

DeltaRobot

v0.4

Generated by Doxygen 1.8.9.1

Tue Apr 14 2015 21:18:26

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	8
3.1.3	Constructor & Destructor Documentation	9
3.1.3.1	AX12	9
3.1.3.2	AX12	9
3.1.3.3	~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	10
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	setGoalPosition	10
3.1.4.9	setID	11
3.1.4.10	setJointMode	11
3.1.4.11	setMinMax	11
3.1.4.12	setSpeed	11
3.1.5	Member Data Documentation	12
3.1.5.1	_ID	12
3.1.5.2	_mode	12

3.1.5.3	dxl	12
3.2	dynamixel2::data Struct Reference	12
3.2.1	Detailed Description	12
3.2.2	Member Data Documentation	12
3.2.2.1	iError	12
3.2.2.2	iID	12
3.2.2.3	iLength	12
3.2.2.4	iStartAddr	12
3.2.2.5	pucTable	12
3.3	dxl_hal Class Reference	13
3.3.1	Detailed Description	13
3.3.2	Member Function Documentation	13
3.3.2.1	change_baudrate	13
3.3.2.2	clear	13
3.3.2.3	close	13
3.3.2.4	get_curr_time	14
3.3.2.5	isOpen	14
3.3.2.6	open	14
3.3.2.7	read	14
3.3.2.8	write	14
3.3.3	Member Data Documentation	14
3.3.3.1	_open	15
3.3.3.2	_serial	15
3.3.3.3	_time	15
3.3.3.4	_timed	15
3.4	dynamixel Class Reference	15
3.4.1	Detailed Description	16
3.4.2	Constructor & Destructor Documentation	16
3.4.2.1	dynamixel	16
3.4.2.2	dynamixel	16
3.4.3	Member Function Documentation	16
3.4.3.1	change_baudrate	16
3.4.3.2	get_comm_result	17
3.4.3.3	get_packet_time	17
3.4.3.4	get_rxpacket_error	17
3.4.3.5	get_rxpacket_error_byte	17
3.4.3.6	get_rxpacket_length	17
3.4.3.7	get_rxpacket_parameter	17
3.4.3.8	initialize	17
3.4.3.9	is_packet_timeout	18

3.4.3.10	isOpen	18
3.4.3.11	ping	18
3.4.3.12	read_byte	18
3.4.3.13	read_word	18
3.4.3.14	rx_packet	18
3.4.3.15	set_packet_timeout	20
3.4.3.16	set_packet_timeout_ms	20
3.4.3.17	set_txpacket_id	20
3.4.3.18	set_txpacket_instruction	20
3.4.3.19	set_txpacket_length	20
3.4.3.20	set_txpacket_parameter	20
3.4.3.21	terminate	20
3.4.3.22	tx_packet	21
3.4.3.23	txrx_packet	21
3.4.3.24	write_byte	22
3.4.3.25	write_word	22
3.4.4	Member Data Documentation	22
3.4.4.1	dH	22
3.4.4.2	gbCommStatus	22
3.4.4.3	gbInstructionPacket	22
3.4.4.4	gbRxGetLength	22
3.4.4.5	gbRxPacketLength	22
3.4.4.6	gbStatusPacket	22
3.4.4.7	gdByteTransTime	22
3.4.4.8	gdPacketStartTime	22
3.4.4.9	gdRcvWaitTime	22
3.4.4.10	giBusUsing	22
3.5	dynamixel2 Class Reference	23
3.5.1	Detailed Description	24
3.5.2	Member Typedef Documentation	24
3.5.2.1	PingData	24
3.5.2.2	SyncBulkData	25
3.5.3	Member Function Documentation	25
3.5.3.1	add_stuffing	25
3.5.3.2	broadcast_ping	25
3.5.3.3	change_baudrate	26
3.5.3.4	factory_reset	26
3.5.3.5	get_bulk_read_data_byte	26
3.5.3.6	get_bulk_read_data_dword	26
3.5.3.7	get_bulk_read_data_word	27

3.5.3.8	get_comm_result	27
3.5.3.9	get_packet_time	27
3.5.3.10	get_ping_result	27
3.5.3.11	get_rxpacket_error_byte	27
3.5.3.12	get_rxpacket_length	27
3.5.3.13	get_rxpacket_parameter	27
3.5.3.14	get_sync_read_data_byte	28
3.5.3.15	get_sync_read_data_dword	28
3.5.3.16	get_sync_read_data_word	28
3.5.3.17	initialize	28
3.5.3.18	is_packet_timeout	29
3.5.3.19	isOpen	29
3.5.3.20	ping	29
3.5.3.21	read_byte	29
3.5.3.22	read_dword	30
3.5.3.23	read_word	30
3.5.3.24	reboot	30
3.5.3.25	remove_stuffing	31
3.5.3.26	rx_packet	31
3.5.3.27	set_packet_timeout	32
3.5.3.28	set_packet_timeout_ms	32
3.5.3.29	set_txpacket_id	32
3.5.3.30	set_txpacket_instruction	32
3.5.3.31	set_txpacket_length	33
3.5.3.32	set_txpacket_parameter	33
3.5.3.33	terminate	33
3.5.3.34	tx_packet	33
3.5.3.35	txrx_packet	34
3.5.3.36	write_byte	37
3.5.3.37	write_dword	38
3.5.3.38	write_word	38
3.5.4	Member Data Documentation	38
3.5.4.1	dH	38
3.5.4.2	gbCommStatus	38
3.5.4.3	gbInstructionPacket	38
3.5.4.4	gbRxGetLength	38
3.5.4.5	gbRxPacketLength	38
3.5.4.6	gbStatusPacket	39
3.5.4.7	gBulkData	39
3.5.4.8	gdByteTransTime	39

3.5.4.9	gdPacketStartTime	39
3.5.4.10	gdRcvWaitTime	39
3.5.4.11	giBusUsing	39
3.5.4.12	gPingData	39
3.5.4.13	gSyncData	39
3.6	MainWindow Class Reference	39
3.6.1	Detailed Description	41
3.6.2	Constructor & Destructor Documentation	41
3.6.2.1	MainWindow	41
3.6.2.2	~MainWindow	42
3.6.3	Member Function Documentation	42
3.6.3.1	joyChanged	42
3.6.3.2	joystickChanged	42
3.6.3.3	on_actionOptions_triggered	43
3.6.3.4	update	43
3.6.4	Member Data Documentation	43
3.6.4.1	_axis	43
3.6.4.2	_axisV	43
3.6.4.3	_buts	43
3.6.4.4	_butsV	43
3.6.4.5	_dataP	43
3.6.4.6	_jAxisX	43
3.6.4.7	_jAxisY	43
3.6.4.8	_jAxisZ	44
3.6.4.9	_joy	44
3.6.4.10	_sT	44
3.6.4.11	_timer	44
3.6.4.12	aSCount	44
3.6.4.13	sCount	44
3.6.4.14	ui	44
3.7	OptionsWindow Class Reference	44
3.7.1	Detailed Description	46
3.7.2	Constructor & Destructor Documentation	46
3.7.2.1	OptionsWindow	46
3.7.2.2	~OptionsWindow	47
3.7.3	Member Function Documentation	47
3.7.3.1	events	47
3.7.3.2	joystickChanged	47
3.7.3.3	on_servoRefresh_clicked	48
3.7.3.4	storeData	48

3.7.4	Member Data Documentation	48
3.7.4.1	_joy	48
3.7.4.2	_portSize	48
3.7.4.3	_servo	48
3.7.4.4	_timer	48
3.7.4.5	ui	48
3.8	dynamixel2::ping_data Struct Reference	48
3.8.1	Detailed Description	49
3.8.2	Member Data Documentation	49
3.8.2.1	iFirmVer	49
3.8.2.2	iID	49
3.8.2.3	iModelNo	49
3.9	ServoThread::Servo Struct Reference	49
3.9.1	Detailed Description	49
3.9.2	Constructor & Destructor Documentation	50
3.9.2.1	Servo	50
3.9.2.2	Servo	50
3.9.3	Member Data Documentation	50
3.9.3.1	ID	50
3.9.3.2	load	50
3.9.3.3	pos	50
3.10	ServoThread Class Reference	50
3.10.1	Detailed Description	53
3.10.2	Member Enumeration Documentation	53
3.10.2.1	Mode	53
3.10.2.2	Version	53
3.10.3	Constructor & Destructor Documentation	53
3.10.3.1	ServoThread	53
3.10.3.2	~ServoThread	53
3.10.4	Member Function Documentation	54
3.10.4.1	cont	54
3.10.4.2	end	54
3.10.4.3	getServoBaud	54
3.10.4.4	getServoPort	54
3.10.4.5	getServoPortInfo	54
3.10.4.6	getServosInfo	55
3.10.4.7	getServosInfo	56
3.10.4.8	load	56
3.10.4.9	pause	56
3.10.4.10	run	56

3.10.4.11 setData	57
3.10.4.12 setServoBaud	57
3.10.4.13 setServoPort	57
3.10.4.14 setSID	58
3.10.4.15 write	58
3.10.5 Member Data Documentation	58
3.10.5.1 _axis	58
3.10.5.2 _buts	58
3.10.5.3 _cBaud	58
3.10.5.4 _cond	59
3.10.5.5 _cPort	59
3.10.5.6 _dChanged	59
3.10.5.7 _end	59
3.10.5.8 _mod	59
3.10.5.9 _mutex	59
3.10.5.10 _pause	59
3.10.5.11 _sBaud	59
3.10.5.12 _servos	59
3.10.5.13 _sPort	59
3.10.5.14 _sPortChanged	59
4 File Documentation	61
4.1 dxl/ax12.cpp File Reference	61
4.1.1 Detailed Description	61
4.2 dxl/ax12.h File Reference	62
4.2.1 Detailed Description	63
4.3 dxl/dxl_hal.cpp File Reference	63
4.3.1 Detailed Description	63
4.4 dxl/dxl_hal.h File Reference	63
4.4.1 Detailed Description	65
4.4.2 Macro Definition Documentation	65
4.4.2.1 MAXNUM_RXPACKET	65
4.4.2.2 MAXNUM_TXPACKET	65
4.5 dxl/dynamixel.cpp File Reference	65
4.5.1 Detailed Description	66
4.5.2 Macro Definition Documentation	66
4.5.2.1 LATENCY_TIME	66
4.5.2.2 PING_STATUS_LENGTH	66
4.5.3 Function Documentation	66
4.5.3.1 update_crc	66

4.6	dxl/dynamixel.h File Reference	67
4.6.1	Detailed Description	69
4.6.2	Macro Definition Documentation	70
4.6.2.1	BROADCAST_ID	70
4.6.2.2	COMM_RXCORRUPT	70
4.6.2.3	COMM_RXFAIL	70
4.6.2.4	COMM_RXSUCCESS	70
4.6.2.5	COMM_RXTIMEOUT	70
4.6.2.6	COMM_RXWAITING	70
4.6.2.7	COMM_TXERROR	70
4.6.2.8	COMM_TXFAIL	70
4.6.2.9	COMM_TXSUCCESS	70
4.6.2.10	ERR_ACCESS	70
4.6.2.11	ERR_CRC	70
4.6.2.12	ERR_DATA_LENGTH	70
4.6.2.13	ERR_DATA_LIMIT	70
4.6.2.14	ERR_DATA_RANGE	70
4.6.2.15	ERR_INSTRUCTION	70
4.6.2.16	ERR_RESULT_FAIL	70
4.6.2.17	ERRBIT_ALERT	70
4.6.2.18	HIBYTE	70
4.6.2.19	HIWORD	70
4.6.2.20	INST_ACTION	70
4.6.2.21	INST_BULK_READ	70
4.6.2.22	INST_BULK_WRITE	70
4.6.2.23	INST_PING	70
4.6.2.24	INST_READ	70
4.6.2.25	INST_REBOOT	70
4.6.2.26	INST_REG_WRITE	70
4.6.2.27	INST_RESET	70
4.6.2.28	INST_STATUS	71
4.6.2.29	INST_SYNC_READ	71
4.6.2.30	INST_SYNC_WRITE	71
4.6.2.31	INST_WRITE	71
4.6.2.32	LOBYTE	71
4.6.2.33	LOWORD	71
4.6.2.34	MAKEDWORD	71
4.6.2.35	MAKEWORD	71
4.6.2.36	MAX_ID	71
4.6.2.37	PING_INFO_FIRM_VER	71

4.6.2.38	PING_INFO_MODEL_NUM	71
4.6.2.39	PRT1_PKT_ERRBIT	71
4.6.2.40	PRT1_PKT_ID	71
4.6.2.41	PRT1_PKT_INSTRUCTION	71
4.6.2.42	PRT1_PKT_LENGTH	71
4.6.2.43	PRT1_PKT_PARAMETER0	71
4.6.2.44	PRT2_INSTRUCTION_PKT_PARAMETER0	71
4.6.2.45	PRT2_PKT_ERRBIT	71
4.6.2.46	PRT2_PKT_HEADER0	71
4.6.2.47	PRT2_PKT_HEADER1	71
4.6.2.48	PRT2_PKT_HEADER2	71
4.6.2.49	PRT2_PKT_ID	71
4.6.2.50	PRT2_PKT_INSTRUCTION	71
4.6.2.51	PRT2_PKT_LENGTH_H	71
4.6.2.52	PRT2_PKT_LENGTH_L	71
4.6.2.53	PRT2_PKT_RESERVED	71
4.6.2.54	PRT2_STATUS_PKT_PARAMETER0	71
4.7	main.cpp File Reference	72
4.7.1	Detailed Description	72
4.7.2	Function Documentation	72
4.7.2.1	main	72
4.8	mainwindow.cpp File Reference	72
4.8.1	Detailed Description	73
4.9	mainwindow.h File Reference	73
4.9.1	Detailed Description	74
4.10	optionswindow.cpp File Reference	74
4.10.1	Detailed Description	75
4.11	optionswindow.h File Reference	75
4.11.1	Detailed Description	76
4.12	servothread.cpp File Reference	76
4.13	servothread.h File Reference	76
4.13.1	Detailed Description	77

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

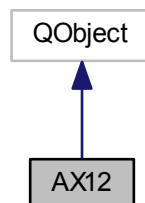
Class Documentation

3.1 AX12 Class Reference

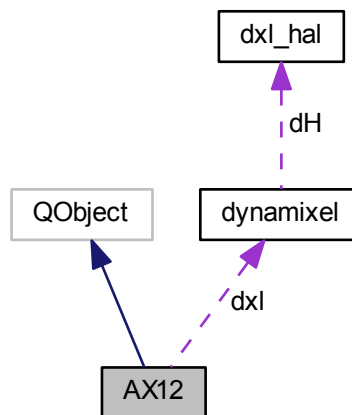
The [AX12](#) class is used to control AX-12 motors from Dynamixel.

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

Inheritance diagram for AX12:



Collaboration diagram for AX12:



Public Member Functions

- [AX12](#) ([dynamixel](#) &[dxi](#), int [ID](#)== -1, [QObject](#) *parent=0)
Default constructor must pass an initialized dynamixel object if ID == -1 no action is done.
- [AX12](#) (const [AX12](#) &a)
Copy constructor.
- [~AX12](#) ()
Default destructor.
- [QVector< int > connectedID](#) ()
Returns all active servos;.
- double [getCurrentLoad](#) ()
Returns the current load from -100% to 100%, 100% is ClockWise and -100% is CounterClockWise.
- double [getCurrentPos](#) ()
Returns the current position from 0° to 300°
- int [getCurrentTemp](#) ()
Returns the current Temperature in Celsius.
- double [getCurrentSpeed](#) ()
Returns the current speed from -100% to 100%, 100% is ClockWise and -100% is CounterClockWise.
- double [getCurrentVoltage](#) ()
Returns the current voltage in Volts.
- int [getID](#) ()
To get the current ID.
- void [setGoalPosition](#) (double goal)
Sets the Goal's position (in degrees) or speed depending on the mode.
- void [setID](#) (int [ID](#))
To set a new ID.
- void [setJointMode](#) (bool mode)
To set Joint/Wheel mode, true if Joint.
- void [setMinMax](#) (double min, double max)
To set the minimum and maximum angle from 0 to 300°

- void [setSpeed](#) (double speed)

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

Private Types

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

Contains all the EEPROM directions enumeration.

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

Contains all the RAM directions enumerations.

Private Attributes

- [dynamixel](#) & [dxl](#)
Contains the dynamixel communication.
- int [_ID](#)
Stores the current ID.
- bool [_mode](#)
True if we use the joint mode.

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

```

00042     {
00043         TorqueEnable      = 24,
00044         LED               = 25,
00045         CWComplianceMargin = 26,
00046         CCWComplianceMargin = 27,
00047         CWComplianceSlope  = 28,
00048         CCWComplianceSlope = 29,
00049         GoalPosition       = 30,
00050         MovingSpeed        = 32,
00051         TorqueLimit        = 34,
00052         PresentPosition    = 36,
00053         PresentSpeed       = 38,
00054         PresentLoad        = 40,
00055         PresentVoltage     = 42,
00056         PresentTemperature = 43,
00057         Registered        = 44,
00058         Moving            = 46,
00059         Lock              = 47,
00060         Punch             = 48
00061     };
00062 
```

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

```

00023     {
00024         ModelNumber      = 0,
00025         VersionFirmware  = 2,
00026         ID               = 3,
00027         BaudRate         = 4,
00028         ReturnDelayTime  = 5,
00029         CWAngleLimit     = 6,
00030         CCWAngleLimit    = 8,
00031         HighestLimitTemp  = 11,
00032         LowestLimitVoltage = 12,
00033         HighestLimitVoltage = 13,
00034         MaxTorque        = 14,
00035         StatusReturnLevel = 16,
00036         AlarmLED         = 17,
00037         AlarmShutdown    = 18
00038     };

```

3.1.3 Constructor & Destructor Documentation

3.1.3.1 AX12::AX12 (dynamixel & dxl, int ID = -1, QObject * parent = 0)

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

```
00005                                     :
00006     QObject (parent),
00007     dxl (dxl),
00008     _ID (ID),
00009     _mode (true)
00010 {
00011     if (_ID < 0) return;
00012     dxl.write_byte(_ID, RAM::TorqueEnable, true);
00013 }
```

3.1.3.2 AX12::AX12 (const AX12 & a)

Copy constructor.

```
00015                                     :
00016     QObject (a.parent()),
00017     dxl (a.dxl),
00018     _ID (a._ID),
00019     _mode (true)
00020 {
00021
00022 }
```

3.1.3.3 AX12::~AX12 ()

Default destructor.

```
00025 {
00026
00027 }
```

3.1.4 Member Function Documentation

3.1.4.1 QVector< int > AX12::connectedID ()

Returns all active servos;

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

3.1.4.2 double AX12::getCurrentLoad ()

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

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

3.1.4.3 double AX12::getCurrentPos ()

Returns the current position from 0° to 300°

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

3.1.4.4 double AX12::getCurrentSpeed ()

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

```
00066 {
00067     if (_ID < 0) return 0;
00068     int speed = dxl.read_word(_ID, RAM::PresentSpeed);
00069     if (dxl.get_comm_result() != COMM_RXSUCCESS) return -1;
00070     speed -= 1024;
00071     if (speed == -1024) speed = 0;
00072     return double((speed/1023.0)*100);
00073 }
```

3.1.4.5 int AX12::getCurrentTemp ()

Returns the current Temperature in Celsius.

```
00058 {
00059     if (_ID < 0) return 0;
00060     int temp = dxl.read_byte(_ID, RAM::PresentTemperature);
00061     if (dxl.get_comm_result() != COMM_RXSUCCESS) return -1;
00062     return temp;
00063 }
```

3.1.4.6 double AX12::getCurrentVoltage ()

Returns the current voltage in Volts.

```
00076 {
00077     if (_ID < 0) return 0;
00078     char voltage = dxl.read_byte(_ID, RAM::PresentVoltage);
00079     if (dxl.get_comm_result() != COMM_RXSUCCESS) return -1;
00080     return double(voltage/10.0);
00081 }
```

3.1.4.7 int AX12::getID () [inline]

To get the current ID.

```
00106 { return _ID; }
```

3.1.4.8 void AX12::setGoalPosition (double goal)

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

```
00084 {
00085     if (_ID < 0) return;
00086     if (goal > 300.0) goal = 300.0;
00087     else if (goal < 0) goal = 0;
00088     dxl.write_word(_ID, RAM::GoalPosition, int((goal/300.0)*1023));
00089 }
```

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

To set a new ID.

```
00092 {
00093     _ID = ID;
00094     if (ID < 0) return;
00095     dxl.write_byte(_ID, RAM::TorqueEnable, true);
00096 }
```

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

To set Joint/Wheel mode, true if Joint.

```
00099 {
00100     if (_ID < 0) return;
00101     _mode = mode;
00102     if (_mode) {
00103         dxl.write_word(_ID, ROM::CWAngleLimit, 0);
00104         dxl.write_word(_ID, ROM::CCWAngleLimit, 1023);
00105     }
00106     else {
00107         dxl.write_word(_ID, ROM::CWAngleLimit, 0);
00108         dxl.write_word(_ID, ROM::CCWAngleLimit, 0);
00109     }
00110 }
```

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

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

```
00113 {
00114     if (_ID < 0) return;
00115
00116     if (min > max) std::swap(min, max);
00117
00118     if (min < 0.0) min = 0;
00119     if (max > 300.0) max = 300;
00120
00121     min = (min/300)*1023;
00122     max = (max/300)*1023;
00123
00124     dxl.write_word(_ID, ROM::CWAngleLimit, int (min));
00125     dxl.write_word(_ID, ROM::CCWAngleLimit, int (max));
00126 }
```

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

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

```
00129 {
00130     if (_ID < 0) return;
00131     if (speed > 100.0) speed = 100.0;
00132     if (_mode) {
00133         if (speed < 0.0) speed = 0.0;
00134
00135         int byte = int((speed/100.0) * 1024.0);
00136         if (speed == 100.0) byte = 0;
00137         dxl.write_byte(_ID, RAM::MovingSpeed, byte);
00138     }
00139     else {
00140         if (speed < -100.0) speed = -100.0;
00141
00142         int byte = int((speed + 100)/100.0 * 1024);
00143         dxl.write_byte(_ID, RAM::MovingSpeed, byte);
00144     }
00145 }
00146 }
```

3.1.5 Member Data Documentation

3.1.5.1 `int AX12::_ID` `[private]`

Stores the current ID.

