DeltaRobot

v0.4

Generated by Doxygen 1.8.9.1

Tue Apr 14 2015 21:18:26

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

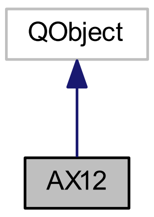
Class Documentation

3.1 AX12 Class Reference

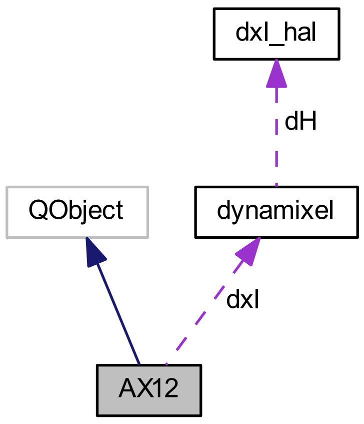
The AX12 class is used to control AX-12 motors from Dynamixel.

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

Inheritance diagram for AX12:



Collaboration diagram for AX12:



Public Member Functions

- AX12 (dynamixel &dxl, int ID=-1, QObject ∗parent=0)

  Default constructor must pass an initialized dynamixel object if ID == -1 no action is done.
- AX12 (const AX12 &a)

  Copy constructor.
- ∼AX12 ()

  Default destructor.
- QVector< int > connectedID ()

  Returns all active servos;.
- double getCurrentLoad ()

  Returns the current load from -100% to 100%, 100% is ClockWise and -100% is CounterClockWise.
- double getCurrentPos ()

  Returns the current position from 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 setGoalPosition (double goal)

  Sets the Goal's position (in degrees) or speed depending on the mode.
- void setID (int ID)

  To set a new ID.
- void setJointMode (bool mode)

  To set Joint/Wheel mode, true if Joint.
- void setMinMax (double min, double max)

  To set the minimum and maximum angle from 0 to 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.

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 (dynamixel & *dxl*, int *ID* = -1, QObject ∗ *parent* = 0)

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

```
                                                 :
         QObject(parent),
         dxl(dxl),
         _ID(ID),
        _mode(true)
     {
         if (_ID < 0) return;
         dxl.write_byte(_ID, RAM::TorqueEnable, true);
     }
```

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

Copy constructor.

```
                           :
         QObject(a.parent()),
         dxl(a.dxl),
         _ID(a._ID),
         _mode(true)
     {
     
     }
```

3.1.3.3 AX12::∼AX12 ( )

Default destructor.

```
     {
     
     }
```

3.1.4 Member Function Documentation

3.1.4.1 QVector< int > AX12::connectedID ( )

Returns all active servos;.

```
     {
         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) 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) return 0;
    int pos = dxl.read_word(_ID, RAM::PresentPosition);
    if (dxl.get_comm_result() != COMM_RXSUCCESS) return -1;
    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) 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) 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) 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::setGoalPosition (double *goal*)

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

```
{
    if (_ID < 0) 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));
}
```

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

To set a new ID.

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

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

To set Joint/Wheel mode, true if Joint.

```
{
    if (_ID < 0) 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);
    }
}
```

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

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

```
{
    if (_ID < 0) return;

    if (min > max) std::swap(min, max);

    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));
}
```

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

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

```
{
    if (_ID < 0) 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 int AX12::_ID [private]

Stores the current ID.

3.1.5.2 bool AX12::_mode [private]

True if we use the joint mode.

3.1.5.3 dynamixel& AX12::dxl [private]

Contains the dynamixel comunication.

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

- dxl/ax12.h
- dxl/ax12.cpp

3.2 dynamixel2::data Struct Reference

Struct used to handle dynamixel data.

Public Attributes

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

3.2.1 Detailed Description

Struct used to handle dynamixel data.

3.2.2 Member Data Documentation

3.2.2.1 unsigned char dynamixel2::data::iError

3.2.2.2 unsigned char dynamixel2::data::iID

3.2.2.3 unsigned short dynamixel2::data::iLength

3.2.2.4 unsigned int dynamixel2::data::iStartAddr

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

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

- dxl/dynamixel.h

3.3 dxl_hal Class Reference

Dynamixel SDK platform dependent.

`#include <dxl_hal.h>`

Public Member Functions

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

Private Attributes

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

3.3.1 Detailed Description

Dynamixel SDK platform dependent.

3.3.2 Member Function Documentation

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

```
{
    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:

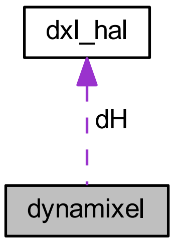
- dxl/dxl_hal.h
- dxl/dxl_hal.cpp

3.4 dynamixel Class Reference

Dynamixel 1.0 protocol class.

`#include <dynamixel.h>`

Collaboration diagram for dynamixel:



Public Member Functions

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

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

Private Attributes

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

3.4.1 Detailed Description

Dynamixel 1.0 protocol class.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 dynamixel::dynamixel ( )

```
{
}
```

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

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

3.4.3 Member Function Documentation

3.4.3.1 int dynamixel::change_baudrate (int *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]

```
{ return gbCommStatus; }
```

3.4.3.3 double dynamixel::get_packet_time (void)

```
{
    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 int dynamixel::get_rxpacket_error (int *error*)

```
{
    if( gbStatusPacket[PRT1_PKT_ERRBIT] & (unsigned char)error )
        return 1;

    return 0;
}
```

3.4.3.5 int dynamixel::get_rxpacket_error_byte (void)

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

3.4.3.6 int dynamixel::get_rxpacket_length ( )

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

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

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

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

```
{
    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 int dynamixel::is_packet_timeout (void)

```
{
    if(this->get_packet_time() > gdRcvWaitTime)
        return 1;
    return 0;
}
```

3.4.3.10 bool dynamixel::isOpen ( ) [inline]

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

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

```
{
    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*)

```
{
    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*)

```
{
    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)

```
{
    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*)

```
{
    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*)

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

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

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

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

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

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

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

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

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

}
```

3.4.3.21 int dynamixel::terminate (void)

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

3.4.3.22 void dynamixel::tx_packet (void)

