

DeltaRobot

v0.4

Generated by Doxygen 1.8.9.1

Fri Apr 24 2015 20:04:29

Contents

1	Main Page	1
2	Namespace Documentation	3
2.1	Ui Namespace Reference	3
2.1.1	Detailed Description	3
3	Class Documentation	5
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	_dxl	12
3.1.5.2	_ID	12
3.1.5.3	_mode	12
3.1.5.4	_rads	12
3.2	dxl_hal Class Reference	12
3.2.1	Detailed Description	13
3.2.2	Member Function Documentation	13
3.2.2.1	change_baudrate	13
3.2.2.2	clear	13
3.2.2.3	close	13
3.2.2.4	get_curr_time	13
3.2.2.5	isOpen	13
3.2.2.6	open	14
3.2.2.7	read	14
3.2.2.8	write	14
3.2.3	Member Data Documentation	14
3.2.3.1	_open	14
3.2.3.2	_serial	14
3.2.3.3	_time	14
3.2.3.4	_timed	14
3.3	dynamixel Class Reference	15
3.3.1	Detailed Description	17
3.3.2	Constructor & Destructor Documentation	17
3.3.2.1	dynamixel	17
3.3.2.2	dynamixel	17
3.3.3	Member Function Documentation	17
3.3.3.1	change_baudrate	17
3.3.3.2	get_comm_result	17
3.3.3.3	get_packet_time	17
3.3.3.4	get_rxpacket_error	18
3.3.3.5	get_rxpacket_error_byte	19
3.3.3.6	get_rxpacket_length	19
3.3.3.7	get_rxpacket_parameter	19
3.3.3.8	initialize	19
3.3.3.9	is_packet_timeout	19
3.3.3.10	isOpen	20
3.3.3.11	ping	20
3.3.3.12	read_byte	20
3.3.3.13	read_word	20
3.3.3.14	rx_packet	21

3.3.3.15	set_packet_timeout	22
3.3.3.16	set_packet_timeout_ms	22
3.3.3.17	set_txpacket_id	22
3.3.3.18	set_txpacket_instruction	23
3.3.3.19	set_txpacket_length	23
3.3.3.20	set_txpacket_parameter	23
3.3.3.21	terminate	23
3.3.3.22	tx_packet	23
3.3.3.23	txrx_packet	24
3.3.3.24	write_byte	24
3.3.3.25	write_word	25
3.3.4	Member Data Documentation	26
3.3.4.1	dH	26
3.3.4.2	gbCommStatus	26
3.3.4.3	gbInstructionPacket	26
3.3.4.4	gbRxGetLength	26
3.3.4.5	gbRxPacketLength	26
3.3.4.6	gbStatusPacket	26
3.3.4.7	gdByteTransTime	26
3.3.4.8	gdPacketStartTime	26
3.3.4.9	gdRcvWaitTime	27
3.3.4.10	giBusUsing	27
3.4	MainWindow Class Reference	27
3.4.1	Detailed Description	29
3.4.2	Constructor & Destructor Documentation	29
3.4.2.1	MainWindow	29
3.4.2.2	~MainWindow	30
3.4.3	Member Function Documentation	30
3.4.3.1	joyChanged	30
3.4.3.2	joystickChanged	30
3.4.3.3	on_actionOptions_triggered	30
3.4.3.4	on_start_clicked	30
3.4.3.5	update	31
3.4.4	Member Data Documentation	31
3.4.4.1	_axis	31
3.4.4.2	_axisV	31
3.4.4.3	_buts	31
3.4.4.4	_butsV	31
3.4.4.5	_dataP	31
3.4.4.6	_jAxisX	31

3.4.4.7	_jAxisY	31
3.4.4.8	_jAxisZ	32
3.4.4.9	_joy	32
3.4.4.10	_sT	32
3.4.4.11	_timer	32
3.4.4.12	aSCount	32
3.4.4.13	sCount	32
3.4.4.14	ui	32
3.5	OptionsWindow Class Reference	32
3.5.1	Detailed Description	34
3.5.2	Constructor & Destructor Documentation	34
3.5.2.1	OptionsWindow	34
3.5.2.2	~OptionsWindow	35
3.5.3	Member Function Documentation	35
3.5.3.1	events	35
3.5.3.2	joystickChanged	36
3.5.3.3	on_servoRefresh_clicked	36
3.5.3.4	storeData	36
3.5.4	Member Data Documentation	36
3.5.4.1	_jAxisX	36
3.5.4.2	_jAxisY	36
3.5.4.3	_jAxisZ	36
3.5.4.4	_joy	37
3.5.4.5	_portSize	37
3.5.4.6	_servo	37
3.5.4.7	_timer	37
3.5.4.8	ui	37
3.6	ServoThread::Servo Struct Reference	37
3.6.1	Detailed Description	37
3.6.2	Constructor & Destructor Documentation	38
3.6.2.1	Servo	38
3.6.2.2	Servo	38
3.6.3	Member Data Documentation	38
3.6.3.1	ID	38
3.6.3.2	load	38
3.6.3.3	pos	38
3.7	ServoThread Class Reference	38
3.7.1	Detailed Description	41
3.7.2	Member Enumeration Documentation	41
3.7.2.1	Mode	41

3.7.2.2	Version	41
3.7.3	Constructor & Destructor Documentation	42
3.7.3.1	ServoThread	42
3.7.3.2	~ServoThread	42
3.7.4	Member Function Documentation	42
3.7.4.1	end	42
3.7.4.2	getServoBaud	42
3.7.4.3	getServoPort	42
3.7.4.4	getServoPortInfo	43
3.7.4.5	getServosInfo	43
3.7.4.6	getServosInfo	43
3.7.4.7	load	43
3.7.4.8	mutex	44
3.7.4.9	pause	44
3.7.4.10	run	44
3.7.4.11	setAngles	44
3.7.4.12	setData	45
3.7.4.13	setServoBaud	45
3.7.4.14	setServoPort	45
3.7.4.15	setServoPortInfo	45
3.7.4.16	setSID	46
3.7.4.17	singleAngle	46
3.7.4.18	statusBar	46
3.7.4.19	wakeUp	46
3.7.4.20	write	46
3.7.5	Member Data Documentation	47
3.7.5.1	_axis	47
3.7.5.2	_buts	47
3.7.5.3	_cBaud	47
3.7.5.4	_cond	47
3.7.5.5	_cPort	47
3.7.5.6	_dChanged	47
3.7.5.7	_end	47
3.7.5.8	_mod	47
3.7.5.9	_mutex	47
3.7.5.10	_pause	48
3.7.5.11	_sBaud	48
3.7.5.12	_servos	48
3.7.5.13	_sPort	48
3.7.5.14	_sPortChanged	48