3.1.5.2 `bool AX12::_mode` `[private]`

True if we use the joint mode.

3.1.5.3 `dynamixel& AX12::dxl` `[private]`

Contains the dynamixel communication.

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

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

3.2 `dynamixel2::data` Struct Reference

Struct used to handle dynamixel data.

Public Attributes

- unsigned char `iID`
- unsigned int `iStartAddr`
- unsigned short `iLength`
- unsigned char `iError`
- unsigned char * `pucTable`

3.2.1 Detailed Description

Struct used to handle dynamixel data.

3.2.2 Member Data Documentation

3.2.2.1 `unsigned char dynamixel2::data::iError`

3.2.2.2 `unsigned char dynamixel2::data::iID`

3.2.2.3 `unsigned short dynamixel2::data::iLength`

3.2.2.4 `unsigned int dynamixel2::data::iStartAddr`

3.2.2.5 `unsigned char* dynamixel2::data::pucTable`

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

- [dxl/dynamixel.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:

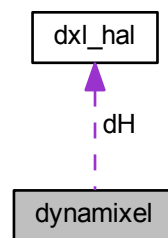
- [dxl/dxl_hal.h](#)
- [dxl/dxl_hal.cpp](#)

3.4 dynamixel Class Reference

Dynamixel 1.0 protocol class.

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

Collaboration diagram for dynamixel:



Public Member Functions

- [dynamixel](#) ()
- [dynamixel](#) (QString port_num, int baud_rate=1000000)
- `bool isOpen` ()
- `int initialize` (QString port_num, int baud_rate)
- `int change_baudrate` (int baud_rate)
- `int terminate` (void)
- `int get_comm_result` ()
- `void tx_packet` (void)
- `void rx_packet` (void)
- `void txrx_packet` (void)
- `void set_txpacket_id` (int id)
- `void set_txpacket_instruction` (int instruction)
- `void set_txpacket_parameter` (int index, int value)
- `void set_txpacket_length` (int length)
- `int get_rxpacket_error` (int error)
- `int get_rxpacket_error_byte` (void)

- int [get_rxpacket_parameter](#) (int index)
- int [get_rxpacket_length](#) ()
- void [ping](#) (int id)
- int [read_byte](#) (int id, int address)
- void [write_byte](#) (int id, int address, int value)
- int [read_word](#) (int id, int address)
- void [write_word](#) (int id, int address, int value)
- double [get_packet_time](#) ()
- void [set_packet_timeout](#) (int NumRcvByte)
- void [set_packet_timeout_ms](#) (int msec)
- int [is_packet_timeout](#) ()

Private Attributes

- [dxl_hal](#) dH
- unsigned char [gbInstructionPacket](#) [[MAXNUM_TXPACKET](#)] = {0}
- unsigned char [gbStatusPacket](#) [[MAXNUM_RXPACKET](#)] = {0}
- unsigned int [gbRxPacketLength](#) = 0
- unsigned int [gbRxGetLength](#) = 0
- double [gdPacketStartTime](#) = 0.0
- double [gdByteTransTime](#) = 0.0
- double [gdRcvWaitTime](#) = 0.0
- int [gbCommStatus](#) = [COMM_RXSUCCESS](#)
- int [giBusUsing](#) = 0

3.4.1 Detailed Description

Dynamixel 1.0 protocol class.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 dynamixel::dynamixel ()

```
00014 {
00015
00016 }
```

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

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

3.4.3 Member Function Documentation

3.4.3.1 int dynamixel::change_baudrate (int baud_rate)

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

3.4.3.2 int dynamixel::get_comm_result() [inline]

```
00112 { return gbCommStatus; }
```

3.4.3.3 double dynamixel::get_packet_time(void)

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

3.4.3.4 int dynamixel::get_rxpacket_error(int error)

```
00279 {  
00280     if( gbStatusPacket[PRT1_PKT_ERRBIT] & (unsigned char)error )  
00281         return 1;  
00282  
00283     return 0;  
00284 }
```

3.4.3.5 int dynamixel::get_rxpacket_error_byte(void)

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

3.4.3.6 int dynamixel::get_rxpacket_length()

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

3.4.3.7 int dynamixel::get_rxpacket_parameter(int index)

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

3.4.3.8 int dynamixel::initialize(QString port_num, int baud_rate)

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

3.4.3.9 int dynamixel::is_packet_timeout (void)

```

00082 {
00083     if(this->get_packet_time() > gdRcvWaitTime)
00084         return 1;
00085     return 0;
00086 }

```

3.4.3.10 bool dynamixel::isOpen () [inline]

```

00103 { return dH.isOpen(); }

```

3.4.3.11 void dynamixel::ping (int id)

```

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

```

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

```

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

```

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

```

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

```

3.4.3.14 void dynamixel::rx_packet (void)

```

00152 {
00153     unsigned char i = 0, j = 0, nRead = 0;

```

```

00154     unsigned char checksum = 0;
00155
00156     if( giBusUsing == 0 )
00157         return;
00158
00159     if( gbInstructionPacket[PRT1_PKT_ID] ==
BROADCAST_ID )
00160     {
00161         gbCommStatus = COMM_RXSUCCESS;
00162         giBusUsing = 0;
00163         return;
00164     }
00165
00166     if( gbCommStatus == COMM_TXSUCCESS )
00167     {
00168         gbRxGetLength = 0;
00169         //gbRxPacketLength = 6; //minimum wait length
00170     }
00171
00172     while(1)
00173     {
00174         nRead = dH.read( &gbStatusPacket[gbRxGetLength],
gbRxPacketLength - gbRxGetLength );
00175         gbRxGetLength += nRead;
00176
00177         if(gbRxGetLength > 4)
00178             gbRxPacketLength = gbStatusPacket[
PRT1_PKT_LENGTH] + 4;
00179
00180         if( gbRxGetLength < gbRxPacketLength )
00181         {
00182             if( is_packet_timeout() == 1 )
00183             {
00184                 if(gbRxGetLength == 0)
00185                     gbCommStatus = COMM_RXTIMEOUT;
00186                 else
00187                     gbCommStatus = COMM_RXCORRUPT;
00188                 giBusUsing = 0;
00189                 return;
00190             }
00191             gbCommStatus = COMM_RXWAITING;
00192             //return;
00193         }
00194         else
00195         {
00196             break;
00197         }
00198     }
00199
00200     // Find packet header
00201     for( i=0; i<(gbRxGetLength-1); i++ )
00202     {
00203         if( gbStatusPacket[i] == 0xff && gbStatusPacket[i+1] == 0xff )
00204             break;
00205         else if( i == gbRxGetLength-2 && gbStatusPacket[gbRxGetLength-1] == 0xff )
00206             break;
00207         else {
00208             gbCommStatus = COMM_RXCORRUPT;
00209             return;
00210         }
00211     }
00212
00213     if( i > 0 )
00214     {
00215         for( j=0; j<(gbRxGetLength-i); j++ )
00216             gbStatusPacket[j] = gbStatusPacket[j + i];
00217
00218         gbRxGetLength -= i;
00219     }
00220
00221     // Check id pairing
00222     if( gbInstructionPacket[PRT1_PKT_ID] !=
gbStatusPacket[PRT1_PKT_ID] )
00223     {
00224         gbCommStatus = COMM_RXCORRUPT;
00225         giBusUsing = 0;
00226         return;
00227     }
00228
00229     // Check checksum
00230     for( i=0; i<(gbStatusPacket[PRT1_PKT_LENGTH]+1); i++ )
00231         checksum += gbStatusPacket[i+2];
00232     checksum = ~checksum;
00233
00234     if( gbStatusPacket[gbStatusPacket[
PRT1_PKT_LENGTH]+3] != checksum )
00235     {

```

```

00236         gbCommStatus = COMM_RXCORRUPT;
00237         giBusUsing = 0;
00238         return;
00239     }
00240
00241     gbCommStatus = COMM_RXSUCCESS;
00242     giBusUsing = 0;
00243 }

```

3.4.3.15 void dynamixel::set_packet_timeout (int NumRcvByte)

```

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

```

3.4.3.16 void dynamixel::set_packet_timeout_ms (int msec)

```

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

```

3.4.3.17 void dynamixel::set_txpacket_id (int id)

```

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

```

3.4.3.18 void dynamixel::set_txpacket_instruction (int instruction)

```

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

```

3.4.3.19 void dynamixel::set_txpacket_length (int length)

```

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

```

3.4.3.20 void dynamixel::set_txpacket_parameter (int index, int value)

```

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

```

3.4.3.21 int dynamixel::terminate (void)

```

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

```


3.4.3.22 void dynamixel::tx_packet (void)

```

00090 {
00091     unsigned char pkt_idx = 0;
00092     unsigned char TxNumByte, RealTxNumByte;
00093     unsigned char checksum = 0;
00094
00095     if( giBusUsing == 1 )
00096     {
00097         gbCommStatus = COMM_TXFAIL;
00098         return;
00099     }
00100
00101     giBusUsing = 1;
00102
00103     if( gbInstructionPacket[PRT1_PKT_INSTRUCTION] !=
INST_READ
00104         && gbInstructionPacket[PRT1_PKT_INSTRUCTION] !=
INST_WRITE
00105         && gbInstructionPacket[PRT1_PKT_INSTRUCTION] !=
INST_REG_WRITE
00106         && gbInstructionPacket[PRT1_PKT_INSTRUCTION] !=
INST_ACTION
00107         && gbInstructionPacket[PRT1_PKT_INSTRUCTION] !=
INST_RESET
00108         && gbInstructionPacket[PRT1_PKT_INSTRUCTION] !=
INST_SYNC_WRITE )
00109     {
00110         gbCommStatus = COMM_TXERROR;
00111         giBusUsing = 0;
00112         return;
00113     }
00114
00115     gbInstructionPacket[0] = 0xff;
00116     gbInstructionPacket[1] = 0xff;
00117     for( pkt_idx = 0; pkt_idx < (gbInstructionPacket[
PRT1_PKT_LENGTH]+1); pkt_idx++ )
00118         checksum += gbInstructionPacket[pkt_idx+2];
00119     gbInstructionPacket[gbInstructionPacket[
PRT1_PKT_LENGTH]+3] = ~checksum;
00120
00121     //if( gbCommStatus == COMM_RXTIMEOUT || gbCommStatus == COMM_RXCORRUPT )
00122     //    dH.clear();
00123
00124     dH.clear();
00125
00126     TxNumByte = gbInstructionPacket[PRT1_PKT_LENGTH] + 4;
00127     RealTxNumByte = dH.write( gbInstructionPacket, TxNumByte );
00128
00129     if( TxNumByte != RealTxNumByte )
00130     {
00131         gbCommStatus = COMM_TXFAIL;
00132         giBusUsing = 0;
00133         return;
00134     }
00135
00136     if( gbInstructionPacket[PRT1_PKT_INSTRUCTION] ==
INST_READ )
00137     {
00138         gbRxPacketLength = gbInstructionPacket[
PRT1_PKT_PARAMETER0+1] + 6;
00139         set_packet_timeout( gbInstructionPacket[
PRT1_PKT_PARAMETER0+1] + 6 );
00140     }
00141     else
00142     {
00143         gbRxPacketLength = 6;
00144         set_packet_timeout( 6 );
00145     }
00146
00147     gbCommStatus = COMM_TXSUCCESS;
00148 }
00149

```

3.4.3.23 void dynamixel::txrx_packet (void)

```

00246 {
00247     tx_packet();
00248
00249     if( gbCommStatus != COMM_TXSUCCESS )
00250         return;
00251

```

```

00252
00253     rx_packet ();
00254 }

```

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

```

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

```

3.4.3.25 void dynamixel::write_word (int id, int address, int value)

```

00356 {
00357     while(giBusUsing);
00358
00359     gbInstructionPacket[PRT1_PKT_ID] = (unsigned char)id;
00360     gbInstructionPacket[PRT1_PKT_INSTRUCTION] =
INST_WRITE;
00361     gbInstructionPacket[PRT1_PKT_PARAMETER0+0] = (unsigned char)
address;
00362     gbInstructionPacket[PRT1_PKT_PARAMETER0+1] = (unsigned char)
LOBYTE(value);
00363     gbInstructionPacket[PRT1_PKT_PARAMETER0+2] = (unsigned char)
HIBYTE(value);
00364     gbInstructionPacket[PRT1_PKT_LENGTH] = 5;
00365
00366     txrx_packet ();
00367 }

```

3.4.4 Member Data Documentation

3.4.4.1 dxl_hal dynamixel::dH [private]

3.4.4.2 int dynamixel::gbCommStatus = COMM_RXSUCCESS [private]

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

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

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

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

3.4.4.7 double dynamixel::gdByteTransTime = 0.0 [private]

3.4.4.8 double dynamixel::gdPacketStartTime = 0.0 [private]

3.4.4.9 double dynamixel::gdRcvWaitTime = 0.0 [private]

3.4.4.10 int dynamixel::giBusUsing = 0 [private]

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

- [dxl/dynamixel.h](#)

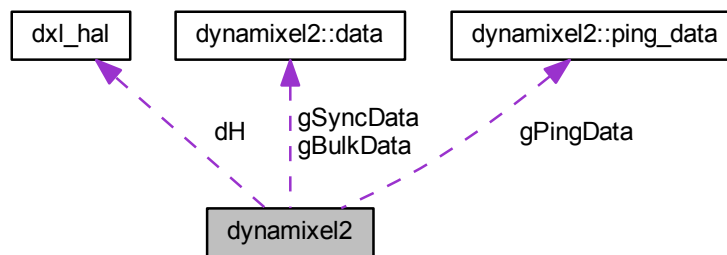
- [dxl/dynamixel.cpp](#)

3.5 dynamixel2 Class Reference

Dynamixel 2.0 protocol class.

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

Collaboration diagram for dynamixel2:



Classes

- struct [data](#)
Struct used to handle dynamixel data.
- struct [ping_data](#)
Struct used to do a ping.

Public Member Functions

- bool [isOpen](#) ()
- int [initialize](#) (QString port_num, int baud_rate)
- int [change_baudrate](#) (int baud_rate)
- int [terminate](#) (void)
- int [get_comm_result](#) (void)
- void [tx_packet](#) (void)
- void [rx_packet](#) (void)
- void [txrx_packet](#) (void)
- void [set_txpacket_id](#) (unsigned char id)
- void [set_txpacket_instruction](#) (unsigned char instruction)
- void [set_txpacket_parameter](#) (unsigned short index, unsigned char value)
- void [set_txpacket_length](#) (unsigned short length)
- int [get_rxpacket_error_byte](#) (void)
- int [get_rxpacket_parameter](#) (int index)
- int [get_rxpacket_length](#) ()
- void [ping](#) (unsigned char id)
- int [get_ping_result](#) (unsigned char id, int info_num)
- void [broadcast_ping](#) ()
- void [reboot](#) (unsigned char id)

- void [factory_reset](#) (unsigned char id, int option)
- unsigned char [read_byte](#) (unsigned char id, int address)
- void [write_byte](#) (unsigned char id, int address, unsigned char value)
- unsigned short [read_word](#) (unsigned char id, int address)
- void [write_word](#) (unsigned char id, int address, unsigned short value)
- unsigned long [read_dword](#) (unsigned char id, int address)
- void [write_dword](#) (unsigned char id, int address, unsigned long value)
- unsigned char [get_bulk_read_data_byte](#) (unsigned char id, unsigned int start_address)
- unsigned short [get_bulk_read_data_word](#) (unsigned char id, unsigned int start_address)
- unsigned long [get_bulk_read_data_dword](#) (unsigned char id, unsigned int start_address)
- unsigned char [get_sync_read_data_byte](#) (unsigned char id, unsigned int start_address)
- unsigned short [get_sync_read_data_word](#) (unsigned char id, unsigned int start_address)
- unsigned long [get_sync_read_data_dword](#) (unsigned char id, unsigned int start_address)
- void [add_stuffing](#) ()
- void [remove_stuffing](#) ()
- double [get_packet_time](#) ()
- int [is_packet_timeout](#) ()
- void [set_packet_timeout](#) (int NumRcvByte)
- void [set_packet_timeout_ms](#) (int msec)

Private Types

- typedef struct [dynamixel2::ping_data](#) PingData
Struct used to do a ping.
- typedef struct [dynamixel2::data](#) SyncBulkData
Struct used to handle dynamixel data.

Private Attributes

- unsigned char [gbInstructionPacket](#) [MAXNUM_TXPACKET] = {0}
- unsigned char [gbStatusPacket](#) [MAXNUM_RXPACKET] = {0}
- unsigned int [gbRxPacketLength](#) = 0
- unsigned int [gbRxGetLength](#) = 0
- double [gdPacketStartTime](#) = 0.0
- double [gdByteTransTime](#) = 0.0
- double [gdRcvWaitTime](#) = 0.0
- int [gbCommStatus](#) = COMM_RXSUCCESS
- int [giBusUsing](#) = 0
- [dxl_hal](#) dH
- [PingData](#) gPingData [MAX_ID+1]
- [SyncBulkData](#) gSyncData [MAX_ID+1]
- [SyncBulkData](#) gBulkData [MAX_ID+1]

3.5.1 Detailed Description

Dynamixel 2.0 protocol class.

3.5.2 Member Typedef Documentation

3.5.2.1 typedef struct [dynamixel2::ping_data](#) [dynamixel2::PingData](#) [private]

Struct used to do a ping.

3.5.2.2 typedef struct dynamixel2::data dynamixel2::SyncBulkData [private]

Struct used to handle dynamixel data.