```
{
    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)

```
{
    tx_packet();

    if( gbCommStatus != COMM_TXSUCCESS )
        return;


    rx_packet();
}
```

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

```
{
    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*)

```
{
    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]

3.4.4.2 int dynamixel::gbCommStatus = COMM_RXSUCCESS [private]

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

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

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

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

3.4.4.7 double dynamixel::gdByteTransTime = 0.0 [private]

3.4.4.8 double dynamixel::gdPacketStartTime = 0.0 [private]

3.4.4.9 double dynamixel::gdRcvWaitTime = 0.0 [private]

3.4.4.10 int dynamixel::giBusUsing = 0 [private]

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

- dxl/dynamixel.h

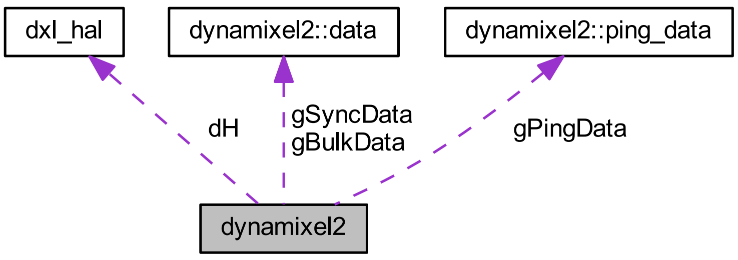
- dxl/dynamixel.cpp

3.5 dynamixel2 Class Reference

Dynamixel 2.0 protocol class.

`#include <dynamixel.h>`

Collaboration diagram for dynamixel2:



Classes

- struct data

    Struct used to handle dynamixel data.
- struct ping_data

    Struct used to do a ping.

Public Member Functions

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

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

Private Types

- typedef struct dynamixel2::ping_data PingData

  Struct used to do a ping.
- typedef struct dynamixel2::data SyncBulkData

  Struct used to handle dynamixel data.

Private Attributes

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

3.5.1 Detailed Description

Dynamixel 2.0 protocol class.

3.5.2 Member Typedef Documentation

3.5.2.1 typedef struct dynamixel2::ping_data dynamixel2::PingData [private]

Struct used to do a ping.

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

Struct used to handle dynamixel data.

3.5.3 Member Function Documentation

3.5.3.1 void dynamixel2::add_stuffing ( )

```
{
    int i = 0, index = 0;
    int packet_length_in = MAKEWORD(gbInstructionPacket[PRT2_PKT_LENGTH_L], gbInstructionPacket[PRT2_PKT_LENGTH_H]);
    int packet_length_out = packet_length_in;
    unsigned char temp[MAXNUM_TXPACKET] = {0};

    memcpy(temp, gbInstructionPacket, PRT2_PKT_LENGTH_H+1);     // FF FF FD XX ID LEN_L LEN_H
    index = PRT2_PKT_INSTRUCTION;
    for( i = 0; i < packet_length_in - 2; i++)  // except CRC
    {
        if((index - 1) == MAXNUM_TXPACKET) {
            gbCommStatus = COMM_TXERROR;
            return;
        }
        temp[index++] = gbInstructionPacket[i+PRT2_PKT_INSTRUCTION];

        if(gbInstructionPacket[i+PRT2_PKT_INSTRUCTION] == 0xFD && gbInstructionPacket[i+PRT2_PKT_INSTRUCTION-1] == 0xFF && gbInstructionPacket[i+PRT2_PKT_INSTRUCTION-2] == 0xFF)
        {
            if((index - 1) == MAXNUM_TXPACKET) {
                gbCommStatus = COMM_TXERROR;
                return;
            }
            // FF FF FD
            temp[index++] = 0xFD;

            packet_length_out++;
        }
    }

    if((index - 1) == MAXNUM_TXPACKET) {
        gbCommStatus = COMM_TXERROR;
        return;
    }
    temp[index++] = gbInstructionPacket[PRT2_PKT_INSTRUCTION+packet_length_in-2];

    if((index - 1) == MAXNUM_TXPACKET) {
        gbCommStatus = COMM_TXERROR;
        return;
    }
    temp[index++] = gbInstructionPacket[PRT2_PKT_INSTRUCTION+packet_length_in-1];

    memcpy(gbInstructionPacket, temp, index);
    gbInstructionPacket[PRT2_PKT_LENGTH_L] = LOBYTE(packet_length_out);
    gbInstructionPacket[PRT2_PKT_LENGTH_H] = HIBYTE(packet_length_out);
}
```

3.5.3.2 void dynamixel2::broadcast_ping ( )

```
{
    int idx = 0;

    gbCommStatus = COMM_TXFAIL;

    gbInstructionPacket[PRT2_PKT_ID]              = BROADCAST_ID;
    gbInstructionPacket[PRT2_PKT_LENGTH_L]        = 0x03;
    gbInstructionPacket[PRT2_PKT_LENGTH_H]        = 0x00;
    gbInstructionPacket[PRT2_PKT_INSTRUCTION]     = INST_PING;

    for(idx = 1; idx <= MAX_ID; idx++)
    {
        gPingData[idx].iID = idx;
        gPingData[idx].iFirmVer = -1;
        gPingData[idx].iModelNo = -1;
    }

    txrx_packet();
}
```

3.5.3.3 int dynamixel2::change_baudrate (int *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;
}
```

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

```
{
    if(id == BROADCAST_ID)
    {
        gbCommStatus = COMM_TXERROR;
        return;
    }

    gbCommStatus = COMM_TXFAIL;

    gbInstructionPacket[PRT2_PKT_ID]            = (unsigned char)id;
    gbInstructionPacket[PRT2_PKT_LENGTH_L]      = 0x04;
    gbInstructionPacket[PRT2_PKT_LENGTH_H]      = 0x00;
    gbInstructionPacket[PRT2_PKT_INSTRUCTION]   =
INST_RESET;
    gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0]
= (unsigned char)option;

    txrx_packet();
}
```

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

```
{
    if((start_address < gBulkData[id].iStartAddr) || ((gBulkData[id].iStartAddr +
gBulkData[id].iLength-1) < start_address))
        return 0;
    return gBulkData[id].pucTable[(start_address-gBulkData[id].
iStartAddr)];
}
```

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

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

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

