

DeltaRobot
v0.4

Generated by Doxygen 1.8.9.1

Sat Jun 6 2015 13:05:15

Contents

1	Main Page	1
2	Namespace Documentation	1
2.1	Ui Namespace Reference	1
2.1.1	Detailed Description	1
3	Class Documentation	1
3.1	AX12 Class Reference	1
3.1.1	Detailed Description	3
3.1.2	Member Enumeration Documentation	3
3.1.3	Constructor & Destructor Documentation	5
3.1.4	Member Function Documentation	6
3.1.5	Member Data Documentation	9
3.2	ServoThread::Dominoe Struct Reference	9
3.2.1	Detailed Description	10
3.2.2	Constructor & Destructor Documentation	10
3.2.3	Member Function Documentation	10
3.2.4	Member Data Documentation	11
3.3	dxl_hal Class Reference	11
3.3.1	Detailed Description	11
3.3.2	Member Function Documentation	12
3.3.3	Member Data Documentation	13
3.4	dynamixel Class Reference	13
3.4.1	Detailed Description	15
3.4.2	Constructor & Destructor Documentation	15
3.4.3	Member Function Documentation	16
3.4.4	Member Data Documentation	23
3.5	MainWindow Class Reference	24
3.5.1	Detailed Description	27
3.5.2	Member Typedef Documentation	27
3.5.3	Member Enumeration Documentation	27
3.5.4	Constructor & Destructor Documentation	27
3.5.5	Member Function Documentation	28
3.5.6	Member Data Documentation	31
3.6	OptionsWindow Class Reference	32
3.6.1	Detailed Description	34
3.6.2	Member Typedef Documentation	34
3.6.3	Constructor & Destructor Documentation	34
3.6.4	Member Function Documentation	35

3.6.5	Member Data Documentation	37
3.7	ServoThread::Servo Struct Reference	38
3.7.1	Detailed Description	38
3.7.2	Constructor & Destructor Documentation	38
3.7.3	Member Function Documentation	39
3.7.4	Member Data Documentation	39
3.8	ServoFind Class Reference	39
3.8.1	Member Typedef Documentation	41
3.8.2	Constructor & Destructor Documentation	41
3.8.3	Member Function Documentation	41
3.8.4	Member Data Documentation	42
3.9	ServoThread Class Reference	42
3.9.1	Detailed Description	47
3.9.2	Member Enumeration Documentation	47
3.9.3	Constructor & Destructor Documentation	47
3.9.4	Member Function Documentation	48
3.9.5	Member Data Documentation	58
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	61
4.2.1	Detailed Description	61
4.3	dxl/dxl_hal.cpp File Reference	61
4.3.1	Detailed Description	61
4.4	dxl/dxl_hal.h File Reference	62
4.4.1	Detailed Description	62
4.5	dxl/dynamixel.cpp File Reference	62
4.5.1	Detailed Description	62
4.6	dxl/dynamixel.h File Reference	62
4.6.1	Detailed Description	62
4.7	main.cpp File Reference	62
4.7.1	Detailed Description	62
4.7.2	Function Documentation	63
4.8	mainwindow.cpp File Reference	63
4.8.1	Detailed Description	63
4.9	mainwindow.h File Reference	63
4.9.1	Detailed Description	63
4.10	optionwindow.cpp File Reference	63
4.10.1	Detailed Description	63

4.11	optionwindow.h File Reference	63
4.11.1	Detailed Description	64
4.12	servofind.cpp File Reference	64
4.13	servofind.h File Reference	64
4.14	servothread.cpp File Reference	64
4.14.1	Detailed Description	64
4.15	servothread.h File Reference	64
4.15.1	Detailed Description	64
4.16	stable.h File Reference	64
4.16.1	Detailed Description	65
Index		67

1 Main Page

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

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.

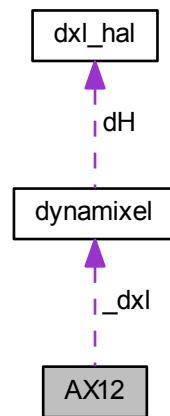
3 Class Documentation

3.1 AX12 Class Reference

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

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

Collaboration diagram for AX12:



Public 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.

Public Member Functions

- `AX12 ()`
Default constructor.
- `AX12 (dynamixel * _dxl, int ID=-1)`
Initializer constructor if ID == -1 no action is done.
- `AX12 (const AX12 &a)`
Copy constructor.
- `~AX12 ()`
Default destructor.
- `QVector< int > connectedID ()`
Returns all active servos;.
- `double getCurrentLoad ()`
Returns the current load from -100% to 100%, 100% is ClockWise and -100% is CounterClockWise.
- `double getCurrentPos ()`

- Returns the current position from 0° to 300°*
 - int `getCurrentTemp` ()
 - Returns the current Temperature in Celsius.*
 - double `getCurrentSpeed` ()
 - Returns the current speed from -100% to 100%, 100% is ClockWise and -100% is CounterClockWise.*
 - double `getCurrentVoltage` ()
 - Returns the current voltage in Volts.*
 - int `getID` ()
 - To get the current ID.*
 - void `setComplianceSlope` (uchar ccw, uchar cw)
 - Sets the compliance slope.*
 - void `setDxl` (dynamixel *dxl)
 - Sets the dynamixel interface.*
 - void `setGoalPosition` (double goal)
 - Sets the Goal's position (in degrees) or speed depending on the mode.*
 - void `setID` (int ID)
 - To set a new ID.*
 - void `setJointMode` (bool mode)
 - To set Joint/Wheel mode.*
 - void `setMinMax` (double min, double max)
 - To set the minimum and maximum angle from 0 to 300°*
 - void `setRadians` (bool rads)
 - Sets the radians mode.*
 - 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 Attributes

- dynamixel * `_dxl`
 - Contains the dynamixel communication.*
- int `_ID`
 - Stores the current ID.*
- bool `_mode`
 - True if we use the joint mode.*
- bool `_rads`
 - True if the angle is returned in radians.*

3.1.1 Detailed Description

The `AX12` class is used to control AX-12 motors from Dynamixel.

3.1.2 Member Enumeration Documentation

3.1.2.1 enum `AX12::RAM`

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

```

00058      {
00059          TorqueEnable      = 24,
00060          LED               = 25,
00061          CWComplianceMargin = 26,
00062          CCWComplianceMargin = 27,
00063          CWComplianceSlope = 28,
00064          CCWComplianceSlope = 29,
00065          GoalPosition      = 30,
00066          MovingSpeed       = 32,
00067          TorqueLimit       = 34,
00068          PresentPosition   = 36,
00069          PresentSpeed      = 38,
00070          PresentLoad       = 40,
00071          PresentVoltage    = 42,
00072          PresentTemperature = 43,
00073          Registered        = 44,
00074          Moving           = 46,
00075          Lock              = 47,
00076          Punch            = 48
00077      };
00078

```

3.1.2.2 enum AX12::ROM

Contains all the EEPROM directions enumeration.

Enumerator

ModelNumber
VersionFirmware
ID
BaudRate
ReturnDelayTime
CWAngleLimit
CCWAngleLimit
HighestLimitTemp
LowestLimitVoltage
HighestLimitVoltage
MaxTorque
StatusReturnLevel

AlarmLED**AlarmShutdown**

```

00039     {
00040         ModelNumber      = 0,
00041         VersionFirmware  = 2,
00042         ID               = 3,
00043         BaudRate         = 4,
00044         ReturnDelayTime  = 5,
00045         CWAngleLimit     = 6,
00046         CCWAngleLimit    = 8,
00047         HighestLimitTemp  = 11,
00048         LowestLimitVoltage = 12,
00049         HighestLimitVoltage = 13,
00050         MaxTorque         = 14,
00051         StatusReturnLevel = 16,
00052         AlarmLED         = 17,
00053         AlarmShutdown    = 18
00054     };

```

3.1.3 Constructor & Destructor Documentation**3.1.3.1 AX12::AX12 ()**

Default constructor.

```

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

```

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

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

```

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

```

3.1.3.3 AX12::AX12 (const AX12 & a)

Copy constructor.

```

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

```

3.1.3.4 AX12::~AX12 ()

Default destructor.

```

00034 {
00035
00036 }

```


3.1.4 Member Function Documentation

3.1.4.1 QVector< int > AX12::connectedID ()

Returns all active servos;

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

3.1.4.2 double AX12::getCurrentLoad ()

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

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

3.1.4.3 double AX12::getCurrentPos ()

Returns the current position from 0° to 300°

```
00060 {
00061     if (_ID < 0 or _dxl == NULL) return 0;
00062     int pos = _dxl->read_word(_ID, RAM::PresentPosition);
00063     if (_dxl->get_comm_result() != COMM_RXSUCCESS) return -1;
00064
00065     if (_rads) return double((pos/1023.0)*(5.0*M_PI)/3.0);
00066     return double((pos/1023.0)*300);
00067 }
```

3.1.4.4 double AX12::getCurrentSpeed ()

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

```
00078 {
00079     if (_ID < 0 or _dxl == NULL) return 0;
00080     int speed = _dxl->read_word(_ID, RAM::PresentSpeed);
00081     if (_dxl->get_comm_result() != COMM_RXSUCCESS) return -1;
00082     speed -= 1024;
00083     if (speed == -1024) speed = 0;
00084     return double((speed/1023.0)*100);
00085 }
```

3.1.4.5 int AX12::getCurrentTemp ()

Returns the current Temperature in Celsius.

```
00070 {
00071     if (_ID < 0 or _dxl == NULL) return 0;
00072     int temp = _dxl->read_byte(_ID, RAM::PresentTemperature);
00073     if (_dxl->get_comm_result() != COMM_RXSUCCESS) return -1;
00074     return temp;
00075 }
```

3.1.4.6 double AX12::getCurrentVoltage ()

Returns the current voltage in Volts.

```
00088 {
00089     if (_ID < 0 or _dxl == NULL) return 0;
00090     char voltage = _dxl->read_byte(_ID, RAM::PresentVoltage);
00091     if (_dxl->get_comm_result() != COMM_RXSUCCESS) return -1;
00092     return double(voltage/10.0);
00093 }
```

3.1.4.7 int AX12::getID () [inline]

To get the current ID.

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

3.1.4.8 void AX12::setComplianceSlope (uchar ccw, uchar cw)

Sets the compliance slope.

Parameters

<i>ccw</i>	Counter Clock Wise Compliance Slope
<i>cw</i>	Clock Wise Compliance Slope

```
00096 {
00097     if (_ID < 0 or _dxl == NULL) return;
00098     _dxl->write_byte(_ID, RAM::CCWComplianceMargin, ccw);
00099     _dxl->write_byte(_ID, RAM::CWComplianceMargin, cw);
00100 }
```

3.1.4.9 void AX12::setDxl (dynamixel * dxl) [inline]

Sets the dynamixel interface.

Parameters

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

```
00123 { _dxl = dxl; }
```

3.1.4.10 void AX12::setGoalPosition (double goal)

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

Parameters

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

```
00103 {
00104     if (_ID < 0 or _dxl == NULL) return;
00105
00106     // Conversion to radians if radians mode
00107     if (_rads) goal *= 180/M_PI;
00108
00109     if (goal > 300.0) goal = 300.0;
00110     else if (goal < 0) goal = 0;
00111     _dxl->write_word(_ID, RAM::GoalPosition, int((goal/300.0)*1023));
00112 }
```

3.1.4.11 void AX12::setID (int ID)

To set a new ID.

Parameters

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

```

00115 {
00116     _ID = ID;
00117     if (_ID < 0 or _dxl == NULL) return;
00118     _dxl->write_byte(_ID, RAM::TorqueEnable, true);
00119 }

```

3.1.4.12 void AX12::setJointMode (bool mode)

To set Joint/Wheel mode.

Parameters

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

```

00122 {
00123     if (_ID < 0 or _dxl == NULL) return;
00124     _mode = mode;
00125     if (_mode) {
00126         _dxl->write_word(_ID, ROM::CWAngleLimit, 0);
00127         _dxl->write_word(_ID, ROM::CCWAngleLimit, 1023);
00128     }
00129     else {
00130         _dxl->write_word(_ID, ROM::CWAngleLimit, 0);
00131         _dxl->write_word(_ID, ROM::CCWAngleLimit, 0);
00132     }
00133 }

```

3.1.4.13 void AX12::setMinMax (double min, double max)

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

Parameters

<i>min</i>	Minimum value from servo
<i>max</i>	Maximum value from servo

```

00136 {
00137     if (_ID < 0 or _dxl == NULL) return;
00138
00139     if (min > max) {
00140         double aux = min;
00141         min = max;
00142         max = aux;
00143     }
00144
00145     if (_rads) min *= 180/M_PI;
00146
00147     if (min < 0.0) min = 0;
00148     if (max > 300.0) max = 300;
00149
00150     min = (min/300)*1023;
00151     max = (max/300)*1023;
00152
00153     _dxl->write_word(_ID, ROM::CWAngleLimit, int (min));
00154     _dxl->write_word(_ID, ROM::CCWAngleLimit, int (max));
00155 }

```

3.1.4.14 void AX12::setRadians (bool rads) [inline]

Sets the radians mode.