3.5.3 Member Function Documentation

3.5.3.1 void dynamixel2::add_stuffing ()

```

00524 {
00525     int i = 0, index = 0;
00526     int packet_length_in = MAKEWORD(gbInstructionPacket[
PRT2_PKT_LENGTH_L], gbInstructionPacket[
PRT2_PKT_LENGTH_H]);
00527     int packet_length_out = packet_length_in;
00528     unsigned char temp[MAXNUM_TXPACKET] = {0};
00529
00530     memcpy(temp, gbInstructionPacket, PRT2_PKT_LENGTH_H+1);    // FF FF
FD XX ID LEN_L LEN_H
00531     index = PRT2_PKT_INSTRUCTION;
00532     for( i = 0; i < packet_length_in - 2; i++)    // except CRC
00533     {
00534         if((index - 1) == MAXNUM_TXPACKET) {
00535             gbCommStatus = COMM_TXERROR;
00536             return;
00537         }
00538         temp[index++] = gbInstructionPacket[i+
PRT2_PKT_INSTRUCTION];
00539
00540         if(gbInstructionPacket[i+PRT2_PKT_INSTRUCTION] == 0xFD &&
gbInstructionPacket[i+PRT2_PKT_INSTRUCTION-1] == 0xFF &&
gbInstructionPacket[i+PRT2_PKT_INSTRUCTION-2] == 0xFF)
00541         {
00542             if((index - 1) == MAXNUM_TXPACKET) {
00543                 gbCommStatus = COMM_TXERROR;
00544                 return;
00545             }
00546             // FF FF FD
00547             temp[index++] = 0xFD;
00548
00549             packet_length_out++;
00550         }
00551     }
00552
00553     if((index - 1) == MAXNUM_TXPACKET) {
00554         gbCommStatus = COMM_TXERROR;
00555         return;
00556     }
00557     temp[index++] = gbInstructionPacket[PRT2_PKT_INSTRUCTION+
packet_length_in-2];
00558
00559     if((index - 1) == MAXNUM_TXPACKET) {
00560         gbCommStatus = COMM_TXERROR;
00561         return;
00562     }
00563     temp[index++] = gbInstructionPacket[PRT2_PKT_INSTRUCTION+
packet_length_in-1];
00564
00565     memcpy(gbInstructionPacket, temp, index);
00566     gbInstructionPacket[PRT2_PKT_LENGTH_L] =
LOBYTE(packet_length_out);
00567     gbInstructionPacket[PRT2_PKT_LENGTH_H] =
HIBYTE(packet_length_out);
00568 }

```

3.5.3.2 void dynamixel2::broadcast_ping ()

```

01049 {
01050     int idx = 0;
01051
01052     gbCommStatus = COMM_TXFAIL;
01053
01054     gbInstructionPacket[PRT2_PKT_ID] =
BROADCAST_ID;
01055     gbInstructionPacket[PRT2_PKT_LENGTH_L] = 0x03;
01056     gbInstructionPacket[PRT2_PKT_LENGTH_H] = 0x00;
01057     gbInstructionPacket[PRT2_PKT_INSTRUCTION] =
INST_PING;
01058 }

```

```

01059     for(idx = 1; idx <= MAX_ID; idx++)
01060     {
01061         gPingData[idx].iID = idx;
01062         gPingData[idx].iFirmVer = -1;
01063         gPingData[idx].iModelNo = -1;
01064     }
01065
01066     txrx_packet();
01067 }

```

3.5.3.3 int dynamixel2::change_baudrate (int *baud_rate*)

```

00411 {
00412     int result = 0;
00413     float baudrate = (float)baud_rate;
00414
00415     result = dH.change_baudrate(baudrate);
00416     if(result == 1)
00417         gByteTransTime = 1000.0f / baudrate * 10.0; // 1000/baudrate(bit per msec) *
10(start bit + data bit + stop bit)
00418
00419     return result;
00420 }

```

3.5.3.4 void dynamixel2::factory_reset (unsigned char *id*, int *option*)

```

01086 {
01087     if(id == BROADCAST_ID)
01088     {
01089         gbCommStatus = COMM_TXERROR;
01090         return;
01091     }
01092
01093     gbCommStatus = COMM_TXFAIL;
01094
01095     gbInstructionPacket[PRT2_PKT_ID] = (unsigned char)id;
01096     gbInstructionPacket[PRT2_PKT_LENGTH_L] = 0x04;
01097     gbInstructionPacket[PRT2_PKT_LENGTH_H] = 0x00;
01098     gbInstructionPacket[PRT2_PKT_INSTRUCTION] =
INST_RESET;
01099     gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0]
= (unsigned char)option;
01100
01101     txrx_packet();
01102 }

```

3.5.3.5 unsigned char dynamixel2::get_bulk_read_data_byte (unsigned char *id*, unsigned int *start_address*)

```

01215 {
01216     if((start_address < gBulkData[id].iStartAddr) || ((gBulkData[id].iStartAddr +
gBulkData[id].iLength-1) < start_address))
01217         return 0;
01218     return gBulkData[id].pucTable[(start_address-gBulkData[id].
iStartAddr)];
01219 }

```

3.5.3.6 unsigned long dynamixel2::get_bulk_read_data_dword (unsigned char *id*, unsigned int *start_address*)

```

01230 {
01231     if((start_address < gBulkData[id].iStartAddr) || ((gBulkData[id].iStartAddr +
gBulkData[id].iLength-1) < start_address))
01232         return 0;
01233     return MAKEDWORD(MAKEWORD(gBulkData[id].pucTable[(start_address-
gBulkData[id].iStartAddr)],
gBulkData[id].pucTable[(start_address-
gBulkData[id].iStartAddr+1)]),
MAKEWORD(gBulkData[id].pucTable[(start_address-
gBulkData[id].iStartAddr+2)],
gBulkData[id].pucTable[(start_address-
gBulkData[id].iStartAddr+3)]));
01237 }

```

3.5.3.7 unsigned short dynamixel2::get_bulk_read_data_word (unsigned char *id*, unsigned int *start_address*)

```

01222 {
01223     if( (start_address < gBulkData[id].iStartAddr) || ((gBulkData[id].iStartAddr +
gBulkData[id].iLength-1) < start_address))
01224         return 0;
01225     return MAKEWORD(gBulkData[id].pucTable[(start_address-
gBulkData[id].iStartAddr)],
01226                   gBulkData[id].pucTable[(start_address-gBulkData[id].iStartAddr+1)
]);
01227 }

```

3.5.3.8 int dynamixel2::get_comm_result (void)**3.5.3.9 double dynamixel2::get_packet_time (void)**

```

00439 {
00440     double elapsed_time;
00441
00442     elapsed_time = (double) (dH.get_curr_time() -
gdPacketStartTime);
00443
00444     // Overflow
00445     if(elapsed_time < 0)
00446         gdPacketStartTime = dH.get_curr_time();
00447
00448     return elapsed_time;
00449 }

```

3.5.3.10 int dynamixel2::get_ping_result (unsigned char *id*, int *info_num*)

```

01036 {
01037     if(id <= MAX_ID && gPingData[id].iModelNo != -1 && gPingData[id].iFirmVer != -1
)
01038     {
01039         if(info_num == PING_INFO_MODEL_NUM )
01040             return gPingData[id].iModelNo;
01041         else if(info_num == PING_INFO_FIRM_VER)
01042             return gPingData[id].iFirmVer;
01043     }
01044
01045     return 0;
01046 }

```

3.5.3.11 int dynamixel2::get_rxpacket_error_byte (void)

```

00999 {
01000     return gbStatusPacket[PRT2_PKT_ERRBIT];
01001 }

```

3.5.3.12 int dynamixel2::get_rxpacket_length ()

```

01009 {
01010     return (int)MAKEWORD(gbStatusPacket[PRT2_PKT_LENGTH_L],
gbStatusPacket[PRT2_PKT_LENGTH_H]);
01011 }

```

3.5.3.13 int dynamixel2::get_rxpacket_parameter (int *index*)

```

01004 {
01005     return (int)gbStatusPacket[PRT2_STATUS_PKT_PARAMETER0+index];
01006 }

```

3.5.3.14 unsigned char dynamixel2::get_sync_read_data_byte (unsigned char *id*, unsigned int *start_address*)

```

01240 {
01241     if((start_address < gSyncData[id].iStartAddr) || ((gSyncData[id].iStartAddr +
gSyncData[id].iLength-1) < start_address))
01242         return 0;
01243     return gBulkData[id].pucTable[(start_address-gSyncData[id].
iStartAddr)];
01244 }
```

3.5.3.15 unsigned long dynamixel2::get_sync_read_data_dword (unsigned char *id*, unsigned int *start_address*)

```

01255 {
01256     if((start_address < gSyncData[id].iStartAddr) || ((gSyncData[id].iStartAddr +
gSyncData[id].iLength-1) < start_address))
01257         return 0;
01258     return MAKEDWORD(MAKEWORD(gSyncData[id].pucTable[(start_address-
gSyncData[id].iStartAddr)+0],
gSyncData[id].pucTable[(start_address-
gSyncData[id].iStartAddr)+1]),
01260     MAKEWORD(gSyncData[id].pucTable[(start_address-
gSyncData[id].iStartAddr)+2],
gSyncData[id].pucTable[(start_address-
gSyncData[id].iStartAddr)+3]));
01262 }
```

3.5.3.16 unsigned short dynamixel2::get_sync_read_data_word (unsigned char *id*, unsigned int *start_address*)

```

01247 {
01248     if((start_address < gSyncData[id].iStartAddr) || ((gSyncData[id].iStartAddr +
gSyncData[id].iLength-1) < start_address))
01249         return 0;
01250     return MAKEDWORD(gSyncData[id].pucTable[(start_address-
gSyncData[id].iStartAddr)],
01251     gSyncData[id].pucTable[(start_address-gSyncData[id].iStartAddr+1)
]);
01252 }
```

3.5.3.17 int dynamixel2::initialize (QString *port_num*, int *baud_rate*)

```

00373 {
00374     unsigned int idx = 0;
00375
00376     if( baud_rate < 1900 )
00377         return 0;
00378
00379     if( dH.open(port_num, baud_rate) == 0 )
00380         return 0;
00381
00382     gdByteTransTime = 1000.0 / (double)baud_rate * 10.0; // 1000/baudrate(bit per msec) *
10(start bit + data bit + stop bit)
00383
00384
00385     for(idx = 1; idx <= MAX_ID; idx++)
00386     {
00387         gSyncData[idx].iID = idx;
00388         gSyncData[idx].iStartAddr = 1;
00389         gSyncData[idx].iLength = 1;
00390         gSyncData[idx].iError = 0;
00391         gSyncData[idx].pucTable = 0;
00392
00393         gBulkData[idx].iID = idx;
00394         gBulkData[idx].iStartAddr = 1;
00395         gBulkData[idx].iLength = 1;
00396         gBulkData[idx].iError = 0;
00397         gBulkData[idx].pucTable = 0;
00398
00399         gPingData[idx].iID = idx;
00400         gPingData[idx].iFirmVer = -1;
00401         gPingData[idx].iModelNo = -1;
00402     }
00403
00404     gbCommStatus = COMM_RXSUCCESS;
00405     giBusUsing = 0;
00406 }
```



```
00407     return 1;
00408 }
```

3.5.3.18 int dynamixel2::is_packet_timeout (void)

```
00452 {
00453     if(this->get_packet_time() > gdRcvWaitTime)
00454         return 1;
00455     return 0;
00456 }
```

3.5.3.19 bool dynamixel2::isOpen () [inline]

```
00189 { return dh.isOpen(); }
```

3.5.3.20 void dynamixel2::ping (unsigned char id)

```
01014 {
01015     gbCommStatus = COMM_TXFAIL;
01016
01017     gbInstructionPacket[PRT2_PKT_ID] = (unsigned char)id;
01018     gbInstructionPacket[PRT2_PKT_LENGTH_L] = 0x03;
01019     gbInstructionPacket[PRT2_PKT_LENGTH_H] = 0x00;
01020     gbInstructionPacket[PRT2_PKT_INSTRUCTION] =
INST_READ;
01021
01022     gPingData[id].iModelNo = -1;
01023     gPingData[id].iFirmVer = -1;
01024
01025     txrx_packet();
01026
01027     if( (id != BROADCAST_ID) && (gbCommStatus ==
COMM_RXSUCCESS) )
01028     {
01029         gPingData[id].iID = id;
01030         gPingData[id].iModelNo = MAKEWORD(
gbStatusPacket[PRT1_PKT_PARAMETER0+1],
gbStatusPacket[PRT1_PKT_PARAMETER0+2] );
01031         gPingData[id].iFirmVer = gbStatusPacket[
PRT1_PKT_PARAMETER0+3];
01032     }
01033 }
```

3.5.3.21 unsigned char dynamixel2::read_byte (unsigned char id, int address)

```
01105 {
01106     unsigned short length = 1;
01107     gbCommStatus = COMM_TXFAIL;
01108
01109     gbInstructionPacket[PRT2_PKT_ID] = id;
01110     gbInstructionPacket[PRT2_PKT_LENGTH_L] = 0x07;
01111     gbInstructionPacket[PRT2_PKT_LENGTH_H] = 0x00;
01112     gbInstructionPacket[PRT2_PKT_INSTRUCTION] =
INST_READ;
01113     gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+0]
] = (unsigned char)LOBYTE(address);
01114     gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+1]
] = (unsigned char)HIBYTE(address);
01115     gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+2]
] = (unsigned char)LOBYTE(length);
01116     gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+3]
] = (unsigned char)HIBYTE(length);
01117
01118     txrx_packet();
01119     //if(gbCommStatus == COMM_RXSUCCESS && id != BROADCAST_ID)
01120     //    memmove(data, &rxpacket[PKT_PARAMETER+1], length);
01121
01122     return gbStatusPacket[PRT2_STATUS_PKT_PARAMETER0];
01123 }
```

3.5.3.22 unsigned long dynamixel2::read_dword (unsigned char *id*, int *address*)

```

01176 {
01177     unsigned short length = 4;
01178     gbCommStatus = COMM_TXFAIL;
01179
01180     gbInstructionPacket[PRT2_PKT_ID] = id;
01181     gbInstructionPacket[PRT2_PKT_LENGTH_L] = 0x07;
01182     gbInstructionPacket[PRT2_PKT_LENGTH_H] = 0x00;
01183     gbInstructionPacket[PRT2_PKT_INSTRUCTION] =
INST_READ;
01184     gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+0
] = LOBYTE(address);
01185     gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+1
] = HIBYTE(address);
01186     gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+2
] = LOBYTE(length);
01187     gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+3
] = HIBYTE(length);
01188
01189     txrx_packet();
01190     //if(gbCommStatus == COMM_RXSUCCESS && id != BROADCAST_ID)
01191     //    memmove(data, &rxpacket[PKT_PARAMETER+1], length);
01192
01193     return MAKEDWORD(MAKEWORD ( gbStatusPacket[
PRT2_STATUS_PKT_PARAMETER0], gbStatusPacket[
PRT2_STATUS_PKT_PARAMETER0+1]),
01194     MAKEWORD ( gbStatusPacket[PRT2_STATUS_PKT_PARAMETER0+2],
gbStatusPacket[PRT2_STATUS_PKT_PARAMETER0+3]));
01195 }

```

3.5.3.23 unsigned short dynamixel2::read_word (unsigned char *id*, int *address*)

```

01140 {
01141     unsigned short length = 2;
01142     gbCommStatus = COMM_TXFAIL;
01143
01144     gbInstructionPacket[PRT2_PKT_ID] = id;
01145     gbInstructionPacket[PRT2_PKT_LENGTH_L] = 0x07;
01146     gbInstructionPacket[PRT2_PKT_LENGTH_H] = 0x00;
01147     gbInstructionPacket[PRT2_PKT_INSTRUCTION] =
INST_READ;
01148     gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+0
] = LOBYTE(address);
01149     gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+1
] = HIBYTE(address);
01150     gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+2
] = LOBYTE(length);
01151     gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+3
] = HIBYTE(length);
01152
01153     txrx_packet();
01154     //if(gbCommStatus == COMM_RXSUCCESS && id != BROADCAST_ID)
01155     //    memmove(data, &rxpacket[PKT_PARAMETER+1], length);
01156
01157     return MAKEWORD ( gbStatusPacket[
PRT2_STATUS_PKT_PARAMETER0], gbStatusPacket[
PRT2_STATUS_PKT_PARAMETER0+1]);
01158 }

```

3.5.3.24 void dynamixel2::reboot (unsigned char *id*)

```

01070 {
01071     if(id == BROADCAST_ID)
01072     {
01073         gbCommStatus = COMM_TXERROR;
01074         return;
01075     }
01076
01077     gbInstructionPacket[PRT2_PKT_ID] = (unsigned char)id;
01078     gbInstructionPacket[PRT2_PKT_LENGTH_L] = 0x03;
01079     gbInstructionPacket[PRT2_PKT_LENGTH_H] = 0x00;
01080     gbInstructionPacket[PRT2_PKT_INSTRUCTION] =
INST_REBOOT;
01081
01082     txrx_packet();
01083 }

```

3.5.3.25 void dynamixel2::remove_stuffing ()

```

00571 {
00572     int i = 0, index = 0;
00573     int packet_length_in = MAKEWORD( gbInstructionPacket[
        PRT2_PKT_LENGTH_L], gbInstructionPacket[
        PRT2_PKT_LENGTH_H]);
00574     int packet_length_out = packet_length_in;
00575
00576     index = PRT2_PKT_INSTRUCTION;
00577     for( i = 0; i < packet_length_in - 2; i++) // except CRC
00578     {
00579         if( gbInstructionPacket[i+PRT2_PKT_INSTRUCTION] == 0xFD &&
        gbInstructionPacket[i+PRT2_PKT_INSTRUCTION+1] == 0xFD &&
        gbInstructionPacket[i+PRT2_PKT_INSTRUCTION-1] == 0xFF &&
        gbInstructionPacket[i+PRT2_PKT_INSTRUCTION-2] == 0xFF)
00580         { // FF FF FD FD
00581             packet_length_out--;
00582             i++;
00583         }
00584         gbInstructionPacket[index++] = gbInstructionPacket[i+
        PRT2_PKT_INSTRUCTION];
00585     }
00586     gbInstructionPacket[index++] = gbInstructionPacket[
        PRT2_PKT_INSTRUCTION+packet_length_in-2];
00587     gbInstructionPacket[index++] = gbInstructionPacket[
        PRT2_PKT_INSTRUCTION+packet_length_in-1];
00588
00589     gbInstructionPacket[PRT2_PKT_LENGTH_L] =
        LOBYTE(packet_length_out);
00590     gbInstructionPacket[PRT2_PKT_LENGTH_H] =
        HIBYTE(packet_length_out);
00591 }

```

3.5.3.26 void dynamixel2::rx_packet (void)

```

00650 {
00651     //int rx_length = 0, wait_length = PRT2_PKT_LENGTH_H + 4 + 1; // 4 : INST ERROR CHKSUM_L CHKSUM_H
00652     unsigned int i;
00653     unsigned short crc = 0;
00654
00655     gbRxGetLength = 0; gbRxPacketLength =
        PRT2_PKT_LENGTH_H + 4 + 1;
00656
00657     // Check Bus Using
00658     //if(bus_using == 0)
00659     //    return 0;
00660
00661     while(1)
00662     {
00663         gbRxGetLength += dH.read( &gbStatusPacket[
        gbRxGetLength], gbRxPacketLength - gbRxGetLength);
00664         if(gbRxGetLength >= gbRxPacketLength) // wait_length minimum : 11
00665         {
00666             // Find packet header
00667             for(i = 0; i < (gbRxGetLength - 2); i++)
00668             {
00669                 if(gbStatusPacket[i] == 0xFF && gbStatusPacket[i+1] == 0xFF &&
        gbStatusPacket[i+2] == 0xFD)
00670                     break;
00671             }
00672
00673             if(i == 0)
00674             {
00675                 // Check length
00676                 gbRxPacketLength = MAKEWORD(
        gbStatusPacket[PRT2_PKT_LENGTH_L], gbStatusPacket[
        PRT2_PKT_LENGTH_H]) + PRT2_PKT_LENGTH_H + 1;
00677                 if(gbRxGetLength < gbRxPacketLength)
00678                 {
00679                     // Check timeout
00680                     if(is_packet_timeout() == 1)
00681                     {
00682                         if(gbRxGetLength == 0)
00683                             gbCommStatus = COMM_RXTIMEOUT;
00684                         else
00685                             gbCommStatus = COMM_RXCORRUPT;
00686                         giBusUsing = 0;
00687                         break;
00688                     }
00689                     continue;
00690                 }
00691             }

