcvWaitTime</a>	27
3.4.4.10	<a href="#">giBusUsing</a>	27
3.5	<a href="#">MainWindow Class Reference</a>	27
3.5.1	<a href="#">Detailed Description</a>	29
3.5.2	<a href="#">Member Enumeration Documentation</a>	29
3.5.2.1	<a href="#">Version</a>	29
3.5.3	<a href="#">Constructor &amp; Destructor Documentation</a>	29
3.5.3.1	<a href="#">MainWindow</a>	29
3.5.3.2	<a href="#">~MainWindow</a>	30
3.5.4	<a href="#">Member Function Documentation</a>	30
3.5.4.1	<a href="#">joyChanged</a>	30
3.5.4.2	<a href="#">joystickChanged</a>	31
3.5.4.3	<a href="#">on_actionOptions_triggered</a>	31
3.5.4.4	<a href="#">on_start_clicked</a>	31

3.5.4.5	<a href="#">update</a>	31
3.5.4.6	<a href="#">write</a>	32
3.5.4.7	<a href="#">write</a>	32
3.5.5	<a href="#">Member Data Documentation</a>	32
3.5.5.1	<a href="#">_axis</a>	32
3.5.5.2	<a href="#">_axisV</a>	32
3.5.5.3	<a href="#">_buts</a>	32
3.5.5.4	<a href="#">_butsV</a>	32
3.5.5.5	<a href="#">_dataP</a>	32
3.5.5.6	<a href="#">_jAxisX</a>	32
3.5.5.7	<a href="#">_jAxisY</a>	32
3.5.5.8	<a href="#">_jAxisZ</a>	33
3.5.5.9	<a href="#">_joy</a>	33
3.5.5.10	<a href="#">_sT</a>	33
3.5.5.11	<a href="#">_timer</a>	33
3.5.5.12	<a href="#">aSCount</a>	33
3.5.5.13	<a href="#">sCount</a>	33
3.5.5.14	<a href="#">ui</a>	33
3.6	<a href="#">OptionsWindow Class Reference</a>	33
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="#">OptionsWindow</a>	35
3.6.2.2	<a href="#">~OptionsWindow</a>	36
3.6.3	<a href="#">Member Function Documentation</a>	36
3.6.3.1	<a href="#">events</a>	36
3.6.3.2	<a href="#">joystickChanged</a>	37
3.6.3.3	<a href="#">on_servoRefresh_clicked</a>	37
3.6.3.4	<a href="#">storeData</a>	38
3.6.4	<a href="#">Member Data Documentation</a>	38
3.6.4.1	<a href="#">_jAxisX</a>	38
3.6.4.2	<a href="#">_jAxisY</a>	38
3.6.4.3	<a href="#">_jAxisZ</a>	38
3.6.4.4	<a href="#">_joy</a>	38
3.6.4.5	<a href="#">_portSize</a>	38
3.6.4.6	<a href="#">_servo</a>	38
3.6.4.7	<a href="#">_timer</a>	38
3.6.4.8	<a href="#">ui</a>	38
3.7	<a href="#">ServoThread::Servo Struct Reference</a>	39
3.7.1	<a href="#">Detailed Description</a>	39
3.7.2	<a href="#">Constructor &amp; Destructor Documentation</a>	39

3.7.2.1	Servo	39
3.7.2.2	Servo	39
3.7.3	Member Data Documentation	39
3.7.3.1	ID	39
3.7.3.2	load	40
3.7.3.3	pos	40
3.8	ServoThread Class Reference	40
3.8.1	Detailed Description	43
3.8.2	Member Enumeration Documentation	43
3.8.2.1	Mode	43
3.8.2.2	Version	43
3.8.3	Constructor & Destructor Documentation	44
3.8.3.1	ServoThread	44
3.8.3.2	~ServoThread	44
3.8.4	Member Function Documentation	44
3.8.4.1	end	44
3.8.4.2	getServoBaud	44
3.8.4.3	getServoPort	45
3.8.4.4	getServoPortInfo	45
3.8.4.5	getServosInfo	45
3.8.4.6	getServosInfo	45
3.8.4.7	load	45
3.8.4.8	mutex	46
3.8.4.9	pause	46
3.8.4.10	read	46
3.8.4.11	readPath	46
3.8.4.12	run	47
3.8.4.13	setAngles	47
3.8.4.14	setData	48
3.8.4.15	setServoBaud	49
3.8.4.16	setServoPort	49
3.8.4.17	setServoPortInfo	49
3.8.4.18	setSID	49
3.8.4.19	setSpeed	50
3.8.4.20	singleAngle	50
3.8.4.21	statusBar	50
3.8.4.22	wakeUp	50
3.8.4.23	write	51
3.8.5	Member Data Documentation	52
3.8.5.1	_axis	52

3.8.5.2	<a href="#">_buts</a>	52
3.8.5.3	<a href="#">_cBaud</a>	52
3.8.5.4	<a href="#">_cond</a>	52
3.8.5.5	<a href="#">_cPort</a>	52
3.8.5.6	<a href="#">_dChanged</a>	52
3.8.5.7	<a href="#">_dominoe</a>	52
3.8.5.8	<a href="#">_end</a>	52
3.8.5.9	<a href="#">_mod</a>	53
3.8.5.10	<a href="#">_mutex</a>	53
3.8.5.11	<a href="#">_pause</a>	53
3.8.5.12	<a href="#">_sBaud</a>	53
3.8.5.13	<a href="#">_servos</a>	53
3.8.5.14	<a href="#">_sNum</a>	53
3.8.5.15	<a href="#">_sPort</a>	53
3.8.5.16	<a href="#">_sPortChanged</a>	53
3.8.5.17	<a href="#">_sSpeed</a>	53
3.8.5.18	<a href="#">a</a>	53
3.8.5.19	<a href="#">b</a>	53
3.8.5.20	<a href="#">cos60</a>	53
3.8.5.21	<a href="#">L1</a>	54
3.8.5.22	<a href="#">L2</a>	54
3.8.5.23	<a href="#">sin60</a>	54
<b>4</b>	<b>File Documentation</b>	<b>55</b>
4.1	<a href="#">dxl/ax12.cpp File Reference</a>	55
4.1.1	<a href="#">Detailed Description</a>	55
4.2	<a href="#">dxl/ax12.h File Reference</a>	55
4.2.1	<a href="#">Detailed Description</a>	55
4.3	<a href="#">dxl/dxl_hal.cpp File Reference</a>	55
4.3.1	<a href="#">Detailed Description</a>	55
4.4	<a href="#">dxl/dxl_hal.h File Reference</a>	55
4.4.1	<a href="#">Detailed Description</a>	56
4.5	<a href="#">dxl/dynamixel.cpp File Reference</a>	56
4.5.1	<a href="#">Detailed Description</a>	56
4.6	<a href="#">dxl/dynamixel.h File Reference</a>	56
4.6.1	<a href="#">Detailed Description</a>	56
4.7	<a href="#">main.cpp File Reference</a>	56
4.7.1	<a href="#">Detailed Description</a>	56
4.7.2	<a href="#">Function Documentation</a>	57
4.7.2.1	<a href="#">main</a>	57



4.8	mainwindow.cpp File Reference . . . . .	57
4.8.1	Detailed Description . . . . .	57
4.9	mainwindow.h File Reference . . . . .	57
4.9.1	Detailed Description . . . . .	57
4.10	optionwindow.cpp File Reference . . . . .	57
4.10.1	Detailed Description . . . . .	57
4.11	optionwindow.h File Reference . . . . .	57
4.11.1	Detailed Description . . . . .	58
4.12	servothread.cpp File Reference . . . . .	58
4.12.1	Detailed Description . . . . .	58
4.13	servothread.h File Reference . . . . .	58
4.13.1	Detailed Description . . . . .	58
4.14	stable.h File Reference . . . . .	58
4.14.1	Detailed Description . . . . .	59
	<b>Index</b>	<b>61</b>



## 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

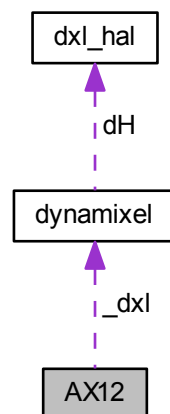
# Class Documentation

### 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](#) ()  
*Default constructor.*
- [AX12](#) ([dynamixel](#) \* `_dxl`, int `ID`== -1)  
*Initializer constructor 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 [setDxl](#) (dynamixel \*dxl)  
*Sets the dynamixel interface.*
- 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.*
- 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 ( )

Default constructor.

```

00005         :
00006         _dxl(NULL),
00007         _ID(-1),
00008         _mode(true),
00009         _rads(false)
00010 {
00011
00012 }

```

#### 3.1.3.2 AX12::AX12( dynamixel \* \_dxl, int ID = -1 )

Initializer constructor if ID == -1 no action is done.

```

00014         :
00015         _dxl(dxl),
00016         _ID(ID),
00017         _mode(true),
00018         _rads(false)
00019 {
00020     if (_ID < 0 or _dxl == NULL) return;
00021     dxl->write_byte(_ID, RAM::TorqueEnable, true);
00022 }

```

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

Copy constructor.

```
00024                                     :
00025     _dxl(a._dxl),
00026     _ID(a._ID),
00027     _mode(a._mode),
00028     _rads(a._rads)
00029 {
00030
00031 }
```

### 3.1.3.4 AX12::~~AX12 ( )

Default destructor.

```
00034 {
00035
00036 }
```

## 3.1.4 Member Function Documentation

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

Returns all active servos;

```
00039 {
00040     if (_dxl == NULL) return QVector<int> (0);
00041
00042     QVector<int> res;
00043     for (int i = 0; i < 256; ++i) {
00044         _dxl->ping(i);
00045         if (_dxl->get_comm_result() == COMM_RXSUCCESS) res.push_back(i);
00046     }
00047     return res;
00048 }
00049 }
```

### 3.1.4.2 double AX12::getCurrentLoad ( )

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

```
00052 {
00053     if (_ID < 0 or _dxl == NULL) return 0;
00054     int load = _dxl->read_word(_ID, RAM::PresentLoad);
00055     load -= 1024;
00056     if (load == -1024) load = 0;
00057     return double((load/1023)*100);
00058 }
```