3.7.5.15	a	48
3.7.5.16	b	48
3.7.5.17	cos60	48
3.7.5.18	L1	48
3.7.5.19	L2	48
3.7.5.20	sin60	48
4	File Documentation	49
4.1	dxl/ax12.cpp File Reference	49
4.1.1	Detailed Description	49
4.2	dxl/ax12.h File Reference	49
4.2.1	Detailed Description	49
4.3	dxl/dxl_hal.cpp File Reference	49
4.3.1	Detailed Description	49
4.4	dxl/dxl_hal.h File Reference	49
4.4.1	Detailed Description	50
4.5	dxl/dynamixel.cpp File Reference	50
4.5.1	Detailed Description	50
4.6	dxl/dynamixel.h File Reference	50
4.6.1	Detailed Description	50
4.7	main.cpp File Reference	50
4.7.1	Detailed Description	50
4.7.2	Function Documentation	51
4.7.2.1	main	51
4.8	mainwindow.cpp File Reference	51
4.8.1	Detailed Description	51
4.9	mainwindow.h File Reference	51
4.9.1	Detailed Description	51
4.10	optionswindow.cpp File Reference	51
4.10.1	Detailed Description	51
4.11	optionswindow.h File Reference	51
4.11.1	Detailed Description	52
4.12	servothread.cpp File Reference	52
4.12.1	Detailed Description	52
4.13	servothread.h File Reference	52
4.13.1	Detailed Description	52
4.14	stable.h File Reference	52
4.14.1	Detailed Description	52
Index		55

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

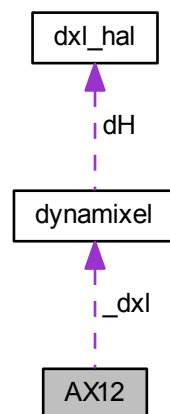
Class Documentation

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 dxl_hal Class Reference

Dynamixel SDK platform dependent.

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

Public Member Functions

- bool [open](#) (QString &devName, int baudrate)
- void [close](#) (void)
- void [clear](#) (void)
- int [change_baudrate](#) (float baudrate)
- int [write](#) (unsigned char *pPacket, int numPacket)

- int `read` (unsigned char *pPacket, int numPacket)
- double `get_curr_time` ()
- bool `isOpen` ()

Private Attributes

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

3.2.1 Detailed Description

Dynamixel SDK platform dependent.

3.2.2 Member Function Documentation

3.2.2.1 int dxl_hal::change_baudrate (float *baudrate*)

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

3.2.2.2 void dxl_hal::clear (void)

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

3.2.2.3 void dxl_hal::close (void)

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

3.2.2.4 double dxl_hal::get_curr_time ()

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

3.2.2.5 bool dxl_hal::isOpen () [inline]

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

3.2.2.6 bool dxl_hal::open (QString & devName, int baudrate)

```

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

```

3.2.2.7 int dxl_hal::read (unsigned char * pPacket, int numPacket)

```

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

```

3.2.2.8 int dxl_hal::write (unsigned char * pPacket, int numPacket)

```

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

```

3.2.3 Member Data Documentation

3.2.3.1 bool dxl_hal::_open = false [private]

3.2.3.2 QSerialPort dxl_hal::_serial [private]

3.2.3.3 int dxl_hal::_time = 30 [private]

3.2.3.4 bool dxl_hal::_timed = false [private]

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

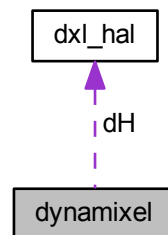
- [dxl/dxl_hal.h](#)
- [dxl/dxl_hal.cpp](#)

3.3 dynamixel Class Reference

Dynamixel 1.0 protocol class.

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

Collaboration diagram for dynamixel:



Public Member Functions

- [dynamixel](#) ()
Default constructor.
- [dynamixel](#) (QString port_num, int baud_rate=1000000)
Initialization constructor.
- bool [isOpen](#) ()
True if the port is open.
- int [initialize](#) (QString port_num, int baud_rate)
Initializes the port.
- int [change_baudrate](#) (int baud_rate)
Changes the current baud rate.
- int [terminate](#) (void)
Closes the communication.
- int [get_comm_result](#) ()
Returns the current com status.
- void [tx_packet](#) (void)
Sends a packet.
- void [rx_packet](#) (void)
Receives a packet.
- void [txrx_packet](#) (void)
Sends and receives a packet.
- void [set_txpacket_id](#) (int id)
Sets the sending packet ID.
- void [set_txpacket_instruction](#) (int instruction)
Sets the sending packet instruction.
- void [set_txpacket_parameter](#) (int index, int value)
Sets the sending packet parameter.
- void [set_txpacket_length](#) (int length)
Sets the sending packet length.

- bool [get_rxpacket_error](#) (int error)
Returns false if no receive error and true if there's an error.
- int [get_rxpacket_error_byte](#) (void)
Returns the error byte.
- int [get_rxpacket_parameter](#) (int index)
Returns the received parameter.
- int [get_rxpacket_length](#) ()
Returns the received packet length.
- void [ping](#) (int id)
Ping to the selected id, check com status for the ping result.
- int [read_byte](#) (int id, int address)
Reads a byte from the selected ID at the selected address.
- void [write_byte](#) (int id, int address, int value)
Writes a byte to the selected ID at the selected address.
- int [read_word](#) (int id, int address)
Reads a word to the selected ID at the selected address.
- void [write_word](#) (int id, int address, int value)
Writes a word to the selected ID at the selected address.
- double [get_packet_time](#) ()
Returns the packet time.
- void [set_packet_timeout](#) (int NumRcvByte)
Sets the timeout in number of received bytes.
- void [set_packet_timeout_ms](#) (int msec)
Sets the timeout in ms.
- bool [is_packet_timeout](#) ()
Returns true if the packet is timeout.

Private Attributes

- [dxl_hal dH](#)
Conains the serial port comunication.
- unsigned char [gblInstructionPacket](#) [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.

3.3.1 Detailed Description

Dynamixel 1.0 protocol class.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 dynamixel::dynamixel() [inline]

Default constructor.

```
00097 {}
```

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

Initialization constructor.

