DeltaRobot

v0.4

Generated by Doxygen 1.8.9.1

Sun Apr 26 2015 20:14:01

Contents

Chapter 1

Main Page

This project is a Delta robot controller using Dynamixel AX12 servos.This type of robot can pick and place objects

Chapter 2

Namespace Documentation

2.1 Ui Namespace Reference

Namespace to work with a User Interface Qt Form.

2.1.1 Detailed Description

Namespace to work with a User Interface Qt Form.

Chapter 3

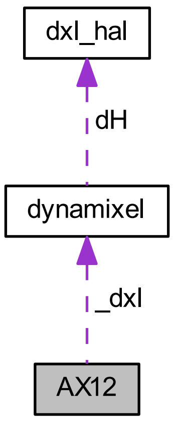
Class Documentation

3.1 AX12 Class Reference

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

`#include <ax12.h>`

Collaboration diagram for AX12:



Public Member Functions

- AX12 ()

  Default constructor.
- AX12 (dynamixel ∗_dxl, int ID=-1)

  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 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 setSpeed (double speed)

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

Private Types

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

  Contains all the EEPROM directions enumeration.
- enum RAM {
  TorqueEnable = 24, LED = 25, CWComplianceMargin = 26, CCWComplianceMargin = 27,
  CWComplianceSlope = 28, CCWComplianceSlope = 29, GoalPosition = 30, MovingSpeed = 32,
  TorqueLimit = 34, PresentPosition = 36, PresentSpeed = 38, PresentLoad = 40,
  PresentVoltage = 42, PresentTemperature = 43, Registered = 44, Moving = 46,
  Lock = 47, Punch = 48 }

  Contains all the RAM directions enumerations.

Private Attributes

- dynamixel ∗ _dxl

  Contains the dynamixel 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 [private]

Contains all the RAM directions enumerations.

Enumerator

- ***TorqueEnable***
- ***LED***
- ***CWComplianceMargin***
- ***CCWComplianceMargin***
- ***CWComplianceSlope***
- ***CCWComplianceSlope***
- ***GoalPosition***
- ***MovingSpeed***
- ***TorqueLimit***
- ***PresentPosition***
- ***PresentSpeed***
- ***PresentLoad***
- ***PresentVoltage***
- ***PresentTemperature***
- ***Registered***
- ***Moving***
- ***Lock***
- ***Punch***

```
    {
        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 [private]

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);
    load -= 1024;
    if (load == -1024) load = 0;
    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*M_PI/3);
    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::setDxl (**dynamixel** * *dxl*) [inline]

Sets the dynamixel interface.

Parameters

dxl	Pointer to the dynamixel control class

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

3.1.4.9 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;
    if (goal > 300.0) goal = 300.0;
    else if (goal < 0) goal = 0;
    _dxl->write_word(_ID, RAM::GoalPosition, int((goal/300.0)*1023));
}
```

3.1.4.10 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);
}
```

3.1.4.11 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);
    }
}
```

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

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

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

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

    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));
}
```

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

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

```
{
    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 Attributes

- double X

  X position.
- double Y

  Y position.
- double ori

  Orientation from X = 0 in degrees.

3.2.1 Detailed Description

Struct to handle the dominoe pieces.

3.2.2 Member Data Documentation

3.2.2.1 double ServoThread::Dominoe::ori

Orientation from X = 0 in degrees.

3.2.2.2 double ServoThread::Dominoe::X

X position.

3.2.2.3 double ServoThread::Dominoe::Y

Y position.

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

- servothread.h

3.3 dxl_hal Class Reference

Dynamixel SDK platform dependent.

`#include <dxl_hal.h>`

Public Member Functions

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

Private Attributes

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

3.3.1 Detailed Description

Dynamixel SDK platform dependent.