### 3.1.4.3 double AX12::getCurrentPos ( )

Returns the current position from 0° to 300°

```
00061 {
00062     if (_ID < 0 or _dxl == NULL) return 0;
00063     int pos = _dxl->read_word(_ID, RAM::PresentPosition);
00064     if (_dxl->get_comm_result() != COMM_RXSUCCESS) return -1;
00065
00066     if (_rads) return double((pos/1023.0)*5*M_PI/3);
00067     return double((pos/1023.0)*300);
00068 }
```

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

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

```
00079 {
00080     if (_ID < 0 or _dxi == NULL) return 0;
00081     int speed = _dxi->read_word(_ID, RAM::PresentSpeed);
00082     if (_dxi->get_comm_result() != COMM_RXSUCCESS) return -1;
00083     speed -= 1024;
00084     if (speed == -1024) speed = 0;
00085     return double((speed/1023.0)*100);
00086 }
```

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

Returns the current Temperature in Celsius.

```
00071 {
00072     if (_ID < 0 or _dxi == NULL) return 0;
00073     int temp = _dxi->read_byte(_ID, RAM::PresentTemperature);
00074     if (_dxi->get_comm_result() != COMM_RXSUCCESS) return -1;
00075     return temp;
00076 }
```

#### 3.1.4.6 double AX12::getCurrentVoltage ( )

Returns the current voltage in Volts.

```
00089 {
00090     if (_ID < 0 or _dxi == NULL) return 0;
00091     char voltage = _dxi->read_byte(_ID, RAM::PresentVoltage);
00092     if (_dxi->get_comm_result() != COMM_RXSUCCESS) return -1;
00093     return double(voltage/10.0);
00094 }
```

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

To get the current ID.

```
00114 { return _ID; }
```

#### 3.1.4.8 void AX12::setDxi ( dynamixel \* dxi ) [inline]

Sets the dynamixel interface.

##### Parameters

<i>dxi</i>	Pointer to the dynamixel control class
------------	--

```
00118 { _dxi = dxi; }
```

#### 3.1.4.9 void AX12::setGoalPosition ( double goal )

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

## Parameters

<i>goal</i>	Position (in degrees if not radian mode) or % speed if used wheel mode
-------------	--

```

00097 {
00098     if (_ID < 0 or _dxl == NULL) return;
00099     if (goal > 300.0) goal = 300.0;
00100     else if (goal < 0) goal = 0;
00101     _dxl->write_word(_ID, RAM::GoalPosition, int((goal/300.0)*1023));
00102 }

```

## 3.1.4.10 void AX12::setID ( int ID )

To set a new ID.

## Parameters

<i>ID</i>	the new ID
-----------	------------

```

00105 {
00106     _ID = ID;
00107     if (_ID < 0 or _dxl == NULL) return;
00108     _dxl->write_byte(_ID, RAM::TorqueEnable, true);
00109 }

```

## 3.1.4.11 void AX12::setJointMode ( bool mode )

To set Joint/Wheel mode.

## Parameters

<i>mode</i>	True if Joint and false if Wheel mode
-------------	---------------------------------------

```

00112 {
00113     if (_ID < 0 or _dxl == NULL) return;
00114     _mode = mode;
00115     if (_mode) {
00116         _dxl->write_word(_ID, ROM::CWAngleLimit, 0);
00117         _dxl->write_word(_ID, ROM::CCWAngleLimit, 1023);
00118     }
00119     else {
00120         _dxl->write_word(_ID, ROM::CWAngleLimit, 0);
00121         _dxl->write_word(_ID, ROM::CCWAngleLimit, 0);
00122     }
00123 }

```

## 3.1.4.12 void AX12::setMinMax ( double min, double max )

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

```

00126 {
00127     if (_ID < 0 or _dxl == NULL) return;
00128
00129     if (min > max) {
00130         double aux = min;
00131         min = max;
00132         max = aux;
00133     }
00134
00135     if (min < 0.0) min = 0;
00136     if (max > 300.0) max = 300;
00137
00138     min = (min/300)*1023;
00139     max = (max/300)*1023;
00140
00141     _dxl->write_word(_ID, ROM::CWAngleLimit, int (min));
00142     _dxl->write_word(_ID, ROM::CCWAngleLimit, int (max));
00143 }

```

### 3.1.4.13 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.

```

00146 {
00147     if (_ID < 0 or _dxl == NULL) return;
00148     if (speed > 100.0) speed = 100.0;
00149     if (_mode) {
00150         if (speed < 0.0) speed = 0.0;
00151
00152         int byte = int((speed/100.0) * 1024.0);
00153         if (speed == 100.0) byte = 0;
00154         _dxl->write_byte(_ID, RAM::MovingSpeed, byte);
00155     }
00156     else {
00157         if (speed < -100.0) speed = -100.0;
00158
00159         int byte = int(((speed + 100)/100.0) * 1024);
00160         _dxl->write_byte(_ID, RAM::MovingSpeed, byte);
00161     }
00162 }
00163 }
```

## 3.1.5 Member Data Documentation

### 3.1.5.1 dynamixel\* AX12::\_dxl [private]

Contains the dynamixel communication.

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

Stores the current ID.

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

True if we use the joint mode.

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

True if the angle is returned in radians.

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

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

## 3.2 ServoThread::Dominoe Struct Reference

Struct to handle the dominoe pieces.

### Public Attributes

- double [X](#)  
*X position.*
- double [Y](#)  
*Y position.*
- double [ori](#)  
*Orientation from X = 0 in degrees.*

### 3.2.1 Detailed Description

Struct to handle the dominoe pieces.

### 3.2.2 Member Data Documentation

#### 3.2.2.1 double ServoThread::Dominoe::ori

Orientation from X = 0 in degrees.

#### 3.2.2.2 double ServoThread::Dominoe::X

X position.

#### 3.2.2.3 double ServoThread::Dominoe::Y

Y position.

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

- [servothread.h](#)

## 3.3 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

### 3.3.1 Detailed Description

Dynamixel SDK platform dependent.

### 3.3.2 Member Function Documentation

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

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

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

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

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

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

#### 3.3.2.4 double dxl\_hal::get\_curr\_time ( )

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

#### 3.3.2.5 bool dxl\_hal::isOpen ( ) [inline]

```
00030 { return _open; }
```

#### 3.3.2.6 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 }
```

#### 3.3.2.7 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 }

```

### 3.3.2.8 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 }

```

## 3.3.3 Member Data Documentation

3.3.3.1 bool dxl\_hal::\_open = false [private]

3.3.3.2 QSerialPort dxl\_hal::\_serial [private]

3.3.3.3 int dxl\_hal::\_time = 30 [private]

3.3.3.4 bool dxl\_hal::\_timed = false [private]

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

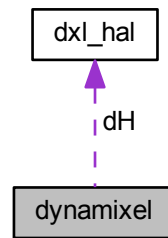
- [dxl/dxl\\_hal.h](#)
- [dxl/dxl\\_hal.cpp](#)

## 3.4 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 communication.*
- 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 if being used.*

#### 3.4.1 Detailed Description

Dynamixel 1.0 protocol class.

### 3.4.2 Constructor & Destructor Documentation

#### 3.4.2.1 dynamixel::dynamixel ( ) [inline]

Default constructor.

```
00097 {}
```

#### 3.4.2.2 dynamixel::dynamixel ( QString port\_num, int baud\_rate = 1000000 )

Initialization constructor.

```
00011 {
00012     initialize(port_num, baud_rate);
00013 }
```

### 3.4.3 Member Function Documentation

#### 3.4.3.1 int dynamixel::change\_baudrate ( int baud\_rate )

Changes the current baud rate.

```
00031 {
00032     int result = 0;
00033     float baudrate = (float)baud_rate;
00034
00035     result = dH.change_baudrate(baudrate);
00036     if(result == 1)
00037         gdByteTransTime = 1000.0f / baudrate * 10.0; // 1000/baudrate(bit per msec) *
00038         10(start bit + data bit + stop bit)
00039     return result;
00040 }
```

#### 3.4.3.2 int dynamixel::get\_comm\_result ( ) [inline]

Returns the current com status.

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

#### 3.4.3.3 double dynamixel::get\_packet\_time ( void )

Returns the packet time.

```
00050 {
00051     double elapsed_time;
00052
00053     elapsed_time = (double)(dH.get_curr_time() -
00054         gdPacketStartTime);
00055     // Overflow
00056     if(elapsed_time < 0) gdPacketStartTime = dH.get_curr_time();
00057     return elapsed_time;
00058 }
00059 }
```

#### 3.4.3.4 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
--------------	----------------------------

```

00271 {
00272     if( gbStatusPacket[PRT1_PKT_ERRBIT] & (unsigned char)error )
00273         return true;
00274
00275     return false;
00276 }
```

## 3.4.3.5 int dynamixel::get\_rxpacket\_error\_byte( void )

Returns the error byte.

```

00279 {
00280     return gbStatusPacket[PRT1_PKT_ERRBIT];
00281 }
```

## 3.4.3.6 int dynamixel::get\_rxpacket\_length( )

Returns the received packet length.

```

00289 {
00290     return (int)gbStatusPacket[PRT1_PKT_LENGTH];
00291 }
```

## 3.4.3.7 int dynamixel::get\_rxpacket\_parameter( int index )

Returns the received parameter.

```

00284 {
00285     return (int)gbStatusPacket[PRT1_PKT_PARAMETER0+index];
00286 }
```

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

Initializes the port.