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

3.3.3 Member Function Documentation

3.3.3.1 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) *
10(start bit + data bit + stop bit)
00038
00039     return result;
00040 }
```

3.3.3.2 int dynamixel::get_comm_result() [inline]

Returns the current com status.

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

3.3.3.3 double dynamixel::get_packet_time(void)

Returns the packet time.

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

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

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

Parameters

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

```

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

3.3.3.5 int dynamixel::get_rxpacket_error_byte(void)

Returns the error byte.

```

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

3.3.3.6 int dynamixel::get_rxpacket_length()

Returns the received packet length.

```

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

3.3.3.7 int dynamixel::get_rxpacket_parameter(int index)

Returns the received parameter.

```

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

3.3.3.8 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 }
```

3.3.3.9 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 }
```

3.3.3.10 bool dynamixel::isOpen () [inline]

True if the port is open.

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

3.3.3.11 void dynamixel::ping (int id)

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

Parameters

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

```
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 }
```

3.3.3.12 int dynamixel::read_byte (int id, int address)

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

Parameters

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

```
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 }
```

3.3.3.13 int dynamixel::read_word (int id, int address)

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

Parameters

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

```

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] = (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], (int)
gbStatusPacket[PRT1_PKT_PARAMETER0+1]);
00345 }

```

3.3.3.14 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 }

```

3.3.3.15 void dynamixel::set_packet_timeout (int NumRcvByte)

Sets the timeout in number of received bytes.

Parameters

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

```

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

```

3.3.3.16 void dynamixel::set_packet_timeout_ms (int msec)

Sets the timeout in ms.

Parameters

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

```

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

```

3.3.3.17 void dynamixel::set_txpacket_id (int id)

Sets the sending packet ID.

```

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

```

3.3.3.18 void dynamixel::set_txpacket_instruction (int *instruction*)

Sets the sending packet instruction.

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

3.3.3.19 void dynamixel::set_txpacket_length (int *length*)

Sets the sending packet length.

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

3.3.3.20 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 }
```

3.3.3.21 int dynamixel::terminate (void)

Closes the communication.

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

3.3.3.22 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 }

```

3.3.3.23 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 }

```

3.3.3.24 void dynamixel::write_byte (int id, int address, int value)

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

Parameters

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

```

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 }

```


3.3.3.25 void dynamixel::write_word (int *id*, int *address*, int *value*)

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

Parameters

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

```

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 }

```

3.3.4 Member Data Documentation**3.3.4.1 dxl_hal dynamixel::dH** [private]

Contains the serial port communication.

3.3.4.2 int dynamixel::gbCommStatus = COMM_RXSUCCESS [private]

Current communication status.

3.3.4.3 unsigned char dynamixel::gbInstructionPacket[MAXNUM_TXPACKET] = {0} [private]

Contains all the instructions.

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

Temporal length from the received packet.

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

Received packet length.

3.3.4.6 unsigned char dynamixel::gbStatusPacket[MAXNUM_RXPACKET] = {0} [private]

Contains the status.

3.3.4.7 double dynamixel::gdByteTransTime = 0.0 [private]

Byte transmission time.

3.3.4.8 double dynamixel::gdPacketStartTime = 0.0 [private]

Packet start time.

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

Receive wait time.

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

True if the bus is being used.

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

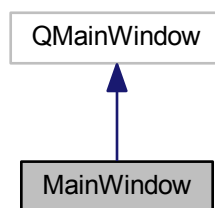
- [dxl/dynamixel.h](#)
- [dxl/dynamixel.cpp](#)

3.4 MainWindow Class Reference

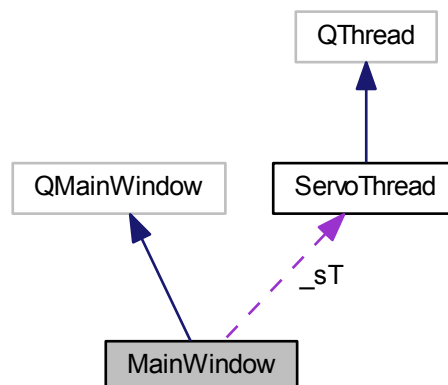
Contains all the windows and other classes.

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

Inheritance diagram for MainWindow:



Collaboration diagram for MainWindow:



Signals

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

Public Member Functions

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

Private Slots

- void [joyChanged](#) ()
Handles a joystick update.
- void [on_actionOptions_triggered](#) ()
To select the options.
- void [update](#) ()
Updates all data to the servo thread.
- void [on_start_clicked](#) ()

Private Attributes

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

Static Private Attributes

- static const int `sCount` = 3
Contains the number of minimum servos to work.
- static const int `aSCount` = 0
Contains the number of additional servos used.

3.4.1 Detailed Description

Contains all the windows and other classes.

3.4.2 Constructor & Destructor Documentation

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

Default constructor.

```

00005                                     :
00006     QMainWindow(parent),
00007     _axis(XJoystick::AxisCount),
00008     _axisV(XJoystick::AxisCount),
00009     _buts(XJoystick::ButtonCount),
00010     _butsV(XJoystick::ButtonCount),
00011     ui(new Ui::MainWindow)
00012 {
00013     ui->setupUi(this);
00014
00015     _sT.start();
00016
00017     connect(&_joy, SIGNAL(changed()), this, SLOT(joyChanged()));
00018     connect(&_timer, SIGNAL(timeout()), this, SLOT(update()));
00019     connect(&_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 }

```

3.4.2.2 MainWindow::~~MainWindow ()

Default destructor.

```
00061 {
00062     delete ui;
00063 }
```

3.4.3 Member Function Documentation

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

Handles a joystick update.

```
00066 {
00067     int sel = _joy.current();
00068
00069     QVector< XJoystick::Info > V(_joy.available());
00070     bool found = false;
00071     int i = 0;
00072     while (i < V.size() and not found) { found = V[i].ID == sel; ++i; }
00073     if (not found) {
00074         if (V.size() > 0) {
00075             _joy.select(V[0].ID);
00076             ui->line->hide();
00077
00078             // Showing axis
00079             ui->joyAxis->show();
00080
00081             // Showing buttons
00082             for (QLabel *l : _buts) l->hide();
00083             ui->joyButs->show();
00084             int n = _joy.buttonCount();
00085             for (int i = 0; i < n; ++i) _buts[i]->show();
00086         }
00087         else {
00088             _joy.select(-1);
00089             ui->joyAxis->hide();
00090             ui->joyButs->hide();
00091             ui->line->hide();
00092         }
00093     }
00094     emit joystickChanged();
00095 }
```

3.4.3.2 void MainWindow::joystickChanged () [signal]

Emmitted when a joystick changes.

3.4.3.3 void MainWindow::on_actionOptions_triggered () [private],[slot]

To select the options.