```
{
    if( (start_address < gBulkData[id].iStartAddr) || ((gBulkData[id].iStartAddr +
gBulkData[id].iLength-1) < start_address))
        return 0;
    return MAKEWORD(gBulkData[id].pucTable[(start_address-
gBulkData[id].iStartAddr)],
                        gBulkData[id].pucTable[(start_address-gBulkData[id].iStartAddr+1)
]);
}
```

3.5.3.8 int dynamixel2::get_comm_result (void)

3.5.3.9 double dynamixel2::get_packet_time (void)

```
{
    double elapsed_time;

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

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

    return elapsed_time;
}
```

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

```
{
    if(id <= MAX_ID && gPingData[id].iModelNo != -1 && gPingData[id].iFirmVer != -1
)
    {
        if(info_num == PING_INFO_MODEL_NUM )
            return gPingData[id].iModelNo;
        else if(info_num == PING_INFO_FIRM_VER)
            return gPingData[id].iFirmVer;
    }

    return 0;
}
```

3.5.3.11 int dynamixel2::get_rxpacket_error_byte (void)

```
{
    return gbStatusPacket[PRT2_PKT_ERRBIT];
}
```

3.5.3.12 int dynamixel2::get_rxpacket_length ( )

```
{
    return (int)MAKEWORD(gbStatusPacket[PRT2_PKT_LENGTH_L],
gbStatusPacket[PRT2_PKT_LENGTH_H]);
}
```

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

```
{
    return (int)gbStatusPacket[PRT2_STATUS_PKT_PARAMETER0+index];
}
```

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

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

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

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

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

```
{
    if((start_address < gSyncData[id].iStartAddr) || ((gSyncData[id].iStartAddr +
gSyncData[id].iLength-1) < start_address))
        return 0;
    return MAKEWORD(gSyncData[id].pucTable[(start_address-
gSyncData[id].iStartAddr)],
                        gSyncData[id].pucTable[(start_address-gSyncData[id].iStartAddr+1)
]);
}
```

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

```
{
    unsigned int idx = 0;

    if( baud_rate < 1900 )
        return 0;

    if( dH.open(port_num, baud_rate) == 0 )
        return 0;

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


    for(idx = 1; idx <= MAX_ID; idx++)
    {
        gSyncData[idx].iID        = idx;
        gSyncData[idx].iStartAddr = 1;
        gSyncData[idx].iLength    = 1;
        gSyncData[idx].iError      = 0;
        gSyncData[idx].pucTable      = 0;

        gBulkData[idx].iID        = idx;
        gBulkData[idx].iStartAddr = 1;
        gBulkData[idx].iLength    = 1;
        gBulkData[idx].iError      = 0;
        gBulkData[idx].pucTable      = 0;

        gPingData[idx].iID = idx;
        gPingData[idx].iFirmVer = -1;
        gPingData[idx].iModelNo = -1;
    }

    gbCommStatus = COMM_RXSUCCESS;
    giBusUsing = 0;

    return 1;
}
```

3.5.3.18 int dynamixel2::is_packet_timeout (void)

```
{
    if(this->get_packet_time() > gdRcvWaitTime)
        return 1;
    return 0;
}
```

3.5.3.19 bool dynamixel2::isOpen ( ) [inline]

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

3.5.3.20 void dynamixel2::ping (unsigned char *id*)

```
{
    gbCommStatus = COMM_TXFAIL;

    gbInstructionPacket[PRT2_PKT_ID]            = (unsigned char)id;
    gbInstructionPacket[PRT2_PKT_LENGTH_L]      = 0x03;
    gbInstructionPacket[PRT2_PKT_LENGTH_H]      = 0x00;
    gbInstructionPacket[PRT2_PKT_INSTRUCTION]   =
INST_PING;

    gPingData[id].iModelNo = -1;
    gPingData[id].iFirmVer = -1;

    txrx_packet();

    if( (id != BROADCAST_ID) && (gbCommStatus ==
COMM_RXSUCCESS) )
    {
        gPingData[id].iID      = id;
        gPingData[id].iModelNo = MAKEWORD(
gbStatusPacket[PRT1_PKT_PARAMETER0+1],
gbStatusPacket[PRT1_PKT_PARAMETER0+2] );
        gPingData[id].iFirmVer = gbStatusPacket[
PRT1_PKT_PARAMETER0+3];
    }
}
```

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

```
{
    unsigned short length = 1;
    gbCommStatus = COMM_TXFAIL;

    gbInstructionPacket[PRT2_PKT_ID]                      = id;
    gbInstructionPacket[PRT2_PKT_LENGTH_L]                  = 0x07;
    gbInstructionPacket[PRT2_PKT_LENGTH_H]                  = 0x00;
    gbInstructionPacket[PRT2_PKT_INSTRUCTION]                =
INST_READ;
    gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+0
]    = (unsigned char)LOBYTE(address);
    gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+1
]    = (unsigned char)HIBYTE(address);
    gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+2
]    = (unsigned char)LOBYTE(length);
    gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+3
]    = (unsigned char)HIBYTE(length);

    txrx_packet();
    //if(gbCommStatus == COMM_RXSUCCESS && id != BROADCAST_ID)
    //  memmove(data, &rxpacket[PKT_PARAMETER+1], length);

    return gbStatusPacket[PRT2_STATUS_PKT_PARAMETER0];
}
```

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

