DeltaRobot

v0.4

Generated by Doxygen 1.8.9.1

Sat Jun 6 2015 13:05:15

Contents

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.

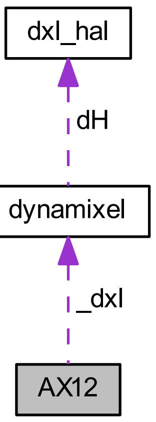
3 Class Documentation

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)

  Initializator 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 0º to 300º

- 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 300º
- 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 comunication.
- 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

```
    {
        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

    };
```

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

```
    {
        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
    };
```

3.1.3 Constructor & Destructor Documentation

3.1.3.1 AX12::AX12 ( )

Default constructor.

```
          :
    _dxl(NULL),
    _ID(-1),
    _mode(true),
    _rads(false)
{

}
```

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

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

```
                                :
    _dxl(dxl),
    _ID(ID),
   _mode(true),
   _rads(false)
{
    if (_ID < 0 or _dxl == NULL) return;
    dxl->write_byte(_ID, RAM::TorqueEnable, true);
}
```

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

Copy constructor.

```
                        :
    _dxl(a._dxl),
    _ID(a._ID),
    _mode(a._mode),
    _rads(a._rads)
{

}
```

3.1.3.4 AX12::∼AX12 ( )

Default destructor.

```
{

}
```

3.1.4 Member Function Documentation

3.1.4.1 QVector< int > AX12::connectedID ( )

Returns all active servos;.

```
{
    if (_dxl == NULL) return QVector<int> (0);

    QVector <int> res;
    for (int i = 0; i < 256; ++i) {
        _dxl->ping(i);
        if (_dxl->get_comm_result() == COMM_RXSUCCESS) res.push_back(i);
    }

    return res;
}
```

3.1.4.2 double AX12::getCurrentLoad ( )

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

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

3.1.4.3 double AX12::getCurrentPos ( )

Returns the current position from 0º to 300º

```
{
    if (_ID < 0 or _dxl == NULL) return 0;
    int pos = _dxl->read_word(_ID, RAM::PresentPosition);
    if (_dxl->get_comm_result() != COMM_RXSUCCESS) return -1;

    if (_rads) return double((pos/1023.0)*(5.0*M_PI)/3.0);
    return double((pos/1023.0)*300);
}
```

3.1.4.4 double AX12::getCurrentSpeed ( )

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

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

3.1.4.5 int AX12::getCurrentTemp ( )

Returns the current Temperature in Celsius.

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

3.1.4.6 double AX12::getCurrentVoltage ( )

Returns the current voltage in Volts.

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

3.1.4.7 int AX12::getID ( ) [inline]

To get the current ID.

```
{ return _ID; }
```

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

Sets the compliance slope.

Parameters

ccw	Counter Clock Wise Compliance Slope
cw	Clock Wise Compliance Slope

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

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

Sets the dynamixel interface.

Parameters

dxl	Pointer to the dynamixel control class

```
{ _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

goal	Position (in degrees if not radian mode) or % speed if used wheel mode

```
{
    if (_ID < 0 or _dxl == NULL) return;

    // Conversion to radians if radians mode
    if (_rads) goal *= 180/M_PI;

    if (goal > 300.0) goal = 300.0;
    else if (goal < 0) goal = 0;
    _dxl->write_word(_ID, RAM::GoalPosition, int((goal/300.0)*1023));
}
```

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

To set a new ID.

Parameters

ID	the new ID

```
{
    _ID = ID;
    if (_ID < 0 or _dxl == NULL) return;
    _dxl->write_byte(_ID, RAM::TorqueEnable, true);
}
```

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

To set Joint/Wheel mode.

Parameters

mode	True if Joint and false if Wheel mode

```
{
    if (_ID < 0 or _dxl == NULL) return;
    _mode = mode;
    if (_mode) {
        _dxl->write_word(_ID, ROM::CWAngleLimit, 0);
        _dxl->write_word(_ID, ROM::CCWAngleLimit, 1023);
    }
    else {
        _dxl->write_word(_ID, ROM::CWAngleLimit, 0);
        _dxl->write_word(_ID, ROM::CCWAngleLimit, 0);
    }
}
```

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

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

Parameters

min	Minimum value from servo
max	Maximum value from servo

```
{
    if (_ID < 0 or _dxl == NULL) return;

    if (min > max)  {
        double aux = min;
        min = max;
        max = aux;
    }

    if (_rads) min *= 180/M_PI;

    if (min < 0.0) min = 0;
    if (max > 300.0) max = 300;

    min = (min/300)*1023;
    max = (max/300)*1023;

    _dxl->write_word(_ID, ROM::CWAngleLimit, int (min));
    _dxl->write_word(_ID, ROM::CCWAngleLimit, int (max));
}
```

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

Sets the radians mode.

Parameters

rads	True if radians mode is used

```
{ _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.

```
{
    if (_ID < 0 or _dxl == NULL) return;
    if (speed > 100.0) speed = 100.0;
    if (_mode) {
        if (speed < 0.0) speed = 0.0;

        int byte = int((speed/100.0) * 1024.0);
        if (speed == 100.0) byte = 0;
        _dxl->write_byte(_ID, RAM::MovingSpeed, byte);
    }
    else {
        if (speed < -100.0) speed = -100.0;

        int byte = int(((speed + 100)/100.0) * 1024);
        _dxl->write_byte(_ID, RAM::MovingSpeed, byte);
    }

}
```

3.1.5 Member Data Documentation

3.1.5.1 dynamixel* AX12::_dxl [private]

Contains the dynamixel comunication.

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 comparisions.
- 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.

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

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

Initialization constructor.

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

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

Initialization constructor with vector.

```
: 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.

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

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

Overloaded operator to copy.

```
        {
            this->X = d.X;
            this->Y = d.Y;
            this->ori = d.ori;
            return *this;
        }
```

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*)

```
{
    bool res = _serial.setBaudRate(qint32(baudrate));
    return int(res);

}
```

3.3.2.2 void dxl_hal::clear (void)

```
{
    // Clear communication buffer

    if (!_serial.isOpen()) return;
    _serial.clear();

}
```

3.3.2.3 void dxl_hal::close (void)

```
{
    // Closing device
    _serial.close();
    _open = false;
}
```

3.3.2.4 double dxl_hal::get_curr_time ( )

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

3.3.2.5 bool dxl_hal::isOpen ( ) [inline]

```
{ return _open; }
```

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

```
{
    // Opening device
    // devIndex: Device index
    // baudrate: Real baudrate (ex> 115200, 57600, 38400...)
    // Return: 0(Failed), 1(Succeed)

    _serial.setPortName(devName);
    _serial.setBaudRate(qint32(baudrate));
    _serial.setDataBits(QSerialPort::Data8);
    _serial.setParity(QSerialPort::NoParity);
    _serial.setStopBits(QSerialPort::OneStop);
    _serial.setFlowControl(QSerialPort::NoFlowControl);
    if(not _serial.open(QIODevice::ReadWrite)) return false;
    _open = true;
    return true;
}
```

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

```
{
    // Recieving date
    // *pPacket: data array pointer
    // numPacket: number of data array
    // Return: number of data recieved. -1 is error.
    _timed = false;
    if (_serial.isOpen()) {
        int n = _serial.read((char*)pPacket, numPacket);
        _timed = _serial.waitForReadyRead(_time);
        _timed = not _timed;
        return n;
    }
    else return -1;