```

```

00692             // Check CRC16
00693             crc = MAKEWORD( gbStatusPacket[
gbRxPacketLength-2], gbStatusPacket[
gbRxPacketLength-1]);
00694             if(update_crc(0, gbStatusPacket,
gbRxPacketLength-2) == crc) // -2 : except CRC16
00695                 gbCommStatus = COMM_RXSUCCESS;
00696             else
00697                 gbCommStatus = COMM_RXCORRUPT;
00698             giBusUsing = 0;
00699             break;
00700         }
00701     else
00702     {
00703         // Remove unnecessary packets
00704         memmove(&gbStatusPacket[0], &gbStatusPacket[i], gbRxGetLength -
i);
00705         gbRxGetLength -= i;
00706     }
00707 }
00708 else
00709 {
00710     // Check timeout
00711     if(is_packet_timeout() == 1)
00712     {
00713         if(gbRxGetLength == 0)
00714             gbCommStatus = COMM_RXTIMEOUT;
00715         else
00716             gbCommStatus = COMM_RXCORRUPT;
00717         giBusUsing = 0;
00718         break;
00719     }
00720 }
00721 }
00722
00723 // Character stuffing
00724 if(gbCommStatus == COMM_RXSUCCESS)
00725     remove_stuffing();
00726
00727 giBusUsing = 0;
00728 }

```

3.5.3.27 void dynamixel2::set_packet_timeout (int NumRcvByte)

```

00459 {
00460     gdPacketStartTime = dH.get_curr_time();
00461     gdRcvWaitTime = (gdByteTransTime*(double)NumRcvByte + 2.0*
LATENCY_TIME + 2.0);
00462 }

```

3.5.3.28 void dynamixel2::set_packet_timeout_ms (int msec)

```

00465 {
00466     gdPacketStartTime = dH.get_curr_time();
00467     gdRcvWaitTime = (double)msec;
00468 }

```

3.5.3.29 void dynamixel2::set_txpacket_id (unsigned char id)

```

00978 {
00979     gbInstructionPacket[PRT2_PKT_ID] = id;
00980 }

```

3.5.3.30 void dynamixel2::set_txpacket_instruction (unsigned char instruction)

```

00983 {
00984     gbInstructionPacket[PRT2_PKT_INSTRUCTION] = (unsigned char)
instruction;
00985 }

```

3.5.3.31 void dynamixel2::set_txpacket_length (unsigned short *length*)

```

00993 {
00994     gbInstructionPacket[PRT2_PKT_LENGTH_L] =
        LOBYTE(length);
00995     gbInstructionPacket[PRT2_PKT_LENGTH_H] =
        HIBYTE(length);
00996 }

```

3.5.3.32 void dynamixel2::set_txpacket_parameter (unsigned short *index*, unsigned char *value*)

```

00988 {
00989     gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+
        index] = value;
00990 }

```

3.5.3.33 int dynamixel2::terminate (void)

```

00423 {
00424     int id = 0;
00425     for(id = 0; id <= MAX_ID; id++)
00426     {
00427         if(gBulkData[id].pucTable != 0)
00428             free((gBulkData[id].pucTable));
00429
00430         if(gSyncData[id].pucTable != 0)
00431             free((gBulkData[id].pucTable));
00432     }
00433     dH.close();
00434     return 0;
00435 }

```

3.5.3.34 void dynamixel2::tx_packet (void)

```

00595 {
00596     int packet_tx_len, real_tx_len;
00597     int length;
00598     unsigned short crc = 0;
00599
00600
00601     // Check Bus Using
00602     if(giBusUsing == 1)
00603     {
00604         gbCommStatus = COMM_TXFAIL;
00605         return;
00606     }
00607     giBusUsing = 1;
00608
00609     // Character stuffing
00610     add_stuffing();
00611     if(gbCommStatus == COMM_TXERROR)
00612         return;
00613
00614     length = MAKEWORD(gbInstructionPacket[
        PRT2_PKT_LENGTH_L], gbInstructionPacket[
        PRT2_PKT_LENGTH_H]);
00615
00616     // Check MAX packet length
00617     if(length > (MAXNUM_TXPACKET))
00618     {
00619         giBusUsing = 0;
00620         gbCommStatus = COMM_TXERROR;
00621         return;
00622     }
00623
00624     // Packet Header
00625     gbInstructionPacket[PRT2_PKT_HEADER0] = 0xFF;
00626     gbInstructionPacket[PRT2_PKT_HEADER1] = 0xFF;
00627     gbInstructionPacket[PRT2_PKT_HEADER2] = 0xFD;
00628     gbInstructionPacket[PRT2_PKT_RESERVED] = 0x00; // RESERVED
00629
00630     // Add CRC16
00631     crc = update_crc(0, gbInstructionPacket, length+
        PRT2_PKT_LENGTH_H+1-2); // -2 : except CRC16
00632     gbInstructionPacket[length+PRT2_PKT_LENGTH_H-1] =

```

```

        LOBYTE( crc );          // last - 1
00633     gbInstructionPacket[ length+PRT2_PKT_LENGTH_H-0 ] =
        HIBYTE( crc );          // last - 0
00634
00635     // Tx Packet
00636     dH.clear();
00637     packet_tx_len = length + PRT2_PKT_LENGTH_H + 1;
00638     real_tx_len = dH.write( gbInstructionPacket, packet_tx_len );
00639     if( packet_tx_len != real_tx_len )
00640     {
00641         giBusUsing = 0;
00642         gbCommStatus = COMM_TXFAIL;
00643         return;
00644     }
00645
00646     gbCommStatus = COMM_TXSUCCESS;
00647 }

```

3.5.3.35 void dynamixel2::txrx_packet (void)

```

00731 {
00732     int n = 0, num = 0;
00733     int id = 0;
00734     int wait_length = 0;
00735     int data_length = 0;
00736     gbCommStatus = COMM_TXFAIL;
00737
00738     // Wait for Bus Idle
00739     while( giBusUsing == 1 )
00740     {
00741         //Sleep(0);
00742     }
00743
00744     if( ( gbInstructionPacket[PRT2_PKT_INSTRUCTION] !=
INST_READ ) &&
00745         ( gbInstructionPacket[PRT2_PKT_INSTRUCTION] !=
INST_WRITE ) &&
00746         ( gbInstructionPacket[PRT2_PKT_INSTRUCTION] !=
INST_REG_WRITE ) &&
00747         ( gbInstructionPacket[PRT2_PKT_INSTRUCTION] !=
INST_ACTION ) &&
00748         ( gbInstructionPacket[PRT2_PKT_INSTRUCTION] !=
INST_RESET ) &&
00749         ( gbInstructionPacket[PRT2_PKT_INSTRUCTION] !=
INST_SYNC_WRITE ) &&
00750         ( gbInstructionPacket[PRT2_PKT_INSTRUCTION] !=
INST_BULK_READ ) &&
00751         ( gbInstructionPacket[PRT2_PKT_INSTRUCTION] !=
INST_REBOOT ) &&
00752         ( gbInstructionPacket[PRT2_PKT_INSTRUCTION] !=
INST_STATUS ) &&
00753         ( gbInstructionPacket[PRT2_PKT_INSTRUCTION] !=
INST_SYNC_READ ) &&
00754         ( gbInstructionPacket[PRT2_PKT_INSTRUCTION] !=
INST_BULK_WRITE ) )
00755     {
00756         gbCommStatus = COMM_TXERROR;
00757         return;
00758     }
00759
00760
00761
00762
00763     //if( ( gbInstructionPacket[PRT2_PKT_INSTRUCTION] != INST_SYNC_READ ) &&
( gbInstructionPacket[PRT2_PKT_INSTRUCTION] != INST_BULK_READ ) )
00764     if( ( gbInstructionPacket[PRT2_PKT_ID] !=
BROADCAST_ID ) )
00765     {
00766         tx_packet();
00767         // Check Tx packet result
00768         if( gbCommStatus != COMM_TXSUCCESS )
00769             return;
00770
00771         // Set Rx Timeout
00772         if( gbInstructionPacket[PRT2_PKT_INSTRUCTION] ==
INST_READ )
00773             set_packet_timeout( MAKEWORD(
gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+2],
gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+3] ) +
11 );
00774         else
00775             set_packet_timeout( PRT2_PKT_LENGTH_H+4+1 ); // 4 : INST
ERROR CHKSUM_L CHKSUM_H

```

```

00776
00778         //if (gbInstructionPacket[PRT2_PKT_ID] == BROADCAST_ID)
00779         //{
00780             //    giBusUsing = 0;
00781             //    gbCommStatus = COMM_RXSUCCESS;
00782             //    return;
00783         //}
00784
00785         rx_packet();
00786         if((gbCommStatus == COMM_RXSUCCESS) && (
gbStatusPacket[PRT2_PKT_ID] != BROADCAST_ID) && (
gbInstructionPacket[PRT2_PKT_ID] != gbStatusPacket[
PRT2_PKT_ID]))
00787             rx_packet();
00788     }
00789     else
00790     {
00791         if (gbInstructionPacket[PRT2_PKT_INSTRUCTION] ==
INST_BULK_READ )
00792         {
00793             num = (MAKEWORD (gbInstructionPacket[
PRT2_PKT_LENGTH_L], gbInstructionPacket[
PRT2_PKT_LENGTH_H]) - 3 )/5;
00794             for(n = 0; n < num; n++)
00795             {
00796                 id = gbInstructionPacket[
PRT2_INSTRUCTION_PKT_PARAMETER0 + n*5];
00797                 gBulkData[id].iError = -1;
00798                 gBulkData[id].iStartAddr = MAKEWORD (
gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0 + n*5
+ 1],
00799                 gbInstructionPacket[
PRT2_INSTRUCTION_PKT_PARAMETER0 + n*5 + 2]);
00800                 gBulkData[id].iLength = MAKEWORD (
gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0 + n*5
+ 3],
00801                 gbInstructionPacket[
PRT2_INSTRUCTION_PKT_PARAMETER0 + n*5 + 4]);
00802
00803                 if(gBulkData[id].pucTable != 0)
00804                     free((gBulkData[id].pucTable));
00805
00806                 gBulkData[id].pucTable = (unsigned char*) calloc(
gbBulkData[id].iLength, sizeof(unsigned char));
00807                 wait_length += gBulkData[id].iLength + 11;
00808             }
00809
00810             while (giBusUsing == 1)
00811             {
00812                 //Sleep(0);
00813             }
00814             tx_packet();
00815             if( gbCommStatus != COMM_TXSUCCESS )
00816                 return;
00817
00818             set_packet_timeout(wait_length);
00819
00820             for(n = 0; n < num; n++)
00821             {
00822                 id = gbInstructionPacket[
PRT2_INSTRUCTION_PKT_PARAMETER0 + n*5];
00823                 // Rx packet
00824                 rx_packet();
00825                 if(gbCommStatus == COMM_RXSUCCESS)
00826                     gBulkData[id].iError = gbStatusPacket[
PRT2_PKT_ERRBIT];
00827                 // rxpacket to rxdata[id]->pucTable
00828                 memcpy(gBulkData[id].pucTable, &gbStatusPacket[
PRT2_STATUS_PKT_PARAMETER0], gBulkData[id].iLength);
00829             }
00830
00831             else if (gbInstructionPacket[PRT2_PKT_INSTRUCTION] ==
INST_SYNC_READ)
00832             {
00833                 num = (MAKEWORD (gbInstructionPacket[
PRT2_PKT_LENGTH_L], gbInstructionPacket[
PRT2_PKT_LENGTH_H]) - 3 - 4); //3 : INST CRC_L CRC_H, 4 : param0->addr_l param0->addr_h
param0->length_l param0->length_h
00834                 data_length = MAKEWORD (gbInstructionPacket[
PRT2_INSTRUCTION_PKT_PARAMETER0+2],
00835                 gbInstructionPacket[
PRT2_INSTRUCTION_PKT_PARAMETER0+3]);
00836
00837
00838                 for(n = 0; n < num; n++)
00839                 {
00840                     id = gbInstructionPacket[

```

```

PRT2_INSTRUCTION_PKT_PARAMETER0 + 4 + n];
00841     gSyncData[id].iID = id;
00842     gSyncData[id].iStartAddr = MAKEWORD(
gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0 + 0],

00843                                     gbInstructionPacket[
PRT2_INSTRUCTION_PKT_PARAMETER0 + 1]);
00844     gSyncData[id].iError = -1;
00845     if(gSyncData[id].pucTable != 0)
00846         free((gSyncData[id].pucTable));
00847
00848     gSyncData[id].pucTable = (unsigned char *) calloc(data_length, sizeof(
unsigned char));
00849 }
00850
00851
00852     wait_length = 11 + data_length;
00853     wait_length *= num;
00854
00855     while(giBusUsing == 1);
00856
00857     tx_packet();
00858
00859     // Check Tx packet result
00860     if( gbCommStatus != COMM_TXSUCCESS )
00861         return;
00862
00863     // Set Rx Timeout (SYNK_READ)
00864     set_packet_timeout(wait_length);
00865
00866     for(n = 0; n < num; n++)
00867     {
00868         id = gbInstructionPacket[
PRT2_INSTRUCTION_PKT_PARAMETER0 + 4 + n];
00869         // Rx packet
00870         rx_packet();
00871         if(gbCommStatus == COMM_RXSUCCESS)
00872             gSyncData[id].iError = gbStatusPacket[
PRT2_PKT_ERRBIT];
00873         // rxpacket to rxdata[id]->pucTable
00874         memcpy(gSyncData[id].pucTable, &gbStatusPacket[
PRT2_STATUS_PKT_PARAMETER0], data_length);
00875     }
00876
00877     return;
00878 }
00879 else if(gbInstructionPacket[PRT2_PKT_INSTRUCTION] ==
INST_PING)
00880 {
00881     int rx_length = 0;
00882     tx_packet();
00883     if(gbCommStatus != COMM_TXSUCCESS)
00884     {
00885         giBusUsing = 0;
00886         return;
00887     }
00888
00889     wait_length = PING_STATUS_LENGTH * MAX_ID;
00890     set_packet_timeout_ms((int)((gdByteTransTime * wait_length)
+ (3 * MAX_ID) + 2 * LATENCY_TIME));
00891
00892     while(1)
00893     {
00894         int _cnt = dH.read(&gbStatusPacket[rx_length], wait_length - rx_length)
;
00895         if(_cnt > 0)
00896         {
00897             rx_length += _cnt;
00898             //printf("cnt : %d, Interval : %f / Wait time : %f\n", _cnt, get_packet_time(),
gdPacketWaitTime);
00899         }
00900         if(is_packet_timeout() == 1 || rx_length >= wait_length)
00901             break;
00902     }
00903     giBusUsing = 0;
00904
00905     if(rx_length== 0)
00906     {
00907         gbCommStatus = COMM_RXTIMEOUT;
00908         return;
00909     }
00910
00911     while(1)
00912     {
00913         int idx = 0;
00914
00915         if(rx_length < PING_STATUS_LENGTH)

```



```

00916         {
00917             gbCommStatus = COMM_RXCORRUPT;
00918             return;
00919         }
00920
00921         // find packet header
00922         while( idx < (rx_length - 2) )
00923         {
00924             if(gbStatusPacket[idx] == 0xFF &&
gbStatusPacket[idx + 1] == 0xFF && gbStatusPacket[idx + 2] == 0xFD)
00925                 break;
00926             else
00927                 idx++;
00928         }
00929
00930         if(idx == 0)
00931         {
00932             // check CRC16
00933             int crc = MAKEWORD(gbStatusPacket[
PING_STATUS_LENGTH - 2], gbStatusPacket[
PING_STATUS_LENGTH - 1]);
00934             if(update_crc(0, gbStatusPacket,
PING_STATUS_LENGTH - 2) == crc) // - 2 : except CRC16
00935             {
00936                 gPingData[gbStatusPacket[
PRT2_PKT_ID]].iID = gbStatusPacket[PRT2_PKT_ID];
00937                 gPingData[gbStatusPacket[
PRT2_PKT_ID]].iModelNo = MAKEWORD(gbStatusPacket[
PRT2_STATUS_PKT_PARAMETER0], gbStatusPacket[
PRT2_STATUS_PKT_PARAMETER0+1]);
00938                 gPingData[gbStatusPacket[
PRT2_PKT_ID]].iFirmVer = gbStatusPacket[
PRT2_STATUS_PKT_PARAMETER0+2];
00939
00940                 memcpy(&gbStatusPacket[0], &gbStatusPacket[
PING_STATUS_LENGTH], rx_length - PING_STATUS_LENGTH);
00941                 rx_length -= PING_STATUS_LENGTH;
00942             }
00943             else
00944             {
00945                 gbCommStatus = COMM_RXCORRUPT;
00946
00947                 // remove header (0xFF 0xFF 0xFD)
00948                 memcpy(&gbStatusPacket[0], &gbStatusPacket[3],
rx_length - 3);
00949                 rx_length -= 3;
00950             }
00951
00952             if(rx_length < PING_STATUS_LENGTH)
00953                 break;
00954         }
00955         else
00956         {
00957             // remove unnecessary packets
00958             memcpy(&gbStatusPacket[0], &gbStatusPacket[idx], rx_length
- idx);
00959             rx_length -= idx;
00960         }
00961     }
00962 }
00963 else // Sync_Write ans Bulk_Write
00964 {
00965     tx_packet();
00966     giBusUsing = 0;
00967     if(gbCommStatus == COMM_TXSUCCESS)
00968         gbCommStatus = COMM_RXSUCCESS;
00969     return;
00970 }
00971 }
00972 }

```

3.5.3.36 void dynamixel2::write_byte (unsigned char id, int address, unsigned char value)

```

01126 {
01127     unsigned short length = 1;
01128     gbInstructionPacket[PRT2_PKT_ID] = id;
01129     gbInstructionPacket[PRT2_PKT_LENGTH_L] =
LOBYTE(length+5);
01130     gbInstructionPacket[PRT2_PKT_LENGTH_H] =
HIBYTE(length+5);
01131     gbInstructionPacket[PRT2_PKT_INSTRUCTION] =
INST_WRITE;
01132     gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+0]
= LOBYTE(address);
]

```

```

01133     gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+1
    ] = HIBYTE(address);
01134     gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+2
    ] = (unsigned char)value;
01135
01136     txrx_packet();
01137 }

```

3.5.3.37 void dynamixel2::write_dword (unsigned char *id*, int *address*, unsigned long *value*)

```

01198 {
01199     unsigned short length = 4;
01200     gbInstructionPacket[PRT2_PKT_ID] = id;
01201     gbInstructionPacket[PRT2_PKT_LENGTH_L] =
    LOBYTE(length+5);
01202     gbInstructionPacket[PRT2_PKT_LENGTH_H] =
    HIBYTE(length+5);
01203     gbInstructionPacket[PRT2_PKT_INSTRUCTION] =
    INST_WRITE;
01204     gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+0
    ] = LOBYTE(address);
01205     gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+1
    ] = HIBYTE(address);
01206     gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+2
    ] = LOBYTE(LOWORD( value ));
01207     gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+3
    ] = HIBYTE(LOWORD( value ));
01208     gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+4
    ] = LOBYTE(HIWORD( value ));
01209     gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+5
    ] = HIBYTE(HIWORD( value ));
01210
01211     txrx_packet();
01212 }