```
00099 {
00100     OptionsWindow o(_joy, &_amp;sT, _jAxisX, _jAxisY,
00101         _jAxisZ, this);
00102     o.exec();
00103     connect(this, SIGNAL(joystickChanged()), &o, SLOT(
00104         joystickChanged()));
00105     if (o.result()) o.storeData();
00106 }
```

3.4.3.4 void MainWindow::on_start_clicked () [private],[slot]

```
00121 {
00122     QString text = ui->start->text();
```

```

00123
00124     if (text == "Start") {
00125         _sT.wakeUp();
00126         ui->start->setText("Stop");
00127     }
00128     else if (text == "Stop") {
00129         _sT.pause();
00130         ui->start->setText("Start");
00131     }
00132 }

```

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

Updates all data to the servo thread.

```

00109 {
00110     _joy.update();
00111     for (int i = 0; i < XJoystick::AxisCount; ++i) _axisV[i] = _joy[i];
00112     for (int i = 0; i < XJoystick::ButtonCount; ++i) _butsV[i] = _joy.button(i);
00113
00114     _sT.setData(_axisV, _butsV);
00115     QVector<ServoThread::Servo> servo(_sT.getServosInfo());
00116
00117     // TODO: Finish update function
00118 }

```

3.4.4 Member Data Documentation

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

Handles all the axis labels.

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

Contains the axis value;.

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

Handles all the button labels.

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

Handles all buttons values.

3.4.4.5 QString MainWindow::_dataP [private]

Contains the path to the data location.

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

Axis for the X value.

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

Axis for the Y value.

3.4.4.8 `int MainWindow::_jAxisZ = -1` `[private]`

AXis for the Z value.

3.4.4.9 `XJoystick MainWindow::_joy` `[private]`

To handle the joystick.

3.4.4.10 `ServoThread MainWindow::_sT` `[private]`

Contains the thread controlling all the servos and external hardware.

3.4.4.11 `QTimer MainWindow::_timer` `[private]`

To update the joystick value.

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

Contains the number of additional servos used.

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

Contains the number of minimun servos to work.

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

Contains the user interface.

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

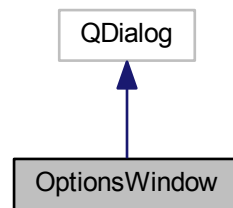
- [mainwindow.h](#)
- [mainwindow.cpp](#)

3.5 OptionsWindow Class Reference

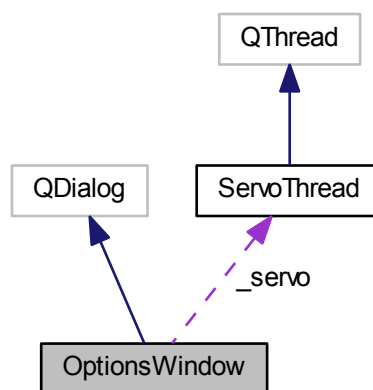
Class used to handle a Window to set the options.

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


Inheritance diagram for OptionsWindow:



Collaboration diagram for OptionsWindow:



Public Slots

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

Public Member Functions

- `OptionsWindow` (XJoystick &J, `ServoThread` *servo, 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.

3.5.1 Detailed Description

Class used to handle a Window to set the options.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 OptionsWindow::OptionsWindow (XJoystick & *J*, ServoThread * *servo*, 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 ServoThread
<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 function
00038     _timer.setInterval(500);
00039     _timer.setSingleShot(false);
00040     _timer.start();
00041     connect(&_timer, SIGNAL(timeout()), this, SLOT(events()));
00042
00043 }

```

3.5.2.2 OptionsWindow::~OptionsWindow ()

Destructor.

```

00046 {
00047     delete ui;
00048 }

```

3.5.3 Member Function Documentation

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

Handles events that need to be updated continuously.

```

00082 {
00083     auto ports = QSerialPortInfo::availablePorts();
00084
00085     if (ports.size() != _portSize) {
00086         _portSize = ports.size();
00087
00088         QString portC(ui->portC->currentData().toString());
00089         QString portS(ui->portS->currentData().toString());
00090
00091         int selC = 0, selS = 0;
00092
00093         ui->portC->clear();
00094         ui->portS->clear();
00095
00096         ui->portC->addItem("None", "");
00097         ui->portS->addItem("None", "");
00098
00099         for (int i = 0; i < ports.size(); ++i) {
00100             QString text(ports[i].portName());
00101             text += ": " + ports[i].description();
00102             ui->portC->addItem(text, ports[i].portName());
00103             ui->portS->addItem(text, ports[i].portName());
00104
00105             if (ports[i].portName() == portC) selC = i + 1;
00106             if (ports[i].portName() == portS) selS = i + 1;
00107         }
00108
00109         ui->portC->setCurrentIndex(selC);
00110         ui->portS->setCurrentIndex(selS);
00111     }
00112 }

```

3.5.3.2 void OptionsWindow::joystickChanged () [slot]

To handle the change of a joystick.

```
00062 {
00063     // Clear all the items and write the new items
00064     ui->joySel->clear();
00065     ui->joySel->addItem("None", -1);
00066
00067     // Adding items and searching the current
00068     int pos = 0;
00069     QVector<XJoystick::Info> V(_joy.available());
00070     for (int i = 0; i < V.size(); ++i) {
00071         QString text(V[i].name);
00072         text += ": " + QString::number(V[i].ID);
00073         if (V[i].ID == _joy.current()) pos = i;
00074         ui->joySel->addItem(text, V[i].ID);
00075     }
00076     ui->joySel->setCurrentIndex(pos);
00077
00078     ui->joyN->setText(QString::number(V.size()));
00079 }
```

3.5.3.3 void OptionsWindow::on_servoRefresh_clicked () [private],[slot]

Refreshes all the servos connected to the port.

```
00115 {
00116     dynamixel dxl;
00117     QString port;
00118     int baud;
00119     _servo->getServoPortInfo(port, baud);
00120
00121     dxl.initialize(port, baud);
00122
00123 }
```

3.5.3.4 void OptionsWindow::storeData ()

Stores all data.

```
00051 {
00052     // Storing joystick data
00053     _joy.select(ui->joySel->currentData().toInt());
00054
00055     // Joystick movement axis
00056     _jAxisX = ui->joyMX->currentData().toInt();
00057     _jAxisY = ui->joyMY->currentData().toInt();
00058     _jAxisZ = ui->joyMZ->currentData().toInt();
00059 }
```

3.5.4 Member Data Documentation

3.5.4.1 int& OptionsWindow::_jAxisX [private]

Reference to axis for the X value.