```

00016 {
00017     if( baud_rate < 1900 ) return 0;
00018
00019     if( not dH.open(port_num, baud_rate) ) return false;
00020
00021     // 1000/baudrate(bit per msec) * 10(start bit + data bit + stop bit)
00022     gdByteTransTime = 1000.0 / (double)baud_rate * 10.0;
00023
00024     gbCommStatus = COMM_RXSUCCESS;
00025     giBusUsing = 0;
00026
00027     return true;
00028 }
```

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

Returns true if the packet is timeout.

**Returns**

True if the packet is timeout

```
00074 {
00075     if(this->get_packet_time() > gdRcvWaitTime)
00076         return true;
00077     return false;
00078 }
```

**3.4.3.10 bool dynamixel::isOpen ( ) [inline]**

True if the port is open.

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

**3.4.3.11 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
-----------	---------------------------

```
00294 {
00295     while(giBusUsing);
00296
00297     gbInstructionPacket[PRT1_PKT_ID] = (unsigned char)id;
00298     gbInstructionPacket[PRT1_PKT_INSTRUCTION] = INST_PING;
00299     gbInstructionPacket[PRT1_PKT_LENGTH] = 2;
00300
00301     txrx_packet();
00302 }
```

**3.4.3.12 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

```
00305 {
00306     while(giBusUsing);
00307
00308     gbInstructionPacket[PRT1_PKT_ID] = (unsigned char)id;
00309     gbInstructionPacket[PRT1_PKT_INSTRUCTION] = INST_READ;
00310     gbInstructionPacket[PRT1_PKT_PARAMETER0+0] = (unsigned char)address;
00311     gbInstructionPacket[PRT1_PKT_PARAMETER0+1] = 1;
00312     gbInstructionPacket[PRT1_PKT_LENGTH] = 4;
00313
00314     txrx_packet();
00315
00316     return (int)gbStatusPacket[PRT1_PKT_PARAMETER0];
00317 }
```

**3.4.3.13 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

```

00333 {
00334     while(giBusUsing);
00335
00336     gbInstructionPacket[PRT1_PKT_ID] = (unsigned char)id;
00337     gbInstructionPacket[PRT1_PKT_INSTRUCTION] = INST_READ;
00338     gbInstructionPacket[PRT1_PKT_PARAMETER0+0] = (unsigned char)address;
00339     gbInstructionPacket[PRT1_PKT_PARAMETER0+1] = 2;
00340     gbInstructionPacket[PRT1_PKT_LENGTH] = 4;
00341
00342     txrx_packet();
00343
00344     return MAKEWORD((int) gbStatusPacket[PRT1_PKT_PARAMETER0+0], (int)
gbStatusPacket[PRT1_PKT_PARAMETER0+1]);
00345 }

```

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

Receives a packet.

```

00144 {
00145     unsigned char i = 0, j = 0, nRead = 0;
00146     unsigned char checksum = 0;
00147
00148     if( giBusUsing == 0 )
00149         return;
00150
00151     if( gbInstructionPacket[PRT1_PKT_ID] == BROADCAST_ID )
00152     {
00153         gbCommStatus = COMM_RXSUCCESS;
00154         giBusUsing = 0;
00155         return;
00156     }
00157
00158     if( gbCommStatus == COMM_TXSUCCESS )
00159     {
00160         gbRxGetLength = 0;
00161         //gbRxPacketLength = 6; //minimum wait length
00162     }
00163
00164     while(1)
00165     {
00166         nRead = dH.read( &gbStatusPacket[gbRxGetLength],
gbRxPacketLength - gbRxGetLength );
00167         gbRxGetLength += nRead;
00168
00169         if(gbRxGetLength > 4)
00170             gbRxPacketLength = gbStatusPacket[PRT1_PKT_LENGTH] + 4;
00171
00172         if( gbRxGetLength < gbRxPacketLength )
00173         {
00174             if( is_packet_timeout() == 1 )
00175             {
00176                 if(gbRxGetLength == 0)
00177                     gbCommStatus = COMM_RXTIMEOUT;
00178                 else
00179                     gbCommStatus = COMM_RXCORRUPT;
00180                 giBusUsing = 0;
00181                 return;
00182             }
00183             gbCommStatus = COMM_RXWAITING;
00184             //return;
00185         }
00186         else
00187         {
00188             break;
00189         }
00190     }
00191
00192     // Find packet header
00193     for( i=0; i<(gbRxGetLength-1); i++ )
00194     {
00195         if( gbStatusPacket[i] == 0xff && gbStatusPacket[i+1] == 0xff )
00196             break;
00197         else if( i == gbRxGetLength-2 && gbStatusPacket[gbRxGetLength-1] == 0xff )
00198             break;

```

```

00199         else {
00200             gbCommStatus = COMM_RXCORRUPT;
00201             return;
00202         }
00203     }
00204
00205     if( i > 0 )
00206     {
00207         for( j=0; j<(gbRxGetLength-i); j++ )
00208             gbStatusPacket[j] = gbStatusPacket[j + i];
00209
00210         gbRxGetLength -= i;
00211     }
00212
00213     // Check id pairing
00214     if( gbInstructionPacket[PRT1_PKT_ID] != gbStatusPacket[PRT1_PKT_ID] )
00215     {
00216         gbCommStatus = COMM_RXCORRUPT;
00217         giBusUsing = 0;
00218         return;
00219     }
00220
00221     // Check checksum
00222     for( i=0; i<(gbStatusPacket[PRT1_PKT_LENGTH]+1); i++ )
00223         checksum += gbStatusPacket[i+2];
00224     checksum = ~checksum;
00225
00226     if( gbStatusPacket[gbStatusPacket[PRT1_PKT_LENGTH]+3] != checksum )
00227     {
00228         gbCommStatus = COMM_RXCORRUPT;
00229         giBusUsing = 0;
00230         return;
00231     }
00232
00233     gbCommStatus = COMM_RXSUCCESS;
00234     giBusUsing = 0;
00235 }

```

#### 3.4.3.15 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
-------------------	--

```

00062 {
00063     gdPacketStartTime = dH.get_curr_time();
00064     gdRcvWaitTime = (gdByteTransTime*(double)NumRcvByte + 2.0*LATENCY_TIME + 2.
00065     0);
00066 }

```

#### 3.4.3.16 void dynamixel::set\_packet\_timeout\_ms ( int msec )

Sets the timeout in ms.

##### Parameters

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

```

00068 {
00069     gdPacketStartTime = dH.get_curr_time();
00070     gdRcvWaitTime = (double)msec;
00071 }

```

#### 3.4.3.17 void dynamixel::set\_txpacket\_id ( int id )

Sets the sending packet ID.

```

00250 {
00251     gbInstructionPacket[PRT1_PKT_ID] = (unsigned char)id;
00252 }

```



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

Sets the sending packet instruction.

```
00255 {
00256     gbInstructionPacket[PRT1_PKT_INSTRUCTION] = (unsigned char)instruction;
00257 }
```

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

Sets the sending packet length.

```
00266 {
00267     gbInstructionPacket[PRT1_PKT_LENGTH] = (unsigned char)length;
00268 }
```

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

Sets the sending packet parameter.

```
00260 {
00261     gbInstructionPacket[PRT1_PKT_PARAMETER0+index] = (unsigned char)value;
00262 }
00263 }
```

**3.4.3.21 int dynamixel::terminate ( void )**

Closes the communication.

```
00043 {
00044     dH.close();
00045     return 0;
00046 }
```

**3.4.3.22 void dynamixel::tx\_packet ( void )**

Sends a packet.

```
00082 {
00083     unsigned char pkt_idx = 0;
00084     unsigned char TxNumByte, RealTxNumByte;
00085     unsigned char checksum = 0;
00086
00087     if ( giBusUsing == 1 )
00088     {
00089         gbCommStatus = COMM_TXFAIL;
00090         return;
00091     }
00092
00093     giBusUsing = 1;
00094
00095     if ( gbInstructionPacket[PRT1_PKT_INSTRUCTION] != INST_PING
00096         && gbInstructionPacket[PRT1_PKT_INSTRUCTION] != INST_READ
00097         && gbInstructionPacket[PRT1_PKT_INSTRUCTION] != INST_WRITE
00098         && gbInstructionPacket[PRT1_PKT_INSTRUCTION] != INST_REG_WRITE
00099         && gbInstructionPacket[PRT1_PKT_INSTRUCTION] != INST_ACTION
00100         && gbInstructionPacket[PRT1_PKT_INSTRUCTION] != INST_RESET
00101         && gbInstructionPacket[PRT1_PKT_INSTRUCTION] != INST_SYNC_WRITE )
00102     {
00103         gbCommStatus = COMM_TXERROR;
00104         giBusUsing = 0;
00105         return;
00106     }
00107 }
```

```

00108     gbInstructionPacket[0] = 0xff;
00109     gbInstructionPacket[1] = 0xff;
00110     for( pkt_idx = 0; pkt_idx < (gbInstructionPacket[PRT1_PKT_LENGTH]+1); pkt_idx++ )
00111         checksum += gbInstructionPacket[pkt_idx+2];
00112     gbInstructionPacket[gbInstructionPacket[PRT1_PKT_LENGTH]+3] = ~
checksum;
00113
00114     //if( gbCommStatus == COMM_RXTIMEOUT || gbCommStatus == COMM_RXCORRUPT )
00115     //    dH.clear();
00116
00117     dH.clear();
00118
00119     TxNumByte = gbInstructionPacket[PRT1_PKT_LENGTH] + 4;
00120     RealTxNumByte = dH.write( gbInstructionPacket, TxNumByte );
00121
00122     if( TxNumByte != RealTxNumByte )
00123     {
00124         gbCommStatus = COMM_TXFAIL;
00125         giBusUsing = 0;
00126         return;
00127     }
00128
00129     if( gbInstructionPacket[PRT1_PKT_INSTRUCTION] == INST_READ )
00130     {
00131         gbRxPacketLength = gbInstructionPacket[PRT1_PKT_PARAMETER0+1] + 6;
00132         set_packet_timeout( gbInstructionPacket[PRT1_PKT_PARAMETER0+1] + 6 );
00133     }
00134     else
00135     {
00136         gbRxPacketLength = 6;
00137         set_packet_timeout( 6 );
00138     }
00139
00140     gbCommStatus = COMM_TXSUCCESS;
00141 }

```

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

Sends and receives a packet.