```

3.5.3.38 void dynamixel2::write_word (unsigned char *id*, int *address*, unsigned short *value*)

```

01161 {
01162     unsigned short length = 2;
01163     gbInstructionPacket[PRT2_PKT_ID] = id;
01164     gbInstructionPacket[PRT2_PKT_LENGTH_L] =
    LOBYTE(length+5);
01165     gbInstructionPacket[PRT2_PKT_LENGTH_H] =
    HIBYTE(length+5);
01166     gbInstructionPacket[PRT2_PKT_INSTRUCTION] =
    INST_WRITE;
01167     gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+0
    ] = LOBYTE(address);
01168     gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+1
    ] = HIBYTE(address);
01169     gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+2
    ] = LOBYTE( value );
01170     gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+3
    ] = HIBYTE( value );
01171
01172     txrx_packet();
01173 }

```

3.5.4 Member Data Documentation

3.5.4.1 dxl_hal dynamixel2::dH [private]

3.5.4.2 int dynamixel2::gbCommStatus = COMM_RXSUCCESS [private]

3.5.4.3 unsigned char dynamixel2::gbInstructionPacket[MAXNUM_TXPACKET] = {0} [private]

3.5.4.4 unsigned int dynamixel2::gbRxGetLength = 0 [private]

3.5.4.5 unsigned int dynamixel2::gbRxPacketLength = 0 [private]

3.5.4.6 `unsigned char dynamixel2::gbStatusPacket[MAXNUM_RXPACKET] = {0}` [private]

3.5.4.7 `SyncBulkData dynamixel2::gBulkData[MAX_ID+1]` [private]

3.5.4.8 `double dynamixel2::gdByteTransTime = 0.0` [private]

3.5.4.9 `double dynamixel2::gdPacketStartTime = 0.0` [private]

3.5.4.10 `double dynamixel2::gdRcvWaitTime = 0.0` [private]

3.5.4.11 `int dynamixel2::giBusUsing = 0` [private]

3.5.4.12 `PingData dynamixel2::gPingData[MAX_ID+1]` [private]

3.5.4.13 `SyncBulkData dynamixel2::gSyncData[MAX_ID+1]` [private]

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

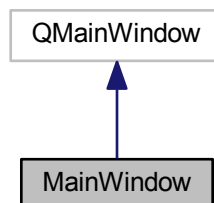
- [dxl/dynamixel.h](#)
- [dxl/dynamixel.cpp](#)

3.6 MainWindow Class Reference

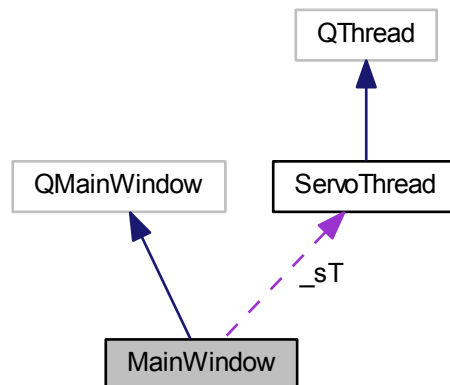
Contains all the windows and other classes.

```
#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.

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.

3.6.1 Detailed Description

Contains all the windows and other classes.

3.6.2 Constructor & Destructor Documentation

3.6.2.1 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     _sT.start();
00015     _timer.setInterval(10);
00016     _timer.start();
00017
00018     connect(&_amp;_joy, SIGNAL(changed()), this, SLOT(joyChanged()));
00019     connect(&_amp;_timer, SIGNAL(timeout()), this, SLOT(update()));
00020
00021     // JOYSTICK
00022     QVector< QString > V(_joy.getAllAxis());
00023     // Adding axis
00024     QGridLayout *wL = new QGridLayout;
00025     for (int i = 0; i < XJoystick::AxisCount; ++i) {
00026         QHBoxLayout *L = new QHBoxLayout;
00027         L->addWidget(new QLabel(V[i].append(":"), this));
00028         _axis[i] = new QLabel("#");
00029         L->addWidget(_axis[i]);
00030         L->addStretch();

```

```

00031         wL->addLayout(L, i%3, i/3);
00032     }
00033     ui->joyAxis->setLayout(wL);
00034
00035     // Adding buttons
00036     wL = new QGridLayout;
00037     for (int i = 0; i < XJoystick::ButtonCount; ++i) {
00038         _buts[i] = new QLabel(QString::number(i + 1));
00039         wL->addWidget(_buts[i], i/8, i%8);
00040         _buts[i]->setEnabled(false);
00041         _buts[i]->hide();
00042     }
00043     ui->joyButs->setLayout(wL);
00044     ui->joyAxis->hide();
00045     ui->joyButs->hide();
00046     ui->line->hide();
00047     // TODO: Create dataPath
00048
00049     _dataP = QStandardPaths::writableLocation(QStandardPaths::AppDataLocation);
00050     QDir dir(_dataP);
00051     if (!dir.exists()) dir.mkpath(_dataP);
00052 }

```

3.6.2.2 MainWindow::~MainWindow ()

Default destructor.

```

00055 {
00056     delete ui;
00057 }

```

3.6.3 Member Function Documentation

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

Handles a joystick update.

```

00060 {
00061     int sel = _joy.current();
00062
00063     QVector< XJoystick::Info > V(_joy.available());
00064     bool found = false;
00065     int i = 0;
00066     while (i < V.size() and not found) { found = V[i].ID == sel; ++i; }
00067     if (not found) {
00068         if (V.size() > 0) {
00069             _joy.select(V[0].ID);
00070             ui->line->hide();
00071
00072             // Showing axis
00073             ui->joyAxis->show();
00074
00075             // Showing buttons
00076             for (QLabel *l : _buts) l->hide();
00077             ui->joyButs->show();
00078             int n = _joy.buttonCount();
00079             for (int i = 0; i < n; ++i) _buts[i]->show();
00080         }
00081         else {
00082             _joy.select(-1);
00083             ui->joyAxis->hide();
00084             ui->joyButs->hide();
00085             ui->line->hide();
00086         }
00087     }
00088     emit joystickChanged();
00089 }

```

3.6.3.2 void MainWindow::joystickChanged () [signal]

Emitted when a joystick changes.

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

To select the options.

```
00093 {
00094     OptionsWindow o(_joy, &_sT, this);
00095     o.exec();
00096
00097     connect(this, SIGNAL(joystickChanged()), &o, SLOT(
00098         joystickChanged()));
00099
00099     if (o.result()) o.storeData();
00100 }
```

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

Updates all data to the servo thread.

```
00103 {
00104     _joy.update();
00105     for (int i = 0; i < XJoystick::AxisCount; ++i) _axisV[i] = _joy[i];
00106     for (int i = 0; i < XJoystick::ButtonCount; ++i) _butsV[i] = _joy.button(i);
00107
00108     _sT.setData(_axisV, _butsV);
00109
00110     // TODO: Finish update function
00111 }
```

3.6.4 Member Data Documentation**3.6.4.1 QVector< QLabel *> MainWindow::_axis [private]**

Handles all the axis labels.

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

Contains the axis value;.

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

Handles all the button labels.

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

Handles all buttons values.

3.6.4.5 QString MainWindow::_dataP [private]

Contains the path to the data location.

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

Axis for the X value.

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

Axis for the Y value.

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

AXis for the Z value.

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

To handle the joystick.

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

Contains the thread controlling all the servos and external hardware.

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

To update the joystick value.

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

Contains the number of additional servos used.

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

Contains the number of minimun servos to work.

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

Contains the user interface.

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

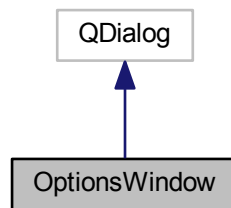
- [mainwindow.h](#)
- [mainwindow.cpp](#)

3.7 OptionsWindow Class Reference

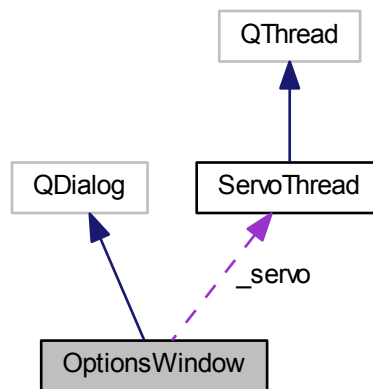
Class used to handle a Window to set the options.

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


Inheritance diagram for OptionsWindow:



Collaboration diagram for OptionsWindow:



Public Slots

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

Public Member Functions

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

Private Slots

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

Private Attributes

- XJoystick & `_joy`
Contains the Joystick to handle options.
- int `_portSize`
Contains the size of the ports.
- ServoThread * `_servo`
Pointer to the servo thread class.
- QTimer `_timer`
Waits for a new COM port.
- Ui::OptionsWindow * `ui`
Containsh the GUI.

3.7.1 Detailed Description

Class used to handle a Window to set the options.

3.7.2 Constructor & Destructor Documentation

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

Default constructor.

```

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

3.7.2.2 OptionsWindow::~OptionsWindow ()

Destructor.

```
00036 {
00037     delete ui;
00038 }
```

3.7.3 Member Function Documentation

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

Handles events that need to be updated continuously.

```
00069 {
00070     auto ports = QSerialPortInfo::availablePorts();
00071
00072     if (ports.size() != _portSize) {
00073         _portSize = ports.size();
00074
00075         QString portC(ui->portC->currentData().toString());
00076         QString portS(ui->portS->currentData().toString());
00077         qDebug() << portC << portS;
00078         int selC = 0, selS = 0;
00079
00080         ui->portC->clear();
00081         ui->portS->clear();
00082
00083         ui->portC->addItem("None", "");
00084         ui->portS->addItem("None", "");
00085
00086         for (int i = 0; i < ports.size(); ++i) {
00087             QString text(ports[i].portName());
00088             text += ": " + ports[i].description();
00089             ui->portC->addItem(text, ports[i].portName());
00090             ui->portS->addItem(text, ports[i].portName());
00091
00092             if (ports[i].portName() == portC) selC = i + 1;
00093             if (ports[i].portName() == portS) selS = i + 1;
00094         }
00095         qDebug() << selC << selS;
00096
00097         ui->portC->setCurrentIndex(selC);
00098         ui->portS->setCurrentIndex(selS);
00099     }
00100 }
```

3.7.3.2 void OptionsWindow::joystickChanged () [slot]

To handle the change of a joystick.

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

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

Refreshes all the servos connected to the port.

```
00103 {
00104     dynamixel dxl;
00105     QString port;
00106     int baud;
00107     _servo->getServoPortInfo(port, baud);
00108
00109     dxl.initialize(port, baud);
00110
00111 }
```

3.7.3.4 void OptionsWindow::storeData ()

Stores all data.

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

3.7.4 Member Data Documentation

3.7.4.1 XJoystick& OptionsWindow::_joy [private]

Contains the Joystick to handle options.

3.7.4.2 int OptionsWindow::_portSize [private]

Contains the size of the ports.

3.7.4.3 ServoThread* OptionsWindow::_servo [private]

Pointer to the servo thread class.

3.7.4.4 QTimer OptionsWindow::_timer [private]

Waits for a new COM port.

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

Containsh the GUI.

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

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

3.8 dynamixel2::ping_data Struct Reference

Struct used to do a ping.

Public Attributes

- int [iID](#)
- int [iModelNo](#)
- int [iFirmVer](#)

3.8.1 Detailed Description

Struct used to do a ping.

3.8.2 Member Data Documentation

3.8.2.1 int dynamixel2::ping_data::iFirmVer

3.8.2.2 int dynamixel2::ping_data::iID

3.8.2.3 int dynamixel2::ping_data::iModelNo

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

- [dxl/dynamixel.h](#)

3.9 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.

3.9.1 Detailed Description

Struct for the [AX12](#) servos.

3.9.2 Constructor & Destructor Documentation

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

Default constructor.

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

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

Copy constructor.

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

3.9.3 Member Data Documentation

3.9.3.1 int ServoThread::Servo::ID

Contains the servo ID.

3.9.3.2 double ServoThread::Servo::load

Contains the servo load.

3.9.3.3 double ServoThread::Servo::pos

Contains the servo position.

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

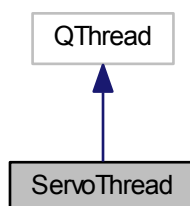
- [servothread.h](#)

3.10 ServoThread Class Reference

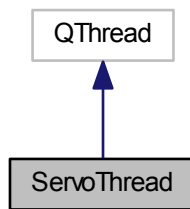
The [ServoThread](#)'s class handles the communication between the delta robot servos and the PC.

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

Inheritance diagram for ServoThread:



Collaboration diagram for ServoThread:



Classes

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

Public Types

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

Public Member Functions

- [ServoThread](#) ()
Default constructor.
- [~ServoThread](#) ()
Default destructor.
- void [cont](#) ()
Continues program's execution.
- 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.
- void [setData](#) (QVector< float > &aV, QVector< bool > &butS)

- *Adds the loaded data.*
- void [setServoBaud](#) (unsigned int baud)
Sets the servos port baud rate.
- void [setServoPort](#) (QString &port)
Sets the servos port.
- void [setSID](#) (QVector< int > &V)
Sets the servos ID.
- 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.

Private Attributes

- 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 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.

3.10.1 Detailed Description

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

3.10.2 Member Enumeration Documentation

3.10.2.1 enum ServoThread::Mode

Contains the working mode.

Enumerator

controlled

manual

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

3.10.2.2 enum ServoThread::Version [private]

Enum containing all the save file versions.

Enumerator

v_1_0

```
00027      {
00028          v_1_0
00029      };
```

3.10.3 Constructor & Destructor Documentation

3.10.3.1 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  }
```

3.10.3.2 ServoThread::~ServoThread ()

Default destructor.

```

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

```

3.10.4 Member Function Documentation

3.10.4.1 void ServoThread::cont () [inline]

Continues program's execution.

```

00063 {
00064     _mutex.lock();
00065     _pause = false;
00066     _cond.wakeOne();
00067     _mutex.unlock();
00068 }

```

3.10.4.2 void ServoThread::end () [inline]

Ends the execution.

```

00072 {
00073     _mutex.lock();
00074     _end = true;
00075     _cond.wakeOne();
00076     _mutex.unlock();
00077
00078     wait();
00079 }

```

3.10.4.3 int ServoThread::getServoBaud () [inline]

Returns the current servo Baud rate.

```

00094 {
00095     QMutexLocker mL(&_mutex);
00096     return _sBaud;
00097 }

```

3.10.4.4 QString ServoThread::getServoPort () [inline]

Returns the current servo Port.

```

00101 {
00102     QMutexLocker mL(&_mutex);
00103     return _sPort;
00104 }

```

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

Returns both servo Port and baud Rate.

```

00108 {
00109     _mutex.lock();
00110     baud = _sBaud;
00111     port = _sPort;
00112     _mutex.unlock();
00113 }

```

3.10.4.6 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
---	-----------------------------------

```

00119     {
00120         _mutex.lock();
00121         V = _servos;
00122         _mutex.unlock();
00123     }

```

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

Overloaded function to get the servo info.

```

00127     {
00128         QMutexLocker mL(&_mutex);
00129         return _servos;
00130     }

```

3.10.4.8 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 }

```

3.10.4.9 void ServoThread::pause () [inline]

Pauses the execution.

```

00086     {
00087         _mutex.lock();
00088         _pause = true;
00089         _mutex.unlock();
00090     }

```

3.10.4.10 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 > (_servos.size(), dxl);

```

```

00086
00087     while (not _end) {
00088
00089         msleep(10);
00090         _mutex.lock();
00091         if (not _end and _pause) {
00092             dxl.terminate();
00093             _cond.wait(&_mutex);
00094             dxl.initialize(sPort, sBaud);
00095         }
00096         if (_dChanged) {
00097             if (sPort != _sPort) {
00098                 sPort = _sPort;
00099                 sBaud = _sBaud;
00100                 dxl.terminate();
00101                 dxl.initialize(sPort, sBaud);
00102             }
00103         }
00104         _dChanged = false;
00105         _mutex.unlock();
00106     }
00107
00108     dxl.terminate();
00109     exit(0);
00110 }

```

3.10.4.11 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 }

```

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

Sets the servos port baud rate.

Parameters

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

```

00140     {
00141         _mutex.lock();
00142         _sBaud = baud;
00143         _mutex.unlock();
00144     }

```

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

Sets the servos port.

Parameters

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

```

00149     {
00150         _mutex.lock();
00151         _sPort = port;
00152         _mutex.unlock();
00153     }

```

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

Sets the servos ID.

Parameters

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

```

00158     {
00159         _mutex.lock();
00160         if (V.size() != _servos.size()) _servos.resize(V.size());
00161         for (int i = 0; i < V.size(); ++i) _servos[i].ID = V[i];
00162         _dChanged = true;
00163         _mutex.unlock();
00164     }
00165 }

```

3.10.4.15 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     df << int(Version::v_1_0) << _cBaud << _cPort << _sBaud <<
00070     _sPort
00071     << _servos.size();
00072     for (const Servo &s : _servos) df << s.ID;
00073     _mutex.unlock();
00074 }
00075 }

```

3.10.5 Member Data Documentation**3.10.5.1 QVector< float > ServoThread::_axis [private]**

Contains the axis value.

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

Contains the buttons value.

3.10.5.3 int ServoThread::_cBaud [private]

Contains the baud rate used to communicate with the clamp.

3.10.5.4 QWaitCondition ServoThread::_cond [private]

To start and pause the thread.

3.10.5.5 QString ServoThread::_cPort [private]

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

3.10.5.6 bool ServoThread::_dChanged [private]

True if the data changes.

3.10.5.7 bool ServoThread::_end [private]

True when we must end executino.

3.10.5.8 Mode ServoThread::_mod [private]

Contains the working mode.

3.10.5.9 QMutex ServoThread::_mutex [private]

To prevent memory errors.

3.10.5.10 bool ServoThread::_pause [private]

Pauses the execution of the thread.

3.10.5.11 int ServoThread::_sBaud [private]

Contains the used baud rate to communicate with the servos.

3.10.5.12 QVector< Servo > ServoThread::_servos [private]

Contains the servos information.

3.10.5.13 QString ServoThread::_sPort [private]

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

3.10.5.14 bool ServoThread::_sPortChanged [private]

True if the servos port changes.

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

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

Chapter 4

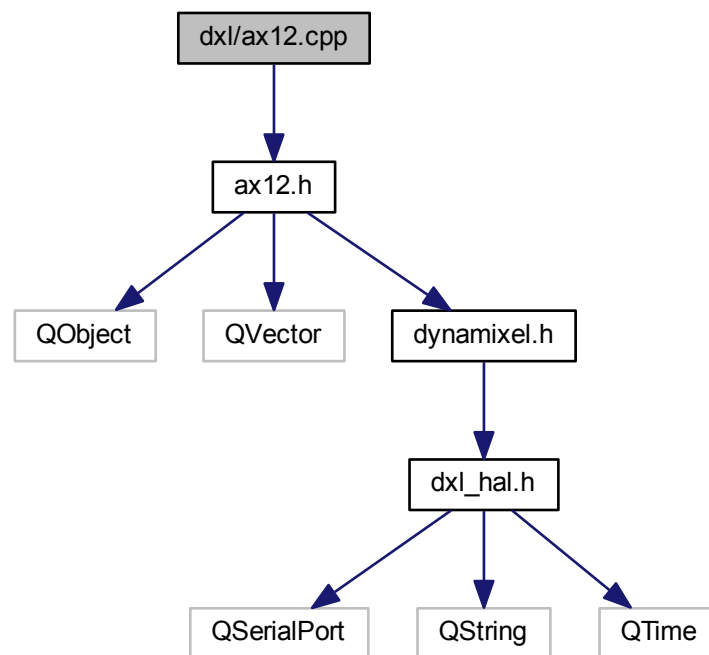
File Documentation

4.1 dxl/ax12.cpp File Reference

Contains the [AX12](#) class implementation.