```
{
    unsigned short length = 4;
    gbCommStatus = COMM_TXFAIL;

    gbInstructionPacket[PRT2_PKT_ID]                           = id;
    gbInstructionPacket[PRT2_PKT_LENGTH_L]                       = 0x07;
    gbInstructionPacket[PRT2_PKT_LENGTH_H]                       = 0x00;
    gbInstructionPacket[PRT2_PKT_INSTRUCTION]                     =
INST_READ;
    gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+0
]     = LOBYTE(address);
    gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+1
]     = HIBYTE(address);
    gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+2
]     = LOBYTE(length);
    gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+3
]     = HIBYTE(length);

    txrx_packet();
    //if(gbCommStatus == COMM_RXSUCCESS && id != BROADCAST_ID)
    //  memmove(data, &rxpacket[PKT_PARAMETER+1], length);

    return MAKEDWORD(MAKEWORD ( gbStatusPacket[
PRT2_STATUS_PKT_PARAMETER0], gbStatusPacket[
PRT2_STATUS_PKT_PARAMETER0+1]),
        MAKEWORD ( gbStatusPacket[PRT2_STATUS_PKT_PARAMETER0+2],
gbStatusPacket[PRT2_STATUS_PKT_PARAMETER0+3]));
}
```

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

```
{
    unsigned short length = 2;
    gbCommStatus = COMM_TXFAIL;

    gbInstructionPacket[PRT2_PKT_ID]                           = id;
    gbInstructionPacket[PRT2_PKT_LENGTH_L]                       = 0x07;
    gbInstructionPacket[PRT2_PKT_LENGTH_H]                       = 0x00;
    gbInstructionPacket[PRT2_PKT_INSTRUCTION]                     =
INST_READ;
    gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+0
]     = LOBYTE(address);
    gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+1
]     = HIBYTE(address);
    gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+2
]     = LOBYTE(length);
    gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+3
]     = HIBYTE(length);

    txrx_packet();
    //if(gbCommStatus == COMM_RXSUCCESS && id != BROADCAST_ID)
    //  memmove(data, &rxpacket[PKT_PARAMETER+1], length);

    return MAKEWORD ( gbStatusPacket[
PRT2_STATUS_PKT_PARAMETER0], gbStatusPacket[
PRT2_STATUS_PKT_PARAMETER0+1]);
}
```

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

```
{
    if(id == BROADCAST_ID)
    {
        gbCommStatus = COMM_TXERROR;
        return;
    }

    gbInstructionPacket[PRT2_PKT_ID]             = (unsigned char)id;
    gbInstructionPacket[PRT2_PKT_LENGTH_L]       = 0x03;
    gbInstructionPacket[PRT2_PKT_LENGTH_H]       = 0x00;
    gbInstructionPacket[PRT2_PKT_INSTRUCTION]    =
INST_REBOOT;

    txrx_packet();
}
```

3.5.3.25 void dynamixel2::remove_stuffing ( )

```
{
    int i = 0, index = 0;
    int packet_length_in = MAKEWORD(gbInstructionPacket[PRT2_PKT_LENGTH_L], gbInstructionPacket[PRT2_PKT_LENGTH_H]);
    int packet_length_out = packet_length_in;

    index = PRT2_PKT_INSTRUCTION;
    for( i = 0; i < packet_length_in - 2; i++)  // except CRC
    {
        if(gbInstructionPacket[i+PRT2_PKT_INSTRUCTION] == 0xFD && gbInstructionPacket[i+PRT2_PKT_INSTRUCTION+1] == 0xFD && gbInstructionPacket[i+PRT2_PKT_INSTRUCTION-1] == 0xFF && gbInstructionPacket[i+PRT2_PKT_INSTRUCTION-2] == 0xFF)
        {   // FF FF FD FD
            packet_length_out--;
            i++;
        }
        gbInstructionPacket[index++] = gbInstructionPacket[i+PRT2_PKT_INSTRUCTION];
    }
    gbInstructionPacket[index++] = gbInstructionPacket[PRT2_PKT_INSTRUCTION+packet_length_in-2];
    gbInstructionPacket[index++] = gbInstructionPacket[PRT2_PKT_INSTRUCTION+packet_length_in-1];

    gbInstructionPacket[PRT2_PKT_LENGTH_L] = LOBYTE(packet_length_out);
    gbInstructionPacket[PRT2_PKT_LENGTH_H] = HIBYTE(packet_length_out);
}
```

3.5.3.26 void dynamixel2::rx_packet (void)

```
{
    //int rx_length = 0, wait_length = PRT2_PKT_LENGTH_H + 4 + 1;    // 4 : INST ERROR CHKSUM_L CHKSUM_H
    unsigned int i;
    unsigned short crc = 0;

    gbRxGetLength = 0; gbRxPacketLength = PRT2_PKT_LENGTH_H + 4 + 1;

    // Check Bus Using
    //if(bus_using == 0)
    //    return 0;

    while(1)
    {
        gbRxGetLength += dH.read( &gbStatusPacket[gbRxGetLength], gbRxPacketLength - gbRxGetLength);
        if(gbRxGetLength >= gbRxPacketLength)    // wait_length minimum : 11
        {
            // Find packet header
            for(i = 0; i < (gbRxGetLength - 2); i++)
            {
                if(gbStatusPacket[i] == 0xFF && gbStatusPacket[i+1] == 0xFF && gbStatusPacket[i+2] == 0xFD)
                    break;
            }

            if(i == 0)
            {
                // Check length
                gbRxPacketLength = MAKEWORD(gbStatusPacket[PRT2_PKT_LENGTH_L], gbStatusPacket[PRT2_PKT_LENGTH_H]) + PRT2_PKT_LENGTH_H + 1;
                if(gbRxGetLength < gbRxPacketLength)
                {
                    // Check timeout
                    if(is_packet_timeout() == 1)
                    {
                        if(gbRxGetLength == 0)
                            gbCommStatus = COMM_RXTIMEOUT;
                        else
                            gbCommStatus = COMM_RXCORRUPT;
                        giBusUsing = 0;
                        break;
                    }
                    continue;
                }

                // Check CRC16
                crc = MAKEWORD(gbStatusPacket[gbRxPacketLength-2], gbStatusPacket[gbRxPacketLength-1]);
                if(update_crc(0, gbStatusPacket, gbRxPacketLength-2) == crc) // -2 : except CRC16
                    gbCommStatus = COMM_RXSUCCESS;
                else
                    gbCommStatus = COMM_RXCORRUPT;
                giBusUsing = 0;
                break;
            }
            else
            {
                // Remove unnecessary packets
                memmove(&gbStatusPacket[0], &gbStatusPacket[i], gbRxGetLength - i);
                gbRxGetLength -= i;
            }
        }
        else
        {
            // Check timeout
            if(is_packet_timeout() == 1)
            {
                if(gbRxGetLength == 0)
                    gbCommStatus = COMM_RXTIMEOUT;
                else
                    gbCommStatus = COMM_RXCORRUPT;
                giBusUsing = 0;
                break;
            }
        }
    }