```

00238 {
00239     tx_packet();
00240
00241     if( gbCommStatus != COMM_TXSUCCESS )
00242         return;
00243
00244     rx_packet();
00245 }
00246 }

```

### 3.4.3.24 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

```

00320 {
00321     while( giBusUsing );
00322
00323     gbInstructionPacket[PRT1_PKT_ID] = (unsigned char)id;
00324     gbInstructionPacket[PRT1_PKT_INSTRUCTION] = INST_WRITE;
00325     gbInstructionPacket[PRT1_PKT_PARAMETER0+0] = (unsigned char)address;
00326     gbInstructionPacket[PRT1_PKT_PARAMETER0+1] = (unsigned char)value;
00327     gbInstructionPacket[PRT1_PKT_LENGTH] = 4;
00328
00329     txrx_packet();
00330 }

```

3.4.3.25 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

```

00348 {
00349     while (giBusUsing);
00350
00351     gbInstructionPacket[PRT1_PKT_ID] = (unsigned char)id;
00352     gbInstructionPacket[PRT1_PKT_INSTRUCTION] = INST_WRITE;
00353     gbInstructionPacket[PRT1_PKT_PARAMETER0+0] = (unsigned char)address;
00354     gbInstructionPacket[PRT1_PKT_PARAMETER0+1] = (unsigned char)LOBYTE(value);
00355     gbInstructionPacket[PRT1_PKT_PARAMETER0+2] = (unsigned char)HIBYTE(value);
00356     gbInstructionPacket[PRT1_PKT_LENGTH] = 5;
00357
00358     txrx_packet();
00359 }

```

**3.4.4 Member Data Documentation****3.4.4.1 dxl\_hal dynamixel::dH** [private]

Contains the serial port communication.

**3.4.4.2 int dynamixel::gbCommStatus = COMM\_RXSUCCESS** [private]

Current communication status.

**3.4.4.3 unsigned char dynamixel::gbInstructionPacket[MAXNUM\_TXPACKET] = {0}** [private]

Contains all the instructions.

**3.4.4.4 unsigned int dynamixel::gbRxGetLength = 0** [private]

Temporal length from the received packet.

**3.4.4.5 unsigned int dynamixel::gbRxPacketLength = 0** [private]

Received packet length.

**3.4.4.6 unsigned char dynamixel::gbStatusPacket[MAXNUM\_RXPACKET] = {0}** [private]

Contains the status.

**3.4.4.7 double dynamixel::gdByteTransTime = 0.0** [private]

Byte transmission time.

**3.4.4.8 double dynamixel::gdPacketStartTime = 0.0** [private]

Packet start time.

3.4.4.9 `double dynamixel::gdRcvWaitTime = 0.0` [private]

Receive wait time.

3.4.4.10 `int dynamixel::giBusUsing = 0` [private]

True if the bus is being used.

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

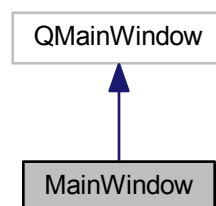
- [dxl/dynamixel.h](#)
- [dxl/dynamixel.cpp](#)

## 3.5 MainWindow Class Reference

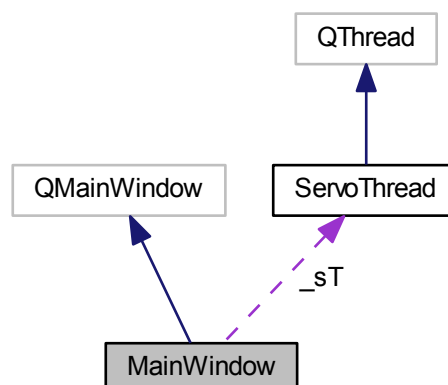
Contains all the windows and other classes.

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

Inheritance diagram for MainWindow:



Collaboration diagram for MainWindow:



## Signals

- void [joystickChanged](#) ()  
*Emmitted when a joystick changes.*

## Public Member Functions

- [MainWindow](#) (QWidget \*parent=0)  
*Default constructor.*
- [~MainWindow](#) ()  
*Default destructor.*

## Private Types

- enum [Version](#) { [v\\_1\\_0](#) }

## 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 Member Functions

- void [write](#) ()  
*Writes the data to the default location.*
- void [write](#) (QString path)  
*Writes the data to disk overloaded function.*

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

### 3.5.1 Detailed Description

Contains all the windows and other classes.

### 3.5.2 Member Enumeration Documentation

#### 3.5.2.1 enum MainWindow::Version [private]

Enumerator

**v\_1\_0**

```
00032         {
00033         v_1_0
00034     };
```

### 3.5.3 Constructor & Destructor Documentation

#### 3.5.3.1 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.start();
00016
00017     connect(&_amp;_joy, SIGNAL(changed()), this, SLOT(joyChanged()));
00018     connect(&_amp;_timer, SIGNAL(timeout()), this, SLOT(update()));
00019     connect(&_amp;_sT, SIGNAL(statusBar(QString)),
00020             ui->statusbar, SLOT(showMessage(QString)));
00021
00022
00023     _timer.setInterval(10);
00024     _timer.start();
00025
00026     // JOYSTICK
00027     QVector< QString > V(_joy.getAllAxis());
```

```

00028     // Adding axis
00029     QGridLayout *wL = new QGridLayout;
00030     for (int i = 0; i < XJoystick::AxisCount; ++i) {
00031         QHBoxLayout *L = new QHBoxLayout;
00032         L->addWidget(new QLabel(V[i].append(":"), this));
00033         _axis[i] = new QLabel("#");
00034         L->addWidget(_axis[i]);
00035         L->addStretch();
00036         wL->addLayout(L, i%3, i/3);
00037     }
00038     ui->joyAxis->setLayout(wL);
00039
00040     // Adding buttons
00041     wL = new QGridLayout;
00042     for (int i = 0; i < XJoystick::ButtonCount; ++i) {
00043         _buts[i] = new QLabel(QString::number(i + 1));
00044         wL->addWidget(_buts[i], i/8, i%8);
00045         _buts[i]->setEnabled(false);
00046         _buts[i]->hide();
00047     }
00048     ui->joyButs->setLayout(wL);
00049     ui->joyAxis->hide();
00050     ui->joyButs->hide();
00051     ui->line->hide();
00052
00053
00054     // Creating data Path
00055     _dataP = QStandardPaths::writableLocation(QStandardPaths::AppDataLocation);
00056     QDir dir(_dataP);
00057     if (!dir.exists()) dir.mkpath(_dataP);
00058 }

```

### 3.5.3.2 MainWindow::~MainWindow ( )

Default destructor.

```

00061 {
00062     delete ui;
00063 }

```

## 3.5.4 Member Function Documentation

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

Handles a joystick update.

```

00081 {
00082     int sel = _joy.current();
00083
00084     QVector< XJoystick::Info > V(_joy.available());
00085     bool found = false;
00086     int i = 0;
00087     while (i < V.size() and not found) { found = V[i].ID == sel; ++i; }
00088     if (not found) {
00089         if (V.size() > 0) {
00090             _joy.select(V[0].ID);
00091             ui->line->hide();
00092
00093             // Showing axis
00094             ui->joyAxis->show();
00095
00096             // Showing buttons
00097             for (QLabel *l : _buts) l->hide();
00098             ui->joyButs->show();
00099             int n = _joy.buttonCount();
00100             for (int i = 0; i < n; ++i) _buts[i]->show();
00101         }
00102         else {
00103             _joy.select(-1);
00104             ui->joyAxis->hide();
00105             ui->joyButs->hide();
00106             ui->line->hide();
00107         }
00108     }
00109     emit joystickChanged();
00110 }

```



**3.5.4.2 void MainWindow::joystickChanged ( ) [signal]**

Emitted when a joystick changes.

**3.5.4.3 void MainWindow::on\_actionOptions\_triggered ( ) [private],[slot]**

To select the options.

```
00114 {
00115     OptionsWindow o(_joy, &_sT, _jAxisX, _jAxisY,
00116                     _jAxisZ, this);
00117     o.exec();
00118     connect(this, SIGNAL(joystickChanged()), &o, SLOT(
00119             joystickChanged()));
00120     if (o.result()) o.storeData();
00121 }
```

**3.5.4.4 void MainWindow::on\_start\_clicked ( ) [private],[slot]**

```
00159 {
00160     QString text = ui->start->text();
00161     if (text == "Start") {
00162         _sT.wakeUp();
00163         ui->start->setText("Stop");
00164     }
00165     else if (text == "Stop") {
00166         _sT.pause();
00167         ui->start->setText("Start");
00168     }
00169 }
00170 }
```

**3.5.4.5 void MainWindow::update ( ) [private],[slot]**

Updates all data to the servo thread.

```
00124 {
00125     // Joystick values
00126     _joy.update();
00127     for (int i = 0; i < XJoystick::AxisCount; ++i) {
00128         float temp = _joy[i];
00129         _axisV[i] = temp;
00130         _axis[i]->setText(QString::number(temp));
00131     }
00132     for (int i = 0; i < XJoystick::ButtonCount; ++i) {
00133         bool temp = _joy.button(i);
00134         _butsV[i] = temp;
00135         _buts[i]->setEnabled(temp);
00136     }
00137     _sT.setData(_axisV, _butsV);
00138     QVector<ServoThread::Servo> servo(_sT.getServosInfo());
00139     // Updating position sliders
00140     ui->servo0S->setValue(servo[0].pos);
00141     ui->servo1S->setValue(servo[1].pos);
00142     ui->servo2S->setValue(servo[2].pos);
00143     // Updating position labels
00144     ui->servo0->setText(QString::number(servo[0].pos));
00145     ui->servo1->setText(QString::number(servo[1].pos));
00146     ui->servo2->setText(QString::number(servo[2].pos));
00147     // Updating load labels
00148     ui->servo0L->setText(QString::number(servo[0].load));
00149     ui->servo1L->setText(QString::number(servo[1].load));
00150     ui->servo2L->setText(QString::number(servo[2].load));
00151 }
00152 }
```

#### 3.5.4.6 void MainWindow::write ( ) [inline],[private]

Writes the data to the default location.

```
00083 { write(_dataP); }
```

#### 3.5.4.7 void MainWindow::write ( QString path ) [private]

Writes the data to disk overloaded function.

```
00066 {
00067     QDir dir(path);
00068     QFile file(dir.filePath("main.opts"));
00069     if(not file.open(QIODevice::WriteOnly)) {
00070         ui->statusbar->showMessage("Error saving file", 1000);
00071         return;
00072     }
00073
00074     QDataStream f(&file);
00075     f << int(Version::v_1_0) << _jAxisX << _jAxisY << _jAxisZ;
00076
00077     _sT.write(dir.filePath("servo.opts"));
00078 }
```

### 3.5.5 Member Data Documentation

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

Handles all the axis labels.

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

Contains the axis value;.

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

Handles all the button labels.

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

Handles all buttons values.

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

Contains the path to the data location.

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

Axis for the X value.

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

Axis for the Y value.

**3.5.5.8** `int MainWindow::_jAxisZ = -1` `[private]`

AXis for the Z value.