Parameters

<i>rads</i>	True if radians mode is used
-------------	------------------------------

```
00145 { _rads = rads; }
```

3.1.4.15 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.

```
00158 {
00159     if (_ID < 0 or _dxl == NULL) return;
00160     if (speed > 100.0) speed = 100.0;
00161     if (_mode) {
00162         if (speed < 0.0) speed = 0.0;
00163
00164         int byte = int((speed/100.0) * 1024.0);
00165         if (speed == 100.0) byte = 0;
00166         _dxl->write_byte(_ID, RAM::MovingSpeed, byte);
00167     }
00168     else {
00169         if (speed < -100.0) speed = -100.0;
00170
00171         int byte = int(((speed + 100)/100.0) * 1024);
00172         _dxl->write_byte(_ID, RAM::MovingSpeed, byte);
00173     }
00174 }
00175 }
```

3.1.5 Member Data Documentation

3.1.5.1 dynamixel* AX12::_dxl [private]

Contains the dynamixel communication.

3.1.5.2 int AX12::_ID [private]

Stores the current ID.

3.1.5.3 bool AX12::_mode [private]

True if we use the joint mode.

3.1.5.4 bool AX12::_rads [private]

True if the angle is returned in radians.

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

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

3.2 ServoThread::Dominoe Struct Reference

Struct to handle the dominoe pieces.

Public Member Functions

- bool [operator<](#) (const [Dominoe](#) &d) const
Overloaded operator for comparisons.
- [Dominoe](#) & [operator=](#) (const [Dominoe](#) &d)
Overloaded operator to copy.

- **Dominoe** ()
Default constructor.
- **Dominoe** (double **X**, double **Y**, double **ori**)
Initialization constructor.
- **Dominoe** (QVector2D point, double **ori**)
Initialization constructor with vector.

Public Attributes

- double **X**
X position.
- double **Y**
Y position.
- double **ori**
Orientation from X = 0 in degrees.

3.2.1 Detailed Description

Struct to handle the dominoe pieces.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 ServoThread::Dominoe::Dominoe () [inline]

Default constructor.

```
00061 : X(0), Y(0), ori(0) {}
```

3.2.2.2 ServoThread::Dominoe::Dominoe (double X, double Y, double ori) [inline]

Initialization constructor.

```
00064 : X(X), Y(Y), ori(ori) {}
```

3.2.2.3 ServoThread::Dominoe::Dominoe (QVector2D point, double ori) [inline]

Initialization constructor with vector.

```
00067 : X(point.x()), Y(point.y()), ori(ori) {}
```

3.2.3 Member Function Documentation

3.2.3.1 bool ServoThread::Dominoe::operator< (const Dominoe & d) const [inline]

Overloaded operator for comparisions.

```
00046     {
00047         if (this->X != d.X) return this->X < d.X;
00048         return this->Y < d.Y;
00049     }
```

3.2.3.2 Dominoe& ServoThread::Dominoe::operator= (const Dominoe & d) [inline]

Overloaded operator to copy.

```

00053         {
00054             this->X = d.X;
00055             this->Y = d.Y;
00056             this->ori = d.ori;
00057             return *this;
00058         }

```

3.2.4 Member Data Documentation

3.2.4.1 double ServoThread::Dominoe::ori

Orientation from X = 0 in degrees.

3.2.4.2 double ServoThread::Dominoe::X

X position.

3.2.4.3 double ServoThread::Dominoe::Y

Y position.

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

- [servothread.h](#)

3.3 dxl_hal Class Reference

Dynamixel SDK platform dependent.

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

Public Member Functions

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

Private Attributes

- QSerialPort [_serial](#)
- int [_time](#) = 30
- bool [_timed](#) = false
- bool [_open](#) = false

3.3.1 Detailed Description

Dynamixel SDK platform dependent.

3.3.2 Member Function Documentation

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

```
00041 {
00042     bool res = _serial.setBaudRate(qint32(baudrate));
00043     return int(res);
00044 }
00045 }
```

3.3.2.2 void dxl_hal::clear (void)

```
00032 {
00033     // Clear communication buffer
00034
00035     if (!_serial.isOpen()) return;
00036     _serial.clear();
00037 }
00038 }
```

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 ()

```
00082 {
00083     return (double)QTime::currentTime().msecsSinceStartOfDay();
00084 }
```

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*)

```
00065 {
00066     // Recieving date
00067     // *pPacket: data array pointer
00068     // numPacket: number of data array
00069     // Return: number of data recieved. -1 is error.
00070     _timed = false;
00071     if (_serial.isOpen()) {
00072         int n = _serial.read((char*)pPacket, numPacket);
00073         _timed = _serial.waitForReadyRead(_time);
00074         _timed = not _timed;
00075         return n;
00076     }
00077     else return -1;
00078 }
00079 }
```

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

```

00048 {
00049     // Transmitting date
00050     // *pPacket: data array pointer
00051     // numPacket: number of data array
00052     // Return: number of data transmitted. -1 is error.
00053     _timed = false;
00054     if (_serial.isOpen()) {
00055         int n = _serial.write((char*)pPacket, numPacket);
00056         _timed = _serial.waitForBytesWritten(_time);
00057         _timed = not _timed;
00058         return n;
00059     }
00060     else return -1;
00061 }
00062 }
```

3.3.3 Member Data Documentation

3.3.3.1 bool dxl_hal::_open = false [private]

3.3.3.2 QSerialPort dxl_hal::_serial [private]

3.3.3.3 int dxl_hal::_time = 30 [private]

3.3.3.4 bool dxl_hal::_timed = false [private]

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

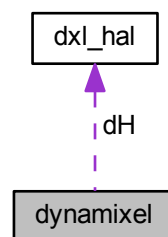
- [dxl/dxl_hal.h](#)
- [dxl/dxl_hal.cpp](#)

3.4 dynamixel Class Reference

Dynamixel 1.0 protocol class.

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

Collaboration diagram for dynamixel:



Public Member Functions

- [dynamixel \(\)](#)
Default constructor.
- [dynamixel \(QString port_num, int baud_rate=1000000\)](#)

- Initialization constructor.*
 - `~dynamixel ()`
- Default destructor.*
 - `bool isOpen ()`
- True if the port is open.*
 - `int initialize (QString port_num, int baud_rate)`
- Initializes the port.*
 - `int change_baudrate (int baud_rate)`
- Changes the current baud rate.*
 - `int terminate (void)`
- Closes the communication.*
 - `int get_comm_result ()`
- Returns the current com status.*
 - `void tx_packet (void)`
- Sends a packet.*
 - `void rx_packet (void)`
- Receives a packet.*
 - `void txrx_packet (void)`
- Sends and receives a packet.*
 - `void set_txpacket_id (int id)`
- Sets the sending packet ID.*
 - `void set_txpacket_instruction (int instruction)`
- Sets the sending packet instruction.*
 - `void set_txpacket_parameter (int index, int value)`
- Sets the sending packet parameter.*
 - `void set_txpacket_length (int length)`
- Sets the sending packet length.*
 - `bool get_rxpacket_error (int error)`
- Returns false if no receive error and true if there's an error.*
 - `int get_rxpacket_error_byte (void)`
- Returns the error byte.*
 - `int get_rxpacket_parameter (int index)`
- Returns the received parameter.*
 - `int get_rxpacket_length ()`
- Returns the received packet length.*
 - `void ping (int id)`
- Ping to the selected id, check com status for the ping result.*
 - `int read_byte (int id, int address)`
- Reads a byte from the selected ID at the selected address.*
 - `void write_byte (int id, int address, int value)`
- Writes a byte to the selected ID at the selected address.*
 - `int read_word (int id, int address)`
- Reads a word to the selected ID at the selected address.*
 - `void write_word (int id, int address, int value)`
- Writes a word to the selected ID at the selected address.*
 - `double get_packet_time ()`
- Returns the packet time.*
 - `void set_packet_timeout (int NumRcvByte)`
- Sets the timeout in number of received bytes.*
 - `void set_packet_timeout_ms (int msec)`
- Sets the timeout in ms.*
 - `bool is_packet_timeout ()`
- Returns true if the packet is timeout.*

Private Attributes

- `dxl_hal dH`
Conains the serial port communication.
- unsigned char `gbInstructionPacket` [MAXNUM_TXPACKET] = {0}
Contains all the instructions.
- unsigned char `gbStatusPacket` [MAXNUM_RXPACKET] = {0}
Contains the status.
- unsigned int `gbRxPacketLength` = 0
Received packet length.
- unsigned int `gbRxGetLength` = 0
Temporal length from the received packet.
- double `gdPacketStartTime` = 0.0
Packet start time.
- double `gdByteTransTime` = 0.0
Byte transmission time.
- double `gdRcvWaitTime` = 0.0
Receive wait time.
- int `gbCommStatus` = COMM_RXSUCCESS
Current communication status.
- int `giBusUsing` = 0
True if the bus if being used.

3.4.1 Detailed Description

Dynamixel 1.0 protocol class.

3.4.2 Constructor & Destructor Documentation

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

Default constructor.

```
00097 {}
```

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

Initialization constructor.

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

3.4.2.3 `dynamixel::~dynamixel ()` [inline]

Default destructor.

```
00103 { dH.close(); }
```

3.4.3 Member Function Documentation

3.4.3.1 `int dynamixel::change_baudrate (int baud_rate)`

Changes the current baud rate.

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

3.4.3.2 `int dynamixel::get_comm_result () [inline]`

Returns the current com status.

```
00118 { return gbCommStatus; }
```

3.4.3.3 `double dynamixel::get_packet_time (void)`

Returns the packet time.

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

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

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

Parameters

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

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

3.4.3.5 `int dynamixel::get_rxpacket_error_byte (void)`

Returns the error byte.

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

3.4.3.6 `int dynamixel::get_rxpacket_length ()`

Returns the received packet length.

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

3.4.3.7 `int dynamixel::get_rxpacket_parameter (int index)`

Returns the received parameter.

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

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

Initializes the port.

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

3.4.3.9 `bool dynamixel::is_packet_timeout (void)`

Returns true if the packet is timeout.

Returns

True if the packet is timeout

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

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

True if the port is open.

```
00106 { return dH.isOpen(); }
```

3.4.3.11 `void dynamixel::ping (int id)`

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

Parameters

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

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

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

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

Parameters

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

```

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

```

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

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

Parameters

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

```

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

```

3.4.3.14 void dynamixel::rx_packet (void)

Receives a packet.

```

00144 {
00145     unsigned char i = 0, j = 0, nRead = 0;
00146     unsigned char checksum = 0;
00147
00148     if( giBusUsing == 0 )
00149         return;
00150
00151     if( gbInstructionPacket[PRT1_PKT_ID] == BROADCAST_ID )
00152     {
00153         gbCommStatus = COMM_RXSUCCESS;
00154         giBusUsing = 0;
00155         return;
00156     }
00157
00158     if( gbCommStatus == COMM_TXSUCCESS )
00159     {
00160         gbRxGetLength = 0;
00161         //gbRxPacketLength = 6; //minimum wait length
00162     }
00163
00164     while(1)
00165     {
00166         nRead = dH.read( &gbStatusPacket[gbRxGetLength],
gbRxPacketLength - gbRxGetLength );
00167         gbRxGetLength += nRead;
00168
00169         if(gbRxGetLength > 4)
00170             gbRxPacketLength = gbStatusPacket[PRT1_PKT_LENGTH] + 4;
00171     }

```

```

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

```

3.4.3.15 void dynamixel::set_packet_timeout (int NumRcvByte)

Sets the timeout in number of received bytes.

Parameters

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

```

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

```

3.4.3.16 `void dynamixel::set_packet_timeout_ms (int msec)`

Sets the timeout in ms.

Parameters

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

```

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

```

3.4.3.17 void dynamixel::set_txpacket_id (int *id*)

Sets the sending packet ID.

```

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

```

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

Sets the sending packet instruction.

```

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

```

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

Sets the sending packet length.

```

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

```

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

Sets the sending packet parameter.

```

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

```

3.4.3.21 int dynamixel::terminate (void)

Closes the communication.

```

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

```

3.4.3.22 void dynamixel::tx_packet (void)

Sends a packet.

```

00082 {
00083     unsigned char pkt_idx = 0;
00084     unsigned char TxNumByte, RealTxNumByte;
00085     unsigned char checksum = 0;
00086
00087     if ( giBusUsing == 1 )
00088     {
00089         gbCommStatus = COMM_TXFAIL;
00090         return;

```



```

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

```

3.4.3.23 void dynamixel::txrx_packet (void)

Sends and receives a packet.

```

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

```

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

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

Parameters

<i>id</i>	Selects the ID to write the byte
-----------	----------------------------------

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

```

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

```

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

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

Parameters

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

```

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

```

3.4.4 Member Data Documentation

3.4.4.1 dxi_hal dynamixel::dH [private]

Contains the serial port communication.

3.4.4.2 int dynamixel::gbCommStatus = COMM_RXSUCCESS [private]

Current communication status.

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

Contains all the instructions.

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

Temporal length from the received packet.

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

Received packet length.

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

Contains the status.

3.4.4.7 double dynamixel::gdByteTransTime = 0.0 [private]

Byte transmission time.

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

Packet start time.

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

Receive wait time.

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

True if the bus is being used.