    // Character stuffing
    if(gbCommStatus == COMM_RXSUCCESS)
        remove_stuffing();

    giBusUsing = 0;
}
```

3.5.3.27 void dynamixel2::set_packet_timeout (int *NumRcvByte*)

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

3.5.3.28 void dynamixel2::set_packet_timeout_ms (int *msec*)

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

3.5.3.29 void dynamixel2::set_txpacket_id (unsigned char *id*)

```
{
    gbInstructionPacket[PRT2_PKT_ID] = id;
}
```

3.5.3.30 void dynamixel2::set_txpacket_instruction (unsigned char *instruction*)

```
{
    gbInstructionPacket[PRT2_PKT_INSTRUCTION] = (unsigned char)instruction;
}
```

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

```
{
    gbInstructionPacket[PRT2_PKT_LENGTH_L] =
LOBYTE(length);
    gbInstructionPacket[PRT2_PKT_LENGTH_H] =
HIBYTE(length);
}
```

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

```
{
    gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+
index] = value;
}
```

3.5.3.33 int dynamixel2::terminate (void)

```
{
    int id = 0;
    for(id = 0; id <= MAX_ID; id++)
    {
        if(gBulkData[id].pucTable != 0)
            free((gBulkData[id].pucTable));

        if(gSyncData[id].pucTable != 0)
            free((gBulkData[id].pucTable));
    }
    dH.close();
    return 0;
}
```

3.5.3.34 void dynamixel2::tx_packet (void)

```
{
    int packet_tx_len, real_tx_len;
    int length;
    unsigned short crc = 0;


    // Check Bus Using
    if(giBusUsing == 1)
    {
        gbCommStatus = COMM_TXFAIL;
        return;
    }
    giBusUsing = 1;

    // Character stuffing
    add_stuffing();
    if(gbCommStatus == COMM_TXERROR)
        return;

    length = MAKEWORD(gbInstructionPacket[
PRT2_PKT_LENGTH_L], gbInstructionPacket[
PRT2_PKT_LENGTH_H]);

    // Check MAX packet length
    if(length > (MAXNUM_TXPACKET))
    {
        giBusUsing = 0;
        gbCommStatus = COMM_TXERROR;
        return;
    }

    // Packet Header
    gbInstructionPacket[PRT2_PKT_HEADER0]    = 0xFF;
    gbInstructionPacket[PRT2_PKT_HEADER1]    = 0xFF;
    gbInstructionPacket[PRT2_PKT_HEADER2]    = 0xFD;
    gbInstructionPacket[PRT2_PKT_RESERVED]   = 0x00; // RESERVED

    // Add CRC16
    crc = update_crc(0, gbInstructionPacket, length+
PRT2_PKT_LENGTH_H+1-2);  // -2 : except CRC16
    gbInstructionPacket[length+PRT2_PKT_LENGTH_H-1] =
LOBYTE(crc);      // last - 1
    gbInstructionPacket[length+PRT2_PKT_LENGTH_H-0] = HIBYTE(crc);      // last - 0

    // Tx Packet
    dH.clear();
    packet_tx_len = length + PRT2_PKT_LENGTH_H + 1;
    real_tx_len = dH.write(gbInstructionPacket, packet_tx_len );
    if( packet_tx_len != real_tx_len )
    {
        giBusUsing = 0;
        gbCommStatus = COMM_TXFAIL;
        return;
    }

    gbCommStatus = COMM_TXSUCCESS;
}
```

3.5.3.35 void dynamixel2::txrx_packet (void)

```
{
    int n = 0, num = 0;
    int id = 0;
    int wait_length = 0;
    int data_length = 0;
    gbCommStatus = COMM_TXFAIL;

    // Wait for Bus Idle
    while(giBusUsing == 1)
    {
        //Sleep(0);
    }

    if( ( gbInstructionPacket[PRT2_PKT_INSTRUCTION] != INST_PING        )  &&
        ( gbInstructionPacket[PRT2_PKT_INSTRUCTION] != INST_READ        )  &&
        ( gbInstructionPacket[PRT2_PKT_INSTRUCTION] != INST_WRITE        )  &&
        ( gbInstructionPacket[PRT2_PKT_INSTRUCTION] != INST_REG_WRITE     )  &&
        ( gbInstructionPacket[PRT2_PKT_INSTRUCTION] != INST_ACTION       )  &&
        ( gbInstructionPacket[PRT2_PKT_INSTRUCTION] != INST_RESET        )  &&
        ( gbInstructionPacket[PRT2_PKT_INSTRUCTION] != INST_SYNC_WRITE   )  &&
        ( gbInstructionPacket[PRT2_PKT_INSTRUCTION] != INST_BULK_READ    )  &&
        ( gbInstructionPacket[PRT2_PKT_INSTRUCTION] != INST_REBOOT       )  &&
        ( gbInstructionPacket[PRT2_PKT_INSTRUCTION] != INST_STATUS       )  &&
        ( gbInstructionPacket[PRT2_PKT_INSTRUCTION] != INST_SYNC_READ    )  &&
        ( gbInstructionPacket[PRT2_PKT_INSTRUCTION] != INST_BULK_WRITE   ) )
    {
        gbCommStatus = COMM_TXERROR;
        return;
    }



    //if( (gbInstructionPacket[PRT2_PKT_INSTRUCTION] != INST_SYNC_READ) && (gbInstructionPacket[PRT2_PKT_INSTRUCTION] != INST_BULK_READ) )
    if( (gbInstructionPacket[PRT2_PKT_ID] != BROADCAST_ID) )
    {
        tx_packet();
        // Check Tx packet result
        if( gbCommStatus != COMM_TXSUCCESS )
            return;