**3.5.5.9** `XJoystick MainWindow::_joy` `[private]`

To handle the joystick.

**3.5.5.10** `ServoThread MainWindow::_sT` `[private]`

Contains the thread controlling all the servos and external hardware.

**3.5.5.11** `QTimer MainWindow::_timer` `[private]`

To update the joystick value.

**3.5.5.12** `const int MainWindow::aSCount = 0` `[static], [private]`

Contains the number of additional servos used.

**3.5.5.13** `const int MainWindow::sCount = 3` `[static], [private]`

Contains the number of minimun servos to work.

**3.5.5.14** `Ui::MainWindow* MainWindow::ui` `[private]`

Contains the user interface.

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

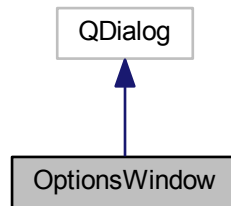
- [mainwindow.h](#)
- [mainwindow.cpp](#)

## 3.6 OptionsWindow Class Reference

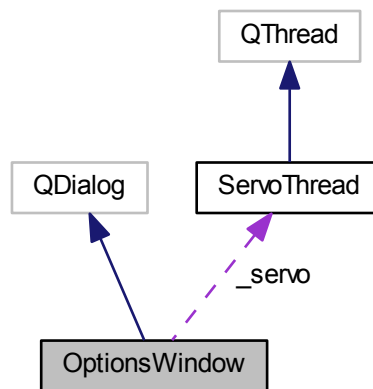
Class used to handle a Window to set the options.

```
#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, int &aX, int &aY, int &aZ, QWidget \*parent=0)  
*Default constructor must be initialized with a few values.*
- `~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

- int & [\\_jAxisX](#)  
*Reference to axis for the X value.*
- int & [\\_jAxisY](#)  
*Reference to axis for the Y value.*
- int & [\\_jAxisZ](#)  
*Reference to axis for the Z value.*
- 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.*

### 3.6.1 Detailed Description

Class used to handle a Window to set the options.

### 3.6.2 Constructor & Destructor Documentation

**3.6.2.1** OptionsWindow::OptionsWindow ( XJoystick & *J*, ServoThread \* *servo*, int & *aX*, int & *aY*, int & *aZ*, QWidget \* *parent* = 0 ) [explicit]

Default constructor must be intialized with a few values.

#### Parameters

<i>J</i>	Reference to the Joystick handler
<i>servo</i>	Pointer to the <a href="#">ServoThread</a>
<i>aX</i>	Axis for the X value
<i>aY</i>	Axis for the Y value
<i>aZ</i>	Axis for the Z value

```

00006                                     :
00007     QDialog(parent),
00008     _jAxisX(aX),
00009     _jAxisY(aY),
00010     _jAxisZ(aZ),
00011     _joy(J),
00012     _portSize(-1),
00013     _servo(servo),
00014     _timer(this),
00015     ui(new Ui::OptionsWindow)
00016 {
00017     ui->setupUi(this);

```

```

00018
00019     QVector< QString > A(_joy.getAllAxis());
00020
00021     // Adding joystick axis movement
00022     ui->joyMX->addItem("None", -1);
00023     ui->joyMY->addItem("None", -1);
00024     ui->joyMZ->addItem("None", -1);
00025
00026     for (int i = 0; i < A.size(); ++i) ui->joyMX->addItem(A[i], i);
00027     for (int i = 0; i < A.size(); ++i) ui->joyMY->addItem(A[i], i);
00028     for (int i = 0; i < A.size(); ++i) ui->joyMZ->addItem(A[i], i);
00029
00030     ui->joyMX->setCurrentIndex(_jAxisX + 1);
00031     ui->joyMY->setCurrentIndex(_jAxisY + 1);
00032     ui->joyMZ->setCurrentIndex(_jAxisZ + 1);
00033
00034     // Updating joystick data
00035     joystickChanged();
00036
00037     // Configuring event funcion
00038     _timer.setInterval(500);
00039     _timer.setSingleShot(false);
00040     _timer.start();
00041     connect(&_timer, SIGNAL(timeout()), this, SLOT(events()));
00042
00043     // Adding servos
00044     ui->servo0->addItem("None", -1);
00045     ui->servo1->addItem("None", -1);
00046     ui->servo2->addItem("None", -1);
00047     ui->servo3->addItem("None", -1);
00048
00049     this->on_servoRefresh_clicked();
00050 }

```

### 3.6.2.2 OptionsWindow::~OptionsWindow ( )

Destructor.

```

00053 {
00054     delete ui;
00055 }

```

## 3.6.3 Member Function Documentation

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

Handles events that need to be updated continously.

```

00099 {
00100     auto ports = QSerialPortInfo::availablePorts();
00101     ui->portN->setText(QString::number(ports.size()));
00102
00103     if (ports.size() != _portSize) {
00104         _portSize = ports.size();
00105
00106         QString portC(ui->portC->currentData().toString());
00107         QString portS(ui->portS->currentData().toString());
00108
00109         int selC = 0, selS = 0;
00110
00111         ui->portC->clear();
00112         ui->portS->clear();
00113
00114         ui->portC->addItem("None", "");
00115         ui->portS->addItem("None", "");
00116
00117         for (int i = 0; i < ports.size(); ++i) {
00118             QString text(ports[i].portName());
00119             text += ": " + ports[i].description();
00120             ui->portC->addItem(text, ports[i].portName());
00121             ui->portS->addItem(text, ports[i].portName());
00122
00123             if (ports[i].portName() == portC) selC = i + 1;
00124             if (ports[i].portName() == portS) selS = i + 1;
00125         }
00126
00127         ui->portC->setCurrentIndex(selC);

```

```

00128         ui->portS->setCurrentIndex(selS);
00129     }
00130 }

```

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

To handle the change of a joystick.

```

00079 {
00080     // Clear all the items and write the new items
00081     ui->joySel->clear();
00082     ui->joySel->addItem("None", -1);
00083
00084     // Adding items and searching the current
00085     int pos = 0;
00086     QVector<XJoystick::Info> V(_joy.available());
00087     for (int i = 0; i < V.size(); ++i) {
00088         QString text(V[i].name);
00089         text += ": " + QString::number(V[i].ID);
00090         if (V[i].ID == _joy.current()) pos = i;
00091         ui->joySel->addItem(text, V[i].ID);
00092     }
00093     ui->joySel->setCurrentIndex(pos);
00094
00095     ui->joyN->setText(QString::number(V.size()));
00096 }

```

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

Refreshes all the servos connected to the port.

```

00133 {
00134
00135     QString port;
00136     int baud;
00137     _servo->getServoPortInfo(port, baud);
00138     dynamixel dxl(port, baud);
00139
00140     int s0 = ui->servo0->currentData().toInt();
00141     int s1 = ui->servo1->currentData().toInt();
00142     int s2 = ui->servo2->currentData().toInt();
00143     int s3 = ui->servo3->currentData().toInt();
00144
00145     ui->servo0->clear();
00146     ui->servo1->clear();
00147     ui->servo2->clear();
00148     ui->servo3->clear();
00149
00150     int index = 0;
00151     int p0 = 0, p1 = 0, p2 = 0, p3 = 0;
00152
00153     for (int i = 0; i <= MAX_ID; ++i) {
00154         dxl.ping(i);
00155         if (dxl.get_comm_result() == COMM_RXSUCCESS) {
00156             if (i == s0) p0 = index;
00157             if (i == s1) p1 = index;
00158             if (i == s2) p2 = index;
00159             if (i == s3) p3 = index;
00160
00161             ui->servo0->addItem(QString::number(i), i);
00162             ui->servo1->addItem(QString::number(i), i);
00163             ui->servo2->addItem(QString::number(i), i);
00164             ui->servo3->addItem(QString::number(i), i);
00165
00166             ++index;
00167         }
00168     }
00169
00170     ui->servo0->setCurrentIndex(p0);
00171     ui->servo1->setCurrentIndex(p1);
00172     ui->servo2->setCurrentIndex(p2);
00173     ui->servo3->setCurrentIndex(p3);
00174
00175 }

```

### 3.6.3.4 void OptionsWindow::storeData ( )

Stores all data.

```

00058 {
00059     // Storing joystick data
00060     _joy.select(ui->joySel->currentData().toInt());
00061
00062     // Joystick movement axis
00063     _jAxisX = ui->joyMX->currentData().toInt();
00064     _jAxisY = ui->joyMY->currentData().toInt();
00065     _jAxisZ = ui->joyMZ->currentData().toInt();
00066
00067     QString portS(ui->portS->currentData().toString());
00068     int baudS(ui->baudRS->value());
00069     _servo->setServoPortInfo(portS, baudS);
00070
00071     QVector<int> sID;
00072     if (ui->servo0->count()) sID.push_back(ui->servo0->currentData().toInt());
00073     if (ui->servo1->count()) sID.push_back(ui->servo1->currentData().toInt());
00074     if (ui->servo2->count()) sID.push_back(ui->servo2->currentData().toInt());
00075     if (ui->servo3->count()) sID.push_back(ui->servo3->currentData().toInt());
00076 }
```

## 3.6.4 Member Data Documentation

### 3.6.4.1 int& OptionsWindow::\_jAxisX [private]

Reference to axis for the X value.

### 3.6.4.2 int& OptionsWindow::\_jAxisY [private]

Reference to axis for the Y value.

### 3.6.4.3 int& OptionsWindow::\_jAxisZ [private]

Reference to axis for the Z value.

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

Contains the Joystick to handle options.

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

Contains the size of the ports.

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

Pointer to the servo thread class.

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

Waits for a new COM port.

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

Containsh the GUI.

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



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

## 3.7 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.*

#### 3.7.1 Detailed Description

Struct for the [AX12](#) servos.

#### 3.7.2 Constructor & Destructor Documentation

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

Default constructor.

```
00042 : ID(ID), load(load), pos(pos) {}
```

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

Copy constructor.