3.5.4.2 int& OptionsWindow::_jAxisY [private]

Reference to axis for the Y value.

3.5.4.3 int& OptionsWindow::_jAxisZ [private]

Reference to axis for the Z value.

3.5.4.4 XJoystick& OptionsWindow::_joy [private]

Contains the Joystick to handle options.

3.5.4.5 int OptionsWindow::_portSize [private]

Contains the size of the ports.

3.5.4.6 ServoThread* OptionsWindow::_servo [private]

Pointer to the servo thread class.

3.5.4.7 QTimer OptionsWindow::_timer [private]

Waits for a new COM port.

3.5.4.8 Ui::OptionsWindow* OptionsWindow::ui [private]

Contains the GUI.

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

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

3.6 ServoThread::Servo Struct Reference

Struct for the [AX12](#) servos.

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

Public Member Functions

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

Public Attributes

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

3.6.1 Detailed Description

Struct for the [AX12](#) servos.

3.6.2 Constructor & Destructor Documentation

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

Default constructor.

```
00034         : ID(ID), load(load), pos(pos) {}
```

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

Copy constructor.

```
00037 : ID(s.ID), load(s.load), pos(s.pos) {}
```

3.6.3 Member Data Documentation

3.6.3.1 int ServoThread::Servo::ID

Contains the servo ID.

3.6.3.2 double ServoThread::Servo::load

Contains the servo load.

3.6.3.3 double ServoThread::Servo::pos

Contains the servo position.

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

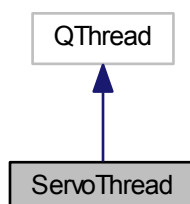
- [servothread.h](#)

3.7 ServoThread Class Reference

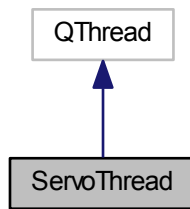
The [ServoThread](#)'s class handles the communication between the delta robot servos and the PC.

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

Inheritance diagram for ServoThread:



Collaboration diagram for ServoThread:



Classes

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

Public Types

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

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 setData (QVector< float > &aV, QVector< bool > &but)`
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 wakeUp ()`
Continues program's execution.
- `void write (QString &file)`
Writes data to the selected directory.

Private Types

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

Private Member Functions

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

Private Attributes

- `const double cos60 = 0.5`
Contains the cosinus of 60.
- `const double sin60 = sqrt(3)/2`
Contains the sinus of 60.
- `const double a = 17.233`
The arm length.
- `const double b = 22.648`
The forearm length.
- `const double L1 = 5.000`
The base center lenght.
- `const double L2 = 6.000`
The platform center length.
- `QVector< float > _axis`
Contains the axis value.
- `QVector< bool > _but`

- 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 comunicate with the clamp.
- `bool _dChanged`

True if the data changes.
- `bool _end`

True when we must end executino.
- `Mode _mod`

Contains the working mode.
- `QMutex _mutex`

To prevent memory errors.
- `bool _pause`

Pauses the execution of the thread.
- `int _sBaud`

Contains the used baud rate to communicate with the servos.
- `QVector< Servo > _servos`

Contains the servos information.
- `QString _sPort`

Contains the selected com port used in the communication with servos.
- `bool _sPortChanged`

True if the servos port changes.

3.7.1 Detailed Description

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

3.7.2 Member Enumeration Documentation

3.7.2.1 enum `ServoThread::Mode`

Contains the working mode.

Enumerator

controlled
manual

```
00042     {
00043         controlled,
00044         manual
00045     };
```

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

Enum containing all the save file versions.

Enumerator

v_1_0

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

3.7.3 Constructor & Destructor Documentation

3.7.3.1 ServoThread::ServoThread ()

Default constructor.

```

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

3.7.3.2 ServoThread::~~ServoThread ()

Default destructor.

```

00022 {
00023     _mutex.lock();
00024     _end = true;
00025     _cond.wakeOne();
00026     _mutex.unlock();
00027
00028     wait();
00029 }
```

3.7.4 Member Function Documentation

3.7.4.1 void ServoThread::end () [inline]

Ends the execution.

```

00055     {
00056         _mutex.lock();
00057         _end = true;
00058         _cond.wakeOne();
00059         _mutex.unlock();
00060
00061         wait();
00062     }
```

3.7.4.2 int ServoThread::getServoBaud () [inline]

Returns the current servo Baud rate.

```

00077     {
00078         QMutexLocker mL(&_mutex);
00079         return _sBaud;
00080     }
```

3.7.4.3 QString ServoThread::getServoPort () [inline]

Returns the current servo Port.

```

00084     {
00085         QMutexLocker mL(&_mutex);
00086         return _sPort;
00087     }

```

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

Returns both servo Port and baud Rate.

```

00091     {
00092         _mutex.lock();
00093         baud = _sBaud;
00094         port = _sPort;
00095         _mutex.unlock();
00096     }

```

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

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

Parameters

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

```

00102     {
00103         _mutex.lock();
00104         V = _servos;
00105         _mutex.unlock();
00106     }

```

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

Overloaded function to get the servo info.

```

00110     {
00111         QMutexLocker mL(&_mutex);
00112         return _servos;
00113     }

```

3.7.4.7 void ServoThread::load (QString & file)

Loads the data from the selected file.

```

00032 {
00033     _mutex.lock();
00034     QFile f(file);
00035     f.open(QIODevice::ReadOnly);
00036     QDataStream df(&f);
00037
00038     int ver;
00039     df >> ver;
00040     if (ver == Version::v_1_0) {
00041         int n;
00042         df >> _cBaud >> _cPort >> _sBaud >> _sPort >> n;
00043
00044         _servos.resize(n);
00045         for (Servo &s : _servos) df >> s.ID;
00046         _dChanged = true;
00047     }
00048     else qWarning() << "Not a valid file";
00049     _mutex.unlock();
00050 }

```

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

Returns the mutex used in the thread.