        // Set Rx Timeout
        if(gbInstructionPacket[PRT2_PKT_INSTRUCTION] == INST_READ)
            set_packet_timeout(MAKEWORD( gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+2], gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+3]) + 11);
        else
            set_packet_timeout(PRT2_PKT_LENGTH_H+4+1);     // 4 : INST ERROR CHKSUM_L CHKSUM_H

        //if(gbInstructionPacket[PRT2_PKT_ID] == BROADCAST_ID)
        //{
        //  giBusUsing = 0;
        //  gbCommStatus = COMM_RXSUCCESS;
        //  return;
        //}

        rx_packet();
        if((gbCommStatus == COMM_RXSUCCESS) && (gbStatusPacket[PRT2_PKT_ID] != BROADCAST_ID) && (gbInstructionPacket[PRT2_PKT_ID] != gbStatusPacket[PRT2_PKT_ID]))
            rx_packet();
    }
    else
    {
        if(gbInstructionPacket[PRT2_PKT_INSTRUCTION] == INST_BULK_READ )
        {
            num = (MAKEWORD(gbInstructionPacket[PRT2_PKT_LENGTH_L], gbInstructionPacket[PRT2_PKT_LENGTH_H]) - 3 )/5;
            for(n = 0; n < num; n++)
            {
                id = gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0 + n*5];
                gBulkData[id].iError = -1;
                gBulkData[id].iStartAddr = MAKEWORD(gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0 + n*5 + 1],
                                                        gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0 + n*5 + 2]);
                gBulkData[id].iLength    = MAKEWORD(gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0 + n*5 + 3],
                                                        gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0 + n*5 + 4]);

                if(gBulkData[id].pucTable != 0)
                    free((gBulkData[id].pucTable));

                gBulkData[id].pucTable = (unsigned char*) calloc(gBulkData[id].iLength, sizeof(unsigned char));
                wait_length += gBulkData[id].iLength + 11;
            }

            while(giBusUsing == 1)
            {
                //Sleep(0);
            }
            tx_packet();
            if( gbCommStatus != COMM_TXSUCCESS )
                return;

            set_packet_timeout(wait_length);

            for(n = 0; n < num; n++)
            {
                id = gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0 + n*5];
                // Rx packet
                rx_packet();
                if(gbCommStatus == COMM_RXSUCCESS)
                    gBulkData[id].iError = gbStatusPacket[PRT2_PKT_ERRBIT];
                // rxpacket to rxdata[id]->pucTable
                memcpy(gBulkData[id].pucTable, &gbStatusPacket[PRT2_STATUS_PKT_PARAMETER0], gBulkData[id].iLength);
            }
        }
        else if(gbInstructionPacket[PRT2_PKT_INSTRUCTION] == INST_SYNC_READ)
        {
            num = (MAKEWORD(gbInstructionPacket[PRT2_PKT_LENGTH_L], gbInstructionPacket[PRT2_PKT_LENGTH_H]) - 3 - 4); //3 : INST  CRC_L CRC H, 4 : param0->addr_l param0->addr_h param0->length_l param0->length_h
            data_length = MAKEWORD(gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+2],
                gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+3]);


            for(n = 0; n < num; n++)
            {
                id = gbInstructionPacket[
PRT2_INSTRUCTION_PKT_PARAMETER0 + 4 + n];
                gSyncData[id].iID = id;
                gSyncData[id].iStartAddr = MAKEWORD(
gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0 + 0],

                                                        gbInstructionPacket[
PRT2_INSTRUCTION_PKT_PARAMETER0 + 1]);
                gSyncData[id].iError = -1;
                if(gSyncData[id].pucTable != 0)
                    free((gSyncData[id].pucTable));

                gSyncData[id].pucTable = (unsigned char *) calloc(data_length, sizeof(
unsigned char));
            }


            wait_length  = 11 + data_length;
            wait_length *= num;

            while(giBusUsing == 1);

            tx_packet();

            // Check Tx packet result
            if( gbCommStatus != COMM_TXSUCCESS )
                return;

            // Set Rx Timeout (SYNK_READ)
            set_packet_timeout(wait_length);

            for(n = 0; n < num; n++)
            {
                id = gbInstructionPacket[
PRT2_INSTRUCTION_PKT_PARAMETER0 + 4 + n];
                // Rx packet
                rx_packet();
                if(gbCommStatus == COMM_RXSUCCESS)
                    gSyncData[id].iError = gbStatusPacket[
PRT2_PKT_ERRBIT];
                // rxpacket to rxdata[id]->pucTable
                memcpy(gSyncData[id].pucTable, &gbStatusPacket[
PRT2_STATUS_PKT_PARAMETER0], data_length);
            }

            return;
        }
        else if(gbInstructionPacket[PRT2_PKT_INSTRUCTION] ==
INST_PING)
        {
            int rx_length = 0;
            tx_packet();
            if(gbCommStatus != COMM_TXSUCCESS)
            {
                giBusUsing = 0;
                return;
            }

            wait_length = PING_STATUS_LENGTH * MAX_ID;
            set_packet_timeout_ms((int)((gdByteTransTime * wait_length)
 + (3 * MAX_ID) + 2 * LATENCY_TIME));

            while(1)
            {
                int _cnt = dH.read(&gbStatusPacket[rx_length], wait_length - rx_length)
;
                if(_cnt > 0)
                {
                    rx_length += _cnt;
                    //printf("cnt : %d, Interval : %f / Wait time : %f\n", _cnt,  get_packet_time(),
 gdPacketWaitTime);
                }
                if(is_packet_timeout() == 1 || rx_length >= wait_length)
                    break;
            }
            giBusUsing = 0;

            if(rx_length== 0)
            {
                gbCommStatus = COMM_RXTIMEOUT;
                return;
            }

            while(1)
            {
                int idx = 0;

                if(rx_length < PING_STATUS_LENGTH)
                {
                    gbCommStatus = COMM_RXCORRUPT;
                    return;
                }