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

Include dependency graph for ax12.cpp:



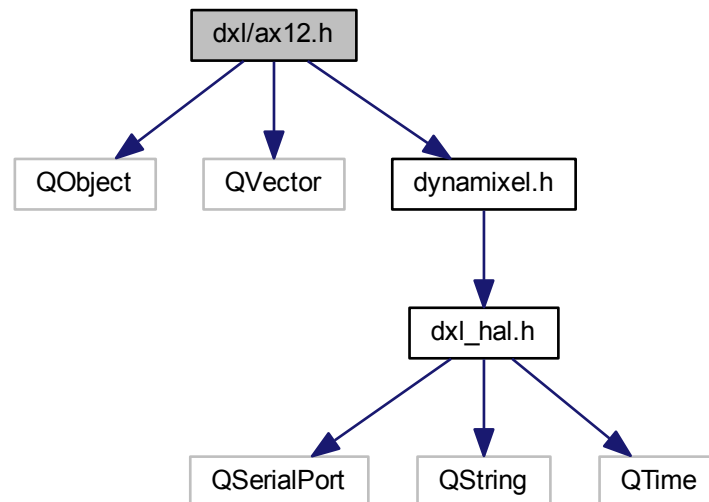
4.1.1 Detailed Description

Contains the [AX12](#) class implementation.

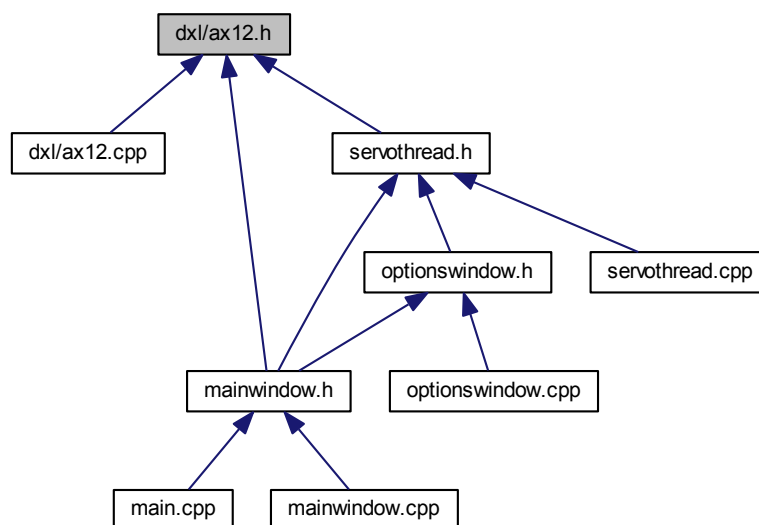
4.2 dxl/ax12.h File Reference

Contains the [AX12](#) class declaration.

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



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



Classes

- class [AX12](#)

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

4.2.1 Detailed Description

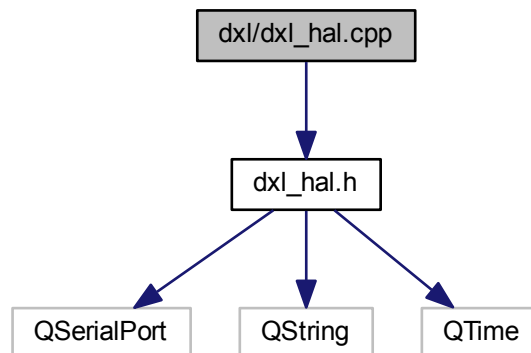
Contains the [AX12](#) class declaration.

4.3 dxl/dxl_hal.cpp File Reference

Contains the Dynamixel SDK platform dependent header source.

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

Include dependency graph for dxl_hal.cpp:



4.3.1 Detailed Description

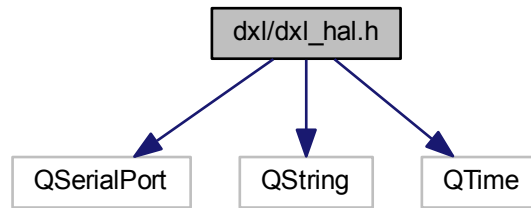
Contains the Dynamixel SDK platform dependent header source.

4.4 dxl/dxl_hal.h File Reference

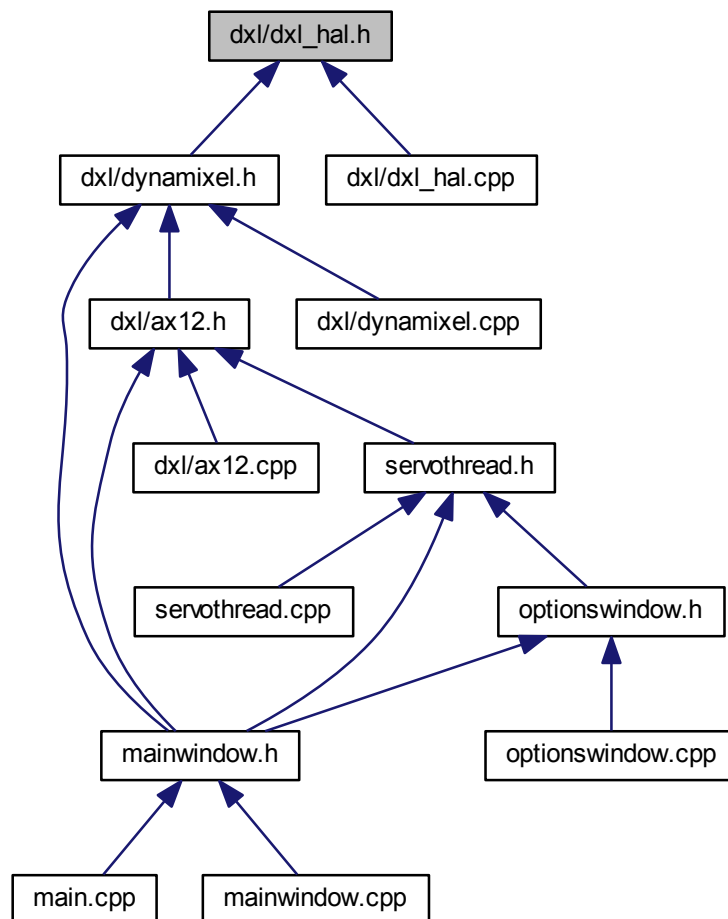
Contains the Dynamixel SDK platform dependent header declaration.

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

Include dependency graph for dxl_hal.h:



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



Classes

- class [dxl_hal](#)
Dynamixel SDK platform dependent.

Macros

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

4.4.1 Detailed Description

Contains the Dynamixel SDK platform dependent header declaration.

4.4.2 Macro Definition Documentation

4.4.2.1 `#define` [MAXNUM_RXPACKET](#) (10000)

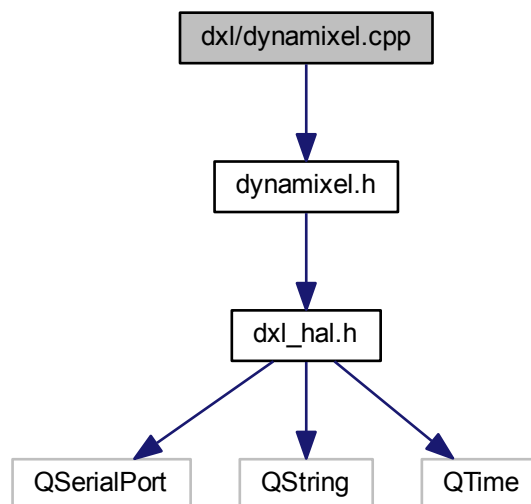
4.4.2.2 `#define` [MAXNUM_TXPACKET](#) (10000)

4.5 dxl/dynamixel.cpp File Reference

Contains the dynamixel and [dynamixel2](#) classes implementation.

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

Include dependency graph for dynamixel.cpp:



Macros

- `#define` [LATENCY_TIME](#) (16)

- `#define PING_STATUS_LENGTH (14)`

Functions

- unsigned short `update_crc` (unsigned short `crc_accum`, unsigned char `*data_blk_ptr`, unsigned short `data_blk_size`)

4.5.1 Detailed Description

Contains the `dynamixel` and `dynamixel2` classes implementation.

4.5.2 Macro Definition Documentation

4.5.2.1 `#define LATENCY_TIME (16)`

4.5.2.2 `#define PING_STATUS_LENGTH (14)`

4.5.3 Function Documentation

4.5.3.1 unsigned short `update_crc` (unsigned short `crc_accum`, unsigned char * `data_blk_ptr`, unsigned short `data_blk_size`)

```

00473 {
00474     unsigned short i, j;
00475     unsigned short crc_table[256] = {0x0000,
00476                                     0x8005, 0x800F, 0x000A, 0x801B, 0x001E, 0x0014, 0x8011,
00477                                     0x8033, 0x0036, 0x003C, 0x8039, 0x0028, 0x802D, 0x8027,
00478                                     0x0022, 0x8063, 0x0066, 0x006C, 0x8069, 0x0078, 0x807D,
00479                                     0x8077, 0x0072, 0x0050, 0x8055, 0x805F, 0x005A, 0x804B,
00480                                     0x004E, 0x0044, 0x8041, 0x80C3, 0x00C6, 0x00CC, 0x80C9,
00481                                     0x00D8, 0x80DD, 0x80D7, 0x00D2, 0x00F0, 0x80F5, 0x80FF,
00482                                     0x00FA, 0x80EB, 0x00EE, 0x00E4, 0x80E1, 0x00A0, 0x80A5,
00483                                     0x80AF, 0x00AA, 0x80BB, 0x00BE, 0x00B4, 0x80B1, 0x8093,
00484                                     0x0096, 0x009C, 0x8099, 0x0088, 0x808D, 0x8087, 0x0082,
00485                                     0x8183, 0x0186, 0x018C, 0x8189, 0x0198, 0x819D, 0x8197,
00486                                     0x0192, 0x01B0, 0x81B5, 0x81BF, 0x01BA, 0x81AB, 0x01AE,
00487                                     0x01A4, 0x81A1, 0x01E0, 0x81E5, 0x81EF, 0x01EA, 0x81FB,
00488                                     0x01FE, 0x01F4, 0x81F1, 0x81D3, 0x01D6, 0x01DC, 0x81D9,
00489                                     0x01C8, 0x81CD, 0x81C7, 0x01C2, 0x0140, 0x8145, 0x814F,
00490                                     0x014A, 0x815B, 0x015E, 0x0154, 0x8151, 0x8173, 0x0176,
00491                                     0x017C, 0x8179, 0x0168, 0x816D, 0x8167, 0x0162, 0x8123,
00492                                     0x0126, 0x012C, 0x8129, 0x0138, 0x813D, 0x8137, 0x0132,
00493                                     0x0110, 0x8115, 0x811F, 0x011A, 0x810B, 0x010E, 0x0104,
00494                                     0x8101, 0x8303, 0x0306, 0x030C, 0x8309, 0x0318, 0x831D,
00495                                     0x8317, 0x0312, 0x0330, 0x8335, 0x833F, 0x033A, 0x832B,
00496                                     0x032E, 0x0324, 0x8321, 0x0360, 0x8365, 0x836F, 0x036A,
00497                                     0x837B, 0x037E, 0x0374, 0x8371, 0x8353, 0x0356, 0x035C,
00498                                     0x8359, 0x0348, 0x834D, 0x8347, 0x0342, 0x03C0, 0x83C5,
00499                                     0x83CF, 0x03CA, 0x83DB, 0x03DE, 0x03D4, 0x83D1, 0x83F3,
00500                                     0x03F6, 0x03FC, 0x83F9, 0x03E8, 0x83ED, 0x83E7, 0x03E2,
00501                                     0x83A3, 0x03A6, 0x03AC, 0x83A9, 0x03B8, 0x83BD, 0x83B7,
00502                                     0x03B2, 0x0390, 0x8395, 0x839F, 0x039A, 0x838B, 0x038E,
00503                                     0x0384, 0x8381, 0x0280, 0x8285, 0x828F, 0x028A, 0x829B,
00504                                     0x029E, 0x0294, 0x8291, 0x82B3, 0x02B6, 0x02BC, 0x82B9,
00505                                     0x02A8, 0x82AD, 0x82A7, 0x02A2, 0x82E3, 0x02E6, 0x02EC,
00506                                     0x82E9, 0x02F8, 0x82FD, 0x82F7, 0x02F2, 0x02D0, 0x82D5,
00507                                     0x82DF, 0x02DA, 0x82CB, 0x02CE, 0x02C4, 0x82C1, 0x8243,
00508                                     0x0246, 0x024C, 0x8249, 0x0258, 0x825D, 0x8257, 0x0252,
00509                                     0x0270, 0x8275, 0x827F, 0x027A, 0x826B, 0x026E, 0x0264,
00510                                     0x8261, 0x0220, 0x8225, 0x822F, 0x022A, 0x823B, 0x0236,
00511                                     0x0234, 0x8231, 0x8213, 0x0216, 0x021C, 0x8219, 0x0208,
00512                                     0x820D, 0x8207, 0x0202 };
00513
00514     for(j = 0; j < data_blk_size; j++)
00515     {
00516         i = ((unsigned short)(crc_accum >> 8) ^ *data_blk_ptr++) & 0xFF;
00517         crc_accum = (crc_accum << 8) ^ crc_table[i];
00518     }
00519     return crc_accum;
00520 }
00521

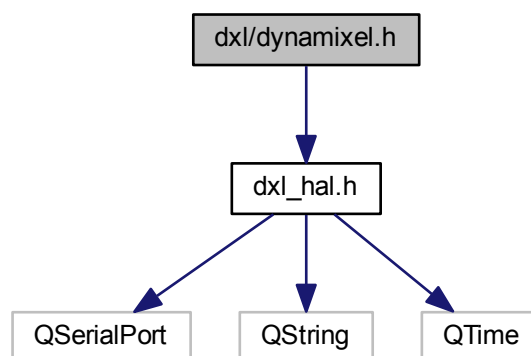
```

4.6 dxl/dynamixel.h File Reference

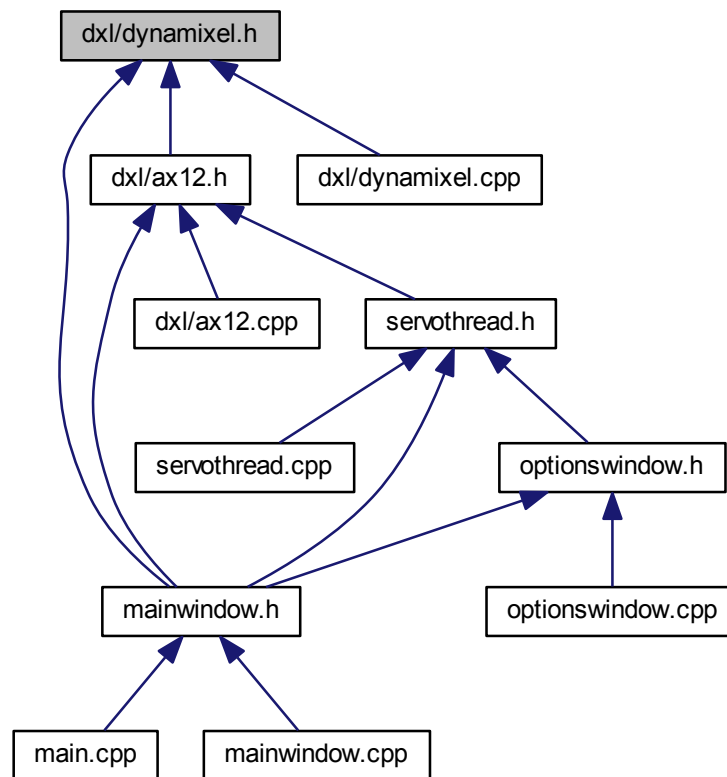
Contains the dynamixel and [dynamixel2](#) classes declaration.

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

Include dependency graph for dynamixel.h:



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



Classes

- class [dynamixel](#)
Dynamixel 1.0 protocol class.
- class [dynamixel2](#)
Dynamixel 2.0 protocol class.
- struct [dynamixel2::ping_data](#)
Struct used to do a ping.
- struct [dynamixel2::data](#)
Struct used to handle dynamixel data.

Macros

- `#define` [MAX_ID](#) (252)
- `#define` [BROADCAST_ID](#) (254)
- `#define` [COMM_TXSUCCESS](#) (0)
- `#define` [COMM_RXSUCCESS](#) (1)
- `#define` [COMM_TXFAIL](#) (2)
- `#define` [COMM_RXFAIL](#) (3)
- `#define` [COMM_TXERROR](#) (4)

- #define [COMM_RXWAITING](#) (5)
- #define [COMM_RXTIMEOUT](#) (6)
- #define [COMM_RXCORRUPT](#) (7)
- #define [ERRBIT_ALERT](#) (128)
- #define [ERR_RESULT_FAIL](#) (1)
- #define [ERR_INSTRUCTION](#) (2)
- #define [ERR_CRC](#) (3)
- #define [ERR_DATA_RANGE](#) (4)
- #define [ERR_DATA_LENGTH](#) (5)
- #define [ERR_DATA_LIMIT](#) (6)
- #define [ERR_ACCESS](#) (7)
- #define [PRT1_PKT_ID](#) (2)
- #define [PRT1_PKT_LENGTH](#) (3)
- #define [PRT1_PKT_INSTRUCTION](#) (4)
- #define [PRT1_PKT_ERRBIT](#) (4)
- #define [PRT1_PKT_PARAMETER0](#) (5)
- #define [PRT2_PKT_HEADER0](#) (0)
- #define [PRT2_PKT_HEADER1](#) (1)
- #define [PRT2_PKT_HEADER2](#) (2)
- #define [PRT2_PKT_RESERVED](#) (3)
- #define [PRT2_PKT_ID](#) (4)
- #define [PRT2_PKT_LENGTH_L](#) (5)
- #define [PRT2_PKT_LENGTH_H](#) (6)
- #define [PRT2_PKT_INSTRUCTION](#) (7)
- #define [PRT2_INSTRUCTION_PKT_PARAMETER0](#) (8)
- #define [PRT2_PKT_ERRBIT](#) (8)
- #define [PRT2_STATUS_PKT_PARAMETER0](#) (9)
- #define [INST_PING](#) (1)
- #define [INST_READ](#) (2)
- #define [INST_WRITE](#) (3)
- #define [INST_REG_WRITE](#) (4)
- #define [INST_ACTION](#) (5)
- #define [INST_RESET](#) (6)
- #define [INST_SYNC_WRITE](#) (131)
- #define [INST_BULK_READ](#) (146)
- #define [INST_REBOOT](#) (8)
- #define [INST_STATUS](#) (85)
- #define [INST_SYNC_READ](#) (130)
- #define [INST_BULK_WRITE](#) (147)
- #define [PING_INFO_MODEL_NUM](#) (1)
- #define [PING_INFO_FIRM_VER](#) (2)
- #define [MAKEWORD](#)(a, b) (((unsigned short)((((unsigned char)((((unsigned long)(a) & 0xff)) | ((unsigned short)((unsigned char)((((unsigned long)(b) & 0xff))) << 8))
- #define [MAKEDWORD](#)(a, b) (((unsigned int)((((unsigned short)((((unsigned long)(a) & 0xffff)) | ((unsigned int)((unsigned short)((((unsigned long)(b) & 0xffff))) << 16))
- #define [LOWORD](#)(l) (((unsigned short)((((unsigned long)(l) & 0xffff))
- #define [HIWORD](#)(l) (((unsigned short)((((unsigned long)(l) >> 16) & 0xffff))
- #define [LOBYTE](#)(w) (((unsigned char)((((unsigned long)(w) & 0xff))
- #define [HIBYTE](#)(w) (((unsigned char)((((unsigned long)(w) >> 8) & 0xff))

4.6.1 Detailed Description

Contains the dynamixel and [dynamixel2](#) classes declaration.