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

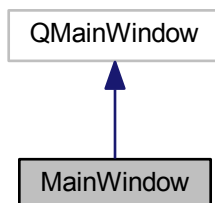
- [dxl/dynamixel.h](#)
- [dxl/dynamixel.cpp](#)

3.5 MainWindow Class Reference

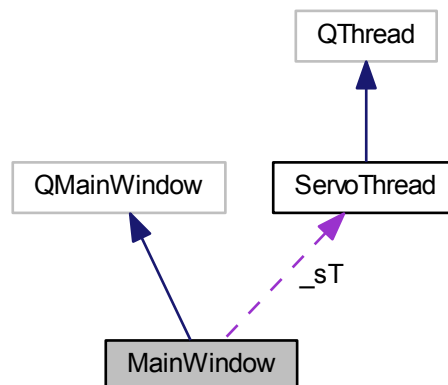
Contains all the windows and other classes.

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

Inheritance diagram for MainWindow:



Collaboration diagram for MainWindow:



Signals

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

Public Member Functions

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

Private Types

- enum [Version](#) { [v_1_0](#) }
- typedef [ServoThread::Mode](#) Mode

Private Slots

- void [joyChanged](#) ()
Handles a joystick update.
- void [modeChanged](#) (Mode m)
Handles the change of a mode in the thread.
- void [on_actionOptions_triggered](#) ()
To select the options.
- void [on_actionImport_triggered](#) ()
Opens the import of Dominoes file.
- void [on_mode_clicked](#) ()
Handles the change of the mode.
- void [on_reset_clicked](#) ()

- Handles a reset.*
- void `on_start_clicked ()`
 - Starts or stops the thread.*
- void `update ()`
 - Updates all data to the servo thread.*

Private Member Functions

- void `keyPressEvent (QKeyEvent *event)`
 - Handles the press of a key.*
- void `keyReleaseEvent (QKeyEvent *event)`
 - Handles the release of a key.*
- void `read ()`
 - Reads the data from the default location.*
- void `read (QString path)`
 - Reads the data from the selected path, overloaded function.*
- void `write ()`
 - Writes the data to the default location.*
- void `write (QString path)`
 - Writes the data to disk overloaded function.*

Private Attributes

- `QVector< QLabel * > _axis`
 - Handles all the axis labels.*
- `QVector< float > _axisV`
 - Contains the axis value;.*
- `QVector< QLabel * > _buts`
 - Handles all the button labels.*
- `QVector< bool > _butsV`
 - Handles all buttons values.*
- `QString _dataP`
 - Contains the path to the data location.*
- `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.*

3.5.1 Detailed Description

Contains all the windows and other classes.

3.5.2 Member Typedef Documentation

3.5.2.1 typedef ServoThread::Mode MainWindow::Mode [private]

3.5.3 Member Enumeration Documentation

3.5.3.1 enum MainWindow::Version [private]

Enumerator

v_1_0

```
00034         {
00035             v_1_0
00036     };
```

3.5.4 Constructor & Destructor Documentation

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

Default constructor.

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

```

00053     _dataP = QStandardPaths::writableLocation(QStandardPaths::AppDataLocation);
00054     QDir dir(_dataP);
00055     if (!dir.exists()) dir.mkpath(_dataP);
00056
00057     read();
00058     _sT.start();
00059 }

```

3.5.4.2 MainWindow::~MainWindow ()

Default destructor.

```

00062 {
00063     delete ui;
00064 }

```

3.5.5 Member Function Documentation

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

Handles a joystick update.

```

00111 {
00112     int sel = _joy.current();
00113
00114     QVector< XJoystick::Info > V(_joy.available());
00115     bool found = false;
00116     int i = 0;
00117     while (i < V.size() and not found) { found = V[i].ID == sel; ++i; }
00118     if (not found) {
00119         if (V.size() > 0) {
00120             _joy.select(V[0].ID);
00121             ui->line->hide();
00122
00123             // Showing axis
00124             ui->joyAxis->show();
00125
00126             // Showing buttons
00127             for (QLabel *l : _buts) l->hide();
00128             ui->joyButs->show();
00129             int n = _joy.buttonCount();
00130             for (int i = 0; i < n; ++i) _buts[i]->show();
00131         }
00132         else {
00133             _joy.select(-1);
00134             ui->joyAxis->hide();
00135             ui->joyButs->hide();
00136             ui->line->hide();
00137         }
00138     }
00139     emit joystickChanged();
00140 }

```

3.5.5.2 void MainWindow::joystickChanged () [signal]

Emmitted when a joystick changes.

3.5.5.3 void MainWindow::keyPressEvent (QKeyEvent * event) [private]

Handles the press of a key.

```

00067 {
00068     if (event->isAutoRepeat()) return;
00069     if (event->key() == Qt::Key_A) _joy.axisPress(0, -100);
00070     else if (event->key() == Qt::Key_D) _joy.axisPress(0, 100);
00071     else if (event->key() == Qt::Key_W) _joy.axisPress(1, 100);
00072     else if (event->key() == Qt::Key_S) _joy.axisPress(1, -100);
00073     else if (event->key() == Qt::Key_Q) _joy.axisPress(2, -100);
00074     else if (event->key() == Qt::Key_E) _joy.axisPress(2, 100);
00075     else if (event->key() == Qt::Key_J) _joy.axisPress(3, -100);
00076     else if (event->key() == Qt::Key_K) _joy.axisPress(3, 100);
00077     else if (event->key() == Qt::Key_R) _sT.reset();
00078     else if (event->key() == Qt::Key_Return) _joy.buttonPress(0, true);
00079
00080     this->update();
00081 }

```

3.5.5.4 void MainWindow::keyReleaseEvent (QKeyEvent * event) [private]

Handles the release of a key.

```
00084 {
00085     if (event->isAutoRepeat()) return;
00086     if (event->key() == Qt::Key_A) _joy.axisRelease(0);
00087     else if (event->key() == Qt::Key_D) _joy.axisRelease(0);
00088     else if (event->key() == Qt::Key_W) _joy.axisRelease(1);
00089     else if (event->key() == Qt::Key_S) _joy.axisRelease(1);
00090     else if (event->key() == Qt::Key_Q) _joy.axisRelease(2);
00091     else if (event->key() == Qt::Key_E) _joy.axisRelease(2);
00092     else if (event->key() == Qt::Key_J) _joy.axisRelease(3);
00093     else if (event->key() == Qt::Key_K) _joy.axisRelease(3);
00094     else if (event->key() == Qt::Key_Return) _joy.buttonRelease(0);
00095     this->update();
00096 }
```

3.5.5.5 void MainWindow::modeChanged (Mode m) [private],[slot]

Handles the change of a mode in the thread.

```
00143 {
00144     qDebug() << int(m);
00145     if (m == Mode::Manual) ui->mode->setText("Manual");
00146     else if (m == Mode::Controlled) ui->mode->setText("Auto");
00147 }
```

3.5.5.6 void MainWindow::on_actionImport_triggered () [private],[slot]

Opens the import of Dominoes file.

```
00166 {
00167     QString caption("Open Dominoes File");
00168     QString dir(QDir::homePath());
00169     QString filter(tr("Dominoes file (*.df)"));
00170
00171     QString file = QFileDialog::getOpenFileName(this, caption, dir, filter);
00172
00173     if (!file.size()) return;
00174
00175     _sT.readPath(file);
00176 }
```

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

To select the options.

```
00151 {
00152     _sT.pause();
00153     ui->start->setText("Start");
00154
00155     OptionsWindow o(_joy, &_sT, this);
00156
00157     connect(this, SIGNAL(joystickChanged()), &o, SLOT(
        joystickChanged()));
00158
00159     if (o.exec()) {
00160         o.storeData();
00161         this->write();
00162     }
00163 }
```

3.5.5.8 void MainWindow::on_mode_clicked () [private],[slot]

Handles the change of the mode.

```
00179 {
00180     if (_sT.isActive()) {
00181         _sT.pause();
00182         ui->start->setText("Start");
00183     }
00184 }
```



```

00183     }
00184     if (ui->mode->text() == "Manual") {
00185         ui->mode->setText("Auto");
00186         _sT.setMode(Mode::Controlled);
00187     }
00188     else if (ui->mode->text() == "Auto") {
00189         ui->mode->setText("Manual");
00190         _sT.setMode(Mode::Manual);
00191     }
00192 }

```

3.5.5.9 void MainWindow::on_reset_clicked () [private],[slot]

Handles a reset.

```

00195 {
00196     _sT.reset();
00197 }

```

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

Starts or stops the thread.

```

00200 {
00201     QString text = ui->start->text();
00202
00203     if (text == "Start") {
00204         _sT.wakeUp();
00205         ui->start->setText("Stop");
00206     }
00207     else if (text == "Stop") {
00208         _sT.pause();
00209         ui->start->setText("Start");
00210     }
00211 }

```

3.5.5.11 void MainWindow::read () [inline],[private]

Reads the data from the default location.

```

00087 { read(_dataP); }

```

3.5.5.12 void MainWindow::read (QString path) [private]

Reads the data from the selected path, overloaded function.

```

00099 {
00100     QDir dir(path);
00101     _sT.read(dir.filePath("servo.opts"));
00102 }

```

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

Updates all data to the servo thread.

```

00214 {
00215     // Joystick values
00216     _joy.update();
00217     for (int i = 0; i < XJoystick::AxisCount; ++i) {
00218         float temp = _joy[i];
00219         _axisV[i] = temp;
00220         _axis[i]->setText(QString::number(temp));
00221     }
00222     for (int i = 0; i < XJoystick::ButtonCount; ++i) {
00223         bool temp = _joy.button(i);
00224         _butsV[i] = temp;
00225         _buts[i]->setEnabled(temp);
00226     }
00227     _sT.setData(_axisV, _butsV);

```

```

00228
00229     QVector<ServoThread::Servo> servo = _sT.getServosInfo();
00230     QVector4D pos = _sT.getCurrentPos();
00231     QString x = QString::number(pos.x());
00232     QString y = QString::number(pos.y());
00233     QString z = QString::number(pos.z());
00234     QString rot = QString::number(pos.w());
00235     ui->pos->setText(x + " " + y + " " + z + " " + rot + "°");
00236
00237     // Updating position sliders
00238     ui->servo0S->setValue(servo[0].pos);
00239     ui->servo1S->setValue(servo[1].pos);
00240     ui->servo2S->setValue(servo[2].pos);
00241     ui->servo3S->setValue(servo[3].pos);
00242
00243     // Updating position labels
00244     ui->servo0->setText(QString::number(servo[0].pos));
00245     ui->servo1->setText(QString::number(servo[1].pos));
00246     ui->servo2->setText(QString::number(servo[2].pos));
00247     ui->servo3->setText(QString::number(servo[3].pos));
00248 }

```

3.5.5.14 void MainWindow::write () [inline], [private]

Writes the data to the default location.

```
00093 { write(_dataP); }
```

3.5.5.15 void MainWindow::write (QString path) [private]

Writes the data to disk overloaded function.

```

00105 {
00106     QDir dir(path);
00107     _sT.write(dir.filePath("servo.opts"));
00108 }

```

3.5.6 Member Data Documentation

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

Handles all the axis labels.

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

Contains the axis value;.

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

Handles all the button labels.

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

Handles all buttons values.

3.5.6.5 QString MainWindow::_dataP [private]

Contains the path to the data location.

3.5.6.6 XJoystick MainWindow::_joy [private]

To handle the joystick.

3.5.6.7 ServoThread MainWindow::_sT [private]

Contains the thread controlling all the servos and external hardware.

3.5.6.8 QTimer MainWindow::_timer [private]

To update the joystick value.

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

Contains the number of additional servos used.

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

Contains the number of minimun servos to work.

3.5.6.11 Ui::MainWindow* MainWindow::ui [private]

Contains the user interface.

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

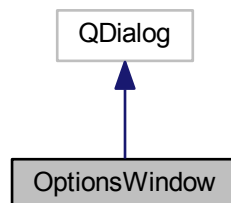
- [mainwindow.h](#)
- [mainwindow.cpp](#)

3.6 OptionsWindow Class Reference

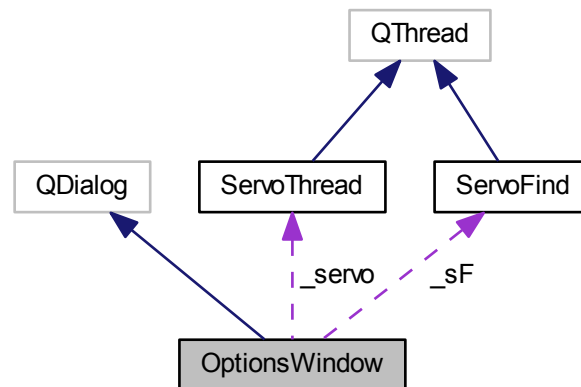
Class used to handle a Window to set the options.

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

Inheritance diagram for OptionsWindow:



Collaboration diagram for OptionsWindow:



Public Slots

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

Public Member Functions

- `OptionsWindow` (XJoystick &J, `ServoThread` *servo, `QWidget` *parent=0)
Default constructor must be initialized with a few values.
- `~OptionsWindow` ()
Destructor.
- void `storeData` ()
Stores all data.

Private Types

- typedef `QDialogButtonBox` `QDB`

Private Slots

- void `events` ()
Handles events that need to be updated continuously.
- void `buttonClicked` (`QAbstractButton` *but)
Handles a button clicked.
- void `on_servoRefresh_clicked` ()
Refreshes all the servos connected to the port.
- void `refreshFinish` ()
Handles the endig of refresh function.