```
00116 { return &_mutex; }
```

3.7.4.9 void ServoThread::pause () [inline]

Pauses the execution.

```
00069 {
00070     _mutex.lock();
00071     _pause = true;
00072     _mutex.unlock();
00073 }
```

3.7.4.10 void ServoThread::run () [private]

Used to create another thread.

```
00078 {
00079     _mutex.lock();
00080     int sBaud = _sBaud;
00081     QString sPort = _sPort;
00082
00083     _mutex.unlock();
00084     dynamixel dxl(sPort, sBaud);
00085     QVector< AX12 > S(_servos.size());
00086
00087     for (int i = 0; i < S.size(); ++i) {
00088         S[i] = AX12(&dxl);
00089         S[i].setID(i);
00090     }
00091
00092     while (not _end) {
00093
00094         msleep(10);
00095         _mutex.lock();
00096         if (not _end and _pause) {
00097             dxl.terminate();
00098             _cond.wait(&_mutex);
00099             emit statusBar("Changed");
00100             dxl.initialize(sPort, sBaud);
00101         }
00102         if (_dChanged) {
00103             if (sPort != _sPort) {
00104                 sPort = _sPort;
00105                 sBaud = _sBaud;
00106                 dxl.terminate();
00107                 dxl.initialize(sPort, sBaud);
00108             }
00109             _dChanged = false;
00110             _mutex.unlock();
00111         }
00112     }
00113
00114     dxl.terminate();
00115     exit(0);
00116 }
```

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

Used to calculate the servos angles.

```
00120 {
00121     double x1 = x0 + L2 - L1;
00122     double y1 = y0;
00123     double z1 = z0;
```

```

00124     theta1 = singleAngle(x1,y1,z1);
00125
00126     double x2 = z0*sin60 - x0*cos60 + L2 - L1;
00127     double y2 = y0;
00128     double z2 = -z0*cos60 - x0*sin60;
00129     theta2 = singleAngle(x2,y2,z2);
00130
00131     double x3 = -z0*sin60 - x0*cos60 + L2 - L1;
00132     double y3 = y0;
00133     double z3 = -z0*cos60 + x0*sin60;
00134     theta3 = singleAngle(x3,y3,z3);
00135 }

```

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

Adds the loaded data.

Parameters

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

```

00053 {
00054     _mutex.lock();
00055     // Copying the joystick values
00056     _axis = aV;
00057     _buts = buts;
00058     _dChanged = true;
00059
00060     _mutex.unlock();
00061 }

```

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

Sets the servos port baud rate.

Parameters

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

```

00126     {
00127         _mutex.lock();
00128         _sBaud = baud;
00129         _mutex.unlock();
00130     }

```

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

Sets the servos port.

Parameters

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

```

00135     {
00136         _mutex.lock();
00137         _sPort = port;
00138         _mutex.unlock();
00139     }

```

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

Sets the servos port info, data and selected port.

Parameters

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

```

00145    {
00146        _mutex.lock();
00147        _sPort = port;
00148        _sBaud = baud;
00149        _mutex.unlock();
00150    }

```

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

Sets the servos ID.

Parameters

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

```

00155    {
00156        _mutex.lock();
00157        if (V.size() != _servos.size()) _servos.resize(V.size());
00158
00159        for (int i = 0; i < V.size(); ++i) _servos[i].ID = V[i];
00160        _dChanged = true;
00161        _mutex.unlock();
00162    }

```

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

Calculates the angle of one servo in the selected position.

```

00138 {
00139     double n = b * b - a * a - z0 * z0 - x0 * x0 - y0 * y0;
00140     double raiz = sqrt (n*n*y0*y0 - 4*(x0*x0 + y0*y0)*(-x0*x0*a*a + n*n/4));
00141
00142     if (x0 < 0) raiz *= -1;
00143     double y = (-n*y0 + raiz ) / (2*(x0*x0 + y0*y0));
00144
00145     int signe = 1;
00146     if ((b*b - (y0 + a)*(y0 + a)) < (x0*x0 + z0*z0) && x0 < 0) signe *= -1;
00147     double x = sqrt(a*a - y*y)*signe;
00148     return atan2 (y,x);
00149 }

```

3.7.4.18 void ServoThread::statusBar (QString) [signal]

Emmitted when the status bar must be changed.

3.7.4.19 void ServoThread::wakeUp () [inline]

Continues program's execution.

```

00166    {
00167        _mutex.lock();
00168        _pause = false;
00169        _cond.wakeOne();
00170        _mutex.unlock();
00171    }

```

3.7.4.20 void ServoThread::write (QString & file)

Writes data to the selected directory.

Parameters

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