}
```

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

```
{
    // Transmiting date
    // *pPacket: data array pointer
    // numPacket: number of data array
    // Return: number of data transmitted. -1 is error.
    _timed = false;
    if (_serial.isOpen()) {
        int n = _serial.write((char*)pPacket, numPacket);
        _timed = _serial.waitForBytesWritten(_time);
        _timed = not _timed;
        return n;
    }
    else return -1;

}
```

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:

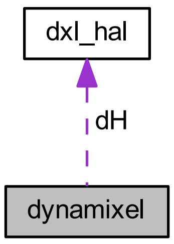
- dxl/dxl_hal.h
- dxl/dxl_hal.cpp

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)

  Initializates the port.
- int change_baudrate (int baud_rate)

  Changes the current baud rate.
- int terminate (void)

  Closes the comunication.
- int get_comm_result ()

  Returns the current com status.
- void tx_packet (void)

  Sends a packet.
- void rx_packet (void)

  Receives a packet.
- void txrx_packet (void)

  Sends and receives a packet.
- void set_txpacket_id (int id)

  Sets the sending packet ID.
- void set_txpacket_instruction (int instruction)

  Sets the sending packet instruction.
- void set_txpacket_parameter (int index, int value)

  Sets the sending packet parameter.
- void set_txpacket_length (int length)

  Sets the sending packet length.
- bool get_rxpacket_error (int error)

  Returns false if no receive error and true if there's an error.
- int get_rxpacket_error_byte (void)

  Returns the error byte.
- int get_rxpacket_parameter (int index)

  Returns the received parameter.
- int get_rxpacket_length ()

  Returns the received packet length.
- void ping (int id)

  Ping to the selected id, check com status for the ping result.
- int read_byte (int id, int address)

  Reads a byte from the selected ID at the selected address.
- void write_byte (int id, int address, int value)

  Writes a byte to the selected ID at the selected address.
- int read_word (int id, int address)

  Reads a word to the selected ID at the selected address.
- void write_word (int id, int address, int value)

  Writes a word to the selected ID at the selected address.
- double get_packet_time ()

  Returns the packet time.
- void set_packet_timeout (int NumRcvByte)

  Sets the timeout in number of received bytes.
- void set_packet_timeout_ms (int msec)

  Sets the timeout in ms.
- bool is_packet_timeout ()

  Returns true if the packet is timeout.

Private Attributes

- dxl_hal dH

  Conains the serial port comunication.
- unsigned char gbInstructionPacket [MAXNUM_TXPACKET] = {0}

  Contains all the instructions.
- unsigned char gbStatusPacket [MAXNUM_RXPACKET] = {0}

  Contains the status.
- unsigned int gbRxPacketLength = 0

  Received packet length.
- unsigned int gbRxGetLength = 0

  Temporal length from the received packet.
- double gdPacketStartTime = 0.0

  Packet start time.
- double gdByteTransTime = 0.0

  Byte transmission time.
- double gdRcvWaitTime = 0.0

  Receive wait time.
- int gbCommStatus = COMM_RXSUCCESS

  Current communication status.
- int giBusUsing = 0

  True if the bus 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.

```
{}
```

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

Initialization constructor.

```
{
    initialize(port_num, baud_rate);
}
```

3.4.2.3 dynamixel::∼dynamixel ( ) [inline]

Default destructor.

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

3.4.3 Member Function Documentation

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

Changes the current baud rate.

```
{
    int result = 0;
    float baudrate = (float)baud_rate;

    result = dH.change_baudrate(baudrate);
    if(result == 1)
        gdByteTransTime = 1000.0f / baudrate * 10.0; // 1000/baudrate(bit per msec) *
 10(start bit + data bit + stop bit)

    return result;
}
```

3.4.3.2 int dynamixel::get_comm_result ( ) [inline]

Returns the current com status.

```
{ return gbCommStatus; }
```

3.4.3.3 double dynamixel::get_packet_time (void)

Returns the packet time.

```
{
    double elapsed_time;

    elapsed_time = (double)(dH.get_curr_time() -
gdPacketStartTime);

    // Overflow
    if(elapsed_time < 0) gdPacketStartTime = dH.get_curr_time();

    return elapsed_time;
}
```

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

error	Selects the error to check

```
{
    if( gbStatusPacket[PRT1_PKT_ERRBIT] & (unsigned char)error )
        return true;

    return false;
}
```

3.4.3.5 int dynamixel::get_rxpacket_error_byte (void)

Returns the error byte.

```
{
    return gbStatusPacket[PRT1_PKT_ERRBIT];
}
```

3.4.3.6 int dynamixel::get_rxpacket_length ( )

Returns the received packet length.

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

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

Returns the received parameter.

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

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

Initializates the port.

```
{
    if( baud_rate < 1900 ) return 0;

    if( not dH.open(port_num, baud_rate) ) return false;

    // 1000/baudrate(bit per msec) * 10(start bit + data bit + stop bit)
    gdByteTransTime = 1000.0 / (double)baud_rate * 10.0;

    gbCommStatus = COMM_RXSUCCESS;
    giBusUsing = 0;

    return true;
}
```

3.4.3.9 bool dynamixel::is_packet_timeout (void)

Returns true if the packet is timeout.

Returns

True if the packet is timeout

```
{
    if(this->get_packet_time() > gdRcvWaitTime)
        return true;
    return false;
}
```

3.4.3.10 bool dynamixel::isOpen ( ) [inline]

True if the port is open.

```
{ 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

id	ID where the ping is done

```
{
    while(giBusUsing);

    gbInstructionPacket[PRT1_PKT_ID] = (unsigned char)id;
    gbInstructionPacket[PRT1_PKT_INSTRUCTION] = INST_PING;
    gbInstructionPacket[PRT1_PKT_LENGTH] = 2;

    txrx_packet();
}
```

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

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

Parameters

id	Selects the ID to read the byte
address	Selects the address to read the byte

```
{
    while(giBusUsing);

    gbInstructionPacket[PRT1_PKT_ID] = (unsigned char)id;
    gbInstructionPacket[PRT1_PKT_INSTRUCTION] = INST_READ;
    gbInstructionPacket[PRT1_PKT_PARAMETER0+0] = (unsigned char)address;
    gbInstructionPacket[PRT1_PKT_PARAMETER0+1] = 1;
    gbInstructionPacket[PRT1_PKT_LENGTH] = 4;

    txrx_packet();

    return (int)gbStatusPacket[PRT1_PKT_PARAMETER0];
}
```

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

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

Parameters

id	Selects the ID to read the word
address	Selects the address to read the word

```
{
    while(giBusUsing);

    gbInstructionPacket[PRT1_PKT_ID] = (unsigned char)id;
    gbInstructionPacket[PRT1_PKT_INSTRUCTION] = INST_READ;
    gbInstructionPacket[PRT1_PKT_PARAMETER0+0] = (unsigned char)address;
    gbInstructionPacket[PRT1_PKT_PARAMETER0+1] = 2;
    gbInstructionPacket[PRT1_PKT_LENGTH] = 4;

    txrx_packet();

    return MAKEWORD((int)gbStatusPacket[PRT1_PKT_PARAMETER0+0], (int)
      gbStatusPacket[PRT1_PKT_PARAMETER0+1]);
}
```