Private Member Functions

- void `keyPressEvent` (QKeyEvent *event)

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.
- QVector< QComboBox * > `_servoC`
Contains all servo QComboBoxes.
- ServoFind `_sF`
Thread to find the servos in a non blocking operation.
- QStatusBar * `status`
Status bar.
- QTimer `_timer`
Waits for a new COM port.
- Ui::OptionsWindow * `ui`
Containsh the GUI.

3.6.1 Detailed Description

Class used to handle a Window to set the options.

3.6.2 Member Typedef Documentation

3.6.2.1 `typedef QDialogButtonBox OptionsWindow::QDB` [private]

3.6.3 Constructor & Destructor Documentation

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

Default constructor must be intialized with a few values.

Parameters

<code>J</code>	Refernce to the Joystick handler
<code>servo</code>	Pointer to the ServoThread
<code>aX</code>	Axis for the X value
<code>aY</code>	Axis for the Y value
<code>aZ</code>	Axis for the Z value

```

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
00015     connect(ui->buttonBox, SIGNAL(clicked(QAbstractButton*)),
00016             this, SLOT(buttonClicked(QAbstractButton*)));
00017
00018     connect(&_sF, SIGNAL(completion(int)),

```

```

00019         ui->progressBar, SLOT(setValue(int)));
00020
00021     connect(&_ampSF, SIGNAL(finished()), this, SLOT(refreshFinish()));
00022
00023     connect(&_ampTimer, SIGNAL(timeout()), this, SLOT(events()));
00024
00025
00026     // Configuring event function
00027     _ampTimer.setInterval(500);
00028     _ampTimer.setSingleShot(false);
00029     _ampTimer.start();
00030
00031     status = new QStatusBar(this);
00032     status->setContentsMargins(0, 0, 0, 0);
00033     this->layout()->addWidget(status);
00034
00035     QVector<QString> A(_ampJoy.getAllAxis());
00036
00037     // Updating joystick data
00038     joystickChanged();
00039
00040     // Adding servos
00041     _ampServoC.push_back(ui->servo0);
00042     _ampServoC.push_back(ui->servo1);
00043     _ampServoC.push_back(ui->servo2);
00044     _ampServoC.push_back(ui->servo3);
00045
00046     for(QComboBox *s : _ampServoC) s->addItem("None", -1);
00047
00048     QVector<ServoThread::Servo> S(_ampServo->getServosInfo());
00049     Q_ASSERT(S.size() == _ampServo->getServosNum());
00050
00051     for (int i = 0; i < S.size(); ++i) {
00052         int ID = S[i].ID;
00053
00054         if (ID >= 0) {
00055             _ampServoC[i]->addItem(QString::number(ID), ID);
00056             _ampServoC[i]->setCurrentIndex(1);
00057         }
00058     }
00059
00060     // Obtaining Servo Port information
00061     QString port;
00062     int baud;
00063     _ampServo->getServoPortInfo(port, baud);
00064     ui->speed->setValue(_ampServo->getSpeed());
00065     ui->baudRS->setValue(baud);
00066     ui->portS->addItem("", port);
00067 }

```

3.6.3.2 OptionsWindow::~OptionsWindow ()

Destructor.

```

00070 {
00071     delete ui;
00072     if (_ampSF.isRunning()) _ampSF.exit();
00073 }

```

3.6.4 Member Function Documentation

3.6.4.1 void OptionsWindow::buttonClicked (QAbstractButton * but) [private],[slot]

Handles a button clicked.

```

00149 {
00150     QDB::ButtonRole role = ui->buttonBox->buttonRole(but);
00151     switch(role) {
00152     case QDB::ApplyRole:
00153         this->storeData();
00154         break;
00155
00156     default:
00157         break;
00158     }
00159 }

```

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

Handles events that need to be updated continuously.

```

00114 {
00115     auto ports = QSerialPortInfo::availablePorts();
00116     ui->portN->setText(QString::number(ports.size()));
00117
00118     if (ports.size() != _portSize) {
00119         _portSize = ports.size();
00120
00121         QString portC(ui->portC->currentData().toString());
00122         QString portS(ui->portS->currentData().toString());
00123
00124         int selC = 0, selS = 0;
00125
00126         ui->portC->clear();
00127         ui->portS->clear();
00128
00129         ui->portC->addItem("None", "");
00130         ui->portS->addItem("None", "");
00131
00132         for (int i = 0; i < ports.size(); ++i) {
00133             QString text(ports[i].portName());
00134             text += ": " + ports[i].description();
00135             ui->portC->addItem(text, ports[i].portName());
00136             ui->portS->addItem(text, ports[i].portName());
00137             if (ports[i].portName() == portC) selC = i + 1;
00138             if (ports[i].portName() == portS) selS = i + 1;
00139         }
00140
00141         if (selS == 0 && ports.size() > 0) selS = 1;
00142
00143         ui->portC->setCurrentIndex(selC);
00144         ui->portS->setCurrentIndex(selS);
00145     }
00146 }

```

3.6.4.3 void OptionsWindow::joystickChanged () [slot]

To handle the change of a joystick.

```

00094 {
00095     // Clear all the items and write the new items
00096     ui->joySel->clear();
00097     ui->joySel->addItem("None", -1);
00098
00099     // Adding items and searching the current
00100     int pos = 0;
00101     QVector<XJoystick::Info> V(_joy.available());
00102     for (int i = 0; i < V.size(); ++i) {
00103         QString text(V[i].name);
00104         text += ": " + QString::number(V[i].ID);
00105         if (V[i].ID == _joy.current()) pos = i;
00106         ui->joySel->addItem(text, V[i].ID);
00107     }
00108     ui->joySel->setCurrentIndex(pos);
00109
00110     ui->joyN->setText(QString::number(V.size()));
00111 }

```

3.6.4.4 void OptionsWindow::keyPressEvent (QKeyEvent * event) [private]

```

00179 {
00180     if (event->key() == Qt::Key_Enter || event->key() == Qt::Key_Return) return;
00181     QDialog::keyPressEvent(event);
00182 }

```

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

Refreshes all the servos connected to the port.

```

00162 {
00163     if (_sF.isRunning()) return;
00164     QString port;
00165     int baud;

```

```

00166     _servo->getServoPortInfo(port, baud);
00167     int min = ui->min->value();
00168     int max = ui->max->value();
00169     _sF.setData(_servoC, port, baud, min, max);
00170     _sF.start();
00171 }

```

3.6.4.6 void OptionsWindow::refreshFinish () [private],[slot]

Handles the endig of refresh function.

```

00174 {
00175     ui->progressBar->setValue(0);
00176 }

```

3.6.4.7 void OptionsWindow::storeData ()

Stores all data.

```

00076 {
00077     status->showMessage("Data Stored", 2000);
00078
00079     // Storing joystick data
00080     _joy.select(ui->joySel->currentData().toInt());
00081
00082     QString portS(ui->portS->currentData().toString());
00083     int baudS(ui->baudRS->value());
00084     _servo->setServoPortInfo(portS, baudS);
00085
00086     QVector<int> sID;
00087     for (QComboBox *s : _servoC) sID.push_back(s->currentData().toInt());
00088
00089     _servo->setSID(sID);
00090     _servo->setSpeed(ui->speed->value());
00091 }

```

3.6.5 Member Data Documentation

3.6.5.1 XJoystick& OptionsWindow::_joy [private]

Contains the Joystick to handle options.

3.6.5.2 int OptionsWindow::_portSize [private]

Contains the size of the ports.

3.6.5.3 ServoThread* OptionsWindow::_servo [private]

Pointer to the servo thread class.

3.6.5.4 QVector< QComboBox *> OptionsWindow::_servoC [private]

Contains all servo QComboBoxes.

3.6.5.5 ServoFind OptionsWindow::_sF [private]

Thread to find the servos in a non blocking operation.

3.6.5.6 QTimer OptionsWindow::_timer [private]

Waits for a new COM port.

3.6.5.7 QStatusBar* OptionsWindow::status [private]

Status bar.

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

Containsh the GUI.

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

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

3.7 ServoThread::Servo Struct Reference

Struct for the [AX12](#) servos.

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

Public Member Functions

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

Public Attributes

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

3.7.1 Detailed Description

Struct for the [AX12](#) servos.

3.7.2 Constructor & Destructor Documentation

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

Default constructor.

```
00080 : ID(ID), pos(pos) {}
```

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

Copy constructor.

```
00083 : ID(s.ID), pos(s.pos) {}
```

3.7.3 Member Function Documentation

3.7.3.1 void ServoThread::Servo::operator=(const Servo & s) [inline]

Operator overloading.

```
00087     {
00088         this->ID = s.ID;
00089         this->pos = s.pos;
00090     }
```

3.7.4 Member Data Documentation

3.7.4.1 int ServoThread::Servo::ID

Contains the servo ID.

3.7.4.2 double ServoThread::Servo::pos

Contains the servo position.

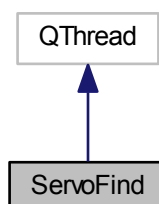
The documentation for this struct was generated from the following file:

- [servothread.h](#)

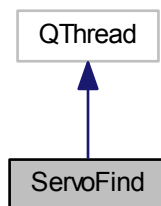
3.8 ServoFind Class Reference

```
#include <servofind.h>
```

Inheritance diagram for ServoFind:



Collaboration diagram for ServoFind:



Signals

- void **completion** (int)
Shows the completion of the process.

Public Member Functions

- **ServoFind** ()
Default constructor.
- **~ServoFind** ()
Default destructor.
- void **run** ()
Main function.
- void **setData** (QVector< QComboBox * > servo, QString port, int baud, int min=0, int max=MAX_ID)
To set all data.

Private Types

- typedef QComboBox **QCB**

Private Attributes

- int **_baud**
Contains the baud rate.
- int **_min** = 0
Minimum value to find.
- int **_max** = MAX_ID
Maximum value to find.
- QString **_port**
Contains the current port.
- QVector< QComboBox * > **_servo**
Contains the pointer to the servos QComboBoxes.

3.8.1 Member Typedef Documentation

3.8.1.1 typedef QComboBox ServoFind::QCB [private]

3.8.2 Constructor & Destructor Documentation

3.8.2.1 ServoFind::ServoFind ()

Default constructor.

```
00004 {
00005
00006 }
```

3.8.2.2 ServoFind::~~ServoFind ()

Default destructor.

```
00009 {
00010
00011 }
```

3.8.3 Member Function Documentation

3.8.3.1 void ServoFind::completion (int) [signal]

Shows the completion of the process.

3.8.3.2 void ServoFind::run ()

Main function.

```
00014 {
00015     QVector<int> data(_servo.size());
00016
00017     for (int i = 0; i < data.size(); ++i)
00018         data[i] = _servo[i]->currentData().toInt();
00019
00020     for (QCB *s : _servo) {
00021         s->clear();
00022         s->addItem("None", -1);
00023     }
00024
00025     int index = 1;
00026     QVector<int> pos(_servo.size(), 0);
00027
00028     dynamixel dxl(_port, _baud);
00029
00030     for (int i = _min; i < _max; ++i) {
00031         dxl.ping(i);
00032         emit completion(((i - _min)/double(_max - _min))*100.0);
00033         if (dxl.get_comm_result() == COMM_RXSUCCESS) {
00034
00035             for (int j = 0; j < _servo.size(); ++j) {
00036                 if (data[j] == i) pos[j] = index;
00037                 _servo[j]->addItem(QString::number(i), i);
00038             }
00039
00040             ++index;
00041         }
00042     }
00043
00044     for (int i = 0; i < _servo.size(); ++i) _servo[i]->setCurrentIndex(pos[i]);
00045 }
```

3.8.3.3 void ServoFind::setData (QVector< QComboBox * > servo, QString port, int baud, int min = 0, int max = MAX_ID)

To set all data.

```

00049 {
00050     if (this->isRunning()) return;
00051     _servo = servo;
00052     _port = port;
00053     _baud = baud;
00054
00055     if (min > max) {
00056         int aux = min;
00057         min = max;
00058         max = aux;
00059     }
00060
00061     if (min < 0) min = 0;
00062     if (max > MAX_ID) max = MAX_ID;
00063
00064
00065     _min = min;
00066     _max = max;
00067 }

```

3.8.4 Member Data Documentation

3.8.4.1 int ServoFind::_baud [private]

Contains the baud rate.

3.8.4.2 int ServoFind::_max = MAX_ID [private]

Maximum value to find.

3.8.4.3 int ServoFind::_min = 0 [private]

Minimum value to find.

3.8.4.4 QString ServoFind::_port [private]

Contains the current port.

3.8.4.5 QVector<QComboBox*> ServoFind::_servo [private]

Contains the pointer to the servos QComboBoxes.

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

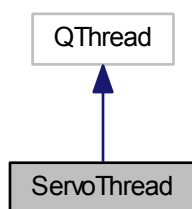
- [servofind.h](#)
- [servofind.cpp](#)

3.9 ServoThread Class Reference

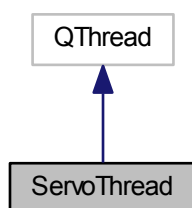
The [ServoThread](#)'s class handles the communication between the delta robot servos and the PC.

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

Inheritance diagram for ServoThread:



Collaboration diagram for ServoThread:



Classes

- struct [Dominoe](#)
Struct to handle the dominoe pieces.
- struct [Servo](#)
Struct for the [AX12](#) servos.