```

00064 {
00065     _mutex.lock();
00066     QFile f(file);
00067     f.open(QIODevice::WriteOnly);
00068     QDataStream df(&f);
00069
00070     df << int (Version::v_1_0) << _cBaud << _cPort << _sBaud <<
    _sPort
00071     << _servos.size();
00072     for (const Servo &s : _servos) df << s.ID;
00073
00074     _mutex.unlock();
00075 }
```

3.7.5 Member Data Documentation

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

Contains the axis value.

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

Contains the buttons value.

3.7.5.3 int ServoThread::_cBaud [private]

Contains the baud rate used to communicate with the clamp.

3.7.5.4 QWaitCondition ServoThread::_cond [private]

To start and pause the thread.

3.7.5.5 QString ServoThread::_cPort [private]

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

3.7.5.6 bool ServoThread::_dChanged [private]

True if the data changes.

3.7.5.7 bool ServoThread::_end [private]

True when we must end execution.

3.7.5.8 Mode ServoThread::_mod [private]

Contains the working mode.

3.7.5.9 QMutex ServoThread::_mutex [private]

To prevent memory errors.

3.7.5.10 `bool ServoThread::_pause` [private]

Pauses the execution of the thread.

3.7.5.11 `int ServoThread::_sBaud` [private]

Contains the used baud rate to communicate with the servos.

3.7.5.12 `QVector< Servo > ServoThread::_servos` [private]

Contains the servos information.

3.7.5.13 `QString ServoThread::_sPort` [private]

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

3.7.5.14 `bool ServoThread::_sPortChanged` [private]

True if the servos port changes.

3.7.5.15 `const double ServoThread::a = 17.233` [private]

The arm length.

3.7.5.16 `const double ServoThread::b = 22.648` [private]

The forearm length.

3.7.5.17 `const double ServoThread::cos60 = 0.5` [private]

Contains the cosinus of 60.

3.7.5.18 `const double ServoThread::L1 = 5.000` [private]

The base center lenght.

3.7.5.19 `const double ServoThread::L2 = 6.000` [private]

The platform center length.

3.7.5.20 `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 and dynamixel2 classes implementation.

4.5.1 Detailed Description

Contains the dynamixel and dynamixel2 classes implementation.

4.6 dxl/dynamixel.h File Reference

Contains the dynamixel and dynamixel2 classes declaration.

Classes

- class [dynamixel](#)

Dynamixel 1.0 protocol class.

4.6.1 Detailed Description

Contains the dynamixel and dynamixel2 classes 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::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

- [_ID](#)
 - [AX12, 12](#)
 - [_axis](#)
 - [MainWindow, 31](#)
 - [ServoThread, 47](#)
 - [_axisV](#)
 - [MainWindow, 31](#)
 - [_buts](#)
 - [MainWindow, 31](#)
 - [ServoThread, 47](#)
 - [_butsV](#)
 - [MainWindow, 31](#)
 - [_cBaud](#)
 - [ServoThread, 47](#)
 - [_cPort](#)
 - [ServoThread, 47](#)
 - [_cond](#)
 - [ServoThread, 47](#)
 - [_dChanged](#)
 - [ServoThread, 47](#)
 - [_dataP](#)
 - [MainWindow, 31](#)
 - [_dxl](#)
 - [AX12, 12](#)
 - [_end](#)
 - [ServoThread, 47](#)
 - [_jAxisX](#)
 - [MainWindow, 31](#)
 - [OptionsWindow, 36](#)
 - [_jAxisY](#)
 - [MainWindow, 31](#)
 - [OptionsWindow, 36](#)
 - [_jAxisZ](#)
 - [MainWindow, 31](#)
 - [OptionsWindow, 36](#)
 - [_joy](#)
 - [MainWindow, 32](#)
 - [OptionsWindow, 36](#)
 - [_mod](#)
 - [ServoThread, 47](#)
 - [_mode](#)
 - [AX12, 12](#)
 - [_mutex](#)
 - [ServoThread, 47](#)
 - [_open](#)
 - [dxl_hal, 14](#)
 - [_pause](#)
 - [ServoThread, 47](#)
 - [_portSize](#)
 - [OptionsWindow, 37](#)
 - [_rads](#)
 - [AX12, 12](#)
 - [_sBaud](#)
 - [ServoThread, 48](#)
 - [_sPort](#)
 - [ServoThread, 48](#)
 - [_sPortChanged](#)
 - [ServoThread, 48](#)
 - [_sT](#)
 - [MainWindow, 32](#)
 - [_serial](#)
 - [dxl_hal, 14](#)
 - [_servo](#)
 - [OptionsWindow, 37](#)
 - [_servos](#)
 - [ServoThread, 48](#)
 - [_time](#)
 - [dxl_hal, 14](#)
 - [_timed](#)
 - [dxl_hal, 14](#)
 - [_timer](#)
 - [MainWindow, 32](#)
 - [OptionsWindow, 37](#)
 - [~AX12](#)
 - [AX12, 9](#)
 - [~MainWindow](#)
 - [MainWindow, 29](#)
 - [~OptionsWindow](#)
 - [OptionsWindow, 35](#)
 - [~ServoThread](#)
 - [ServoThread, 42](#)
- [a](#)
 - [ServoThread, 48](#)
- [aSCount](#)
 - [MainWindow, 32](#)
- [AX12, 5](#)
 - [_ID, 12](#)
 - [_dxl, 12](#)
 - [_mode, 12](#)
 - [_rads, 12](#)
 - [~AX12, 9](#)
 - [AX12, 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, [48](#)
- 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, [13](#)
 - dynamixel, [17](#)
- clear
 - dxl_hal, [13](#)
- close
 - dxl_hal, [13](#)
- connectedID
 - AX12, [9](#)
- controlled
 - ServoThread, [41](#)
- cos60
 - ServoThread, [48](#)
- dH
 - dynamixel, [26](#)
- dxl/ax12.cpp, [49](#)
- dxl/ax12.h, [49](#)
- dxl/dxl_hal.cpp, [49](#)
- dxl/dxl_hal.h, [49](#)
- dxl/dynamixel.cpp, [50](#)
- dxl/dynamixel.h, [50](#)
- dxl_hal, [12](#)
 - _open, [14](#)
 - _serial, [14](#)
 - _time, [14](#)
 - _timed, [14](#)
 - change_baudrate, [13](#)
 - clear, [13](#)
 - close, [13](#)
 - get_curr_time, [13](#)
 - isOpen, [13](#)
 - open, [13](#)
 - read, [14](#)
 - write, [14](#)
- dynamixel, [15](#)
 - change_baudrate, [17](#)
 - dH, [26](#)
 - dynamixel, [17](#)
 - gbCommStatus, [26](#)
 - gbInstructionPacket, [26](#)
 - gbRxGetLength, [26](#)