3.4.3.14 void dynamixel::rx_packet (void)

Receives a packet.

```
{
    unsigned char i = 0, j = 0, nRead = 0;
    unsigned char checksum = 0;

    if( giBusUsing == 0 )
        return;

    if( gbInstructionPacket[PRT1_PKT_ID] == BROADCAST_ID )
    {
        gbCommStatus = COMM_RXSUCCESS;
        giBusUsing = 0;
        return;
    }

    if( gbCommStatus == COMM_TXSUCCESS )
    {
        gbRxGetLength = 0;
        //gbRxPacketLength = 6; //minimum wait length
    }

    while(1)
    {
        nRead = dH.read( &gbStatusPacket[gbRxGetLength],
      gbRxPacketLength - gbRxGetLength );
        gbRxGetLength += nRead;

        if(gbRxGetLength > 4)
            gbRxPacketLength = gbStatusPacket[PRT1_PKT_LENGTH] + 4;

        if( gbRxGetLength < gbRxPacketLength )
        {
            if( is_packet_timeout() == 1 )
            {
                if(gbRxGetLength == 0)
                    gbCommStatus = COMM_RXTIMEOUT;
                else
                    gbCommStatus = COMM_RXCORRUPT;
                giBusUsing = 0;
                return;
            }
            gbCommStatus = COMM_RXWAITING;
            //return;
        }
        else
        {
            break;
        }
    }

    // Find packet header
    for( i=0; i<(gbRxGetLength-1); i++ )
    {
        if( gbStatusPacket[i] == 0xff && gbStatusPacket[i+1] == 0xff )
            break;
        else if( i == gbRxGetLength-2 && gbStatusPacket[gbRxGetLength-1] == 0xff )
            break;
        else {
            gbCommStatus = COMM_RXCORRUPT;
            return;
        }
    }

    if( i > 0 )
    {
        for( j=0; j<(gbRxGetLength-i); j++ )
            gbStatusPacket[j] = gbStatusPacket[j + i];

        gbRxGetLength -= i;
    }

    // Check id pairing
    if( gbInstructionPacket[PRT1_PKT_ID] != gbStatusPacket[PRT1_PKT_ID])
    {
        gbCommStatus = COMM_RXCORRUPT;
        giBusUsing = 0;
        return;
    }

    // Check checksum
    for( i=0; i<(gbStatusPacket[PRT1_PKT_LENGTH]+1); i++ )
        checksum += gbStatusPacket[i+2];
    checksum = ~checksum;

    if( gbStatusPacket[gbStatusPacket[PRT1_PKT_LENGTH]+3] != checksum )
    {
        gbCommStatus = COMM_RXCORRUPT;
        giBusUsing = 0;
        return;
    }

    gbCommStatus = COMM_RXSUCCESS;
    giBusUsing = 0;
}
```

3.4.3.15 void dynamixel::set_packet_timeout (int *NumRcvByte*)

Sets the timeout in number of received bytes.

Parameters

NumRcvByte	Number of received bytes to do a timeout

```
{
    gdPacketStartTime = dH.get_curr_time();
    gdRcvWaitTime = (gdByteTransTime*(double)NumRcvByte + 2.0*LATENCY_TIME + 2.
0);
}
```

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

Sets the timeout in ms.

Parameters

msec	Miliseconds for the timeout

```
{
    gdPacketStartTime = dH.get_curr_time();
    gdRcvWaitTime = (double)msec;
}
```

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

Sets the sending packet ID.

```
{
    gbInstructionPacket[PRT1_PKT_ID] = (unsigned char)id;
}
```

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

Sets the sending packet instruction.

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

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

Sets the sending packet length.

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

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

Sets the sending packet parameter.

```
{
    gbInstructionPacket[PRT1_PKT_PARAMETER0+index] = (unsigned char)value;

}
```

3.4.3.21 int dynamixel::terminate (void)

Closes the comunication.

```
{
    dH.close();
    return 0;
}
```

3.4.3.22 void dynamixel::tx_packet (void)

Sends a packet.

```
{
    unsigned char pkt_idx = 0;
    unsigned char TxNumByte, RealTxNumByte;
    unsigned char checksum = 0;

    if( giBusUsing == 1 )
    {
        gbCommStatus = COMM_TXFAIL;
        return;
    }

    giBusUsing = 1;

    if( gbInstructionPacket[PRT1_PKT_INSTRUCTION] != INST_PING
        && gbInstructionPacket[PRT1_PKT_INSTRUCTION] != INST_READ
        && gbInstructionPacket[PRT1_PKT_INSTRUCTION] != INST_WRITE
        && gbInstructionPacket[PRT1_PKT_INSTRUCTION] != INST_REG_WRITE
        && gbInstructionPacket[PRT1_PKT_INSTRUCTION] != INST_ACTION
        && gbInstructionPacket[PRT1_PKT_INSTRUCTION] != INST_RESET
        && gbInstructionPacket[PRT1_PKT_INSTRUCTION] != INST_SYNC_WRITE )
    {
        gbCommStatus = COMM_TXERROR;
        giBusUsing = 0;
        return;
    }

    gbInstructionPacket[0] = 0xff;
    gbInstructionPacket[1] = 0xff;
    for( pkt_idx = 0; pkt_idx < (gbInstructionPacket[PRT1_PKT_LENGTH]+1); pkt_idx++ )
        checksum += gbInstructionPacket[pkt_idx+2];
    gbInstructionPacket[gbInstructionPacket[PRT1_PKT_LENGTH]+3] = ~
checksum;

    //if( gbCommStatus == COMM_RXTIMEOUT || gbCommStatus == COMM_RXCORRUPT )
    //  dH.clear();

    dH.clear();

    TxNumByte = gbInstructionPacket[PRT1_PKT_LENGTH] + 4;
    RealTxNumByte = dH.write( gbInstructionPacket, TxNumByte );

    if( TxNumByte != RealTxNumByte )
    {
        gbCommStatus = COMM_TXFAIL;
        giBusUsing = 0;
        return;
    }

    if( gbInstructionPacket[PRT1_PKT_INSTRUCTION] == INST_READ )
    {
        gbRxPacketLength =  gbInstructionPacket[PRT1_PKT_PARAMETER0+1] + 6;
        set_packet_timeout( gbInstructionPacket[PRT1_PKT_PARAMETER0+1] + 6 );
    }
    else
    {
        gbRxPacketLength = 6;
        set_packet_timeout( 6 );
    }

    gbCommStatus = COMM_TXSUCCESS;
}
```

3.4.3.23 void dynamixel::txrx_packet (void)

Sends and receives a packet.

```
{
    tx_packet();

    if( gbCommStatus != COMM_TXSUCCESS )
        return;


    rx_packet();
}
```

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

id	Selects the ID to write the byte

address	Selects the address to write the byte
value	Value to set at the selected location

```
{
    while(giBusUsing);

    gbInstructionPacket[PRT1_PKT_ID] = (unsigned char)id;
    gbInstructionPacket[PRT1_PKT_INSTRUCTION] = INST_WRITE;
    gbInstructionPacket[PRT1_PKT_PARAMETER0+0] = (unsigned char)address;
    gbInstructionPacket[PRT1_PKT_PARAMETER0+1] = (unsigned char)value;
    gbInstructionPacket[PRT1_PKT_LENGTH] = 4;

    txrx_packet();
}
```

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

id	Selects the ID to write the word
address	Selects the address to write the word
value	Value to set at the selected location

```
{
    while(giBusUsing);

    gbInstructionPacket[PRT1_PKT_ID] = (unsigned char)id;
    gbInstructionPacket[PRT1_PKT_INSTRUCTION] = INST_WRITE;
    gbInstructionPacket[PRT1_PKT_PARAMETER0+0] = (unsigned char)address;
    gbInstructionPacket[PRT1_PKT_PARAMETER0+1] = (unsigned char)LOBYTE(value);
    gbInstructionPacket[PRT1_PKT_PARAMETER0+2] = (unsigned char)HIBYTE(value);
    gbInstructionPacket[PRT1_PKT_LENGTH] = 5;

    txrx_packet();
}
```

3.4.4 Member Data Documentation

3.4.4.1 dxl_hal dynamixel::dH [private]

Conains the serial port comunication.