3.3.2 Member Function Documentation

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

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

}
```

3.3.2.2 void dxl_hal::clear (void)

```
{
    // Clear communication buffer
    _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:

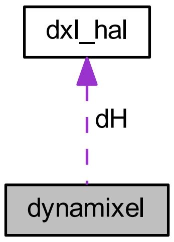
- dxl/dxl_hal.h
- dxl/dxl_hal.cpp

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.
- 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);
}
```

3.4.3 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:

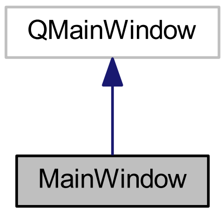
- dxl/dynamixel.h
- dxl/dynamixel.cpp

3.5 MainWindow Class Reference

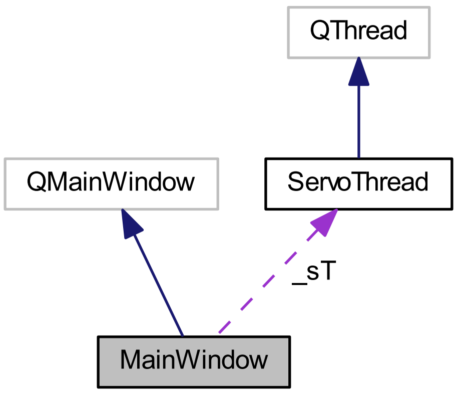
Contains all the windows and other classes.

`#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 }

Private Slots

- void joyChanged ()

  Handles a joystick update.
- void on_actionOptions_triggered ()

  To select the options.
- void update ()

  Updates all data to the servo thread.
- void on_start_clicked ()

Private Member Functions

- void write ()

  Writes the data to the default location.
- void write (QString path)

  Writes the data to disk overloaded function.

Private Attributes

- QVector< QLabel ∗ > _axis

  Handles all the axis labels.
- QVector< float > _axisV

  Contains the axis value;.
- QVector< QLabel ∗ > _buts

  Handles all the button labels.
- QVector< bool > _butsV

  Handles all buttons values.
- QString _dataP

  Contains the path to the data location.
- int _jAxisX = -1

  Axis for the X value.
- int _jAxisY = -1

  Axis for the Y value.
- int _jAxisZ = -1

AXis for the Z value.

- XJoystick _joy

  To handle the joystick.
- ServoThread _sT

  Contains the thread controlling all the servos and external hardware.
- QTimer _timer

  To update the joystick value.
- Ui::MainWindow ∗ ui

  Contains the user interface.

Static Private Attributes

- static const int sCount = 3

  Contains the number of minimun servos to work.
- static const int aSCount = 0

  Contains the number of additional servos used.

3.5.1 Detailed Description

Contains all the windows and other classes.

3.5.2 Member Enumeration Documentation

3.5.2.1 enum MainWindow::Version [private]

Enumerator

v_1_0

```
                 {
        v_1_0
    };
```

3.5.3 Constructor & Destructor Documentation

3.5.3.1 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);

    _sT.start();

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


    _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);
}
```

3.5.3.2 MainWindow::∼MainWindow ( )

Default destructor.

```
{
    delete ui;
}
```

3.5.4 Member Function Documentation

3.5.4.1 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();
}
```

3.5.4.2 void MainWindow::joystickChanged ( ) [signal]

Emmitted when a joystick changes.

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

To select the options.

```
{
    OptionsWindow o(_joy, &_sT, _jAxisX, _jAxisY,
  _jAxisZ, this);
    o.exec();

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

    if (o.result()) o.storeData();
}
```

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

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

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

3.5.4.5 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());

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

    // Updating load labels
    ui->servo0L->setText(QString::number(servo[0].load));
    ui->servo1L->setText(QString::number(servo[1].load));
    ui->servo2L->setText(QString::number(servo[2].load));

}
```

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

Writes the data to the default location.