Public Types

- enum [Mode](#) { [Controlled](#), [Manual](#), [Reset](#) }
Contains the working mode.

Signals

- void [modeChanged](#) ([Mode](#))
To show the change of a mode.
- void [statusBar](#) (QString, int)
Emmitted when the status bar must be changed.

Public Member Functions

- [ServoThread \(\)](#)
Default constructor.
- [~ServoThread \(\)](#)
Default destructor.
- void [end \(\)](#)
Ends the execution.
- QVector4D [getCurrentPos \(\)](#)
Returns the current position.
- 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.
- int [getServosNum \(\)](#)
Returns the number of servos to handle.
- int [getSpeed \(\)](#)
Returns the current speed.
- bool [isActive \(\)](#)
Returns true if the servos are active.
- QMutex * [mutex \(\)](#)
Returns the mutex used in the thread.
- void [pause \(\)](#)
Pauses the execution.
- void [read](#) (QString file)
Reads and loads the data from the selected file.
- void [readPath](#) (QString file)
Reads the path where to put the selected pieces.
- void [reset \(\)](#)
Resets to default positions (used when the mode changes or when some data has changed).
- void [setMode](#) ([Mode](#) m)
Sets the current working mode.
- void [setData](#) (QVector< float > &aV, QVector< bool > &butts)
Adds the loaded data.
- void [setServoBaud](#) (unsigned int baud)
Sets the servos port baud rate.
- void [setServoPort](#) (QString &port)
Sets the servos port.
- void [setServoPortInfo](#) (QString &port, unsigned int baud)
Sets the servos port info, data and selected port.
- void [setSID](#) (QVector< int > &V)
Sets the servos ID.
- void [setSpeed](#) (unsigned char speed)
Sets the servos speed.
- void [wakeUp \(\)](#)
Continues program's execution.
- void [write](#) (QString file)
Writes data to the selected directory.

Private Types

- enum `Version` { `v_1_0` }
Enum containing all the save file versions.
- enum `Status` {
 `begin`, `take`, `waiting`, `rotate`,
 `going`, `ending` }
Contains the available status for the Controlled mode.

Private Member Functions

- bool `isPosAvailable` (const QVector4D &newPos)
Returns true if the position is available.
- bool `isReady` (const QVector< double > &S, const QVector4D &pos, double err)
- void `run` ()
Used to create another thread.
- void `setAngles` (const QVector4D &pos, QVector< double > &D)
Used to calculate the servos angles.
- void `setGoalPosition` (const QVector< int > &ID, const QVector< double > &pos, `dynamixel` &dxl)
- double `singleAngle` (double x0, double y0, double z0)
Calculates the angle of one servo in the selected position.

Private Attributes

- const double `cos60` = 0.5
Contains the cosinus of 60.
- const double `sin60` = sqrt(3)/2
Contains the sinus of 60.
- const double `a` = 11.6
The arm length.
- const double `b` = 22.648
The forearm length.
- const double `L1` = 5.499
The base center length.
- const double `L2` = 6.000
The clamp support center lenght.
- const double `maxErr` = 3.0
Max available error.
- const double `minAngle` = 60.0
Minimum servo angle.
- const double `maxAngle` = 240.0
Maximum servo angle.
- const double `workRadSq` = 144.0
Working radius squared.
- const uchar `ccwCS` = 2
The Counter Clock Wise Compliance Slope.
- const uchar `cwCS` = 2
The Clock Wise Compliance Slope.
- const double `workHeigh` = 23.3
Working heigh.
- const double `idleHeigh` = 22.0

- Idle heigh.*
- const double `descHeigh` [3] = { 23.0, 22.7, 22.3 }
- Descent height.*
- const QVector4D `posStart` = QVector4D(11.5, 0.0f, idleHeigh, 150)
- Starting position for the controlled mode.*
- const QVector4D `posIdle` = QVector4D(0.0f, 0.0f, idleHeigh, 150)
- Idle position.*
- QVector4D `_axis`
- Contains the axis value.*
- QVector< bool > `_buts`
- Contains the buttons value.*
- int `_cBaud`
- Contains the baud rate used to communicate with the clamp.*
- QWaitCondition `_cond`
- To start and pause the thread.*
- QString `_cPort`
- Contains the selected com port used to comunitate with the clamp.*
- bool `_dChanged`
- True if the data changes.*
- QVector< QVector< `Dominoe` > > `_dominoe`
- Contains all the dominoes information.*
- bool `_end`
- True when we must end executino.*
- bool `_enter`
- True if the enter key is pressed.*
- `Mode _mod`
- Contains the working mode.*
- QMutex `_mutex`
- To prevent memory errors between threads.*
- bool `_pause`
- Pauses the execution of the thread.*
- QVector4D `_pos`
- Contains the current position to show to the window.*
- 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.*
- unsigned int `_sSpeed`
- Speed of the robot.*
- `Status _status`
- Current status.*

Static Private Attributes

- static const int `_sNum` = 4
- Number of servos to manage.*

3.9.1 Detailed Description

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

3.9.2 Member Enumeration Documentation

3.9.2.1 enum ServoThread::Mode

Contains the working mode.

Enumerator

Controlled

Manual

Reset

```
00095     {
00096         Controlled,
00097         Manual,
00098         Reset
00099     };
```

3.9.2.2 enum ServoThread::Status [private]

Contains the available status for the Controlled mode.

Enumerator

begin

take

waiting

rotate

going

ending

```
00028     {
00029         begin,
00030         take,
00031         waiting,
00032         rotate,
00033         going,
00034         ending
00035     };
```

3.9.2.3 enum ServoThread::Version [private]

Enum containing all the save file versions.

Enumerator

v_1_0

```
00023     {
00024         v_1_0
00025     };
```

3.9.3 Constructor & Destructor Documentation

3.9.3.1 ServoThread::ServoThread ()

Default constructor.

```

00004         :
00005         _axis(0, 0, 0, 0),
00006         _buts(XJoystick::ButtonCount),
00007         _cBaud(9600),
00008         _cPort("COM3"),
00009         _dChanged(true),
00010         _end(false),
00011         _mod(Mode::Manual),
00012         _pause(true),
00013         _sBaud(1000000),
00014         _servos(_sNum),
00015         _sPort("COM9"),
00016         _sPortChanged(false),
00017         _sSpeed(100),
00018         _status(Status::begin)
00019 {
00020     for (Servo &s : _servos) s.ID = -1;
00021 }

```

3.9.3.2 ServoThread::~ServoThread ()

Default destructor.

```

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

```

3.9.4 Member Function Documentation

3.9.4.1 void ServoThread::end () [inline]

Ends the execution.

```

00109 {
00110     _mutex.lock();
00111     _end = true;
00112     _cond.wakeOne();
00113     _mutex.unlock();
00114
00115     wait();
00116 }

```

3.9.4.2 QVector4D ServoThread::getCurrentPos () [inline]

Returns the current position.

```

00120 {
00121     QMutexLocker m(&_mutex);
00122     return _pos;
00123 }

```

3.9.4.3 int ServoThread::getServoBaud () [inline]

Returns the current servo Baud rate.

```

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

```

3.9.4.4 QString ServoThread::getServoPort () [inline]

Returns the current servo Port.

```

00134 {
00135     QMutexLocker mL(&_mutex);
00136     return _sPort;
00137 }

```

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

Returns both servo Port and baud Rate.

```
00141    {
00142        _mutex.lock();
00143        baud = _sBaud;
00144        port = _sPort;
00145        _mutex.unlock();
00146    }
```

3.9.4.6 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
---	-----------------------------------

```
00151    {
00152        _mutex.lock();
00153        V = _servos;
00154        _mutex.unlock();
00155    }
```

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

Overloaded function to get the servo info.

```
00159    {
00160        QMutexLocker mL(&_mutex);
00161        return _servos;
00162    }
```

3.9.4.8 int ServoThread::getServosNum () [inline]

Returns the number of servos to handle.

```
00165 { return _sNum; }
```

3.9.4.9 int ServoThread::getSpeed () [inline]

Returns the current speed.

```
00169    {
00170        QMutexLocker m(&_mutex);
00171        return _sSpeed;
00172    }
```

3.9.4.10 bool ServoThread::isActive () [inline]

Returns true if the servos are active.

```
00176    {
00177        QMutexLocker m(&_mutex);
00178        return not _pause;
00179    }
```

3.9.4.11 bool ServoThread::isPosAvailable (const QVector4D & *newPos*) [private]

Returns true if the position is available.

```

00151 {
00152     if (newPos.toVector2D().lengthSquared() > workRadSq) return false;
00153
00154     QVector<double> D(4);
00155     this->setAngles(newPos, D);
00156
00157     for (int i = 0; i < 3; ++i) {
00158         if (qIsNaN(D[i])) return false;
00159         if (D[i] > maxAngle or D[i] < minAngle) return false;
00160     }
00161
00162     return true;
00163 }

```

3.9.4.12 bool ServoThread::isReady (const QVector< double > & S, const QVector4D & pos, double err) [private]

```

00167 {
00168     QVector<double> D(4);
00169     this->setAngles(pos, D);
00170
00171     for (int i = 0; i < 3; ++i) if (abs(S[i] - D[i]) > err) return false;
00172     return true;
00173 }

```

3.9.4.13 void ServoThread::modeChanged (Mode) [signal]

To show the change of a mode.

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

Returns the mutex used in the thread.

```

00182 { return &_mutex; }

```

3.9.4.15 void ServoThread::pause () [inline]

Pauses the execution.

```

00186 {
00187     _mutex.lock();
00188     _pause = true;
00189     _mutex.unlock();
00190 }

```

3.9.4.16 void ServoThread::read (QString file)

Reads and loads the data from the selected file.

Parameters

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

```

00033 {
00034     // Opening file for reading
00035     QFile f(file);
00036     if (!f.open(QIODevice::ReadOnly)) {
00037         emit statusBar("Cannot read stored data", 2000);
00038         return;
00039     }
00040     QDataStream df(&f);
00041
00042     QMutexLocker mL(&_mutex);
00043
00044     int version;
00045     df >> version;
00046     if (version != Version::v_1_0) {
00047         emit statusBar("Error opening file", 2000);
00048         return;
00049     }
00050
00051     df >> _cBaud >> _cPort >> _sBaud >> _sPort >>

```

```

    _sSpeed;
00052     unsigned int en;
00053     df >> en;
00054     _mod = static_cast<Mode>(en);
00055
00056     int size;
00057     df >> size;
00058     _servos.resize(size);
00059     for (Servo &s : _servos) df >> s.ID;
00060     _dChanged = true;
00061
00062 }

```

3.9.4.17 void ServoThread::readPath (QString file)

Reads the path where to put the selected pieces.

Parameters

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

```

00065 {
00066     // Opening file for reading
00067     QFile f(file);
00068     if (!f.open(QIODevice::ReadOnly)) {
00069         emit statusBar("Error opening file", 2000);
00070         return;
00071     }
00072
00073     QTextStream pF(&f);
00074
00075     int size;
00076     pF >> size;
00077     QVector<Dominoe> temp(size);
00078     for (Dominoe &d : temp) pF >> d.X >> d.Y >> d.ori;
00079     std::sort(temp.begin(), temp.end());
00080
00081     _mutex.lock();
00082     double sep = 0.8; // 2cm of separation
00083     QVector2D ori(posStart.toVector2D());
00084
00085     _dominoe.clear();
00086     for (int i = 0; i < temp.size(); ++i) {
00087         QVector2D aux(temp[i].X, temp[i].Y);
00088
00089         // Checking if its a valid position
00090         QVector4D aux2(aux);
00091         aux2[2] = workHeigh;
00092         if (not this->isPosAvailable(aux2)) continue;
00093
00094         double angle = temp[i].ori + 60.0;
00095         if (angle >= 180.0) angle -= 180.0;
00096         else if (angle >= 360.0) angle -= 360.0;
00097
00098         // Calculating direction vector
00099         aux -= ori;
00100         int steps = aux.length()/sep;
00101
00102         QVector<Dominoe> V(steps + 1);
00103
00104         // Initial point
00105         V[0] = Dominoe(ori, angle);
00106
00107         // Adding intermediate positions
00108         for (int j = 1; j <= steps; ++j){
00109             Dominoe dAux(j*aux/double(steps) + ori, angle);
00110             V[j] = dAux;
00111         }
00112         _dominoe.push_back(V);
00113     }
00114     _dChanged = true;
00115     _mutex.unlock();
00116
00117     f.close();
00118
00119     emit statusBar("File loaded succesfully", 1000);
00120 }

```

3.9.4.18 void ServoThread::reset () [inline]

Resets to default positions (used when the mode changes or when some data has changed).