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 if being used.

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

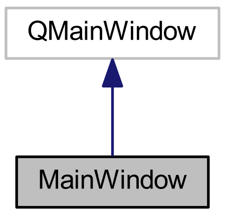
- dxl/dynamixel.h
- dxl/dynamixel.cpp

3.5 MainWindow Class Reference

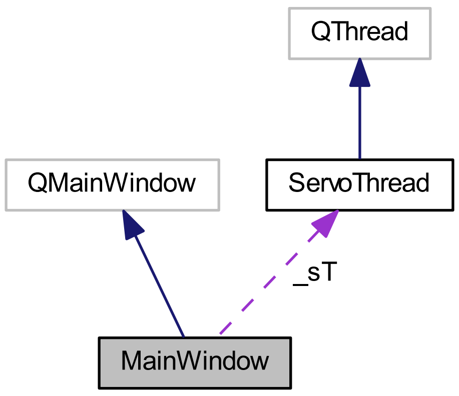
Contains all the windows and other classes.

`#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 realease 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 minimun servos to work.
- static const int aSCount = 0

  Contains the number of additional servos used.

3.5.1 Detailed Description

Contains all the windows and other classes.

3.5.2 Member 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

```
                    {
        v_1_0
    };
```

3.5.4 Constructor & Destructor Documentation

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

Default constructor.

```
                                        :
    QMainWindow(parent),
    _axis(XJoystick::AxisCount),
    _axisV(XJoystick::AxisCount),
    _buts(XJoystick::ButtonCount),
    _butsV(XJoystick::ButtonCount),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    connect(&_joy, SIGNAL(changed()), this, SLOT(joyChanged()));
    connect(&_timer, SIGNAL(timeout()), this, SLOT(update()));
    connect(&_sT, SIGNAL(statusBar(QString, int)),
            ui->statusbar, SLOT(showMessage(QString,int)));
    connect(&_sT, SIGNAL(modeChanged(Mode)), this, SLOT(
modeChanged(Mode)));


    _timer.setInterval(10);
    _timer.start();

    // JOYSTICK
    QVector< QString > V(_joy.getAllAxis());
    // Adding axis
    QGridLayout *wL = new QGridLayout;
    for (int i = 0; i < XJoystick::AxisCount; ++i) {
        QHBoxLayout *L = new QHBoxLayout;
        L->addWidget(new QLabel(V[i].append(":"), this));
        _axis[i] = new QLabel("#");
        L->addWidget(_axis[i]);
        L->addStretch();
        wL->addLayout(L, i%3, i/3);
    }
    ui->joyAxis->setLayout(wL);

    // Adding buttons
    wL = new QGridLayout;
    for (int i = 0; i < XJoystick::ButtonCount; ++i) {
        _buts[i] = new QLabel(QString::number(i + 1));
        wL->addWidget(_buts[i], i/8, i%8);
        _buts[i]->setEnabled(false);
        _buts[i]->hide();
    }
    ui->joyButs->setLayout(wL);
    ui->joyAxis->hide();
    ui->joyButs->hide();
    ui->line->hide();

    // Creating data Path
    _dataP = QStandardPaths::writableLocation(QStandardPaths::AppDataLocation);
    QDir dir(_dataP);
    if (!dir.exists()) dir.mkpath(_dataP);

    read();
    _sT.start();
}
```

3.5.4.2 MainWindow::∼MainWindow ( )

Default destructor.

```
{
    delete ui;
}
```

3.5.5 Member Function Documentation

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

Handles a joystick update.

```
{
    int sel = _joy.current();

    QVector< XJoystick::Info > V(_joy.available());
    bool found = false;
    int i = 0;
    while (i < V.size() and not found) { found = V[i].ID == sel; ++i; }
    if (not found) {
        if (V.size() > 0) {
            _joy.select(V[0].ID);
            ui->line->hide();

            // Showing axis
            ui->joyAxis->show();

            // Showing buttons
            for (QLabel *l : _buts) l->hide();
            ui->joyButs->show();
            int n = _joy.buttonCount();
            for (int i = 0; i < n; ++i) _buts[i]->show();
        }
        else {
            _joy.select(-1);
            ui->joyAxis->hide();
            ui->joyButs->hide();
            ui->line->hide();
        }
    }
    emit joystickChanged();
}
```

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.

```
{
    if (event->isAutoRepeat()) return;
    if (event->key() == Qt::Key_A) _joy.axisPress(0, -100);
    else if (event->key() == Qt::Key_D) _joy.axisPress(0, 100);
    else if (event->key() == Qt::Key_W) _joy.axisPress(1, 100);
    else if (event->key() == Qt::Key_S) _joy.axisPress(1, -100);
    else if (event->key() == Qt::Key_Q) _joy.axisPress(2, -100);
    else if (event->key() == Qt::Key_E) _joy.axisPress(2, 100);
    else if (event->key() == Qt::Key_J) _joy.axisPress(3, -100);
    else if (event->key() == Qt::Key_K) _joy.axisPress(3, 100);
    else if (event->key() == Qt::Key_R) _sT.reset();
    else if (event->key() == Qt::Key_Return) _joy.buttonPress(0, true);

    this->update();
}
```

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

Handles the realease of a key.

```
{
    if (event->isAutoRepeat()) return;
    if (event->key() == Qt::Key_A) _joy.axisRelease(0);
    else if (event->key() == Qt::Key_D) _joy.axisRelease(0);
    else if (event->key() == Qt::Key_W) _joy.axisRelease(1);
    else if (event->key() == Qt::Key_S) _joy.axisRelease(1);
    else if (event->key() == Qt::Key_Q) _joy.axisRelease(2);
    else if (event->key() == Qt::Key_E) _joy.axisRelease(2);
    else if (event->key() == Qt::Key_J) _joy.axisRelease(3);
    else if (event->key() == Qt::Key_K) _joy.axisRelease(3);
    else if (event->key() == Qt::Key_Return) _joy.buttonRelease(0);
    this->update();
}
```

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

Handles the change of a mode in the thread.