                // find packet header
                while( idx < (rx_length - 2) )
                {
                    if(gbStatusPacket[idx] == 0xFF && gbStatusPacket[idx + 1] == 0xFF && gbStatusPacket[idx + 2] == 0xFD)
                        break;
                    else
                        idx++;
                }

                if(idx == 0)
                {
                    // check CRC16
                    int crc = MAKEWORD(gbStatusPacket[PING_STATUS_LENGTH - 2], gbStatusPacket[PING_STATUS_LENGTH - 1]);
                    if(update_crc(0, gbStatusPacket, PING_STATUS_LENGTH - 2) == crc) // -2 : except CRC16
                    {
                        gPingData[gbStatusPacket[PRT2_PKT_ID]].iID = gbStatusPacket[PRT2_PKT_ID];
                        gPingData[gbStatusPacket[PRT2_PKT_ID]].iModelNo = MAKEWORD(gbStatusPacket[PRT2_STATUS_PKT_PARAMETER0], gbStatusPacket[PRT2_STATUS_PKT_PARAMETER0+1]);
                        gPingData[gbStatusPacket[PRT2_PKT_ID]].iFirmVer = gbStatusPacket[PRT2_STATUS_PKT_PARAMETER0+2];

                        memcpy(&gbStatusPacket[0], &gbStatusPacket[PING_STATUS_LENGTH], rx_length - PING_STATUS_LENGTH);
                        rx_length -= PING_STATUS_LENGTH;
                    }
                    else
                    {
                        gbCommStatus = COMM_RXCORRUPT;

                        // remove header (0xFF 0xFF 0xFD)
                        memcpy(&gbStatusPacket[0], &gbStatusPacket[3], rx_length - 3);
                        rx_length -= 3;
                    }

                    if(rx_length < PING_STATUS_LENGTH)
                        break;
                }
                else
                {
                    // remove unnecessary packets
                    memcpy(&gbStatusPacket[0], &gbStatusPacket[idx], rx_length - idx);
                    rx_length -= idx;
                }
            }
        }
        else // Sync_Write ans Bulk Write
        {
            tx_packet();
            giBusUsing = 0;
            if(gbCommStatus == COMM_TXSUCCESS)
                gbCommStatus = COMM_RXSUCCESS;
            return;
        }
    }
}
```

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

```
{
    unsigned short length = 1;
    gbInstructionPacket[PRT2_PKT_ID]                    = id;
    gbInstructionPacket[PRT2_PKT_LENGTH_L]                = LOBYTE(length+5);
    gbInstructionPacket[PRT2_PKT_LENGTH_H]                = HIBYTE(length+5);
    gbInstructionPacket[PRT2_PKT_INSTRUCTION]              = INST_WRITE;
    gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+0]     = LOBYTE(address);
    gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+1
]     = HIBYTE(address);
    gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+2
]     = (unsigned char)value;

    txrx_packet();
}
```

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

```
{
    unsigned short length = 4;
    gbInstructionPacket[PRT2_PKT_ID]                      = id;
    gbInstructionPacket[PRT2_PKT_LENGTH_L]                   =
LOBYTE(length+5);
    gbInstructionPacket[PRT2_PKT_LENGTH_H]                   =
HIBYTE(length+5);
    gbInstructionPacket[PRT2_PKT_INSTRUCTION]                 =
INST_WRITE;
    gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+0
]     = LOBYTE(address);
    gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+1
]     = HIBYTE(address);
    gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+2
]     = LOBYTE(LOWORD( value ));
    gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+3
]     = HIBYTE(LOWORD( value ));
    gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+4
]     = LOBYTE(HIWORD( value ));
    gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+5
]     = HIBYTE(HIWORD( value ));

    txrx_packet();
}
```

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

```
{
    unsigned short length = 2;
    gbInstructionPacket[PRT2_PKT_ID]                      = id;
    gbInstructionPacket[PRT2_PKT_LENGTH_L]                   =
LOBYTE(length+5);
    gbInstructionPacket[PRT2_PKT_LENGTH_H]                   =
HIBYTE(length+5);
    gbInstructionPacket[PRT2_PKT_INSTRUCTION]                 =
INST_WRITE;
    gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+0
]     = LOBYTE(address);
    gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+1
]     = HIBYTE(address);
    gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+2
]     = LOBYTE( value );
    gbInstructionPacket[PRT2_INSTRUCTION_PKT_PARAMETER0+3
]     = HIBYTE( value );

    txrx_packet();
}
```

3.5.4 Member Data Documentation

3.5.4.1 dxl_hal dynamixel2::dH [private]

3.5.4.2 int dynamixel2::gbCommStatus = COMM_RXSUCCESS [private]

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

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

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

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

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

3.5.4.8 double dynamixel2::gdByteTransTime = 0.0 [private]

3.5.4.9 double dynamixel2::gdPacketStartTime = 0.0 [private]

3.5.4.10 double dynamixel2::gdRcvWaitTime = 0.0 [private]

3.5.4.11 int dynamixel2::giBusUsing = 0 [private]

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

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

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

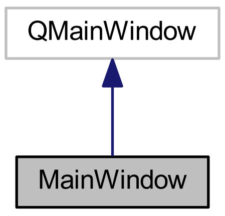
- dxl/dynamixel.h
- dxl/dynamixel.cpp

3.6 MainWindow Class Reference

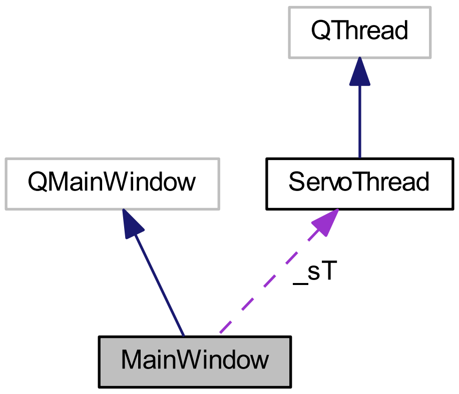
Contains all the windows and other classes.

`#include <mainwindow.h>`

Inheritance diagram for MainWindow:



Collaboration diagram for MainWindow:



Signals

- void joystickChanged ()

  Emmitted when a joystick changes.

Public Member Functions

- MainWindow (QWidget ∗parent=0)

  Default constructor.
- ∼MainWindow ()

  Default destructor.

Private Slots

- void joyChanged ()

  Handles a joystick update.
- void on_actionOptions_triggered ()

  To select the options.
- void update ()

  Updates all data to the servo thread.

Private Attributes

- QVector< QLabel ∗ > _axis

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

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

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

Handles all buttons values.
- QString _dataP

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

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

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

  AXis for the Z value.
- XJoystick _joy

  To handle the joystick.
- ServoThread _sT

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

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

  Contains the user interface.

Static Private Attributes

- static const int sCount = 3

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

  Contains the number of additional servos used.

3.6.1 Detailed Description

Contains all the windows and other classes.

3.6.2 Constructor & Destructor Documentation

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

Default constructor.