Precondition

The thread is sleeping

```
00204     {
00205         _mutex.lock();
00206         _mod = Mode::Reset;
00207         _mutex.unlock();
00208     }
```

3.9.4.19 void ServoThread::run() [private]

Used to create another thread.

```
00176 {
00177     // First initializations
00178     _mutex.lock();
00179     int sBaud = _sBaud;
00180     QString sPort = _sPort;
00181     _mutex.unlock();
00182
00183     // Serial port interface
00184     dynamixel dxl(sPort, sBaud);
00185
00186     // Contains the servos communication
00187     QVector<AX12> A(4);
00188
00189     // Contains the servos ID
00190     QVector< int > ID(_sNum);
00191
00192     // Contains the current servo data
00193     QVector< double > S(_sNum);
00194
00195     // Contains the servos angles
00196     QVector<double> D(4);
00197     D[3] = 150.0;
00198
00199     // First initialization
00200     _mutex.lock();
00201     for (int i = 0; i < A.size(); ++i) {
00202         A[i] = AX12(&dxl);
00203         A[i].setID(_servos[i].ID);
00204         ID[i] = _servos[i].ID;
00205         A[i].setSpeed(_sSpeed);
00206         A[i].setComplianceSlope(ccwCS, cwCS);
00207     }
00208     _mutex.unlock();
00209
00210     QVector4D pos(posIdle);
00211     QVector4D axis(0, 0, 0, 0);
00212     QVector< bool > buts;
00213
00214     // Contains the domino number to put
00215     int dom = 0;
00216     int pas = 0;
00217     double speed = 100.0;
00218     QVector< QVector< Dominoe > > Dom;
00219
00220     // Main while
00221     while (not _end) {
00222
00223         // Pause
00224         _mutex.lock();
00225         if (not _end and _pause) {
00226             dxl.terminate();
00227
00228             // Thread pause
00229             _cond.wait(&_mutex);
00230
00231             if (_end) exit(0);
00232             dxl.initialize(sPort, sBaud);
00233         }
00234         _mutex.unlock();
00235
00236         // Get current servo position
00237         for (int i = 0; i < 3; ++i) S[i] = A[i].getCurrentPos();
00238         if (_mod == Mode::Manual) S[3] = A[3].getCurrentPos();
00239
00240
00241         /***** MUTEX *****/
00242         // Handling changes of data
00243         _mutex.lock();
00244         if (_dChanged) {
```

```

00245         if (sPort != _sPort or sBaud != _sBaud) {
00246             sPort = _sPort;
00247             sBaud = _sBaud;
00248             dxl.terminate();
00249             dxl.initialize(sPort, sBaud);
00250         }
00251         for (int i = 0; i < S.size(); ++i) {
00252             A[i].setID(_servos[i].ID);
00253             ID[i] = _servos[i].ID;
00254             A[i].setSpeed(_sSpeed);
00255             A[i].setComplianceSlope(ccwCS, cwCS);
00256         }
00257
00258         speed = _sSpeed;
00259         Dom = _dominoe;
00260         dom = 0;
00261         pas = 0;
00262         pos = posIdle;
00263         this->setAngles(pos, D);
00264         this->setGoalPosition(ID, D, dxl);
00265         _dChanged = false;
00266     }
00267
00268     // Joystick and buttons update, must use mutex
00269     for (int i = 0; i < _sNum; ++i) _servos[i].pos = S[i];
00270     axis = _axis;
00271     buts = _buts;
00272     for (bool &b : _buts) b = 0;
00273     _pos = pos;
00274     _mutex.unlock();
00275
00276     /***** MODE *****/
00277     // Main function with data updated
00278
00279     if (_mod == Mode::Manual) {
00280         QVector4D posAux = pos + 0.5*axis;
00281
00282         bool ok = this->isPosAvailable(posAux);
00283         ok &= this->isReady(S, pos, maxErr + 5.0);
00284         if (ok) pos = posAux;
00285     }
00286     else if (_mod == Mode::Controlled) {
00287         switch(_status) {
00288             case Status::begin:
00289                 pos = posStart;
00290                 if (this->isReady(S, pos, maxErr)) {
00291                     _status = Status::take;
00292                     QThread::msleep(500);
00293                 }
00294                 break;
00295             case Status::take:
00296                 pos[2] = workHeigh;
00297                 if (this->isReady(S, pos, maxErr)) {
00298                     emit statusBar("Esperant peça", -1);
00299                     _status = Status::waiting;
00300                 }
00301                 break;
00302             case Status::waiting:
00303                 if (buts[0]) {
00304                     pas = 0;
00305                     _status = Status::rotate;
00306                     emit statusBar("Girant!", -1);
00307                     for (AX12 &a : A) a.setSpeed(speed/3.5);
00308                 }
00309                 else break;
00310             case Status::rotate:
00311                 {
00312                     S[3] = A[3].getCurrentPos();
00313                     pos[3] = Dom[dom][0].ori;
00314                     double aux = abs(S[3] - Dom[dom][0].ori);
00315                     if (aux < maxErr) {
00316                         _status = Status::going;
00317                         QThread::msleep(1000);
00318                         emit statusBar("Posicionant", -1);
00319                     }
00320                 }
00321                 break;
00322             case Status::going:
00323                 {
00324                     Dominoe &domi = Dom[dom][pas];
00325                     pos = QVector4D (domi.X, domi.Y, workHeigh, domi.ori);
00326                 }
00327         }
00328     }
00329
00330     case Status::going:
00331     {
00332         Dominoe &domi = Dom[dom][pas];
00333         pos = QVector4D (domi.X, domi.Y, workHeigh, domi.ori);

```



```

00334         double err;
00335         pas == Dom[dom].size() - 1 ? err = maxErr : err = 2*maxErr;
00336
00337         if (this->isReady(S, pos, err)) {
00338             ++pas;
00339             if (pas == Dom[dom].size()) {
00340                 pas = 0;
00341                 _status = Status::ending;
00342                 QThread::msleep(1000);
00343                 emit statusBar("Col·locada", 1500);
00344
00345                 for (AX12 &a : A) a.setSpeed(speed);
00346             }
00347         }
00348     }
00349     break;
00350
00351     case Status::ending:
00352         pos[2] = descHeigh[pas];
00353
00354         if (this->isReady(S, pos, maxErr)) {
00355             ++pas;
00356             if (pas == 3) {
00357                 _status = Status::begin;
00358                 if (dom == Dom.size() - 1) {
00359                     dom = 0;
00360                     pas = 0;
00361                     _mod = Mode::Reset;
00362                 }
00363                 else ++dom;
00364             }
00365             break;
00366         }
00367
00368     default:
00369         _status = Status::begin;
00370
00371     }
00372 }
00373 else if (_mod == Mode::Reset) {
00374     _mod = Mode::Manual;
00375     pos = posIdle;
00376     dom = 0;
00377 }
00378
00379 this->setAngles(pos, D);
00380 this->setGoalPosition(ID, D, dx1);
00381 }
00382 dx1.terminate();
00383 exit(0);
00384 }

```

3.9.4.20 void ServoThread::setAngles (const QVector4D & pos, QVector< double > & D) [private]

Used to calculate the servos angles.

```

00387 {
00388     double x1 = pos.x() + L2 - L1;
00389     double y1 = -pos.z();
00390     double z1 = pos.y();
00391     D[0] = singleAngle(x1,y1,z1);
00392
00393     double x2 = pos.y()*sin60 - pos.x()*cos60 + L2 - L1;
00394     double y2 = -pos.z();
00395     double z2 = -pos.y()*cos60 - pos.x()*sin60;
00396     D[1] = singleAngle(x2,y2,z2);
00397
00398     double x3 = -pos.y()*sin60 - pos.x()*cos60 + L2 - L1;
00399     double y3 = -pos.z();
00400     double z3 = -pos.y()*cos60 + pos.x()*sin60;
00401     D[2] = singleAngle(x3,y3,z3);
00402
00403     for (double &d : D) d = 150.0 - d*180/M_PI;
00404     D[3] = pos.w();
00405 }

```

3.9.4.21 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

```

00123 {
00124     _mutex.lock();
00125     // Copying the joystick values
00126     _axis = QVector4D(aV[0], aV[1], aV[2], aV[3]);
00127     _axis.normalize();
00128     _axis[3] *= 5;
00129     for (int i = 0; i < butS.size(); ++i) _butS[i] |= butS[i];
00130     _mutex.unlock();
00131 }

```

3.9.4.22 void ServoThread::setGoalPosition (const QVector< int > &ID, const QVector< double > &pos, dynamixel &dxl) [private]

```

00409 {
00410     dxl.set_txpacket_id(BROADCAST_ID);
00411     dxl.set_txpacket_instruction(INST_SYNC_WRITE);
00412     dxl.set_txpacket_parameter(0, AX12::RAM::GoalPosition);
00413     dxl.set_txpacket_parameter(1, 2);
00414
00415     Q_ASSERT(ID.size() == pos.size());
00416     for (int i = 0; i < ID.size(); ++i) {
00417         unsigned int data = (pos[i]/300.0)*1023.0;
00418
00419         dxl.set_txpacket_parameter(2 + 3*i, ID[i]);
00420         dxl.set_txpacket_parameter(2 + 3*i + 1, LOBYTE(data));
00421         dxl.set_txpacket_parameter(2 + 3*i + 2, HIBYTE(data));
00422     }
00423     dxl.set_txpacket_length(4 + 3*ID.size());
00424     dxl.txrx_packet();
00425 }

```

3.9.4.23 void ServoThread::setMode (Mode m) [inline]

Sets the current working mode.

Precondition

The thread must be on pause

Parameters

<i>m</i>	Contains the desired working mode
----------	-----------------------------------

```

00214 {
00215     QMutexLocker mut(&_mutex);
00216     if (!_pause) return;
00217     _mod = m;
00218     _dChanged = true;
00219 }

```

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

Sets the servos port baud rate.

Parameters

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

```

00229 {
00230     _mutex.lock();
00231     _sBaud = baud;
00232     _dChanged = true;
00233     _mutex.unlock();
00234 }

```

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

Sets the servos port.

Parameters

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

```

00239     {
00240         _mutex.lock();
00241         _sPort = port;
00242         _dChanged = true;
00243         _mutex.unlock();
00244     }

```

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

Sets the servos port info, data and selected port.

Parameters

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

```

00250     {
00251         _mutex.lock();
00252         _sPort = port;
00253         _sBaud = baud;
00254         _dChanged = true;
00255         _mutex.unlock();
00256     }

```

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

Sets the servos ID.

Parameters

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

```

00262     {
00263         // Error passing the data
00264         if (V.size() != _sNum) {
00265             qDebug() << "Error setting servos";
00266             return;
00267         }
00268
00269         _mutex.lock();
00270         for (int i = 0; i < V.size(); ++i) _servos[i].ID = V[i];
00271         _dChanged = true;
00272         _mutex.unlock();
00273     }
00274

```

3.9.4.28 void ServoThread::setSpeed (unsigned char *speed*) [inline]

Sets the servos speed.

Parameters

<i>speed</i>	unsigned char from 0 to 100 containing the % of speed
--------------	---

```

00279     {
00280         if (speed > 100) speed = 100;
00281
00282         _mutex.lock();
00283         _sSpeed = speed;
00284         _dChanged = true;
00285         _mutex.unlock();
00286     }

```

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

Calculates the angle of one servo in the selected position.

```
00428 {
00429     double n = b*b - a*a - z0*z0 - x0*x0 - y0*y0;
00430     double raiz = sqrt (n*n*y0*y0 - 4*(x0*x0 + y0*y0)*(-x0*x0*a*a + n*n/4));
00431
00432     if (x0 < 0) raiz *= -1;
00433     double y = (-n*y0 + raiz ) / (2*(x0*x0 + y0*y0));
00434
00435     int signe = 1;
00436     if ((b*b - (y0 + a)*(y0 + a)) < (x0*x0 + z0*z0) && x0 < 0) signe *= -1;
00437     double x = sqrt(a*a - y*y)*signe;
00438     return atan2 (y,x);
00439 }
```

3.9.4.30 void ServoThread::statusBar (QString , int) [signal]

Emmitted when the status bar must be changed.

3.9.4.31 void ServoThread::wakeUp () [inline]

Continues program's execution.

```
00290 {
00291     _mutex.lock();
00292     _pause = false;
00293     _cond.wakeOne();
00294     _mutex.unlock();
00295 }
```

3.9.4.32 void ServoThread::write (QString file)

Writes data to the selected directory.

Parameters

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

```
00134 {
00135     // Opening file for writing
00136     QFile f(file);
00137     f.open(QIODevice::WriteOnly);
00138     QDataStream df(&f);
00139
00140     _mutex.lock();
00141
00142     // Clamp and servos baud rate and port must be written
00143     df << int(Version::v_l_0) << _cBaud << _cPort << _sBaud <<
_sPort << _sSpeed
00144     << int(_mod) << _servos.size();
00145     for (const Servo &s : _servos) df << s.ID;
00146
00147     _mutex.unlock();
00148 }
```

3.9.5 Member Data Documentation

3.9.5.1 QVector4D ServoThread::_axis [private]

Contains the axis value.

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

Contains the buttons value.

3.9.5.3 int ServoThread::_cBaud [private]

Contains the baud rate used to communicate with the clamp.

3.9.5.4 QWaitCondition ServoThread::_cond [private]

To start and pause the thread.

3.9.5.5 QString ServoThread::_cPort [private]

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

3.9.5.6 bool ServoThread::_dChanged [private]

True if the data changes.

3.9.5.7 QVector< QVector< Dominoe > > ServoThread::_dominoe [private]

Contains all the dominoes information.

3.9.5.8 bool ServoThread::_end [private]

True when we must end executino.

3.9.5.9 bool ServoThread::_enter [private]

True if the enter key is pressed.

3.9.5.10 Mode ServoThread::_mod [private]

Contains the working mode.

3.9.5.11 QMutex ServoThread::_mutex [private]

To prevent memory errors between threads.

3.9.5.12 bool ServoThread::_pause [private]

Pauses the execution of the thread.