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

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

Opens the import of Dominoes file.

```
{
    QString caption("Open Dominoes File");
    QString dir(QDir::homePath());
    QString filter(tr("Dominoes file (*.df)"));

    QString file = QFileDialog::getOpenFileName(this, caption, dir, filter);

    if (!file.size()) return;

    _sT.readPath(file);
}
```

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

To select the options.

```
{
    _sT.pause();
    ui->start->setText("Start");

    OptionsWindow o(_joy, &_sT, this);

    connect(this, SIGNAL(joystickChanged()), &o, SLOT(
joystickChanged()));

    if (o.exec()) {
        o.storeData();
        this->write();
    }
}
```

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

Handles the change of the mode.

```
{
    if (_sT.isActive()) {
        _sT.pause();
        ui->start->setText("Start");
    }
    if (ui->mode->text() == "Manual") {
        ui->mode->setText("Auto");
        _sT.setMode(Mode::Controlled);
    }
    else if (ui->mode->text() == "Auto") {
        ui->mode->setText("Manual");
        _sT.setMode(Mode::Manual);
    }
}
```

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

Handles a reset.

```
{
    _sT.reset();
}
```

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

Starts or stops the thread.

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

    if (text == "Start") {
        _sT.wakeUp();
        ui->start->setText("Stop");
    }
    else if (text == "Stop") {
        _sT.pause();
        ui->start->setText("Start");
    }
}
```

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

Reads the data from the default location.

```
{ read(_dataP); }
```

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

Reads the data from the selected path, overloaded function.

```
{
    QDir dir(path);
    _sT.read(dir.filePath("servo.opts"));
}
```

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

Updates all data to the servo thread.

```
{
    // Joystick values
    _joy.update();
    for (int i = 0; i < XJoystick::AxisCount; ++i) {
        float temp = _joy[i];
        _axisV[i] = temp;
        _axis[i]->setText(QString::number(temp));
    }
    for (int i = 0; i < XJoystick::ButtonCount; ++i) {
        bool temp = _joy.button(i);
        _butsV[i] = temp;
        _buts[i]->setEnabled(temp);
    }
    _sT.setData(_axisV, _butsV);

    QVector<ServoThread::Servo> servo = _sT.getServosInfo();
    QVector4D pos = _sT.getCurrentPos();
    QString x = QString::number(pos.x());
    QString y = QString::number(pos.y());
    QString z = QString::number(pos.z());
    QString rot = QString::number(pos.w());
    ui->pos->setText(x + "  " + y + "  " + z + "  " + rot + "°");

    // Updating position sliders
    ui->servo0S->setValue(servo[0].pos);
    ui->servo1S->setValue(servo[1].pos);
    ui->servo2S->setValue(servo[2].pos);
    ui->servo3S->setValue(servo[3].pos);

    // Updating position labels
    ui->servo0->setText(QString::number(servo[0].pos));
    ui->servo1->setText(QString::number(servo[1].pos));
    ui->servo2->setText(QString::number(servo[2].pos));
    ui->servo3->setText(QString::number(servo[3].pos));
}
```

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

Writes the data to the default location.

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

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

Writes the data to disk overloaded function.

```
{
    QDir dir(path);
    _sT.write(dir.filePath("servo.opts"));
}
```

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:

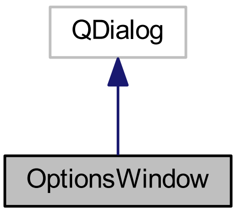
- mainwindow.h
- mainwindow.cpp

3.6 OptionsWindow Class Reference

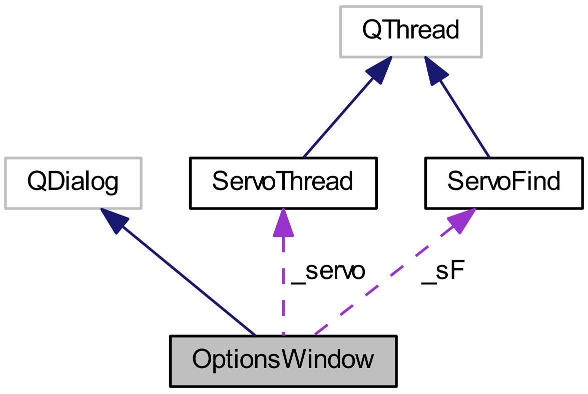
Class used to handle a Window to set the options.

`#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 intialized 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 continously.
- 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

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

```
                                                                           :
    QDialog(parent),
    _joy(J),
    _portSize(-1),
    _servo(servo),
    _timer(this),
    ui(new Ui::OptionsWindow)
{
    ui->setupUi(this);

    connect(ui->buttonBox, SIGNAL(clicked(QAbstractButton*)),
            this, SLOT(buttonClicked(QAbstractButton*)));

    connect(&_sF, SIGNAL(completion(int)),
            ui->progressBar, SLOT(setValue(int)));

    connect(&_sF, SIGNAL(finished()), this, SLOT(refreshFinish()));

    connect(&_timer, SIGNAL(timeout()), this, SLOT(events()));


    // Configuring event funcion
    _timer.setInterval(500);
    _timer.setSingleShot(false);
    _timer.start();

    status = new QStatusBar(this);
    status->setContentsMargins(0, 0, 0, 0);
    this->layout()->addWidget(status);

    QVector< QString > A(_joy.getAllAxis());

    // Updating joystick data
    joystickChanged();

    // Adding servos
    _servoC.push_back(ui->servo0);
    _servoC.push_back(ui->servo1);
    _servoC.push_back(ui->servo2);
    _servoC.push_back(ui->servo3);

    for(QComboBox *s : _servoC) s->addItem("None", -1);

    QVector<ServoThread::Servo> S(_servo->getServosInfo());
    Q_ASSERT(S.size() == _servo->getServosNum());

    for (int i = 0; i < S.size(); ++i) {
        int ID = S[i].ID;

        if (ID >= 0) {
            _servoC[i]->addItem(QString::number(ID), ID);
            _servoC[i]->setCurrentIndex(1);
        }
    }

    // Obtaining Servo Port information
    QString port;
    int baud;
    _servo->getServoPortInfo(port, baud);
    ui->speed->setValue(_servo->getSpeed());
    ui->baudRS->setValue(baud);
    ui->portS->addItem("", port);
}
```

3.6.3.2 OptionsWindow::∼OptionsWindow ( )

Destructor.

```
{
    delete ui;
    if (_sF.isRunning()) _sF.exit();
}
```

3.6.4 Member Function Documentation

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

Handles a button clicked.

```
{
    QDB::ButtonRole role = ui->buttonBox->buttonRole(but);
    switch(role) {
    case QDB::ApplyRole:
        this->storeData();
        break;

    default:
        break;
    }
}
```

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

Handles events that need to be updated continously.

```
{
    auto ports = QSerialPortInfo::availablePorts();
    ui->portN->setText(QString::number(ports.size()));

    if (ports.size() != _portSize) {
        _portSize = ports.size();

        QString portC(ui->portC->currentData().toString());
        QString portS(ui->portS->currentData().toString());

        int selC = 0, selS = 0;

        ui->portC->clear();
        ui->portS->clear();

        ui->portC->addItem("None", "");
        ui->portS->addItem("None", "");

        for (int i = 0; i < ports.size(); ++i) {
            QString text(ports[i].portName());
            text += ": " + ports[i].description();
            ui->portC->addItem(text, ports[i].portName());
            ui->portS->addItem(text, ports[i].portName());
            if (ports[i].portName() == portC) selC = i + 1;
            if (ports[i].portName() == portS) selS = i + 1;
        }

        if (selS == 0 && ports.size() > 0) selS = 1;

        ui->portC->setCurrentIndex(selC);
        ui->portS->setCurrentIndex(selS);
    }
}
```

3.6.4.3 void OptionsWindow::joystickChanged ( ) [slot]

To handle the change of a joystick.