```
00045 : ID(s.ID), load(s.load), pos(s.pos) {}
```

#### 3.7.3 Member Data Documentation

##### 3.7.3.1 int ServoThread::Servo::ID

Contains the servo ID.

### 3.7.3.2 double ServoThread::Servo::load

Contains the servo load.

### 3.7.3.3 double ServoThread::Servo::pos

Contains the servo position.

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

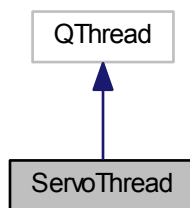
- [servothread.h](#)

## 3.8 ServoThread Class Reference

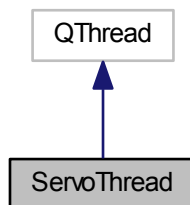
The [ServoThread](#)'s class handles the communication between the delta robot servos and the PC.

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

Inheritance diagram for ServoThread:



Collaboration diagram for ServoThread:



## Classes

- struct [Dominoe](#)

*Struct to handle the dominoe pieces.*

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

## Public Types

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

## Signals

- void [statusBar](#) (QString)  
*Emmitted when the status bar must be changed.*

## 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 [read](#) (QString file)  
*Reads and loads the data from the selected file.*
- void [readPath](#) (QString file)  
*Reads the path where to put the selected pieces.*
- 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 `setSpeed` (int speed)  
*Sets the servos speed.*
- 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` = 6.000  
*The clamp center lenght.*
- const double `L2` = 6.374  
*The support 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.*
- QVector< `Dominoe` > `_dominoe`  
*Contains all the dominoes information.*

- `bool _end`  
*True when we must end executino.*
- `Mode _mod`  
*Contains the working mode.*
- `QMutex _mutex`  
*To prevent memory errors between threads.*
- `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.*
- `const int _sNum = 4`  
*Number of servos to manage.*
- `QString _sPort`  
*Contains the selected com port used in the communication with servos.*
- `bool _sPortChanged`  
*True if the servos port changes.*
- `int _sSpeed`  
*Speed of the robot.*

### 3.8.1 Detailed Description

The `ServoThread`'s class handles the communication between the delta robot servos and the PC.

### 3.8.2 Member Enumeration Documentation

#### 3.8.2.1 `enum ServoThread::Mode`

Contains the working mode.

Enumerator

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

```
00050     {
00051         controlled,
00052         manual
00053     };
```

#### 3.8.2.2 `enum ServoThread::Version` [private]

Enum containing all the save file versions.

Enumerator

***v\_1\_0***

```
00019     {
00020         v_1_0
00021     };
```

### 3.8.3 Constructor & Destructor Documentation

#### 3.8.3.1 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(_sNum),
00015     _sPort("COM9"),
00016     _sPortChanged(false),
00017     _sSpeed(60)
00018 {
00019
00020 }
```

#### 3.8.3.2 ServoThread::~~ServoThread ( )

Default destructor.

```

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

### 3.8.4 Member Function Documentation

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

Ends the execution.

```

00063     {
00064         _mutex.lock();
00065         _end = true;
00066         _cond.wakeOne();
00067         _mutex.unlock();
00068
00069         wait();
00070     }
```

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

Returns the current servo Baud rate.

```

00085     {
00086         QMutexLocker mL(&_mutex);
00087         return _sBaud;
00088     }
```

**3.8.4.3** QString ServoThread::getServoPort ( ) [inline]

Returns the current servo Port.

```
00092     {
00093         QMutexLocker mL(&_mutex);
00094         return _sPort;
00095     }
```

**3.8.4.4** void ServoThread::getServoPortInfo ( QString & port, int & baud ) [inline]

Returns both servo Port and baud Rate.

```
00099     {
00100         _mutex.lock();
00101         baud = _sBaud;
00102         port = _sPort;
00103         _mutex.unlock();
00104     }
```

**3.8.4.5** 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
---	-----------------------------------

```
00110     {
00111         _mutex.lock();
00112         V = _servos;
00113         _mutex.unlock();
00114     }
```

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

Overloaded function to get the servo info.

```
00118     {
00119         QMutexLocker mL(&_mutex);
00120         return _servos;
00121     }
```

**3.8.4.7** 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_1_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 }
```

#### 3.8.4.8 QMutex\* ServoThread::mutex ( ) [inline]

Returns the mutex used in the thread.

```
00124 { return &_mutex; }
```

#### 3.8.4.9 void ServoThread::pause ( ) [inline]

Pauses the execution.

```
00077 {
00078     _mutex.lock();
00079     _pause = true;
00080     _mutex.unlock();
00081 }
```

#### 3.8.4.10 void ServoThread::read ( QString file )

Reads and loads the data from the selected file.

Parameters

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

```
00054 {
00055     // Opening file for reading
00056     QFile f(file);
00057     f.open(QIODevice::ReadOnly);
00058     QDataStream df(&f);
00059
00060     QMutexLocker mL(&_mutex);
00061
00062     int version;
00063     df >> version;
00064     if (version != Version::v_1_0) {
00065         emit statusBar("Error opening file");
00066         return;
00067     }
00068
00069     df >> _cBaud >> _cPort >> _sBaud >> _sPort;
00070     int size;
00071     df >> size;
00072     _servos.resize(size);
00073     for (Servo &s : _servos) df >> s.ID;
00074
00075 }
```

#### 3.8.4.11 void ServoThread::readPath ( QString file )

Reads the path where to put the selected pieces.

Parameters

<i>file</i>	Path to the file where to read the pieces
-------------	---

```
00078 {
00079     // Opening file for reading
00080     QFile f(file);
00081     if (!f.open(QIODevice::ReadOnly)) {
00082         emit statusBar("Error opening file");
00083         return;
00084     }
00085
00086     QTextStream pF(&f);
00087
00088     int size;
00089     pF >> size;
```



```

00090     _mutex.lock();
00091     _dominoe.resize(size);
00092     for (Dominoe &d : _dominoe) {
00093         pF >> d.X >> d.Y >> d.ori;
00094     }
00095
00096     _mutex.unlock();
00097 }

```

#### 3.8.4.12 void ServoThread::run( ) [private]

Used to create another thread.

```

00128 {
00129     _mutex.lock();
00130     int sBaud = _sBaud;
00131     QString sPort = _sPort;
00132
00133     _mutex.unlock();
00134     dynamixel dxl(sPort, sBaud);
00135     QVector< AX12 > S(_servos.size());
00136
00137     for (int i = 0; i < S.size(); ++i) {
00138         S[i] = AX12(&dxl);
00139         S[i].setID(i);
00140     }
00141
00142     while (not _end) {
00143         msleep(10);
00144         _mutex.lock();
00145         if (not _end and _pause) {
00146             dxl.terminate();
00147             _cond.wait(&_mutex);
00148             emit statusBar("Changed");
00149             dxl.initialize(sPort, sBaud);
00150         }
00151         if (_dChanged) {
00152             if (sPort != _sPort) {
00153                 sPort = _sPort;
00154                 sBaud = _sBaud;
00155                 dxl.terminate();
00156                 dxl.initialize(sPort, sBaud);
00157             }
00158         }
00159         _dChanged = false;
00160         _mutex.unlock();
00161     }
00162
00163     dxl.terminate();
00164     exit(0);
00165 }
00166 }

```

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

Used to calculate the servos angles.

```

00170 {
00171     double x1 = x0 + L2 - L1;
00172     double y1 = y0;
00173     double z1 = z0;
00174     theta1 = singleAngle(x1,y1,z1);
00175
00176     double x2 = z0*sin60 - x0*cos60 + L2 - L1;
00177     double y2 = y0;
00178     double z2 = -z0*cos60 - x0*sin60;
00179     theta2 = singleAngle(x2,y2,z2);
00180
00181     double x3 = -z0*sin60 - x0*cos60 + L2 - L1;
00182     double y3 = y0;
00183     double z3 = -z0*cos60 + x0*sin60;
00184     theta3 = singleAngle(x3,y3,z3);
00185 }

```

3.8.4.14 void ServoThread::setData ( QVector< float > & *aV*, QVector< bool > & *butts* )

Adds the loaded data.

## Parameters

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

```

00100 {
00101     _mutex.lock();
00102     // Copying the joystick values
00103     _axis = aV;
00104     _buts = buts;
00105     _dChanged = true;
00106
00107     _mutex.unlock();
00108 }

```

**3.8.4.15** void ServoThread::setServoBaud ( unsigned int *baud* ) [inline]

Sets the servos port baud rate.

## Parameters

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

```

00142 {
00143     _mutex.lock();
00144     _sBaud = baud;
00145     _mutex.unlock();
00146 }

```

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

Sets the servos port.

## Parameters

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

```

00151 {
00152     _mutex.lock();
00153     _sPort = port;
00154     _mutex.unlock();
00155 }

```

**3.8.4.17** 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

```

00161 {
00162     _mutex.lock();
00163     _sPort = port;
00164     _sBaud = baud;
00165     _mutex.unlock();
00166 }

```

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

Sets the servos ID.

## Parameters

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

```

00171     {
00172         if (V.size() != _sNum) {
00173             qDebug() << "Error setting servos";
00174             return;
00175         }
00176
00177         _mutex.lock();
00178         for (int i = 0; i < V.size(); ++i) _servos[i].ID = V[i];
00179         _dChanged = true;
00180         _mutex.unlock();
00181     }
00182 
```

**3.8.4.19 void ServoThread::setSpeed ( int speed ) [inline]**

Sets the servos speed.

## Parameters

<b>speed</b>	Integer from 0 to 100 containing the % of speed
--------------	---

```

00187     {
00188         if (speed < 0) speed = 0;
00189         else if (speed > 100) speed = 100;
00190
00191         _mutex.lock();
00192         _sSpeed = speed;
00193         _mutex.unlock();
00194     }

```

**3.8.4.20 double ServoThread::singleAngle ( double x0, double y0, double z0 ) [private]**

Calculates the angle of one servo in the selected position.

```

00188 {
00189     double n = b * b - a * a - z0 * z0 - x0 * x0 - y0 * y0;
00190     double raiz = sqrt (n*n*y0*y0 - 4*(x0*x0 + y0*y0)*(-x0*x0*a*a + n*n/4));
00191
00192     if (x0 < 0) raiz *= -1;
00193     double y = (-n*y0 + raiz ) / (2*(x0*x0 + y0*y0));
00194
00195     int signe = 1;
00196     if ((b*b - (y0 + a)*(y0 + a)) < (x0*x0 + z0*z0) && x0 < 0) signe *= -1;
00197     double x = sqrt(a*a - y*y)*signe;
00198     return atan2 (y,x);
00199 }

```

**3.8.4.21 void ServoThread::statusBar ( QString ) [signal]**

Emmitted when the status bar must be changed.