3.9.5.13 QVector4D ServoThread::_pos [private]

Contains the current position to show to the window.

3.9.5.14 int ServoThread::_sBaud [private]

Contains the used baud rate to communicate with the servos.

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

Contains the servos information.

3.9.5.16 const int ServoThread::_sNum = 4 [static], [private]

Number of servos to manage.

3.9.5.17 QString ServoThread::_sPort [private]

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

3.9.5.18 bool ServoThread::_sPortChanged [private]

True if the servos port changes.

3.9.5.19 `unsigned int ServoThread::_sSpeed` [private]

Speed of the robot.

3.9.5.20 `Status ServoThread::_status` [private]

Current status.

3.9.5.21 `const double ServoThread::a = 11.6` [private]

The arm length.

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

The forearm length.

3.9.5.23 `const uchar ServoThread::ccwCS = 2` [private]

The Counter Clock Wise Compliance Slope.

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

Contains the cosinus of 60.

3.9.5.25 `const uchar ServoThread::cwCS = 2` [private]

The Clock Wise Compliance Slope.

3.9.5.26 `const double ServoThread::descHeigh[3] = { 23.0, 22.7, 22.3 }` [private]

Descent height.

3.9.5.27 `const double ServoThread::idleHeigh = 22.0` [private]

Idle heigh.

3.9.5.28 `const double ServoThread::L1 = 5.499` [private]

The base center length.

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

The clamp support center lenght.

3.9.5.30 `const double ServoThread::maxAngle = 240.0` [private]

Maximum servo angle.

3.9.5.31 `const double ServoThread::maxErr = 3.0` [private]

Max available error.

3.9.5.32 `const double ServoThread::minAngle = 60.0` [private]

Minimum servo angle.

3.9.5.33 `const QVector4D ServoThread::posIdle = QVector4D(0.0f, 0.0f, idleHeigh, 150)` [private]

Idle position.

3.9.5.34 `const QVector4D ServoThread::posStart = QVector4D(11.5, 0.0f, idleHeigh, 150) [private]`

Starting position for the controlled mode.

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

Contains the sinus of 60.

3.9.5.36 `const double ServoThread::workHeigh = 23.3 [private]`

Working heigh.

3.9.5.37 `const double ServoThread::workRadSq = 144.0 [private]`

Working radius squared.

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

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

4 File Documentation

4.1 dxl/ax12.cpp File Reference

Contains the [AX12](#) class implementation.

4.1.1 Detailed Description

Contains the [AX12](#) class implementation.

4.2 dxl/ax12.h File Reference

Contains the [AX12](#) class declaration.

Classes

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

4.2.1 Detailed Description

Contains the [AX12](#) class declaration.

4.3 dxl/dxl_hal.cpp File Reference

Contains the Dynamixel SDK platform dependent header source.

4.3.1 Detailed Description

Contains the Dynamixel SDK platform dependent header source.

4.4 dxl/dxl_hal.h File Reference

Contains the Dynamixel SDK platform dependent header declaration.

Classes

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

4.4.1 Detailed Description

Contains the Dynamixel SDK platform dependent header declaration.

4.5 dxl/dynamixel.cpp File Reference

Contains the dynamixel class implementation.

4.5.1 Detailed Description

Contains the dynamixel class implementation.

4.6 dxl/dynamixel.h File Reference

Contains the dynamixel class declaration.

Classes

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

4.6.1 Detailed Description

Contains the dynamixel class declaration.

4.7 main.cpp File Reference

Contains the Main of the program.

Functions

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

4.7.1 Detailed Description

Contains the Main of the program.

4.7.2 Function Documentation

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

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

4.8 mainwindow.cpp File Reference

Contains the [MainWindow](#) class implementation.

4.8.1 Detailed Description

Contains the [MainWindow](#) class implementation.

4.9 mainwindow.h File Reference

Contains the [MainWindow](#) class declaration.

Classes

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

Namespaces

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

4.9.1 Detailed Description

Contains the [MainWindow](#) class declaration.

4.10 optionswindow.cpp File Reference

Contains the [OptionsWindow](#) class implementation.

4.10.1 Detailed Description

Contains the [OptionsWindow](#) class implementation.

4.11 optionswindow.h File Reference

Contains the [OptionsWindow](#) class declaration.

Classes

- class [OptionsWindow](#)
Class used to handle a Window to set the options.

Namespaces

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

4.11.1 Detailed Description

Contains the [OptionsWindow](#) class declaration.

4.12 servofind.cpp File Reference

4.13 servofind.h File Reference

Classes

- class [ServoFind](#)

4.14 servothread.cpp File Reference

Contains the [ServoThread](#) class implementation.

4.14.1 Detailed Description

Contains the [ServoThread](#) class implementation.

4.15 servothread.h File Reference

Contains the [ServoThread](#) class declaration.

Classes

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

4.15.1 Detailed Description

Contains the [ServoThread](#) class declaration.

4.16 stable.h File Reference

Contains all includes in a precompiled header.

4.16.1 Detailed Description

Contains all includes in a precompiled header.

The includes are:

- Algorithm
- QAbstractButton
- QApplication
- QComboBox
- QElapsedTimer
- QDebug
- QDialog
- QDialogButtonBox
- QDir
- QFileDialog
- QKeyEvent
- QLabel
- QMainWindow
- QMutex
- QSerialPortInfo
- QStandardPaths
- QStatusBar
- QString
- QtGlobal
- QThread
- QTime
- QTimer
- QVector
- QVector3D
- QVector4D
- QWaitCondition
- XJoystick

Index

- _ID
 - AX12, [9](#)
 - _axis
 - MainWindow, [31](#)
 - ServoThread, [58](#)
 - _axisV
 - MainWindow, [31](#)
 - _baud
 - ServoFind, [42](#)
 - _buts
 - MainWindow, [31](#)
 - ServoThread, [58](#)
 - _butsV
 - MainWindow, [31](#)
 - _cBaud
 - ServoThread, [58](#)
 - _cPort
 - ServoThread, [59](#)
 - _cond
 - ServoThread, [58](#)
 - _dChanged
 - ServoThread, [59](#)
 - _dataP
 - MainWindow, [31](#)
 - _dominoe
 - ServoThread, [59](#)
 - _dxl
 - AX12, [9](#)
 - _end
 - ServoThread, [59](#)
 - _enter
 - ServoThread, [59](#)
 - _joy
 - MainWindow, [31](#)
 - OptionsWindow, [37](#)
 - _max
 - ServoFind, [42](#)
 - _min
 - ServoFind, [42](#)
 - _mod
 - ServoThread, [59](#)
 - _mode
 - AX12, [9](#)
 - _mutex
 - ServoThread, [59](#)
 - _open
 - dxl_hal, [13](#)
 - _pause
 - ServoThread, [59](#)
 - _port
 - ServoFind, [42](#)
 - _portSize
 - OptionsWindow, [37](#)
 - _pos
 - ServoThread, [59](#)
 - _rads
 - AX12, [9](#)
 - _sBaud
 - ServoThread, [59](#)
 - _sF
 - OptionsWindow, [37](#)
 - _sNum
 - ServoThread, [59](#)
 - _sPort
 - ServoThread, [59](#)
 - _sPortChanged
 - ServoThread, [59](#)
 - _sSpeed
 - ServoThread, [59](#)
 - _sT
 - MainWindow, [31](#)
 - _serial
 - dxl_hal, [13](#)
 - _servo
 - OptionsWindow, [37](#)
 - ServoFind, [42](#)
 - _servoC
 - OptionsWindow, [37](#)
 - _servos
 - ServoThread, [59](#)
 - _status
 - ServoThread, [60](#)
 - _time
 - dxl_hal, [13](#)
 - _timed
 - dxl_hal, [13](#)
 - _timer
 - MainWindow, [31](#)
 - OptionsWindow, [37](#)
 - ~AX12
 - AX12, [5](#)
 - ~MainWindow
 - MainWindow, [28](#)
 - ~OptionsWindow
 - OptionsWindow, [35](#)
 - ~ServoFind
 - ServoFind, [41](#)
 - ~ServoThread
 - ServoThread, [48](#)
 - ~dynamixel
 - dynamixel, [15](#)
 - a
 - ServoThread, [60](#)
 - aSCount
 - MainWindow, [32](#)
 - AX12, [1](#)
 - _ID, [9](#)
 - _dxl, [9](#)
 - _mode, [9](#)

- [_rads](#), [9](#)
- [~AX12](#), [5](#)
- [AX12](#), [5](#)
- [AlarmLED](#), [4](#)
- [AlarmShutdown](#), [5](#)
- [BaudRate](#), [4](#)
- [CCWAngleLimit](#), [4](#)
- [CCWComplianceMargin](#), [4](#)
- [CCWComplianceSlope](#), [4](#)
- [CWAngleLimit](#), [4](#)
- [CWComplianceMargin](#), [3](#)
- [CWComplianceSlope](#), [4](#)
- [connectedID](#), [6](#)
- [getCurrentLoad](#), [6](#)
- [getCurrentPos](#), [6](#)
- [getCurrentSpeed](#), [6](#)
- [getCurrentTemp](#), [6](#)
- [getCurrentVoltage](#), [6](#)
- [getID](#), [7](#)
- [GoalPosition](#), [4](#)
- [HighestLimitTemp](#), [4](#)
- [HighestLimitVoltage](#), [4](#)
- [ID](#), [4](#)
- [LED](#), [3](#)
- [Lock](#), [4](#)
- [LowestLimitVoltage](#), [4](#)
- [MaxTorque](#), [4](#)
- [ModelNumber](#), [4](#)
- [Moving](#), [4](#)
- [MovingSpeed](#), [4](#)
- [PresentLoad](#), [4](#)
- [PresentPosition](#), [4](#)
- [PresentSpeed](#), [4](#)
- [PresentTemperature](#), [4](#)
- [PresentVoltage](#), [4](#)
- [Punch](#), [4](#)
- [RAM](#), [3](#)
- [ROM](#), [4](#)
- [Registered](#), [4](#)
- [ReturnDelayTime](#), [4](#)
- [setComplianceSlope](#), [7](#)
- [setDxl](#), [7](#)
- [setGoalPosition](#), [7](#)
- [setID](#), [7](#)
- [setJointMode](#), [8](#)
- [setMinMax](#), [8](#)
- [setRadians](#), [8](#)
- [setSpeed](#), [9](#)
- [StatusReturnLevel](#), [4](#)
- [TorqueEnable](#), [3](#)
- [TorqueLimit](#), [4](#)
- [VersionFirmware](#), [4](#)
- [AlarmLED](#)
 - [AX12](#), [4](#)
- [AlarmShutdown](#)
 - [AX12](#), [5](#)
- [b](#)
 - [ServoThread](#), [60](#)
- [BaudRate](#)
 - [AX12](#), [4](#)
- [begin](#)
 - [ServoThread](#), [47](#)
- [buttonClicked](#)
 - [OptionsWindow](#), [35](#)
- [CCWAngleLimit](#)
 - [AX12](#), [4](#)
- [CCWComplianceMargin](#)
 - [AX12](#), [4](#)
- [CCWComplianceSlope](#)
 - [AX12](#), [4](#)
- [CWAngleLimit](#)
 - [AX12](#), [4](#)
- [CWComplianceMargin](#)
 - [AX12](#), [3](#)
- [CWComplianceSlope](#)
 - [AX12](#), [4](#)
- [ccwCS](#)
 - [ServoThread](#), [60](#)
- [change_baudrate](#)
 - [dxl_hal](#), [12](#)
 - [dynamixel](#), [16](#)
- [clear](#)
 - [dxl_hal](#), [12](#)
- [close](#)
 - [dxl_hal](#), [12](#)
- [completion](#)
 - [ServoFind](#), [41](#)
- [connectedID](#)
 - [AX12](#), [6](#)
- [Controlled](#)
 - [ServoThread](#), [47](#)
- [cos60](#)
 - [ServoThread](#), [60](#)
- [cwCS](#)
 - [ServoThread](#), [60](#)
- [dH](#)
 - [dynamixel](#), [23](#)
- [descHeigh](#)
 - [ServoThread](#), [60](#)
- [Dominoe](#)
 - [ServoThread::Dominoe](#), [10](#)
- [dxl/ax12.cpp](#), [61](#)
- [dxl/ax12.h](#), [61](#)
- [dxl/dxl_hal.cpp](#), [61](#)
- [dxl/dxl_hal.h](#), [62](#)
- [dxl/dynamixel.cpp](#), [62](#)
- [dxl/dynamixel.h](#), [62](#)
- [dxl_hal](#), [11](#)
 - [_open](#), [13](#)
 - [_serial](#), [13](#)
 - [_time](#), [13](#)
 - [_timed](#), [13](#)
 - [change_baudrate](#), [12](#)
 - [clear](#), [12](#)
 - [close](#), [12](#)