```
{
    // Clear all the items and write the new items
    ui->joySel->clear();
    ui->joySel->addItem("None", -1);

    // Adding items and searching the current
    int pos = 0;
    QVector<XJoystick::Info> V(_joy.available());
    for (int i = 0; i < V.size(); ++i) {
        QString text(V[i].name);
        text += ": " + QString::number(V[i].ID);
        if (V[i].ID == _joy.current()) pos = i;
        ui->joySel->addItem(text, V[i].ID);
    }
    ui->joySel->setCurrentIndex(pos);

    ui->joyN->setText(QString::number(V.size()));
}
```

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

```
{
    if (event->key() == Qt::Key_Enter || event->key() == Qt::Key_Return) return;
    QDialog::keyPressEvent(event);
}
```

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

Refreshes all the servos connected to the port.

```
{
    if (_sF.isRunning()) return;
    QString port;
    int baud;
    _servo->getServoPortInfo(port, baud);
    int min = ui->min->value();
    int max = ui->max->value();
    _sF.setData(_servoC, port, baud, min, max);
    _sF.start();
}
```

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

Handles the endig of refresh function.

```
{
    ui->progressBar->setValue(0);
}
```

3.6.4.7 void OptionsWindow::storeData ( )

Stores all data.

```
{
    status->showMessage("Data Stored", 2000);

    // Storing joystick data
    _joy.select(ui->joySel->currentData().toInt());

    QString portS(ui->portS->currentData().toString());
    int baudS(ui->baudRS->value());
    _servo->setServoPortInfo(portS, baudS);

    QVector<int> sID;
    for (QComboBox *s : _servoC) sID.push_back(s->currentData().toInt());

    _servo->setSID(sID);
    _servo->setSpeed(ui->speed->value());
}
```

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.

```
        {
            this->ID = s.ID;
            this->pos = s.pos;
        }
```

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.

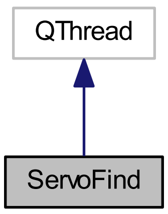
The documentation for this struct was generated from the following file:

- servothread.h

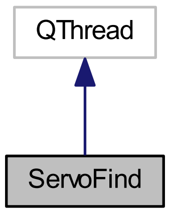
3.8 ServoFind Class Reference

`#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.

```
{

}
```

3.8.2.2 ServoFind::∼ServoFind ( )

Default destructor.

```
{

}
```

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.

```
{
    QVector<int> data(_servo.size());

    for (int i = 0; i < data.size(); ++i)
        data[i] = _servo[i]->currentData().toInt();

    for (QCB *s : _servo) {
        s->clear();
        s->addItem("None", -1);
    }

    int index = 1;
    QVector<int> pos(_servo.size(), 0);

    dynamixel dxl(_port, _baud);

    for (int i = _min; i < _max; ++i) {
        dxl.ping(i);
        emit completion(((i - _min)/double(_max - _min))*100.0);
        if (dxl.get_comm_result() == COMM_RXSUCCESS) {

            for (int j = 0; j < _servo.size(); ++j) {
                if (data[j] == i) pos[j] = index;
                _servo[j]->addItem(QString::number(i), i);
            }

            ++index;
        }
    }

    for (int i = 0; i < _servo.size(); ++i) _servo[i]->setCurrentIndex(pos[i]);
}
```

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.

```
{
    if (this->isRunning()) return;
    _servo = servo;
    _port = port;
    _baud = baud;

    if (min > max) {
        int aux = min;
        min = max;
        max = aux;
    }

    if (min < 0) min = 0;
    if (max > MAX_ID) max = MAX_ID;


    _min = min;
    _max = max;
}
```

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:

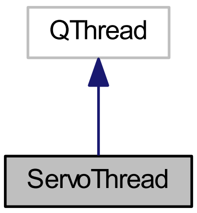
- servofind.h
- servofind.cpp

3.9 ServoThread Class Reference

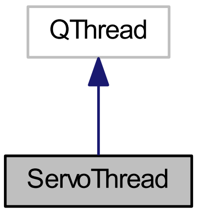
The ServoThread's class handles the comunication between the delta robot servos and the PC.

`#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 > &buts)
  Adds the loaded data.
- void setServoBaud (unsigned int baud)
  Sets the servos port baud rate.
- void setServoPort (QString &port)
  Sets the servos port.
- void 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 comunicate 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 comunicate with the servos.
- QVector< Servo > _servos

  Contains the servos information.
- QString _sPort

  Contains the selected com port used in the comunication 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 comunication 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

```
    {
        Controlled,
        Manual,
        Reset
    };
```

3.9.2.2 enum ServoThread::Status [private]

Contains the available status for the Controlled mode.

Enumerator

begin

take

waiting

rotate

going

ending

```
                {
        begin,
        take,
        waiting,
        rotate,
        going,
        ending
    };
```

3.9.2.3 enum ServoThread::Version [private]

Enum containing all the save file versions.

Enumerator

v_1_0

```
    {
        v_1_0
    };
```

3.9.3 Constructor & Destructor Documentation

3.9.3.1 ServoThread::ServoThread ( )

Default constructor.

```
                        :
    _axis(0, 0, 0, 0),
    _buts(XJoystick::ButtonCount),
    _cBaud(9600),
    _cPort("COM3"),
    _dChanged(true),
    _end(false),
    _mod(Mode::Manual),
    _pause(true),
    _sBaud(1000000),
    _servos(_sNum),
    _sPort("COM9"),
    _sPortChanged(false),
    _sSpeed(100),
    _status(Status::begin)
{
    for (Servo &s : _servos) s.ID = -1;
}
```

3.9.3.2 ServoThread::∼ServoThread ( )

Default destructor.

```
{
    _mutex.lock();
    _end = true;
    _cond.wakeOne();
    _mutex.unlock();
    wait();
}
```

3.9.4 Member Function Documentation

3.9.4.1 void ServoThread::end ( ) [inline]

Ends the execution.

```
    {
        _mutex.lock();
        _end = true;
        _cond.wakeOne();
        _mutex.unlock();

        wait();
    }
```

3.9.4.2 QVector4D ServoThread::getCurrentPos ( ) [inline]

Returns the current position.

```
    {
        QMutexLocker m(&_mutex);
        return _pos;
    }
```

3.9.4.3 int ServoThread::getServoBaud ( ) [inline]

Returns the current servo Baud rate.

```
    {
        QMutexLocker mL(&_mutex);
        return _sBaud;
    }
```

3.9.4.4 QString ServoThread::getServoPort ( ) [inline]

Returns the current servo Port.

```
    {
        QMutexLocker mL(&_mutex);
        return _sPort;
    }
```

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

Returns both servo Port and baud Rate.

```
    {
        _mutex.lock();
        baud = _sBaud;
        port = _sPort;
        _mutex.unlock();
    }
```

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

```
    {
        _mutex.lock();
        V = _servos;
        _mutex.unlock();
    }
```

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

Overloaded function to get the servo info.

```
    {
        QMutexLocker mL(&_mutex);
        return _servos;
    }
```

3.9.4.8 int ServoThread::getServosNum ( ) [inline]

Returns the number of servos to handle.