**3.8.4.22 void ServoThread::wakeUp ( ) [inline]**

Continues program's execution.

```

00198     {
00199         _mutex.lock();
00200         _pause = false;
00201         _cond.wakeOne();
00202         _mutex.unlock();
00203     }

```

#### 3.8.4.23 void ServoThread::write ( QString *file* )

Writes data to the selected directory.

## Parameters

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

```

00111 {
00112     // Opening file for writing
00113     QFile f(file);
00114     f.open(QIODevice::WriteOnly);
00115     QDataStream df(&f);
00116
00117     _mutex.lock();
00118
00119     // Clamp and servos baud rate and port must be written
00120     df << int (Version::v_l_0) << _cBaud << _cPort << _sBaud <<
    _sPort
00121     << _servos.size();
00122     for (const Servo &s : _servos) df << s.ID;
00123
00124     _mutex.unlock();
00125 }

```

### 3.8.5 Member Data Documentation

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

Contains the axis value.

#### 3.8.5.2 QVector< bool > ServoThread::\_buts [private]

Contains the buttons value.

#### 3.8.5.3 int ServoThread::\_cBaud [private]

Contains the baud rate used to communicate with the clamp.

#### 3.8.5.4 QWaitCondition ServoThread::\_cond [private]

To start and pause the thread.

#### 3.8.5.5 QString ServoThread::\_cPort [private]

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

#### 3.8.5.6 bool ServoThread::\_dChanged [private]

True if the data changes.

#### 3.8.5.7 QVector< Dominoe > ServoThread::\_dominoe [private]

Contains all the dominoes information.

#### 3.8.5.8 bool ServoThread::\_end [private]

True when we must end executino.

**3.8.5.9** `Mode ServoThread::_mod` [private]

Contains the working mode.

**3.8.5.10** `QMutex ServoThread::_mutex` [private]

To prevent memory errors between threads.

**3.8.5.11** `bool ServoThread::_pause` [private]

Pauses the execution of the thread.

**3.8.5.12** `int ServoThread::_sBaud` [private]

Contains the used baud rate to communicate with the servos.

**3.8.5.13** `QVector< Servo > ServoThread::_servos` [private]

Contains the servos information.

**3.8.5.14** `const int ServoThread::_sNum = 4` [private]

Number of servos to manage.

**3.8.5.15** `QString ServoThread::_sPort` [private]

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

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

True if the servos port changes.

**3.8.5.17** `int ServoThread::_sSpeed` [private]

Speed of the robot.

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

The arm length.

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

The forearm length.

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

Contains the cosinus of 60.

**3.8.5.21** `const double ServoThread::L1 = 6.000` `[private]`

The clamp center lenght.

**3.8.5.22** `const double ServoThread::L2 = 6.374` `[private]`

The support center length.

**3.8.5.23** `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.

#### 4.1.1 Detailed Description

Contains the [AX12](#) class implementation.

### 4.2 dxl/ax12.h File Reference

Contains the [AX12](#) class declaration.

#### Classes

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

#### 4.2.1 Detailed Description

Contains the [AX12](#) class declaration.

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

Contains the Dynamixel SDK platform dependent header source.

#### 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.

## Classes

- class [dxl\\_hal](#)

*Dynamixel SDK platform dependent.*

### 4.4.1 Detailed Description

Contains the Dynamixel SDK platform dependent header declaration.

## 4.5 dxl/dynamixel.cpp File Reference

Contains the dynamixel class implementation.

### 4.5.1 Detailed Description

Contains the dynamixel class implementation.

## 4.6 dxl/dynamixel.h File Reference

Contains the dynamixel class declaration.

## Classes

- class [dynamixel](#)

*Dynamixel 1.0 protocol class.*

### 4.6.1 Detailed Description

Contains the dynamixel class declaration.

## 4.7 main.cpp File Reference

Contains the Main of the program.

## 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.

### 4.8.1 Detailed Description

Contains the [MainWindow](#) class implementation.

## 4.9 mainwindow.h File Reference

Contains the [MainWindow](#) class declaration.

### 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.

### 4.10.1 Detailed Description

Contains the [OptionsWindow](#) class implementation.

## 4.11 optionswindow.h File Reference

Contains the [OptionsWindow](#) class declaration.

## 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

Contains the [ServoThread](#) class implementation.

### 4.12.1 Detailed Description

Contains the [ServoThread](#) class implementation.

## 4.13 servothread.h File Reference

Contains the [ServoThread](#) class declaration.

## Classes

- class [ServoThread](#)

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

- struct [ServoThread::Dominoe](#)

*Struct to handle the dominoe pieces.*

- struct [ServoThread::Servo](#)

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

### 4.13.1 Detailed Description

Contains the [ServoThread](#) class declaration.

## 4.14 stable.h File Reference

Contains all includes in a precompiled header.

### 4.14.1 Detailed Description

Contains all includes in a precompiled header.

The includes are:

- QApplication
- QDebug
- QDir
- QDialog
- QLabel
- QMainWindow
- QMutex
- QSerialPortInfo
- QStandardPaths
- QStatusBar
- QString
- QThread
- QTime
- QTimer
- QVector
- QWaitCondition
- XJoystick



# Index

<code>_ID</code>	<code>ServoThread</code> , 53
<code>AX12</code> , 12	<code>_portSize</code>
<code>_axis</code>	<code>OptionsWindow</code> , 38
<code>MainWindow</code> , 32	<code>_rads</code>
<code>ServoThread</code> , 52	<code>AX12</code> , 12
<code>_axisV</code>	<code>_sBaud</code>
<code>MainWindow</code> , 32	<code>ServoThread</code> , 53
<code>_buts</code>	<code>_sNum</code>
<code>MainWindow</code> , 32	<code>ServoThread</code> , 53
<code>ServoThread</code> , 52	<code>_sPort</code>
<code>_butsV</code>	<code>ServoThread</code> , 53
<code>MainWindow</code> , 32	<code>_sPortChanged</code>
<code>_cBaud</code>	<code>ServoThread</code> , 53
<code>ServoThread</code> , 52	<code>_sSpeed</code>
<code>_cPort</code>	<code>ServoThread</code> , 53
<code>ServoThread</code> , 52	<code>_sT</code>
<code>_cond</code>	<code>MainWindow</code> , 33
<code>ServoThread</code> , 52	<code>_serial</code>
<code>_dChanged</code>	<code>dxl_hal</code> , 15
<code>ServoThread</code> , 52	<code>_servo</code>
<code>_dataP</code>	<code>OptionsWindow</code> , 38
<code>MainWindow</code> , 32	<code>_servos</code>
<code>_dominoe</code>	<code>ServoThread</code> , 53
<code>ServoThread</code> , 52	<code>_time</code>
<code>_dxl</code>	<code>dxl_hal</code> , 15
<code>AX12</code> , 12	<code>_timed</code>
<code>_end</code>	<code>dxl_hal</code> , 15
<code>ServoThread</code> , 52	<code>_timer</code>
<code>_jAxisX</code>	<code>MainWindow</code> , 33
<code>MainWindow</code> , 32	<code>OptionsWindow</code> , 38
<code>OptionsWindow</code> , 38	<code>~AX12</code>
<code>_jAxisY</code>	<code>AX12</code> , 9
<code>MainWindow</code> , 32	<code>~MainWindow</code>
<code>OptionsWindow</code> , 38	<code>MainWindow</code> , 30
<code>_jAxisZ</code>	<code>~OptionsWindow</code>
<code>MainWindow</code> , 32	<code>OptionsWindow</code> , 36
<code>OptionsWindow</code> , 38	<code>~ServoThread</code>
<code>_joy</code>	<code>ServoThread</code> , 44
<code>MainWindow</code> , 33	<code>a</code>
<code>OptionsWindow</code> , 38	<code>ServoThread</code> , 53
<code>_mod</code>	<code>aSCount</code>
<code>ServoThread</code> , 52	<code>MainWindow</code> , 33
<code>_mode</code>	<code>AX12</code> , 5
<code>AX12</code> , 12	<code>_ID</code> , 12
<code>_mutex</code>	<code>_dxl</code> , 12
<code>ServoThread</code> , 53	<code>_mode</code> , 12
<code>_open</code>	<code>_rads</code> , 12
<code>dxl_hal</code> , 15	<code>~AX12</code> , 9
<code>pause</code>	<code>AX12</code> , 8

- AlarmLED, 8
- AlarmShutdown, 8
- BaudRate, 8
- CCWAngleLimit, 8
- CCWComplianceMargin, 7
- CCWComplianceSlope, 7
- CWAngleLimit, 8
- CWComplianceMargin, 7
- CWComplianceSlope, 7
- connectedID, 9
- getCurrentLoad, 9
- getCurrentPos, 9
- getCurrentSpeed, 9
- getCurrentTemp, 10
- getCurrentVoltage, 10
- getID, 10
- GoalPosition, 7
- HighestLimitTemp, 8
- HighestLimitVoltage, 8
- ID, 8
- LED, 7
- Lock, 7
- LowestLimitVoltage, 8
- MaxTorque, 8
- ModelNumber, 8
- Moving, 7
- MovingSpeed, 7
- PresentLoad, 7
- PresentPosition, 7
- PresentSpeed, 7
- PresentTemperature, 7
- PresentVoltage, 7
- Punch, 7
- RAM, 7
- ROM, 7
- Registered, 7
- ReturnDelayTime, 8
- setDxl, 10
- setGoalPosition, 10
- setID, 11
- setJointMode, 11
- setMinMax, 11
- setSpeed, 11
- StatusReturnLevel, 8
- TorqueEnable, 7
- TorqueLimit, 7
- VersionFirmware, 8
- AlarmLED
  - AX12, 8
- AlarmShutdown
  - AX12, 8
- b
  - ServoThread, 53
- BaudRate
  - AX12, 8
- CCWAngleLimit
  - AX12, 8
- CCWComplianceMargin
  - AX12, 7
- CCWComplianceSlope
  - AX12, 7
- CWAngleLimit
  - AX12, 8
- CWComplianceMargin
  - AX12, 7
- CWComplianceSlope
  - AX12, 7
- change\_baudrate
  - dxl\_hal, 14
  - dynamixel, 18
- clear
  - dxl\_hal, 14
- close
  - dxl\_hal, 14
- connectedID
  - AX12, 9
- controlled
  - ServoThread, 43
- cos60
  - ServoThread, 53
- dH
  - dynamixel, 26
- dxl/ax12.cpp, 55
- dxl/ax12.h, 55
- dxl/dxl\_hal.cpp, 55
- dxl/dxl\_hal.h, 55
- dxl/dynamixel.cpp, 56
- dxl/dynamixel.h, 56
- dxl\_hal, 13
  - \_open, 15
  - \_serial, 15
  - \_time, 15
  - \_timed, 15
  - change\_baudrate, 14
  - clear, 14
  - close, 14
  - get\_curr\_time, 14
  - isOpen, 14
  - open, 14
  - read, 14
  - write, 15
- dynamixel, 15
  - change\_baudrate, 18
  - dH, 26
  - dynamixel, 18
  - gbCommStatus, 26
  - gbInstructionPacket, 26
  - gbRxGetLength, 26
  - gbRxPacketLength, 26
  - gbStatusPacket, 26
  - gdByteTransTime, 26
  - gdPacketStartTime, 26
  - gdRcvWaitTime, 26
  - get\_comm\_result, 18
  - get\_packet\_time, 18