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

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

Writes the data to disk overloaded function.

```
{
    QDir dir(path);
    QFile file(dir.filePath("main.opts"));
    if(not file.open(QIODevice::WriteOnly)) {
        ui->statusbar->showMessage("Error saving file", 1000);
        return;
    }

    QDataStream f(&file);
    f << int(Version::v_1_0) << _jAxisX << _jAxisY << _jAxisZ;

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

3.5.5 Member Data Documentation

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

Handles all the axis labels.

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

Contains the axis value;.

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

Handles all the button labels.

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

Handles all buttons values.

3.5.5.5 QString MainWindow::_dataP [private]

Contains the path to the data location.

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

Axis for the X value.

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

Axis for the Y value.

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

AXis for the Z value.

3.5.5.9 XJoystick MainWindow::_joy `[private]`

To handle the joystick.

3.5.5.10 ServoThread MainWindow::_sT `[private]`

Contains the thread controlling all the servos and external hardware.

3.5.5.11 QTimer MainWindow::_timer `[private]`

To update the joystick value.

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

Contains the number of additional servos used.

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

Contains the number of minimun servos to work.

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

Contains the user interface.

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

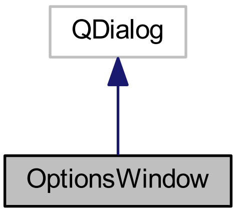
- mainwindow.h
- mainwindow.cpp

3.6 OptionsWindow Class Reference

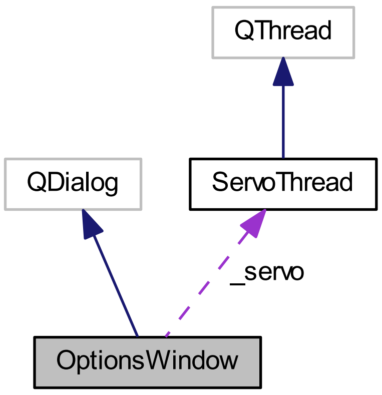
Class used to handle a Window to set the options.

`#include <optionswindow.h>`

Inheritance diagram for OptionsWindow:



Collaboration diagram for OptionsWindow:



Public Slots

- void joystickChanged ()

  To handle the change of a joystick.

Public Member Functions

- OptionsWindow (XJoystick &J, ServoThread ∗servo, int &aX, int &aY, int &aZ, QWidget ∗parent=0)

  Default constructor must be intialized with a few values.
- ∼OptionsWindow ()

  Destructor.
- void storeData ()

  Stores all data.

Private Slots

- void events ()

  Handles events that need to be updated continously.
- void on_servoRefresh_clicked ()

  Refreshes all the servos connected to the port.

Private Attributes

- int & _jAxisX

  Reference to axis for the X value.
- int & _jAxisY

  Reference to axis for the Y value.
- int & _jAxisZ

  Reference to axis for the Z value.
- XJoystick & _joy

  Contains the Joystick to handle options.
- int _portSize

  Contains the size of the ports.
- ServoThread ∗ _servo

  Pointer to the servo thread class.
- QTimer _timer

  Waits for a new COM port.
- Ui::OptionsWindow ∗ ui

  Containsh the GUI.

3.6.1 Detailed Description

Class used to handle a Window to set the options.

3.6.2 Constructor & Destructor Documentation

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

Default constructor must be intialized with a few values.

Parameters

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),
    _jAxisX(aX),
    _jAxisY(aY),
    _jAxisZ(aZ),
    _joy(J),
    _portSize(-1),
    _servo(servo),
    _timer(this),
    ui(new Ui::OptionsWindow)
{
    ui->setupUi(this);

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

    // Adding joystick axis movement
    ui->joyMX->addItem("None", -1);
    ui->joyMY->addItem("None", -1);
    ui->joyMZ->addItem("None", -1);

    for (int i = 0; i < A.size(); ++i) ui->joyMX->addItem(A[i], i);
    for (int i = 0; i < A.size(); ++i) ui->joyMY->addItem(A[i], i);
    for (int i = 0; i < A.size(); ++i) ui->joyMZ->addItem(A[i], i);

    ui->joyMX->setCurrentIndex(_jAxisX + 1);
    ui->joyMY->setCurrentIndex(_jAxisY + 1);
    ui->joyMZ->setCurrentIndex(_jAxisZ + 1);

    // Updating joystick data
    joystickChanged();

    // Configuring event funcion
    _timer.setInterval(500);
    _timer.setSingleShot(false);
    _timer.start();
    connect(&_timer, SIGNAL(timeout()), this, SLOT(events()));

    // Adding servos
    ui->servo0->addItem("None", -1);
    ui->servo1->addItem("None", -1);
    ui->servo2->addItem("None", -1);
    ui->servo3->addItem("None", -1);

    this->on_servoRefresh_clicked();
}
```

3.6.2.2 OptionsWindow::∼OptionsWindow ( )

Destructor.

```
{
    delete ui;
}
```

3.6.3 Member Function Documentation

3.6.3.1 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;
        }

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

3.6.3.2 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()));
}
```

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

Refreshes all the servos connected to the port.