```
{ return _sNum; }
```

3.9.4.9 int ServoThread::getSpeed ( ) [inline]

Returns the current speed.

```
    {
        QMutexLocker m(&_mutex);
        return _sSpeed;
    }
```

3.9.4.10 bool ServoThread::isActive ( ) [inline]

Returns true if the servos are active.

```
    {
        QMutexLocker m(&_mutex);
        return not _pause;
    }
```

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

Returns true if the position is available.

```
{
    if (newPos.toVector2D().lengthSquared() > workRadSq) return false;

    QVector<double> D(4);
    this->setAngles(newPos, D);

    for (int i = 0; i < 3; ++i) {
        if (qIsNaN(D[i])) return false;
        if (D[i] > maxAngle or D[i] < minAngle) return false;
    }

    return true;
}
```

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

```
{
    QVector<double> D(4);
    this->setAngles(pos, D);

    for (int i = 0; i < 3; ++i) if (abs(S[i] - D[i]) > err) return false;
    return true;
}
```

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.

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

3.9.4.15 void ServoThread::pause ( ) [inline]

Pauses the execution.

```
    {
        _mutex.lock();
        _pause = true;
        _mutex.unlock();
    }
```

3.9.4.16 void ServoThread::read (QString *file*)

Reads and loads the data from the selected file.

Parameters

file	Path to the selected file

```
{
    // Opening file for reading
    QFile f(file);
    if (!f.open(QIODevice::ReadOnly)) {
        emit statusBar("Cannot read stored data", 2000);
        return;
    }
    QDataStream df(&f);

    QMutexLocker mL(&_mutex);

    int version;
    df >> version;
    if (version != Version::v_1_0) {
        emit statusBar("Error opening file", 2000);
        return;
    }

    df >> _cBaud >> _cPort >> _sBaud >> _sPort >>
```

```
      _sSpeed;
    unsigned int en;
    df >> en;
    _mod = static_cast<Mode>(en);

    int size;
    df >> size;
    _servos.resize(size);
    for (Servo &s : _servos) df >> s.ID;
    _dChanged = true;

}
```

3.9.4.17 void ServoThread::readPath (QString *file*)

Reads the path where to put the selected pieces.

Parameters

file	Path to the file where to read the pieces

```
{
    // Opening file for reading
    QFile f(file);
    if (!f.open(QIODevice::ReadOnly)) {
        emit statusBar("Error opening file", 2000);
        return;
    }

    QTextStream pF(&f);

    int size;
    pF >> size;
    QVector<Dominoe> temp(size);
    for (Dominoe &d : temp) pF >> d.X >> d.Y >> d.ori;
    std::sort(temp.begin(), temp.end());

    _mutex.lock();
    double sep = 0.8; // 2cm of separation
    QVector2D ori(posStart.toVector2D());

    _dominoe.clear();
    for (int i = 0; i < temp.size(); ++i) {
        QVector2D aux(temp[i].X, temp[i].Y);

        // Checking if its a vàlid position
        QVector4D aux2(aux);
        aux2[2] = workHeigh;
        if (not this->isPosAvailable(aux2)) continue;

        double angle = temp[i].ori + 60.0;
        if (angle >= 180.0) angle -= 180.0;
        else if (angle >= 360.0) angle -= 360.0;

        // Calculating direction vector
        aux -= ori;
        int steps = aux.length()/sep;

        QVector<Dominoe> V(steps + 1);

        // Initial point
        V[0] = Dominoe(ori, angle);

        // Adding intermediate positions
        for (int j = 1; j <= steps; ++j){
            Dominoe dAux(j*aux/double(steps) + ori, angle);
            V[j] = dAux;
        }
        _dominoe.push_back(V);
    }
    _dChanged = true;
    _mutex.unlock();

    f.close();

    emit statusBar("File loaded succesfully", 1000);
}
```

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

```
    {
        _mutex.lock();
        _mod = Mode::Reset;
        _mutex.unlock();
    }
```

3.9.4.19 void ServoThread::run ( ) [private]

Used to create another thread.

```
{
    // First initializations
    _mutex.lock();
    int sBaud = _sBaud;
    QString sPort = _sPort;
    _mutex.unlock();

    // Serial port interface
    dynamixel dxl(sPort, sBaud);

    // Contains the servos comunication
    QVector<AX12> A(4);

    // Contains the servos ID
    QVector< int > ID(_sNum);

    // Contains the current servo data
    QVector< double > S(_sNum);

    // Contains the servos angles
    QVector<double> D(4);
    D[3] = 150.0;

    // First initialization
    _mutex.lock();
    for (int i = 0; i < A.size(); ++i) {
        A[i] = AX12(&dxl);
        A[i].setID(_servos[i].ID);
        ID[i] = _servos[i].ID;
        A[i].setSpeed(_sSpeed);
        A[i].setComplianceSlope(ccwCS, cwCS);
    }
    _mutex.unlock();

    QVector4D pos(posIdle);
    QVector4D axis(0, 0, 0, 0);
    QVector< bool > buts;

    // Contains the domino number to put
    int dom = 0;
    int pas = 0;
    double speed = 100.0;
    QVector< QVector< Dominoe > > Dom;

    // Main while
    while (not _end) {

        // Pause
        _mutex.lock();
        if (not _end and _pause) {
            dxl.terminate();

            // Thread pause
            _cond.wait(&_mutex);

            if (_end) exit(0);
            dxl.initialize(sPort, sBaud);
        }
        _mutex.unlock();

        // Get current servo position
        for (int i = 0; i < 3; ++i) S[i] = A[i].getCurrentPos();
        if (_mod == Mode::Manual) S[3] = A[3].getCurrentPos();


        /*********** MUTEX ***********/
        // Handling changes of data
        _mutex.lock();
        if (_dChanged) {
            if (sPort != _sPort or sBaud != _sBaud) {
                sPort = _sPort;
                sBaud = _sBaud;
                dxl.terminate();
                dxl.initialize(sPort, sBaud);
            }
            for (int i = 0; i < S.size(); ++i) {
                A[i].setID(_servos[i].ID);
                ID[i] = _servos[i].ID;
                A[i].setSpeed(_sSpeed);
                A[i].setComplianceSlope(ccwCS, cwCS);
            }

            speed = _sSpeed;
            Dom = _dominoe;
            dom = 0;
            pas = 0;
            pos = posIdle;
            this->setAngles(pos, D);
            this->setGoalPosition(ID, D, dxl);
            _dChanged = false;
        }

        // Joystick and buttons update, must use mutex
        for (int i = 0; i < _sNum; ++i) _servos[i].pos =  S[i];
        axis = _axis;
        buts = _buts;
        for (bool &b : _buts) b = 0;
        _pos = pos;
        _mutex.unlock();