- get\_rxpacket\_error, 18
- get\_rxpacket\_error\_byte, 19
- get\_rxpacket\_length, 19
- get\_rxpacket\_parameter, 19
- giBusUsing, 27
- initialize, 19
- is\_packet\_timeout, 19
- isOpen, 20
- ping, 20
- read\_byte, 20
- read\_word, 20
- rx\_packet, 21
- set\_packet\_timeout, 22
- set\_packet\_timeout\_ms, 22
- set\_txpacket\_id, 22
- set\_txpacket\_instruction, 22
- set\_txpacket\_length, 23
- set\_txpacket\_parameter, 23
- terminate, 23
- tx\_packet, 23
- txrx\_packet, 24
- write\_byte, 24
- write\_word, 24
- end
  - ServoThread, 44
- events
  - OptionsWindow, 36
- gbCommStatus
  - dynamixel, 26
- gbInstructionPacket
  - dynamixel, 26
- gbRxGetLength
  - dynamixel, 26
- gbRxPacketLength
  - dynamixel, 26
- gbStatusPacket
  - dynamixel, 26
- gdByteTransTime
  - dynamixel, 26
- gdPacketStartTime
  - dynamixel, 26
- gdRcvWaitTime
  - dynamixel, 26
- get\_comm\_result
  - dynamixel, 18
- get\_curr\_time
  - dxl\_hal, 14
- get\_packet\_time
  - dynamixel, 18
- get\_rxpacket\_error
  - dynamixel, 18
- get\_rxpacket\_error\_byte
  - dynamixel, 19
- get\_rxpacket\_length
  - dynamixel, 19
- get\_rxpacket\_parameter
  - dynamixel, 19
- getCurrentLoad
  - AX12, 9
- getCurrentPos
  - AX12, 9
- getCurrentSpeed
  - AX12, 9
- getCurrentTemp
  - AX12, 10
- getCurrentVoltage
  - AX12, 10
- getID
  - AX12, 10
- getServoBaud
  - ServoThread, 44
- getServoPort
  - ServoThread, 44
- getServoPortInfo
  - ServoThread, 45
- getServosInfo
  - ServoThread, 45
- giBusUsing
  - dynamixel, 27
- GoalPosition
  - AX12, 7
- HighestLimitTemp
  - AX12, 8
- HighestLimitVoltage
  - AX12, 8
- ID
  - AX12, 8
  - ServoThread::Servo, 39
- initialize
  - dynamixel, 19
- is\_packet\_timeout
  - dynamixel, 19
- isOpen
  - dxl\_hal, 14
  - dynamixel, 20
- joyChanged
  - MainWindow, 30
- joystickChanged
  - MainWindow, 30
  - OptionsWindow, 37
- L1
  - ServoThread, 53
- L2
  - ServoThread, 54
- LED
  - AX12, 7
- load
  - ServoThread, 45
  - ServoThread::Servo, 39
- Lock
  - AX12, 7
- LowestLimitVoltage

- AX12, 8
- main
  - main.cpp, 57
- main.cpp, 56
  - main, 57
- MainWindow, 27
  - \_axis, 32
  - \_axisV, 32
  - \_buts, 32
  - \_butsV, 32
  - \_dataP, 32
  - \_jAxisX, 32
  - \_jAxisY, 32
  - \_jAxisZ, 32
  - \_joy, 33
  - \_sT, 33
  - \_timer, 33
  - ~MainWindow, 30
  - aSCount, 33
  - joyChanged, 30
  - joystickChanged, 30
  - MainWindow, 29
  - on\_actionOptions\_triggered, 31
  - on\_start\_clicked, 31
  - sCount, 33
  - ui, 33
  - update, 31
  - v\_1\_0, 29
  - Version, 29
  - write, 31, 32
- mainwindow.cpp, 57
- mainwindow.h, 57
- manual
  - ServoThread, 43
- MaxTorque
  - AX12, 8
- Mode
  - ServoThread, 43
- ModelNumber
  - AX12, 8
- Moving
  - AX12, 7
- MovingSpeed
  - AX12, 7
- mutex
  - ServoThread, 45
- on\_actionOptions\_triggered
  - MainWindow, 31
- on\_servoRefresh\_clicked
  - OptionsWindow, 37
- on\_start\_clicked
  - MainWindow, 31
- open
  - dxl\_hal, 14
- OptionsWindow, 33
  - \_jAxisX, 38
  - \_jAxisY, 38
  - \_jAxisZ, 38
  - \_joy, 38
  - \_portSize, 38
  - \_servo, 38
  - \_timer, 38
  - ~OptionsWindow, 36
  - events, 36
  - joystickChanged, 37
  - on\_servoRefresh\_clicked, 37
  - OptionsWindow, 35
  - storeData, 37
  - ui, 38
- optionswindow.cpp, 57
- optionswindow.h, 57
- ori
  - ServoThread::Dominoe, 13
- pause
  - ServoThread, 46
- ping
  - dynamixel, 20
- pos
  - ServoThread::Servo, 40
- 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, 14
  - ServoThread, 46
- read\_byte
  - dynamixel, 20
- read\_word
  - dynamixel, 20
- readPath
  - ServoThread, 46
- Registered
  - AX12, 7
- ReturnDelayTime
  - AX12, 8
- run
  - ServoThread, 47
- rx\_packet
  - dynamixel, 21
- sCount

- MainWindow, 33
- Servo
  - ServoThread::Servo, 39
- ServoThread, 40
  - \_axis, 52
  - \_buts, 52
  - \_cBaud, 52
  - \_cPort, 52
  - \_cond, 52
  - \_dChanged, 52
  - \_dominoe, 52
  - \_end, 52
  - \_mod, 52
  - \_mutex, 53
  - \_pause, 53
  - \_sBaud, 53
  - \_sNum, 53
  - \_sPort, 53
  - \_sPortChanged, 53
  - \_sSpeed, 53
  - \_servos, 53
  - ~ServoThread, 44
  - a, 53
  - b, 53
  - controlled, 43
  - cos60, 53
  - end, 44
  - getServoBaud, 44
  - getServoPort, 44
  - getServoPortInfo, 45
  - getServosInfo, 45
  - L1, 53
  - L2, 54
  - load, 45
  - manual, 43
  - Mode, 43
  - mutex, 45
  - pause, 46
  - read, 46
  - readPath, 46
  - run, 47
  - ServoThread, 44
  - setAngles, 47
  - setData, 47
  - setSID, 49
  - setServoBaud, 49
  - setServoPort, 49
  - setServoPortInfo, 49
  - setSpeed, 50
  - sin60, 54
  - singleAngle, 50
  - statusBar, 50
  - v\_1\_0, 43
  - Version, 43
  - wakeUp, 50
  - write, 50
- ServoThread::Dominoe, 12
  - ori, 13
  - X, 13
  - Y, 13
- ServoThread::Servo, 39
  - ID, 39
  - load, 39
  - pos, 40
  - Servo, 39
- servothread.cpp, 58
- servothread.h, 58
- set\_packet\_timeout
  - dynamixel, 22
- set\_packet\_timeout\_ms
  - dynamixel, 22
- set\_txpacket\_id
  - dynamixel, 22
- set\_txpacket\_instruction
  - dynamixel, 22
- set\_txpacket\_length
  - dynamixel, 23
- set\_txpacket\_parameter
  - dynamixel, 23
- setAngles
  - ServoThread, 47
- setData
  - ServoThread, 47
- setDxl
  - AX12, 10
- setGoalPosition
  - AX12, 10
- setID
  - AX12, 11
- setJointMode
  - AX12, 11
- setMinMax
  - AX12, 11
- setSID
  - ServoThread, 49
- setServoBaud
  - ServoThread, 49
- setServoPort
  - ServoThread, 49
- setServoPortInfo
  - ServoThread, 49
- setSpeed
  - AX12, 11
  - ServoThread, 50
- sin60
  - ServoThread, 54
- singleAngle
  - ServoThread, 50
- stable.h, 58
- statusBar
  - ServoThread, 50
- StatusReturnLevel
  - AX12, 8
- storeData
  - OptionsWindow, 37
- terminate

- dynamixel, [23](#)
- TorqueEnable
  - AX12, [7](#)
- TorqueLimit
  - AX12, [7](#)
- tx\_packet
  - dynamixel, [23](#)
- txrx\_packet
  - dynamixel, [24](#)
- Ui, [3](#)
- ui
  - MainWindow, [33](#)
  - OptionsWindow, [38](#)
- update
  - MainWindow, [31](#)
- v\_1\_0
  - MainWindow, [29](#)
  - ServoThread, [43](#)
- Version
  - MainWindow, [29](#)
  - ServoThread, [43](#)
- VersionFirmware
  - AX12, [8](#)
- wakeUp
  - ServoThread, [50](#)
- write
  - dxl\_hal, [15](#)
  - MainWindow, [31](#), [32](#)
  - ServoThread, [50](#)
- write\_byte
  - dynamixel, [24](#)
- write\_word
  - dynamixel, [24](#)
- X
  - ServoThread::Dominoe, [13](#)
- Y
  - ServoThread::Dominoe, [13](#)