```
{

    QString port;
    int baud;
    _servo->getServoPortInfo(port, baud);
    dynamixel dxl(port, baud);

    int s0 = ui->servo0->currentData().toInt();
    int s1 = ui->servo1->currentData().toInt();
    int s2 = ui->servo2->currentData().toInt();
    int s3 = ui->servo3->currentData().toInt();

    ui->servo0->clear();
    ui->servo1->clear();
    ui->servo2->clear();
    ui->servo3->clear();

    int index = 0;
    int p0 = 0, p1 = 0, p2 = 0, p3 = 0;

    for (int i = 0; i <= MAX_ID; ++i) {
        dxl.ping(i);
        if (dxl.get_comm_result() == COMM_RXSUCCESS) {
            if (i == s0) p0 = index;
            if (i == s1) p1 = index;
            if (i == s2) p2 = index;
            if (i == s3) p3 = index;

            ui->servo0->addItem(QString::number(i), i);
            ui->servo1->addItem(QString::number(i), i);
            ui->servo2->addItem(QString::number(i), i);
            ui->servo3->addItem(QString::number(i), i);

            ++index;
        }
    }

    ui->servo0->setCurrentIndex(p0);
    ui->servo1->setCurrentIndex(p1);
    ui->servo2->setCurrentIndex(p2);
    ui->servo3->setCurrentIndex(p3);

}
```

3.6.3.4 void OptionsWindow::storeData ( )

Stores all data.

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

    // Joystick movement axis
    _jAxisX = ui->joyMX->currentData().toInt();
    _jAxisY = ui->joyMY->currentData().toInt();
    _jAxisZ = ui->joyMZ->currentData().toInt();

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

    QVector<int> sID;
    if (ui->servo0->count()) sID.push_back(ui->servo0->currentData().toInt());
    if (ui->servo1->count()) sID.push_back(ui->servo1->currentData().toInt());
    if (ui->servo2->count()) sID.push_back(ui->servo2->currentData().toInt());
    if (ui->servo3->count()) sID.push_back(ui->servo3->currentData().toInt());
}
```

3.6.4 Member Data Documentation

3.6.4.1 int& OptionsWindow::_jAxisX [private]

Reference to axis for the X value.

3.6.4.2 int& OptionsWindow::_jAxisY [private]

Reference to axis for the Y value.

3.6.4.3 int& OptionsWindow::_jAxisZ [private]

Reference to axis for the Z value.

3.6.4.4 XJoystick& OptionsWindow::_joy [private]

Contains the Joystick to handle options.

3.6.4.5 int OptionsWindow::_portSize [private]

Contains the size of the ports.

3.6.4.6 ServoThread* OptionsWindow::_servo [private]

Pointer to the servo thread class.

3.6.4.7 QTimer OptionsWindow::_timer [private]

Waits for a new COM port.

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

Containsh the GUI.

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

- optionswindow.h
- optionswindow.cpp

3.7 ServoThread::Servo Struct Reference

Struct for the AX12 servos.

`#include <servothread.h>`

Public Member Functions

- Servo (int ID=-1, double load=-1, double pos=-1)

  Default constructor.
- Servo (const Servo &s)

  Copy constructor.

Public Attributes

- int ID

  Contains the servo ID.
- double load

  Contains the servo load.
- double pos

  Contains the servo position.

3.7.1 Detailed Description

Struct for the AX12 servos.

3.7.2 Constructor & Destructor Documentation

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

Default constructor.

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

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

Copy constructor.

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

3.7.3 Member Data Documentation

3.7.3.1 int ServoThread::Servo::ID

Contains the servo ID.

3.7.3.2 double ServoThread::Servo::load

Contains the servo load.