4.6.2 Macro Definition Documentation

4.6.2.1 `#define BROADCAST_ID (254)`

4.6.2.2 `#define COMM_RXCORRUPT (7)`

4.6.2.3 `#define COMM_RXFAIL (3)`

4.6.2.4 `#define COMM_RXSUCCESS (1)`

4.6.2.5 `#define COMM_RXTIMEOUT (6)`

4.6.2.6 `#define COMM_RXWAITING (5)`

4.6.2.7 `#define COMM_TXERROR (4)`

4.6.2.8 `#define COMM_TXFAIL (2)`

4.6.2.9 `#define COMM_TXSUCCESS (0)`

4.6.2.10 `#define ERR_ACCESS (7)`

4.6.2.11 `#define ERR_CRC (3)`

4.6.2.12 `#define ERR_DATA_LENGTH (5)`

4.6.2.13 `#define ERR_DATA_LIMIT (6)`

4.6.2.14 `#define ERR_DATA_RANGE (4)`

4.6.2.15 `#define ERR_INSTRUCTION (2)`

4.6.2.16 `#define ERR_RESULT_FAIL (1)`

4.6.2.17 `#define ERRBIT_ALERT (128)`

4.6.2.18 `#define HIBYTE(w) (((unsigned char)((((unsigned long)(w)) >> 8) & 0xff))`

4.6.2.19 `#define HIWORD(l) (((unsigned short)((((unsigned long)(l)) >> 16) & 0xffff))`

4.6.2.20 `#define INST_ACTION (5)`

4.6.2.21 `#define INST_BULK_READ (146)`

4.6.2.22 `#define INST_BULK_WRITE (147)`

4.6.2.23 `#define INST_PING (1)`

4.6.2.24 `#define INST_READ (2)`

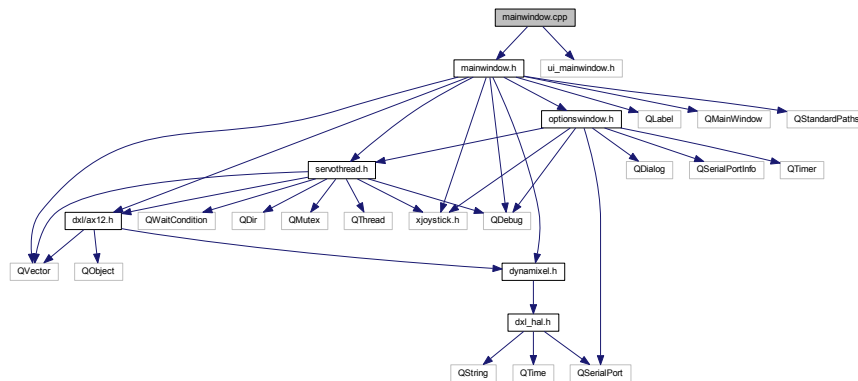
4.6.2.25 `#define INST_REBOOT (8)`

4.6.2.26 `#define INST_REG_WRITE (4)`

4.6.2.27 `#define INST_RESET (6)`

- 4.6.2.28 `#define INST_STATUS (85)`
- 4.6.2.29 `#define INST_SYNC_READ (130)`
- 4.6.2.30 `#define INST_SYNC_WRITE (131)`
- 4.6.2.31 `#define INST_WRITE (3)`
- 4.6.2.32 `#define LOBYTE(w) ((unsigned char)((((unsigned long)(w)) & 0xff))`
- 4.6.2.33 `#define LOWORD(l) ((unsigned short)((((unsigned long)(l)) & 0xffff))`
- 4.6.2.34 `#define MAKEDWORD(a, b) (((unsigned int)((((unsigned short)((((unsigned long)(a)) & 0xffff)) | ((unsigned int)((((unsigned short)((((unsigned long)(b)) & 0xffff))) << 16))`
- 4.6.2.35 `#define MAKEWORD(a, b) (((unsigned short)((((unsigned char)((((unsigned long)(a)) & 0xff)) | ((unsigned short)((((unsigned char)((((unsigned long)(b)) & 0xff))) << 8))`
- 4.6.2.36 `#define MAX_ID (252)`
- 4.6.2.37 `#define PING_INFO_FIRM_VER (2)`
- 4.6.2.38 `#define PING_INFO_MODEL_NUM (1)`
- 4.6.2.39 `#define PRT1_PKT_ERRBIT (4)`
- 4.6.2.40 `#define PRT1_PKT_ID (2)`
- 4.6.2.41 `#define PRT1_PKT_INSTRUCTION (4)`
- 4.6.2.42 `#define PRT1_PKT_LENGTH (3)`
- 4.6.2.43 `#define PRT1_PKT_PARAMETER0 (5)`
- 4.6.2.44 `#define PRT2_INSTRUCTION_PKT_PARAMETER0 (8)`
- 4.6.2.45 `#define PRT2_PKT_ERRBIT (8)`
- 4.6.2.46 `#define PRT2_PKT_HEADER0 (0)`
- 4.6.2.47 `#define PRT2_PKT_HEADER1 (1)`
- 4.6.2.48 `#define PRT2_PKT_HEADER2 (2)`
- 4.6.2.49 `#define PRT2_PKT_ID (4)`
- 4.6.2.50 `#define PRT2_PKT_INSTRUCTION (7)`
- 4.6.2.51 `#define PRT2_PKT_LENGTH_H (6)`
- 4.6.2.52 `#define PRT2_PKT_LENGTH_L (5)`
- 4.6.2.53 `#define PRT2_PKT_RESERVED (3)`
- 4.6.2.54 `#define PRT2_STATUS_PKT_PARAMETER0 (9)`

Include dependency graph for/mainwindow.cpp:



4.8.1 Detailed Description

Contains the [MainWindow](#) class implementation.

4.9/mainwindow.h File Reference

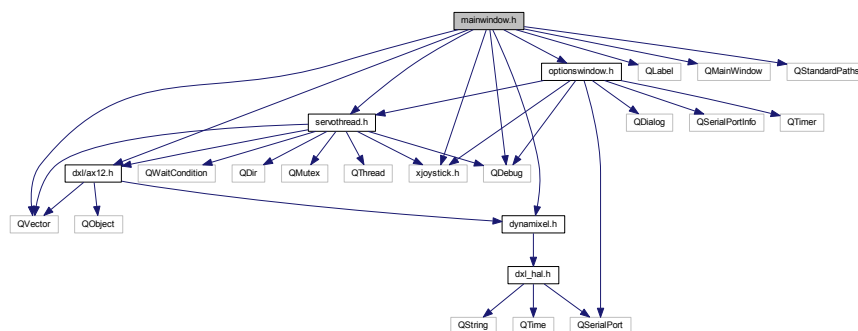
Contains the [MainWindow](#) class declaration.

```

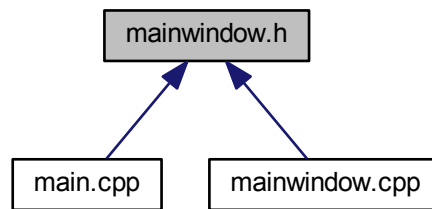
#include <QDebug>
#include <QLabel>
#include <QMainWindow>
#include <QVector>
#include <QStandardPaths>
#include <x joystick.h>
#include "dxl/ax12.h"
#include "dxl/dynamixel.h"
#include "optionswindow.h"
#include "servothread.h"

```

Include dependency graph for/mainwindow.h:



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



Classes

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

Namespaces

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

4.9.1 Detailed Description

Contains the [MainWindow](#) class declaration.

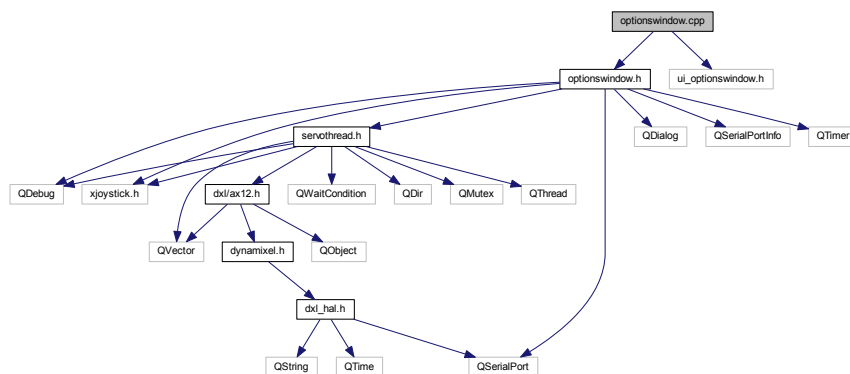
4.10 optionswindow.cpp File Reference

Contains the [OptionsWindow](#) class implementation.

```
#include "optionswindow.h"
```

```
#include "ui_optionswindow.h"
```

Include dependency graph for `optionswindow.cpp`:



4.10.1 Detailed Description

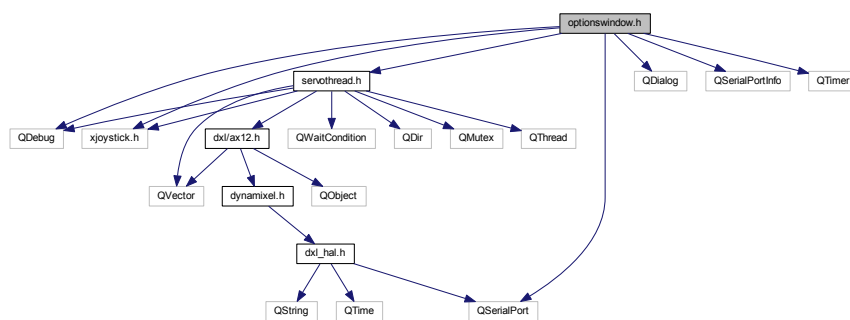
Contains the [OptionsWindow](#) class implementation.

4.11 optionswindow.h File Reference

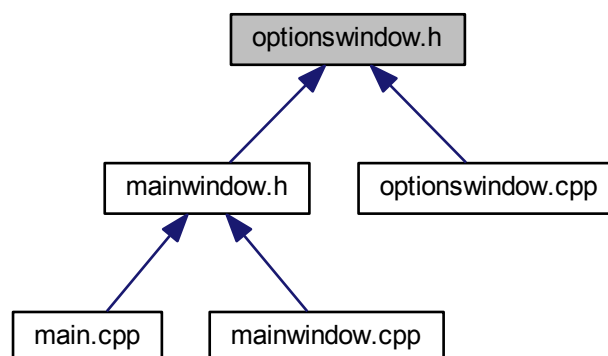
Contains the [OptionsWindow](#) class declaration.

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

Include dependency graph for optionswindow.h:



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



Classes

- class [OptionsWindow](#)

Class used to handle a Window to set the options.

Namespaces

- [Ui](#)

Namespace to work with a User Interface Qt Form.

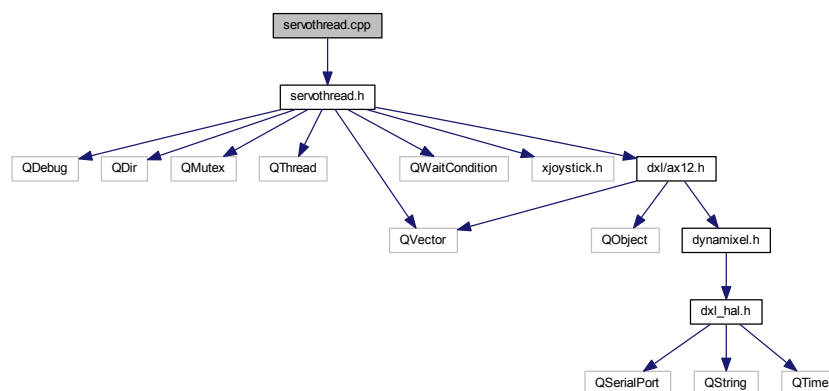
4.11.1 Detailed Description

Contains the [OptionsWindow](#) class declaration.

4.12 servothread.cpp File Reference

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

Include dependency graph for servothread.cpp:



4.13 servothread.h File Reference

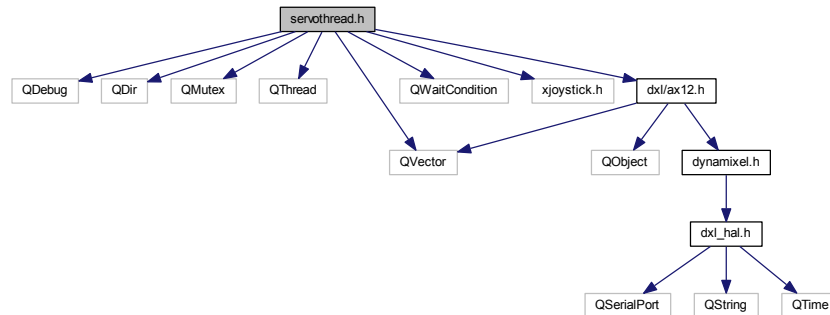
Contains the [ServoThread](#) class implementation.

```

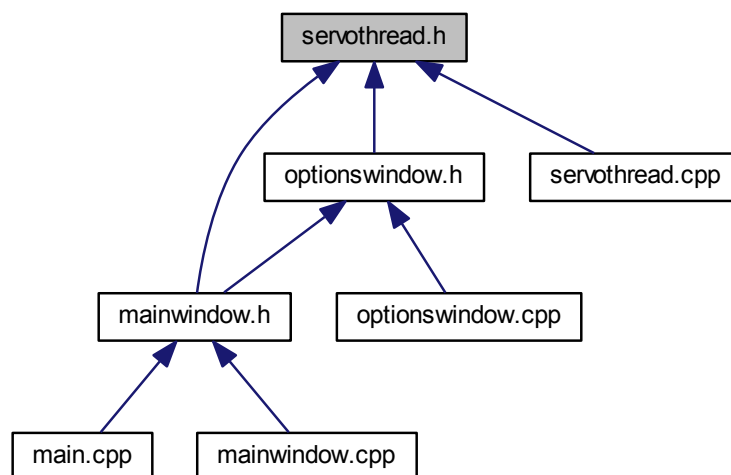
#include <QDebug>
#include <QDir>
#include <QMutex>
#include <QThread>
#include <QVector>
#include <QWaitCondition>
#include <xjoystick.h>
#include "dxl/ax12.h"

```


Include dependency graph for servothread.h:



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



Classes

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

4.13.1 Detailed Description

Contains the [ServoThread](#) class implementation.

Contains the [ServoThread](#) class declaration.

Index

- `_ID`
 - AX12, [12](#)
 - `_axis`
 - MainWindow, [43](#)
 - ServoThread, [58](#)
 - `_axisV`
 - MainWindow, [43](#)
 - `_buts`
 - MainWindow, [43](#)
 - ServoThread, [58](#)
 - `_butsV`
 - MainWindow, [43](#)
 - `_cBaud`
 - ServoThread, [58](#)
 - `_cPort`
 - ServoThread, [59](#)
 - `_cond`
 - ServoThread, [58](#)
 - `_dChanged`
 - ServoThread, [59](#)
 - `_dataP`
 - MainWindow, [43](#)
 - `_end`
 - ServoThread, [59](#)
 - `_jAxisX`
 - MainWindow, [43](#)
 - `_jAxisY`
 - MainWindow, [43](#)
 - `_jAxisZ`
 - MainWindow, [43](#)
 - `_joy`
 - MainWindow, [44](#)
 - OptionsWindow, [48](#)
 - `_mod`
 - ServoThread, [59](#)
 - `_mode`
 - AX12, [12](#)
 - `_mutex`
 - ServoThread, [59](#)
 - `_open`
 - dxl_hal, [14](#)
 - `_pause`
 - ServoThread, [59](#)
 - `_portSize`
 - OptionsWindow, [48](#)
 - `_sBaud`
 - ServoThread, [59](#)
 - `_sPort`
 - ServoThread, [59](#)
 - `_sPortChanged`
 - ServoThread, [59](#)
 - `_sT`
 - MainWindow, [44](#)
 - `_serial`
 - dxl_hal, [15](#)
 - `_servo`
 - OptionsWindow, [48](#)
 - `_servos`
 - ServoThread, [59](#)
 - `_time`
 - dxl_hal, [15](#)
 - `_timed`
 - dxl_hal, [15](#)
 - `_timer`
 - MainWindow, [44](#)
 - OptionsWindow, [48](#)
 - `~AX12`
 - AX12, [9](#)
 - `~MainWindow`
 - MainWindow, [42](#)
 - `~OptionsWindow`
 - OptionsWindow, [46](#)
 - `~ServoThread`
 - ServoThread, [53](#)
- `aSCount`
 - MainWindow, [44](#)
- `AX12`, [5](#)
 - `_ID`, [12](#)
 - `_mode`, [12](#)
 - `~AX12`, [9](#)
 - AX12, [9](#)
 - AlarmLED, [8](#)
 - AlarmShutdown, [8](#)
 - BaudRate, [8](#)
 - CCWAngleLimit, [8](#)
 - CCWComplianceMargin, [7](#)
 - CCWComplianceSlope, [7](#)
 - CWAngleLimit, [8](#)
 - CWComplianceMargin, [7](#)
 - CWComplianceSlope, [7](#)
 - connectedID, [9](#)
 - dxl, [12](#)
 - getCurrentLoad, [9](#)
 - getCurrentPos, [9](#)
 - getCurrentSpeed, [10](#)
 - getCurrentTemp, [10](#)
 - getCurrentVoltage, [10](#)
 - getID, [10](#)

- GoalPosition, [7](#)
- HighestLimitTemp, [8](#)
- HighestLimitVoltage, [8](#)
- ID, [8](#)
- LED, [7](#)
- Lock, [8](#)
- LowestLimitVoltage, [8](#)
- MaxTorque, [8](#)
- ModelNumber, [8](#)
- Moving, [8](#)
- MovingSpeed, [7](#)
- PresentLoad, [7](#)
- PresentPosition, [7](#)
- PresentSpeed, [7](#)
- PresentTemperature, [8](#)
- PresentVoltage, [8](#)
- Punch, [8](#)
- RAM, [7](#)
- ROM, [8](#)
- Registered, [8](#)
- ReturnDelayTime, [8](#)
- setGoalPosition, [10](#)
- setID, [10](#)
- setJointMode, [11](#)
- setMinMax, [11](#)
- setSpeed, [11](#)
- StatusReturnLevel, [8](#)
- TorqueEnable, [7](#)
- TorqueLimit, [7](#)
- VersionFirmware, [8](#)
- add_stuffing
 - dynamixel2, [25](#)
- AlarmLED
 - AX12, [8](#)
- AlarmShutdown
 - AX12, [8](#)
- BROADCAST_ID
 - dynamixel.h, [70](#)
- BaudRate
 - AX12, [8](#)
- broadcast_ping
 - dynamixel2, [25](#)
- CCWAngleLimit
 - AX12, [8](#)
- CCWComplianceMargin
 - AX12, [7](#)
- CCWComplianceSlope
 - AX12, [7](#)
- COMM_RXCORRUPT
 - dynamixel.h, [70](#)
- COMM_RXFAIL
 - dynamixel.h, [70](#)
- COMM_RXSUCCESS
 - dynamixel.h, [70](#)
- COMM_RXTIMEOUT
 - dynamixel.h, [70](#)
- COMM_RXWAITING
 - dynamixel.h, [70](#)
- COMM_TXERROR
 - dynamixel.h, [70](#)
- COMM_TXFAIL
 - dynamixel.h, [70](#)
- COMM_TXSUCCESS
 - dynamixel.h, [70](#)
- CWAngleLimit
 - AX12, [8](#)
- CWComplianceMargin
 - AX12, [7](#)
- CWComplianceSlope
 - AX12, [7](#)
- change_baudrate
 - dxl_hal, [13](#)
 - dynamixel, [16](#)
 - dynamixel2, [26](#)
- clear
 - dxl_hal, [13](#)
- close
 - dxl_hal, [13](#)
- connectedID
 - AX12, [9](#)
- cont
 - ServoThread, [54](#)
- controlled
 - ServoThread, [53](#)
- dH
 - dynamixel, [22](#)
 - dynamixel2, [38](#)
- dxl
 - AX12, [12](#)
- dxl/ax12.cpp, [61](#)
- dxl/ax12.h, [62](#)
- dxl/dxl_hal.cpp, [63](#)
- dxl/dxl_hal.h, [63](#)
- dxl/dynamixel.cpp, [65](#)
- dxl/dynamixel.h, [67](#)
- dxl_hal, [13](#)
 - _open, [14](#)
 - _serial, [15](#)
 - _time, [15](#)
 - _timed, [15](#)
 - change_baudrate, [13](#)
 - clear, [13](#)
 - close, [13](#)
 - get_curr_time, [13](#)
 - isOpen, [14](#)
 - open, [14](#)
 - read, [14](#)
 - write, [14](#)
- dxl_hal.h
 - MAXNUM_RXPACKET, [65](#)
 - MAXNUM_TXPACKET, [65](#)
- dynamixel, [15](#)
 - change_baudrate, [16](#)
 - dH, [22](#)
 - dynamixel, [16](#)