- get_curr_time, 12
- isOpen, 12
- open, 12
- read, 12
- write, 12
- dynamixel, 13
 - ~dynamixel, 15
 - change_baudrate, 16
 - dH, 23
 - dynamixel, 15
 - gbCommStatus, 23
 - gbInstructionPacket, 23
 - gbRxGetLength, 23
 - gbRxPacketLength, 23
 - gbStatusPacket, 23
 - gdByteTransTime, 23
 - gdPacketStartTime, 23
 - gdRcvWaitTime, 24
 - get_comm_result, 16
 - get_packet_time, 16
 - get_rxpacket_error, 16
 - get_rxpacket_error_byte, 16
 - get_rxpacket_length, 16
 - get_rxpacket_parameter, 16
 - giBusUsing, 24
 - initialize, 17
 - is_packet_timeout, 17
 - isOpen, 17
 - ping, 17
 - read_byte, 17
 - read_word, 18
 - rx_packet, 18
 - set_packet_timeout, 19
 - set_packet_timeout_ms, 19
 - set_txpacket_id, 21
 - set_txpacket_instruction, 21
 - set_txpacket_length, 21
 - set_txpacket_parameter, 21
 - terminate, 21
 - tx_packet, 21
 - txrx_packet, 22
 - write_byte, 22
 - write_word, 23
- end
 - ServoThread, 48
- ending
 - ServoThread, 47
- events
 - OptionsWindow, 35
- gbCommStatus
 - dynamixel, 23
- gbInstructionPacket
 - dynamixel, 23
- gbRxGetLength
 - dynamixel, 23
- gbRxPacketLength
 - dynamixel, 23
- gbStatusPacket
 - dynamixel, 23
- gdByteTransTime
 - dynamixel, 23
- gdPacketStartTime
 - dynamixel, 23
- gdRcvWaitTime
 - dynamixel, 24
- get_comm_result
 - dynamixel, 16
- get_curr_time
 - dH, 12
- get_packet_time
 - dynamixel, 16
- get_rxpacket_error
 - dynamixel, 16
- get_rxpacket_error_byte
 - dynamixel, 16
- get_rxpacket_length
 - dynamixel, 16
- get_rxpacket_parameter
 - dynamixel, 16
- getCurrentLoad
 - AX12, 6
- getCurrentPos
 - AX12, 6
 - ServoThread, 48
- getCurrentSpeed
 - AX12, 6
- getCurrentTemp
 - AX12, 6
- getCurrentVoltage
 - AX12, 6
- getID
 - AX12, 7
- getServoBaud
 - ServoThread, 48
- getServoPort
 - ServoThread, 48
- getServoPortInfo
 - ServoThread, 48
- getServosInfo
 - ServoThread, 49
- getServosNum
 - ServoThread, 49
- getSpeed
 - ServoThread, 49
- giBusUsing
 - dynamixel, 24
- GoalPosition
 - AX12, 4
- going
 - ServoThread, 47
- HighestLimitTemp
 - AX12, 4
- HighestLimitVoltage
 - AX12, 4

- ID
 - AX12, 4
 - ServoThread::Servo, 39
- idleHeight
 - ServoThread, 60
- initialize
 - dynamixel, 17
- is_packet_timeout
 - dynamixel, 17
- isActive
 - ServoThread, 49
- isOpen
 - dxl_hal, 12
 - dynamixel, 17
- isPosAvailable
 - ServoThread, 49
- isReady
 - ServoThread, 50
- joyChanged
 - MainWindow, 28
- joystickChanged
 - MainWindow, 28
 - OptionsWindow, 36
- keyPressEvent
 - MainWindow, 28
 - OptionsWindow, 36
- keyReleaseEvent
 - MainWindow, 29
- L1
 - ServoThread, 60
- L2
 - ServoThread, 60
- LED
 - AX12, 3
- Lock
 - AX12, 4
- LowestLimitVoltage
 - AX12, 4
- main
 - main.cpp, 63
- main.cpp, 62
 - main, 63
- MainWindow, 24
 - _axis, 31
 - _axisV, 31
 - _buts, 31
 - _butsV, 31
 - _dataP, 31
 - _joy, 31
 - _sT, 31
 - _timer, 31
 - ~MainWindow, 28
 - aSCount, 32
 - joyChanged, 28
 - joystickChanged, 28
 - keyPressEvent, 28
 - keyReleaseEvent, 29
 - MainWindow, 27
 - Mode, 27
 - modeChanged, 29
 - on_actionImport_triggered, 29
 - on_actionOptions_triggered, 29
 - on_mode_clicked, 29
 - on_reset_clicked, 30
 - on_start_clicked, 30
 - read, 30
 - sCount, 32
 - ui, 32
 - update, 30
 - v_1_0, 27
 - Version, 27
 - write, 31
- mainwindow.cpp, 63
- mainwindow.h, 63
- Manual
 - ServoThread, 47
- maxAngle
 - ServoThread, 60
- maxErr
 - ServoThread, 60
- MaxTorque
 - AX12, 4
- minAngle
 - ServoThread, 60
- Mode
 - MainWindow, 27
 - ServoThread, 47
- modeChanged
 - MainWindow, 29
 - ServoThread, 50
- ModelNumber
 - AX12, 4
- Moving
 - AX12, 4
- MovingSpeed
 - AX12, 4
- mutex
 - ServoThread, 50
- on_actionImport_triggered
 - MainWindow, 29
- on_actionOptions_triggered
 - MainWindow, 29
- on_mode_clicked
 - MainWindow, 29
- on_reset_clicked
 - MainWindow, 30
- on_servoRefresh_clicked
 - OptionsWindow, 36
- on_start_clicked
 - MainWindow, 30
- open
 - dxl_hal, 12
- operator<

- ServoThread::Dominoe, 10
- operator=
 - ServoThread::Dominoe, 10
 - ServoThread::Servo, 39
- OptionsWindow, 32
 - _joy, 37
 - _portSize, 37
 - _sF, 37
 - _servo, 37
 - _servoC, 37
 - _timer, 37
 - ~OptionsWindow, 35
 - buttonClicked, 35
 - events, 35
 - joystickChanged, 36
 - keyPressEvent, 36
 - on_servoRefresh_clicked, 36
 - OptionsWindow, 34
 - QDB, 34
 - refreshFinish, 37
 - status, 37
 - storeData, 37
 - ui, 37
- optionwindow.cpp, 63
- optionwindow.h, 63
- ori
 - ServoThread::Dominoe, 11
- pause
 - ServoThread, 50
- ping
 - dynamixel, 17
- pos
 - ServoThread::Servo, 39
- posIdle
 - ServoThread, 60
- posStart
 - ServoThread, 60
- PresentLoad
 - AX12, 4
- PresentPosition
 - AX12, 4
- PresentSpeed
 - AX12, 4
- PresentTemperature
 - AX12, 4
- PresentVoltage
 - AX12, 4
- Punch
 - AX12, 4
- QCB
 - ServoFind, 41
- QDB
 - OptionsWindow, 34
- RAM
 - AX12, 3
- ROM
 - AX12, 4
- read
 - dxl_hal, 12
 - MainWindow, 30
 - ServoThread, 50
- read_byte
 - dynamixel, 17
- read_word
 - dynamixel, 18
- readPath
 - ServoThread, 51
- refreshFinish
 - OptionsWindow, 37
- Registered
 - AX12, 4
- Reset
 - ServoThread, 47
- reset
 - ServoThread, 51
- ReturnDelayTime
 - AX12, 4
- rotate
 - ServoThread, 47
- run
 - ServoFind, 41
 - ServoThread, 52
- rx_packet
 - dynamixel, 18
- sCount
 - MainWindow, 32
- Servo
 - ServoThread::Servo, 38
- ServoFind, 39
 - _baud, 42
 - _max, 42
 - _min, 42
 - _port, 42
 - _servo, 42
 - ~ServoFind, 41
 - completion, 41
 - QCB, 41
 - run, 41
 - ServoFind, 41
 - setData, 41
- ServoThread, 42
 - _axis, 58
 - _buts, 58
 - _cBaud, 58
 - _cPort, 59
 - _cond, 58
 - _dChanged, 59
 - _dominoe, 59
 - _end, 59
 - _enter, 59
 - _mod, 59
 - _mutex, 59
 - _pause, 59
 - _pos, 59

- [_sBaud, 59](#)
- [_sNum, 59](#)
- [_sPort, 59](#)
- [_sPortChanged, 59](#)
- [_sSpeed, 59](#)
- [_servos, 59](#)
- [_status, 60](#)
- [~ServoThread, 48](#)
- [a, 60](#)
- [b, 60](#)
- [begin, 47](#)
- [ccwCS, 60](#)
- [Controlled, 47](#)
- [cos60, 60](#)
- [cwCS, 60](#)
- [descHeigh, 60](#)
- [end, 48](#)
- [ending, 47](#)
- [getCurrentPos, 48](#)
- [getServoBaud, 48](#)
- [getServoPort, 48](#)
- [getServoPortInfo, 48](#)
- [getServosInfo, 49](#)
- [getServosNum, 49](#)
- [getSpeed, 49](#)
- [going, 47](#)
- [idleHeigh, 60](#)
- [isActive, 49](#)
- [isPosAvailable, 49](#)
- [isReady, 50](#)
- [L1, 60](#)
- [L2, 60](#)
- [Manual, 47](#)
- [maxAngle, 60](#)
- [maxErr, 60](#)
- [minAngle, 60](#)
- [Mode, 47](#)
- [modeChanged, 50](#)
- [mutex, 50](#)
- [pause, 50](#)
- [posIdle, 60](#)
- [posStart, 60](#)
- [read, 50](#)
- [readPath, 51](#)
- [Reset, 47](#)
- [reset, 51](#)
- [rotate, 47](#)
- [run, 52](#)
- [ServoThread, 47](#)
- [setAngles, 54](#)
- [setData, 54](#)
- [setGoalPosition, 55](#)
- [setMode, 55](#)
- [setSID, 57](#)
- [setServoBaud, 55](#)
- [setServoPort, 55](#)
- [setServoPortInfo, 57](#)
- [setSpeed, 57](#)
- [sin60, 61](#)
- [singleAngle, 57](#)
- [Status, 47](#)
- [statusBar, 58](#)
- [take, 47](#)
- [v_1_0, 47](#)
- [Version, 47](#)
- [waiting, 47](#)
- [wakeUp, 58](#)
- [workHeigh, 61](#)
- [workRadSq, 61](#)
- [write, 58](#)
- [ServoThread::Dominoe, 9](#)
 - [Dominoe, 10](#)
 - [operator<, 10](#)
 - [operator=, 10](#)
 - [ori, 11](#)
 - [X, 11](#)
 - [Y, 11](#)
- [ServoThread::Servo, 38](#)
 - [ID, 39](#)
 - [operator=, 39](#)
 - [pos, 39](#)
 - [Servo, 38](#)
- [servofind.cpp, 64](#)
- [servofind.h, 64](#)
- [servothread.cpp, 64](#)
- [servothread.h, 64](#)
- [set_packet_timeout](#)
 - [dynamixel, 19](#)
- [set_packet_timeout_ms](#)
 - [dynamixel, 19](#)
- [set_txpacket_id](#)
 - [dynamixel, 21](#)
- [set_txpacket_instruction](#)
 - [dynamixel, 21](#)
- [set_txpacket_length](#)
 - [dynamixel, 21](#)
- [set_txpacket_parameter](#)
 - [dynamixel, 21](#)
- [setAngles](#)
 - [ServoThread, 54](#)
- [setComplianceSlope](#)
 - [AX12, 7](#)
- [setData](#)
 - [ServoFind, 41](#)
 - [ServoThread, 54](#)
- [setDxl](#)
 - [AX12, 7](#)
- [setGoalPosition](#)
 - [AX12, 7](#)
 - [ServoThread, 55](#)
- [setID](#)
 - [AX12, 7](#)
- [setJointMode](#)
 - [AX12, 8](#)
- [setMinMax](#)
 - [AX12, 8](#)

- setMode
 - ServoThread, [55](#)
- setRadians
 - AX12, [8](#)
- setSID
 - ServoThread, [57](#)
- setServoBaud
 - ServoThread, [55](#)
- setServoPort
 - ServoThread, [55](#)
- setServoPortInfo
 - ServoThread, [57](#)
- setSpeed
 - AX12, [9](#)
 - ServoThread, [57](#)
- sin60
 - ServoThread, [61](#)
- singleAngle
 - ServoThread, [57](#)
- stable.h, [64](#)
- Status
 - ServoThread, [47](#)
- status
 - OptionsWindow, [37](#)
- statusBar
 - ServoThread, [58](#)
- StatusReturnLevel
 - AX12, [4](#)
- storeData
 - OptionsWindow, [37](#)
- take
 - ServoThread, [47](#)
- terminate
 - dynamixel, [21](#)
- TorqueEnable
 - AX12, [3](#)
- TorqueLimit
 - AX12, [4](#)
- tx_packet
 - dynamixel, [21](#)
- txrx_packet
 - dynamixel, [22](#)
- Ui, [1](#)
- ui
 - MainWindow, [32](#)
 - OptionsWindow, [37](#)
- update
 - MainWindow, [30](#)
- v_1_0
 - MainWindow, [27](#)
 - ServoThread, [47](#)
- Version
 - MainWindow, [27](#)
 - ServoThread, [47](#)
- VersionFirmware
 - AX12, [4](#)
- waiting
 - ServoThread, [47](#)
- wakeUp
 - ServoThread, [58](#)
- workHeigh
 - ServoThread, [61](#)
- workRadSq
 - ServoThread, [61](#)
- write
 - dxl_hal, [12](#)
 - MainWindow, [31](#)
 - ServoThread, [58](#)
- write_byte
 - dynamixel, [22](#)
- write_word
 - dynamixel, [23](#)
- X
 - ServoThread::Dominoe, [11](#)
- Y
 - ServoThread::Dominoe, [11](#)