3.7.3.3 double ServoThread::Servo::pos

Contains the servo position.

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

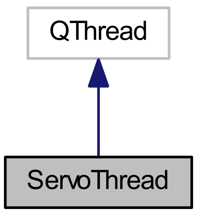
- servothread.h

3.8 ServoThread Class Reference

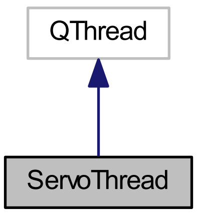
The ServoThread's class handles the comunication between the delta robot servos and the PC.

`#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 }

  Contains the working mode.

Signals

- void statusBar (QString)

  Emmitted when the status bar must be changed.

Public Member Functions

- ServoThread ()

  Default constructor.
- ∼ServoThread ()

  Default destructor.
- void end ()

  Ends the execution.
- void load (QString &file)

  Loads the data from the selected file.
- void pause ()

  Pauses the execution.
- int getServoBaud ()

  Returns the current servo Baud rate.
- QString getServoPort ()

  Returns the current servo Port.
- void getServoPortInfo (QString &port, int &baud)

  Returns both servo Port and baud Rate.
- void getServosInfo (QVector< Servo > &V)

  Returns the servos info, with all its load and current position.
- QVector< Servo > getServosInfo ()

  Overloaded function to get the servo info.
- QMutex ∗ mutex ()

  Returns the mutex used in the thread.
- void read (QString file)

  Reads and loads the data from the selected file.
- void readPath (QString file)

  Reads the path where to put the selected pieces.
- void setData (QVector< float > &aV, QVector< bool > &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 (int speed)

  Sets the servos speed.
- void wakeUp ()

  Continues program's execution.
- void write (QString file)

  Writes data to the selected directory.

Private Types

- enum Version { v_1_0 }

  Enum containing all the save file versions.

Private Member Functions

- void run ()

  Used to create another thread.
- void setAngles (double x0, double y0, double z0, double &theta1, double &theta2, double &theta3)

  Used to calculate the servos angles.
- double singleAngle (double x0, double y0, double z0)

  Calculates the angle of one servo in the selected position.

Private Attributes

- const double cos60 = 0.5

  Contains the cosinus of 60.
- const double sin60 = sqrt(3)/2

  Contains the sinus of 60.
- const double a = 17.233

  The arm length.
- const double b = 22.648

  The forearm length.
- const double L1 = 6.000

  The clamp center lenght.
- const double L2 = 6.374

  The support center length.
- QVector< float > _axis

  Contains the axis value.
- QVector< bool > _buts

  Contains the buttons value.
- int _cBaud

  Contains the baud rate used to 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< Dominoe > _dominoe

  Contains all the dominoes information.

- bool _end

  True when we must end executino.
- Mode _mod

  Contains the working mode.
- QMutex _mutex

  To prevent memory errors between threads.
- bool _pause

  Pauses the execution of the thread.
- int _sBaud

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

  Contains the servos information.
- const int _sNum = 4

  Number of servos to manage.
- QString _sPort

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

  True if the servos port changes.
- int _sSpeed

  Speed of the robot.

3.8.1 Detailed Description

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

3.8.2 Member Enumeration Documentation

3.8.2.1 enum ServoThread::Mode

Contains the working mode.

Enumerator

controlled

manual

```
    {
        controlled,
        manual
    };
```

3.8.2.2 enum ServoThread::Version [private]

Enum containing all the save file versions.

Enumerator

v_1_0

```
    {
        v_1_0
    };
```

3.8.3 Constructor & Destructor Documentation

3.8.3.1 ServoThread::ServoThread ( )

Default constructor.

```
                        :
    _axis(XJoystick::AxisCount),
    _buts(XJoystick::ButtonCount),
    _cBaud(9600),
    _cPort("COM3"),
    _dChanged(false),
    _end(false),
    _mod(Mode::manual),
    _pause(true),
    _sBaud(1000000),
    _servos(_sNum),
    _sPort("COM9"),
    _sPortChanged(false),
    _sSpeed(60)
{

}
```

3.8.3.2 ServoThread::∼ServoThread ( )

Default destructor.