 - gbRxPacketLength, [26](#)
 - gbStatusPacket, [26](#)
 - gdByteTransTime, [26](#)
 - gdPacketStartTime, [26](#)
 - gdRcvWaitTime, [26](#)
 - get_comm_result, [17](#)
 - get_packet_time, [17](#)
 - get_rxpacket_error, [17](#)
 - 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, 42

events

- OptionsWindow, 35

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, 17

get_curr_time

- dxl_hal, 13

get_packet_time

- dynamixel, 17

get_rxpacket_error

- dynamixel, 17

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, 42

getServoPort

- ServoThread, 42

getServoPortInfo

- ServoThread, 43

getServosInfo

- ServoThread, 43

giBusUsing

- dynamixel, 27

GoalPosition

- AX12, 7

HighestLimitTemp

- AX12, 8

HighestLimitVoltage

- AX12, 8

ID

- AX12, 8
- ServoThread::Servo, 38

initialize

- dynamixel, 19

is_packet_timeout

- dynamixel, 19

isOpen

- dxl_hal, 13
- dynamixel, 20

joyChanged

- MainWindow, 30

joystickChanged

- MainWindow, 30
- OptionsWindow, 35

L1

- ServoThread, 48

L2

- ServoThread, 48

LED

- AX12, 7

load

- ServoThread, 43
- ServoThread::Servo, 38

Lock

- AX12, 7

LowestLimitVoltage

- AX12, 8

main

- main.cpp, 51

main.cpp, 50

main, 51

- MainWindow, 27
 - _axis, 31
 - _axisV, 31
 - _buts, 31
 - _butsV, 31
 - _dataP, 31
 - _jAxisX, 31
 - _jAxisY, 31
 - _jAxisZ, 31
 - _joy, 32
 - _sT, 32
 - _timer, 32
 - ~MainWindow, 29
 - aSCount, 32
 - joyChanged, 30
 - joystickChanged, 30
 - MainWindow, 29
 - on_actionOptions_triggered, 30
 - on_start_clicked, 30
 - sCount, 32
 - ui, 32
 - update, 31
- mainwindow.cpp, 51
- mainwindow.h, 51
- manual
 - ServoThread, 41
- MaxTorque
 - AX12, 8
- Mode
 - ServoThread, 41
- ModelNumber
 - AX12, 8
- Moving
 - AX12, 7
- MovingSpeed
 - AX12, 7
- mutex
 - ServoThread, 43
- on_actionOptions_triggered
 - MainWindow, 30
- on_servoRefresh_clicked
 - OptionsWindow, 36
- on_start_clicked
 - MainWindow, 30
- open
 - dxl_hal, 13
- OptionsWindow, 32
 - _jAxisX, 36
 - _jAxisY, 36
 - _jAxisZ, 36
 - _joy, 36
 - _portSize, 37
 - _servo, 37
 - _timer, 37
 - ~OptionsWindow, 35
 - events, 35
 - joystickChanged, 35
 - on_servoRefresh_clicked, 36
 - OptionsWindow, 34
 - storeData, 36
 - ui, 37
 - optionswindow.cpp, 51
 - optionswindow.h, 51
- pause
 - ServoThread, 44
- ping
 - dynamixel, 20
- pos
 - ServoThread::Servo, 38
- 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
- read_byte
 - dynamixel, 20
- read_word
 - dynamixel, 20
- Registered
 - AX12, 7
- ReturnDelayTime
 - AX12, 8
- run
 - ServoThread, 44
- rx_packet
 - dynamixel, 21
- sCount
 - MainWindow, 32
- Servo
 - ServoThread::Servo, 38
- ServoThread, 38
 - _axis, 47
 - _buts, 47
 - _cBaud, 47
 - _cPort, 47
 - _cond, 47
 - _dChanged, 47
 - _end, 47
 - _mod, 47
 - _mutex, 47
 - _pause, 47

- [_sBaud](#), [48](#)
- [_sPort](#), [48](#)
- [_sPortChanged](#), [48](#)
- [_servos](#), [48](#)
- [~ServoThread](#), [42](#)
- [a](#), [48](#)
- [b](#), [48](#)
- [controlled](#), [41](#)
- [cos60](#), [48](#)
- [end](#), [42](#)
- [getServoBaud](#), [42](#)
- [getServoPort](#), [42](#)
- [getServoPortInfo](#), [43](#)
- [getServosInfo](#), [43](#)
- [L1](#), [48](#)
- [L2](#), [48](#)
- [load](#), [43](#)
- [manual](#), [41](#)
- [Mode](#), [41](#)
- [mutex](#), [43](#)
- [pause](#), [44](#)
- [run](#), [44](#)
- [ServoThread](#), [42](#)
- [setAngles](#), [44](#)
- [setData](#), [45](#)
- [setSID](#), [46](#)
- [setServoBaud](#), [45](#)
- [setServoPort](#), [45](#)
- [setServoPortInfo](#), [45](#)
- [sin60](#), [48](#)
- [singleAngle](#), [46](#)
- [statusBar](#), [46](#)
- [v_1_0](#), [41](#)
- [Version](#), [41](#)
- [wakeUp](#), [46](#)
- [write](#), [46](#)
- [ServoThread::Servo](#), [37](#)
 - [ID](#), [38](#)
 - [load](#), [38](#)
 - [pos](#), [38](#)
 - [Servo](#), [38](#)
- [servothread.cpp](#), [52](#)
- [servothread.h](#), [52](#)
- [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](#), [44](#)
- [setData](#)
 - [ServoThread](#), [45](#)
- [setDxl](#)
 - [AX12](#), [10](#)
- [setGoalPosition](#)
 - [AX12](#), [10](#)
- [setID](#)
 - [AX12](#), [11](#)
- [setJointMode](#)
 - [AX12](#), [11](#)
- [setMinMax](#)
 - [AX12](#), [11](#)
- [setSID](#)
 - [ServoThread](#), [46](#)
- [setServoBaud](#)
 - [ServoThread](#), [45](#)
- [setServoPort](#)
 - [ServoThread](#), [45](#)
- [setServoPortInfo](#)
 - [ServoThread](#), [45](#)
- [setSpeed](#)
 - [AX12](#), [11](#)
- [sin60](#)
 - [ServoThread](#), [48](#)
- [singleAngle](#)
 - [ServoThread](#), [46](#)
- [stable.h](#), [52](#)
- [statusBar](#)
 - [ServoThread](#), [46](#)
- [StatusReturnLevel](#)
 - [AX12](#), [8](#)
- [storeData](#)
 - [OptionsWindow](#), [36](#)
- [terminate](#)
 - [dynamixel](#), [23](#)
- [TorqueEnable](#)
 - [AX12](#), [7](#)
- [TorqueLimit](#)
 - [AX12](#), [7](#)
- [tx_packet](#)
 - [dynamixel](#), [23](#)
- [txrx_packet](#)
 - [dynamixel](#), [24](#)
- [Ui](#), [3](#)
- [ui](#)
 - [MainWindow](#), [32](#)
 - [OptionsWindow](#), [37](#)
- [update](#)
 - [MainWindow](#), [31](#)
- [v_1_0](#)
 - [ServoThread](#), [41](#)
- [Version](#)
 - [ServoThread](#), [41](#)
- [VersionFirmware](#)
 - [AX12](#), [8](#)
- [wakeUp](#)

- ServoThread, [46](#)
- write
 - dxl_hal, [14](#)
 - ServoThread, [46](#)
- write_byte
 - dynamixel, [24](#)
- write_word
 - dynamixel, [24](#)