        /******** MODE ********/
        // Main function with data updated

        if (_mod == Mode::Manual) {
            QVector4D posAux = pos + 0.5*axis;

            bool ok = this->isPosAvailable(posAux);
            ok &= this->isReady(S, pos, maxErr + 5.0);
            if (ok) pos = posAux;
        }
        else if (_mod == Mode::Controlled) {
            switch(_status) {
            case Status::begin:
                pos = posStart;
                if (this->isReady(S, pos, maxErr))   {
                    _status = Status::take;
                    QThread::msleep(500);
                }
                break;

            case Status::take:
                pos[2] = workHeigh;
                if (this->isReady(S, pos, maxErr)) {
                    emit statusBar("Esperant peça", -1);
                    _status = Status::waiting;
                }
                break;

            case Status::waiting:
                if (buts[0]) {
                    pas = 0;
                    _status = Status::rotate;
                    emit statusBar("Girant!", -1);

                    for (AX12 &a : A) a.setSpeed(speed/3.5);
                }
                else break;

            case Status::rotate:
            {
                S[3] = A[3].getCurrentPos();
                pos[3] = Dom[dom][0].ori;
                double aux = abs(S[3] - Dom[dom][0].ori);
                if (aux < maxErr) {
                    _status = Status::going;
                    QThread::msleep(1000);
                    emit statusBar("Posicionant", -1);
                }
            }
                break;

            case Status::going:
            {
                Dominoe &domi = Dom[dom][pas];
                pos = QVector4D (domi.X, domi.Y, workHeigh, domi.ori);
                double err;
                pas == Dom[dom].size() - 1 ? err = maxErr : err = 2*maxErr;

                if (this->isReady(S, pos, err)) {
                    ++pas;
                    if (pas == Dom[dom].size()) {
                        pas = 0;
                        _status = Status::ending;
                        QThread::msleep(1000);
                        emit statusBar("Col·locada", 1500);

                        for (AX12 &a : A) a.setSpeed(speed);
                    }
                }
            }
                break;

            case Status::ending:
                pos[2] = descHeigh[pas];

                if (this->isReady(S, pos, maxErr)) {
                    ++pas;
                    if (pas == 3) {
                        _status = Status::begin;
                        if (dom == Dom.size() - 1) {
                            dom = 0;
                            pas = 0;
                            _mod = Mode::Reset;
                        }
                        else ++dom;
                    }
                }
                break;

            default:
                _status = Status::begin;

            }
        }
        else if (_mod == Mode::Reset) {
            _mod = Mode::Manual;
            pos = posIdle;
            dom = 0;
        }

        this->setAngles(pos, D);
        this->setGoalPosition(ID, D, dxl);
    }
    dxl.terminate();
    exit(0);
}
```

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

Used to calculate the servos angles.

```
{
    double x1 = pos.x() + L2 - L1;
    double y1 = -pos.z();
    double z1 = pos.y();
    D[0] = singleAngle(x1,y1,z1);

    double x2 = pos.y()*sin60 - pos.x()*cos60 + L2 - L1;
    double y2 = -pos.z();
    double z2 = -pos.y()*cos60 - pos.x()*sin60;
    D[1] = singleAngle(x2,y2,z2);

    double x3 = -pos.y()*sin60 - pos.x()*cos60 + L2 - L1;
    double y3 = -pos.z();
    double z3 = -pos.y()*cos60 + pos.x()*sin60;
    D[2] = singleAngle(x3,y3,z3);

    for (double &d : D) d = 150.0 - d*180/M_PI;
    D[3] = pos.w();
}
```

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

Adds the loaded data.

Parameters

aV	Contains the axis values
buts	Contains the buttons values

```
{
    _mutex.lock();
    // Copying the joystick values
    _axis = QVector4D(aV[0], aV[1], aV[2], aV[3]);
    _axis.normalize();
    _axis[3] *= 5;
    for (int i = 0; i < buts.size(); ++i) _buts[i] |= buts[i];
    _mutex.unlock();
}
```

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

```
{
    dxl.set_txpacket_id(BROADCAST_ID);
    dxl.set_txpacket_instruction(INST_SYNC_WRITE);
    dxl.set_txpacket_parameter(0, AX12::RAM::GoalPosition);
    dxl.set_txpacket_parameter(1, 2);

    Q_ASSERT(ID.size() == pos.size());
    for (int i = 0; i < ID.size(); ++i) {
        unsigned int data = (pos[i]/300.0)*1023.0;

        dxl.set_txpacket_parameter(2 + 3*i, ID[i]);
        dxl.set_txpacket_parameter(2 + 3*i + 1, LOBYTE(data));
        dxl.set_txpacket_parameter(2 + 3*i + 2, HIBYTE(data));
    }
    dxl.set_txpacket_length(4 + 3*ID.size());
    dxl.txrx_packet();
}
```

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

Sets the current working mode.

Precondition

The thread must be on pause

Parameters

m	Contains the desired working mode

```
    {
        QMutexLocker mut(&_mutex);
        if (!_pause) return;
        _mod = m;
        _dChanged = true;
    }
```

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

Sets the servos port baud rate.

Parameters

baud	Positive number containing the baud rate

```
    {
        _mutex.lock();
        _sBaud = baud;
        _dChanged = true;
        _mutex.unlock();
    }
```

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

Sets the servos port.

Parameters

port	String containing the port name

```
    {
        _mutex.lock();
        _sPort = port;
        _dChanged = true;
        _mutex.unlock();
    }
```

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

Sets the servos port info, data and selected port.

Parameters

port	String containing the selected port
baud	Contains the selected baud rate

```
    {
        _mutex.lock();
        _sPort = port;
        _sBaud = baud;
        _dChanged = true;
        _mutex.unlock();
    }
```

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

Sets the servos ID.

Parameters

V	Vector containing all the servos ID

```
    {
        // Error passing the data
        if (V.size() != _sNum) {
            qDebug() << "Error setting servos";
            return;
        }


        _mutex.lock();
        for (int i = 0; i < V.size(); ++i) _servos[i].ID = V[i];
        _dChanged = true;
        _mutex.unlock();
    }
```

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

Sets the servos speed.

Parameters

speed	unsigned char from 0 to 100 containing the % of speed

```
    {
        if (speed > 100) speed = 100;

        _mutex.lock();
        _sSpeed = speed;
        _dChanged = true;
        _mutex.unlock();
    }
```

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

Calculates the angle of one servo in the selected position.

```
{
    double n = b*b - a*a - z0*z0 - x0*x0 - y0*y0;
    double raiz = sqrt (n*n*y0*y0 - 4*(x0*x0 + y0*y0)*(-x0*x0*a*a + n*n/4));

    if (x0 < 0) raiz *= -1;
    double y = (-n*y0 + raiz ) / (2*(x0*x0 + y0*y0));

    int signe = 1;
    if ((b*b - (y0 + a)*(y0 + a)) < (x0*x0 + z0*z0) && x0 < 0) signe *= -1;
    double x = sqrt(a*a - y*y)*signe;
    return atan2 (y,x);
}
```

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.

```
    {
        _mutex.lock();
        _pause = false;
        _cond.wakeOne();
        _mutex.unlock();
    }
```

3.9.4.32 void ServoThread::write (QString *file*)

Writes data to the selected directory.

Parameters

file	Path to the file

```
{
    // Opening file for writing
    QFile f(file);
    f.open(QIODevice::WriteOnly);
    QDataStream df(&f);

    _mutex.lock();

    // Clamp and servos baud rate and port must be writen
    df << int(Version::v_1_0) << _cBaud << _cPort << _sBaud <<
_sPort << _sSpeed
       << int(_mod) << _servos.size();
    for (const Servo &s : _servos) df << s.ID;

    _mutex.unlock();
}
```

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 comunicate 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 comunitate 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 comunicate 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 comunication 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[ ]*)

```
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}
```

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 comunication 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