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

    wait();
}
```

3.8.4 Member Function Documentation

3.8.4.1 void ServoThread::end ( ) [inline]

Ends the execution.

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

        wait();
    }
```

3.8.4.2 int ServoThread::getServoBaud ( ) [inline]

Returns the current servo Baud rate.

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

3.8.4.3 QString ServoThread::getServoPort ( ) [inline]

Returns the current servo Port.

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

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

Returns both servo Port and baud Rate.

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

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

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

Parameters

V	Servo vector to store information

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

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

Overloaded function to get the servo info.

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

3.8.4.7 void ServoThread::load (QString & *file*)

Loads the data from the selected file.

```
{
    _mutex.lock();
    QFile f(file);
    f.open(QIODevice::ReadOnly);
    QDataStream df(&f);

    int ver;
    df >> ver;
    if (ver == Version::v_1_0) {
        int n;
        df >> _cBaud >> _cPort >> _sBaud >> _sPort >> n;

        _servos.resize(n);
        for (Servo &s : _servos) df >> s.ID;
        _dChanged = true;
    }
    else qWarning() << "Not a valid file";
    _mutex.unlock();
}
```

3.8.4.8 QMutex∗ ServoThread::mutex ( ) [inline]

Returns the mutex used in the thread.

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

3.8.4.9 void ServoThread::pause ( ) [inline]

Pauses the execution.

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

3.8.4.10 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);
    f.open(QIODevice::ReadOnly);
    QDataStream df(&f);

    QMutexLocker mL(&_mutex);

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

    df >> _cBaud >> _cPort >> _sBaud >> _sPort;
    int size;
    df >> size;
    _servos.resize(size);
    for (Servo &s : _servos) df >> s.ID;

}
```

3.8.4.11 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");
        return;
    }

    QTextStream pF(&f);

    int size;
    pF >> size;
    _mutex.lock();
    _dominoe.resize(size);
    for (Dominoe &d : _dominoe) {
        pF >> d.X >> d.Y >> d.ori;
    }

    _mutex.unlock();
}
```

3.8.4.12 void ServoThread::run ( ) [private]

Used to create another thread.

```
{
    _mutex.lock();
    int sBaud = _sBaud;
    QString sPort = _sPort;

    _mutex.unlock();
    dynamixel dxl(sPort, sBaud);
    QVector< AX12 > S(_servos.size());

    for (int i = 0; i < S.size(); ++i) {
        S[i] = AX12(&dxl);
        S[i].setID(i);
    }

    while (not _end) {

        msleep(10);
        _mutex.lock();
        if (not _end and _pause) {
            dxl.terminate();
            _cond.wait(&_mutex);
            emit statusBar("Changed");
            dxl.initialize(sPort, sBaud);
        }
        if (_dChanged) {
            if (sPort != _sPort) {
                sPort = _sPort;
                sBaud = _sBaud;
                dxl.terminate();
                dxl.initialize(sPort, sBaud);
            }
        }
        _dChanged = false;
        _mutex.unlock();
    }

    dxl.terminate();
    exit(0);
}
```

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

Used to calculate the servos angles.

```
{
    double x1 = x0 + L2 - L1;
    double y1 = y0;
    double z1 = z0;
    theta1 = singleAngle(x1,y1,z1);

    double x2 = z0*sin60 - x0*cos60 + L2 - L1;
    double y2 = y0;
    double z2 = -z0*cos60 - x0*sin60;
    theta2 = singleAngle(x2,y2,z2);

    double x3 = -z0*sin60 - x0*cos60 + L2 - L1;
    double y3 = y0;
    double z3 = -z0*cos60 + x0*sin60;
    theta3 = singleAngle(x3,y3,z3);
}
```

3.8.4.14 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 = aV;
    _buts = buts;
    _dChanged = true;

    _mutex.unlock();
}
```

3.8.4.15 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;
        _mutex.unlock();
    }
```

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

Sets the servos port.