- gbCommStatus, [22](#)
- gbInstructionPacket, [22](#)
- gbRxGetLength, [22](#)
- gbRxPacketLength, [22](#)
- gbStatusPacket, [22](#)
- gdByteTransTime, [22](#)
- gdPacketStartTime, [22](#)
- gdRcvWaitTime, [22](#)
- get_comm_result, [16](#)
- get_packet_time, [17](#)
- get_rxpacket_error, [17](#)
- get_rxpacket_error_byte, [17](#)
- get_rxpacket_length, [17](#)
- get_rxpacket_parameter, [17](#)
- giBusUsing, [22](#)
- initialize, [17](#)
- is_packet_timeout, [17](#)
- isOpen, [18](#)
- ping, [18](#)
- read_byte, [18](#)
- read_word, [18](#)
- rx_packet, [18](#)
- set_packet_timeout, [20](#)
- set_packet_timeout_ms, [20](#)
- set_txpacket_id, [20](#)
- set_txpacket_instruction, [20](#)
- set_txpacket_length, [20](#)
- set_txpacket_parameter, [20](#)
- terminate, [20](#)
- tx_packet, [20](#)
- txrx_packet, [21](#)
- write_byte, [22](#)
- write_word, [22](#)
- dynamixel.cpp
 - LATENCY_TIME, [66](#)
 - PING_STATUS_LENGTH, [66](#)
 - update_crc, [66](#)
- dynamixel.h
 - BROADCAST_ID, [70](#)
 - COMM_RXCORRUPT, [70](#)
 - COMM_RXFAIL, [70](#)
 - COMM_RXSUCCESS, [70](#)
 - COMM_RXTIMEOUT, [70](#)
 - COMM_RXWAITING, [70](#)
 - COMM_TXERROR, [70](#)
 - COMM_TXFAIL, [70](#)
 - COMM_TXSUCCESS, [70](#)
 - ERR_ACCESS, [70](#)
 - ERR_CRC, [70](#)
 - ERR_DATA_LENGTH, [70](#)
 - ERR_DATA_LIMIT, [70](#)
 - ERR_DATA_RANGE, [70](#)
 - ERR_INSTRUCTION, [70](#)
 - ERR_RESULT_FAIL, [70](#)
 - ERRBIT_ALERT, [70](#)
 - HIBYTE, [70](#)
 - HIWORD, [70](#)
 - INST_ACTION, [70](#)
 - INST_BULK_READ, [70](#)
 - INST_BULK_WRITE, [70](#)
 - INST_PING, [70](#)
 - INST_READ, [70](#)
 - INST_REBOOT, [70](#)
 - INST_REG_WRITE, [70](#)
 - INST_RESET, [70](#)
 - INST_STATUS, [70](#)
 - INST_SYNC_READ, [71](#)
 - INST_SYNC_WRITE, [71](#)
 - INST_WRITE, [71](#)
 - LOBYTE, [71](#)
 - LOWORD, [71](#)
 - MAKEDWORD, [71](#)
 - MAKEWORD, [71](#)
 - MAX_ID, [71](#)
 - PING_INFO_FIRM_VER, [71](#)
 - PING_INFO_MODEL_NUM, [71](#)
 - PRT1_PKT_ERRBIT, [71](#)
 - PRT1_PKT_ID, [71](#)
 - PRT1_PKT_INSTRUCTION, [71](#)
 - PRT1_PKT_LENGTH, [71](#)
 - PRT1_PKT_PARAMETER0, [71](#)
 - PRT2_INSTRUCTION_PKT_PARAMETER0, [71](#)
 - PRT2_PKT_ERRBIT, [71](#)
 - PRT2_PKT_HEADER0, [71](#)
 - PRT2_PKT_HEADER1, [71](#)
 - PRT2_PKT_HEADER2, [71](#)
 - PRT2_PKT_ID, [71](#)
 - PRT2_PKT_INSTRUCTION, [71](#)
 - PRT2_PKT_LENGTH_H, [71](#)
 - PRT2_PKT_LENGTH_L, [71](#)
 - PRT2_PKT_RESERVED, [71](#)
 - PRT2_STATUS_PKT_PARAMETER0, [71](#)
- dynamixel2, [23](#)
 - add_stuffing, [25](#)
 - broadcast_ping, [25](#)
 - change_baudrate, [26](#)
 - dH, [38](#)
 - factory_reset, [26](#)
 - gBulkData, [39](#)
 - gPingData, [39](#)
 - gSyncData, [39](#)
 - gbCommStatus, [38](#)
 - gbInstructionPacket, [38](#)
 - gbRxGetLength, [38](#)
 - gbRxPacketLength, [38](#)
 - gbStatusPacket, [38](#)
 - gdByteTransTime, [39](#)
 - gdPacketStartTime, [39](#)
 - gdRcvWaitTime, [39](#)
 - get_bulk_read_data_byte, [26](#)
 - get_bulk_read_data_dword, [26](#)
 - get_bulk_read_data_word, [26](#)
 - get_comm_result, [27](#)
 - get_packet_time, [27](#)
 - get_ping_result, [27](#)
 - get_rxpacket_error_byte, [27](#)

- get_rxpacket_length, 27
- get_rxpacket_parameter, 27
- get_sync_read_data_byte, 27
- get_sync_read_data_dword, 28
- get_sync_read_data_word, 28
- giBusUsing, 39
- initialize, 28
- is_packet_timeout, 29
- isOpen, 29
- ping, 29
- PingData, 24
- read_byte, 29
- read_dword, 29
- read_word, 30
- reboot, 30
- remove_stuffing, 30
- rx_packet, 31
- set_packet_timeout, 32
- set_packet_timeout_ms, 32
- set_txpacket_id, 32
- set_txpacket_instruction, 32
- set_txpacket_length, 32
- set_txpacket_parameter, 33
- SyncBulkData, 24
- terminate, 33
- tx_packet, 33
- txrx_packet, 34
- write_byte, 37
- write_dword, 38
- write_word, 38
- dynamixel2::data, 12
 - iError, 12
 - iID, 12
 - iLength, 12
 - iStartAddr, 12
 - pucTable, 12
- dynamixel2::ping_data, 48
 - iFirmVer, 49
 - iID, 49
 - iModelNo, 49
- ERR_ACCESS
 - dynamixel.h, 70
- ERR_CRC
 - dynamixel.h, 70
- ERR_DATA_LENGTH
 - dynamixel.h, 70
- ERR_DATA_LIMIT
 - dynamixel.h, 70
- ERR_DATA_RANGE
 - dynamixel.h, 70
- ERR_INSTRUCTION
 - dynamixel.h, 70
- ERR_RESULT_FAIL
 - dynamixel.h, 70
- ERRBIT_ALERT
 - dynamixel.h, 70
- end
 - ServoThread, 54
- events
 - OptionsWindow, 47
- factory_reset
 - dynamixel2, 26
- gBulkData
 - dynamixel2, 39
- gPingData
 - dynamixel2, 39
- gSyncData
 - dynamixel2, 39
- gbCommStatus
 - dynamixel, 22
 - dynamixel2, 38
- gbInstructionPacket
 - dynamixel, 22
 - dynamixel2, 38
- gbRxGetLength
 - dynamixel, 22
 - dynamixel2, 38
- gbRxPacketLength
 - dynamixel, 22
 - dynamixel2, 38
- gbStatusPacket
 - dynamixel, 22
 - dynamixel2, 38
- gdByteTransTime
 - dynamixel, 22
 - dynamixel2, 39
- gdPacketStartTime
 - dynamixel, 22
 - dynamixel2, 39
- gdRcvWaitTime
 - dynamixel, 22
 - dynamixel2, 39
- get_bulk_read_data_byte
 - dynamixel2, 26
- get_bulk_read_data_dword
 - dynamixel2, 26
- get_bulk_read_data_word
 - dynamixel2, 26
- get_comm_result
 - dynamixel, 16
 - dynamixel2, 27
- get_curr_time
 - dxl_hal, 13
- get_packet_time
 - dynamixel, 17
 - dynamixel2, 27
- get_ping_result
 - dynamixel2, 27
- get_rxpacket_error
 - dynamixel, 17
- get_rxpacket_error_byte
 - dynamixel, 17
 - dynamixel2, 27
- get_rxpacket_length
 - dynamixel, 17

- dynamixel2, [27](#)
- get_rxpacket_parameter
 - dynamixel, [17](#)
 - dynamixel2, [27](#)
- get_sync_read_data_byte
 - dynamixel2, [27](#)
- get_sync_read_data_dword
 - dynamixel2, [28](#)
- get_sync_read_data_word
 - dynamixel2, [28](#)
- getCurrentLoad
 - AX12, [9](#)
- getCurrentPos
 - AX12, [9](#)
- getCurrentSpeed
 - AX12, [10](#)
- getCurrentTemp
 - AX12, [10](#)
- getCurrentVoltage
 - AX12, [10](#)
- getID
 - AX12, [10](#)
- getServoBaud
 - ServoThread, [54](#)
- getServoPort
 - ServoThread, [54](#)
- getServoPortInfo
 - ServoThread, [54](#)
- getServosInfo
 - ServoThread, [54](#), [56](#)
- giBusUsing
 - dynamixel, [22](#)
 - dynamixel2, [39](#)
- GoalPosition
 - AX12, [7](#)
- HIBYTE
 - dynamixel.h, [70](#)
- HIWORD
 - dynamixel.h, [70](#)
- HighestLimitTemp
 - AX12, [8](#)
- HighestLimitVoltage
 - AX12, [8](#)
- ID
 - AX12, [8](#)
 - ServoThread::Servo, [50](#)
- iError
 - dynamixel2::data, [12](#)
- iFirmVer
 - dynamixel2::ping_data, [49](#)
- iID
 - dynamixel2::data, [12](#)
 - dynamixel2::ping_data, [49](#)
- iLength
 - dynamixel2::data, [12](#)
- iModelNo
 - dynamixel2::ping_data, [49](#)
- INST_ACTION
 - dynamixel.h, [70](#)
- INST_BULK_READ
 - dynamixel.h, [70](#)
- INST_BULK_WRITE
 - dynamixel.h, [70](#)
- INST_PING
 - dynamixel.h, [70](#)
- INST_READ
 - dynamixel.h, [70](#)
- INST_REBOOT
 - dynamixel.h, [70](#)
- INST_REG_WRITE
 - dynamixel.h, [70](#)
- INST_RESET
 - dynamixel.h, [70](#)
- INST_STATUS
 - dynamixel.h, [70](#)
- INST_SYNC_READ
 - dynamixel.h, [71](#)
- INST_SYNC_WRITE
 - dynamixel.h, [71](#)
- INST_WRITE
 - dynamixel.h, [71](#)
- iStartAddr
 - dynamixel2::data, [12](#)
- initialize
 - dynamixel, [17](#)
 - dynamixel2, [28](#)
- is_packet_timeout
 - dynamixel, [17](#)
 - dynamixel2, [29](#)
- isOpen
 - dxl_hal, [14](#)
 - dynamixel, [18](#)
 - dynamixel2, [29](#)
- joyChanged
 - MainWindow, [42](#)
- joystickChanged
 - MainWindow, [42](#)
 - OptionsWindow, [47](#)
- LATENCY_TIME
 - dynamixel.cpp, [66](#)
- LED
 - AX12, [7](#)
- LOBYTE
 - dynamixel.h, [71](#)
- LOWORD
 - dynamixel.h, [71](#)
- load
 - ServoThread, [56](#)
 - ServoThread::Servo, [50](#)
- Lock
 - AX12, [8](#)
- LowestLimitVoltage
 - AX12, [8](#)

- MAKEDWORD
 - dynamixel.h, 71
- MAKEWORD
 - dynamixel.h, 71
- MAX_ID
 - dynamixel.h, 71
- MAXNUM_RXPACKET
 - dxl_hal.h, 65
- MAXNUM_TXPACKET
 - dxl_hal.h, 65
- main
 - main.cpp, 72
- main.cpp, 72
 - main, 72
- MainWindow, 39
 - _axis, 43
 - _axisV, 43
 - _buts, 43
 - _butsV, 43
 - _dataP, 43
 - _jAxisX, 43
 - _jAxisY, 43
 - _jAxisZ, 43
 - _joy, 44
 - _sT, 44
 - _timer, 44
 - ~MainWindow, 42
 - aSCount, 44
 - joyChanged, 42
 - joystickChanged, 42
 - MainWindow, 41
 - on_actionOptions_triggered, 42
 - sCount, 44
 - ui, 44
 - update, 43
- mainwindow.cpp, 72
- mainwindow.h, 73
- manual
 - ServoThread, 53
- MaxTorque
 - AX12, 8
- Mode
 - ServoThread, 53
- ModelNumber
 - AX12, 8
- Moving
 - AX12, 8
- MovingSpeed
 - AX12, 7
- on_actionOptions_triggered
 - MainWindow, 42
- on_servoRefresh_clicked
 - OptionsWindow, 47
- open
 - dxl_hal, 14
- OptionsWindow, 44
 - _joy, 48
 - _portSize, 48
 - _servo, 48
 - _timer, 48
 - ~OptionsWindow, 46
 - events, 47
 - joystickChanged, 47
 - on_servoRefresh_clicked, 47
 - OptionsWindow, 46
 - storeData, 48
 - ui, 48
- optionswindow.cpp, 74
- optionswindow.h, 75
- PING_INFO_FIRM_VER
 - dynamixel.h, 71
- PING_INFO_MODEL_NUM
 - dynamixel.h, 71
- PING_STATUS_LENGTH
 - dynamixel.cpp, 66
- PRT1_PKT_ERRBIT
 - dynamixel.h, 71
- PRT1_PKT_ID
 - dynamixel.h, 71
- PRT1_PKT_INSTRUCTION
 - dynamixel.h, 71
- PRT1_PKT_LENGTH
 - dynamixel.h, 71
- PRT1_PKT_PARAMETER0
 - dynamixel.h, 71
- PRT2_INSTRUCTION_PKT_PARAMETER0
 - dynamixel.h, 71
- PRT2_PKT_ERRBIT
 - dynamixel.h, 71
- PRT2_PKT_HEADER0
 - dynamixel.h, 71
- PRT2_PKT_HEADER1
 - dynamixel.h, 71
- PRT2_PKT_HEADER2
 - dynamixel.h, 71
- PRT2_PKT_ID
 - dynamixel.h, 71
- PRT2_PKT_INSTRUCTION
 - dynamixel.h, 71
- PRT2_PKT_LENGTH_H
 - dynamixel.h, 71
- PRT2_PKT_LENGTH_L
 - dynamixel.h, 71
- PRT2_PKT_RESERVED
 - dynamixel.h, 71
- PRT2_STATUS_PKT_PARAMETER0
 - dynamixel.h, 71
- pause
 - ServoThread, 56
- ping
 - dynamixel, 18
 - dynamixel2, 29
- PingData
 - dynamixel2, 24
- pos
 - ServoThread::Servo, 50

- PresentLoad
 - AX12, [7](#)
- PresentPosition
 - AX12, [7](#)
- PresentSpeed
 - AX12, [7](#)
- PresentTemperature
 - AX12, [8](#)
- PresentVoltage
 - AX12, [8](#)
- pucTable
 - dynamixel2::data, [12](#)
- Punch
 - AX12, [8](#)
- RAM
 - AX12, [7](#)
- ROM
 - AX12, [8](#)
- read
 - dxl_hal, [14](#)
- read_byte
 - dynamixel, [18](#)
 - dynamixel2, [29](#)
- read_dword
 - dynamixel2, [29](#)
- read_word
 - dynamixel, [18](#)
 - dynamixel2, [30](#)
- reboot
 - dynamixel2, [30](#)
- Registered
 - AX12, [8](#)
- remove_stuffing
 - dynamixel2, [30](#)
- ReturnDelayTime
 - AX12, [8](#)
- run
 - ServoThread, [56](#)
- rx_packet
 - dynamixel, [18](#)
 - dynamixel2, [31](#)
- sCount
 - MainWindow, [44](#)
- Servo
 - ServoThread::Servo, [50](#)
- ServoThread, [50](#)
 - _axis, [58](#)
 - _buts, [58](#)
 - _cBaud, [58](#)
 - _cPort, [59](#)
 - _cond, [58](#)
 - _dChanged, [59](#)
 - _end, [59](#)
 - _mod, [59](#)
 - _mutex, [59](#)
 - _pause, [59](#)
 - _sBaud, [59](#)
 - _sPort, [59](#)
 - _sPortChanged, [59](#)
 - _servos, [59](#)
 - ~ServoThread, [53](#)
 - cont, [54](#)
 - controlled, [53](#)
 - end, [54](#)
 - getServoBaud, [54](#)
 - getServoPort, [54](#)
 - getServoPortInfo, [54](#)
 - getServosInfo, [54](#), [56](#)
 - load, [56](#)
 - manual, [53](#)
 - Mode, [53](#)
 - pause, [56](#)
 - run, [56](#)
 - ServoThread, [53](#)
 - setData, [57](#)
 - setSID, [58](#)
 - setServoBaud, [57](#)
 - setServoPort, [57](#)
 - v_1_0, [53](#)
 - Version, [53](#)
 - write, [58](#)
- ServoThread::Servo, [49](#)
 - ID, [50](#)
 - load, [50](#)
 - pos, [50](#)
 - Servo, [50](#)
- servothread.cpp, [76](#)
- servothread.h, [76](#)
- set_packet_timeout
 - dynamixel, [20](#)
 - dynamixel2, [32](#)
- set_packet_timeout_ms
 - dynamixel, [20](#)
 - dynamixel2, [32](#)
- set_txpacket_id
 - dynamixel, [20](#)
 - dynamixel2, [32](#)
- set_txpacket_instruction
 - dynamixel, [20](#)
 - dynamixel2, [32](#)
- set_txpacket_length
 - dynamixel, [20](#)
 - dynamixel2, [32](#)
- set_txpacket_parameter
 - dynamixel, [20](#)
 - dynamixel2, [33](#)
- setData
 - ServoThread, [57](#)
- setGoalPosition
 - AX12, [10](#)
- setID
 - AX12, [10](#)
- setJointMode
 - AX12, [11](#)
- setMinMax

- AX12, [11](#)
- setSID
 - ServoThread, [58](#)
- setServoBaud
 - ServoThread, [57](#)
- setServoPort
 - ServoThread, [57](#)
- setSpeed
 - AX12, [11](#)
- StatusReturnLevel
 - AX12, [8](#)
- storeData
 - OptionsWindow, [48](#)
- SyncBulkData
 - dynamixel2, [24](#)
- terminate
 - dynamixel, [20](#)
 - dynamixel2, [33](#)
- TorqueEnable
 - AX12, [7](#)
- TorqueLimit
 - AX12, [7](#)
- tx_packet
 - dynamixel, [20](#)
 - dynamixel2, [33](#)
- txrx_packet
 - dynamixel, [21](#)
 - dynamixel2, [34](#)
- Ui, [3](#)
- ui
 - MainWindow, [44](#)
 - OptionsWindow, [48](#)
- update
 - MainWindow, [43](#)
- update_crc
 - dynamixel.cpp, [66](#)
- v_1_0
 - ServoThread, [53](#)
- Version
 - ServoThread, [53](#)
- VersionFirmware
 - AX12, [8](#)
- write
 - dxl_hal, [14](#)
 - ServoThread, [58](#)
- write_byte
 - dynamixel, [22](#)
 - dynamixel2, [37](#)
- write_dword
 - dynamixel2, [38](#)
- write_word
 - dynamixel, [22](#)
 - dynamixel2, [38](#)