Parameters

port	String containing the port name

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

3.8.4.17 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;
        _mutex.unlock();
    }
```

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

Sets the servos ID.

Parameters

V	Vector containing all the servos ID

```
    {
        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();
    }
```

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

Sets the servos speed.

Parameters

speed	Integer from 0 to 100 containing the % of speed

```
    {
        if (speed < 0) speed = 0;
        else if (speed > 100) speed = 100;

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

3.8.4.20 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);
}
```

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

Emmitted when the status bar must be changed.

3.8.4.22 void ServoThread::wakeUp ( ) [inline]

Continues program's execution.

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

3.8.4.23 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
        << _servos.size();
    for (const Servo &s : _servos) df << s.ID;

    _mutex.unlock();
}
```

3.8.5 Member Data Documentation

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

Contains the axis value.

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

Contains the buttons value.

3.8.5.3 int ServoThread::_cBaud [private]

Contains the baud rate used to comunicate with the clamp.

3.8.5.4 QWaitCondition ServoThread::_cond [private]

To start and pause the thread.

3.8.5.5 QString ServoThread::_cPort [private]

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

3.8.5.6 bool ServoThread::_dChanged [private]

True if the data changes.

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

Contains all the dominoes information.

3.8.5.8 bool ServoThread::_end [private]

True when we must end executino.

3.8.5.9 Mode ServoThread::_mod [private]

Contains the working mode.

3.8.5.10 QMutex ServoThread::_mutex [private]

To prevent memory errors between threads.

3.8.5.11 bool ServoThread::_pause [private]

Pauses the execution of the thread.

3.8.5.12 int ServoThread::_sBaud [private]

Contains the used baud rate to comunicate with the servos.

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

Contains the servos information.

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

Number of servos to manage.

3.8.5.15 QString ServoThread::_sPort [private]

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

3.8.5.16 bool ServoThread::_sPortChanged [private]

True if the servos port changes.

3.8.5.17 int ServoThread::_sSpeed [private]

Speed of the robot.

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

The arm length.

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

The forearm length.

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

Contains the cosinus of 60.

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

The clamp center lenght.

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

The support center length.

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

Contains the sinus of 60.

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

- servothread.h
- servothread.cpp

Chapter 4

File Documentation

4.1 dxl/ax12.cpp File Reference

Contains the AX12 class implementation.

4.1.1 Detailed Description

Contains the AX12 class implementation.

4.2 dxl/ax12.h File Reference

Contains the AX12 class declaration.

Classes

- class AX12

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

4.2.1 Detailed Description

Contains the AX12 class declaration.

4.3 dxl/dxl_hal.cpp File Reference

Contains the Dynamixel SDK platform dependent header source.

4.3.1 Detailed Description

Contains the Dynamixel SDK platform dependent header source.

4.4 dxl/dxl_hal.h File Reference

Contains the Dynamixel SDK platform dependent header declaration.

Classes

- class dxl_hal

  Dynamixel SDK platform dependent.

4.4.1 Detailed Description

Contains the Dynamixel SDK platform dependent header declaration.

4.5 dxl/dynamixel.cpp File Reference

Contains the dynamixel class implementation.

4.5.1 Detailed Description

Contains the dynamixel class implementation.

4.6 dxl/dynamixel.h File Reference

Contains the dynamixel class declaration.

Classes

- class dynamixel

  Dynamixel 1.0 protocol class.

4.6.1 Detailed Description

Contains the dynamixel class declaration.

4.7 main.cpp File Reference

Contains the Main of the program.

Functions

- int main (int argc, char ∗argv[ ])

4.7.1 Detailed Description

Contains the Main of the program.

4.7.2 Function Documentation

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

```
{
    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 servothread.cpp File Reference

Contains the ServoThread class implementation.

4.12.1 Detailed Description

Contains the ServoThread class implementation.

4.13 servothread.h File Reference

Contains the ServoThread class declaration.

Classes

- class ServoThread

  The ServoThread's class handles the 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.

4.13.1 Detailed Description

Contains the ServoThread class declaration.

4.14 stable.h File Reference

Contains all includes in a precompiled header.

4.14.1 Detailed Description

Contains all includes in a precompiled header.

The includes are:

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

Index