```
                                        :
         QMainWindow(parent),
         _axis(XJoystick::AxisCount),
         _axisV(XJoystick::AxisCount),
         _buts(XJoystick::ButtonCount),
         _butsV(XJoystick::ButtonCount),
         ui(new Ui::MainWindow)
{
         ui->setupUi(this);
         _sT.start();
         _timer.setInterval(10);
         _timer.start();

         connect(&_joy, SIGNAL(changed()), this, SLOT(joyChanged()));
         connect(&_timer, SIGNAL(timeout()), this, SLOT(update()));

         // 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();
    // TODO: Create dataPath

    _dataP = QStandardPaths::writableLocation(QStandardPaths::AppDataLocation);
    QDir dir(_dataP);
    if (!dir.exists()) dir.mkpath(_dataP);
}
```

3.6.2.2 MainWindow::∼MainWindow ( )

Default destructor.

```
{
    delete ui;
}
```

3.6.3 Member Function Documentation

3.6.3.1 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();
}
```

3.6.3.2 void MainWindow::joystickChanged ( ) [signal]

Emmitted when a joystick changes.

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

To select the options.

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

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

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

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

Updates all data to the servo thread.

```
{
    _joy.update();
    for (int i = 0; i < XJoystick::AxisCount; ++i) _axisV[i] = _joy[i];
    for (int i = 0; i < XJoystick::ButtonCount; ++i) _butsV[i] = _joy.button(i);

    _sT.setData(_axisV, _butsV);

    // TODO: Finish update function
}
```

3.6.4 Member Data Documentation

3.6.4.1 QVector< QLabel ∗> MainWindow::_axis [private]

Handles all the axis labels.

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

Contains the axis value;.

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

Handles all the button labels.

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

Handles all buttons values.

3.6.4.5 QString MainWindow::_dataP [private]

Contains the path to the data location.

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

Axis for the X value.

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

Axis for the Y value.

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

AXis for the Z value.

3.6.4.9 XJoystick MainWindow::_joy [private]

To handle the joystick.

3.6.4.10 ServoThread MainWindow::_sT [private]

Contains the thread controlling all the servos and external hardware.

3.6.4.11 QTimer MainWindow::_timer [private]

To update the joystick value.

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

Contains the number of additional servos used.

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

Contains the number of minimun servos to work.

3.6.4.14 Ui::MainWindow∗ MainWindow::ui [private]

Contains the user interface.

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

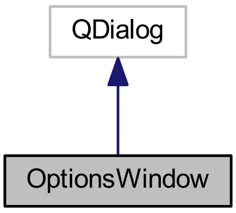
- mainwindow.h
- mainwindow.cpp

3.7 OptionsWindow Class Reference

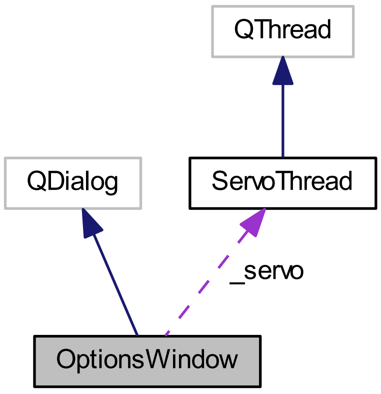
Class used to handle a Window to set the options.

`#include <optionswindow.h>`

Inheritance diagram for OptionsWindow:



Collaboration diagram for OptionsWindow:



Public Slots

- void joystickChanged ()

  To handle the change of a joystick.

Public Member Functions

- OptionsWindow (XJoystick &J, ServoThread ∗servo, QWidget ∗parent=0)

  Default constructor.

- ∼OptionsWindow ()

  Destructor.

- void storeData ()

  Stores all data.

Private Slots

- void events ()

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

    Refreshes all the servos connected to the port.

Private Attributes

- XJoystick & _joy

    Contains the Joystick to handle options.
- int _portSize

    Contains the size of the ports.
- ServoThread ∗ _servo

    Pointer to the servo thread class.
- QTimer _timer

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

    Containsh the GUI.

3.7.1 Detailed Description

Class used to handle a Window to set the options.

3.7.2 Constructor & Destructor Documentation

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

Default constructor.

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

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

    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);

    joystickChanged();

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

}
```

3.7.2.2 OptionsWindow::∼OptionsWindow ( )

Destructor.

```
{
    delete ui;
}
```

3.7.3 Member Function Documentation

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

Handles events that need to be updated continously.

```
{
    auto ports = QSerialPortInfo::availablePorts();

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

        QString portC(ui->portC->currentData().toString());
        QString portS(ui->portS->currentData().toString());
        qDebug() << portC << portS;
        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;
        }
        qDebug() << selC << selS;

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

3.7.3.2 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()));
}
```

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

Refreshes all the servos connected to the port.

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

    dxl.initialize(port, baud);

}
```

3.7.3.4 void OptionsWindow::storeData ( )

Stores all data.

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


}
```

3.7.4 Member Data Documentation

3.7.4.1 XJoystick& OptionsWindow::_joy [private]

Contains the Joystick to handle options.

3.7.4.2 int OptionsWindow::_portSize [private]

Contains the size of the ports.

3.7.4.3 ServoThread∗ OptionsWindow::_servo [private]

Pointer to the servo thread class.

3.7.4.4 QTimer OptionsWindow::_timer [private]

Waits for a new COM port.

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

Containsh the GUI.

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

- optionswindow.h
- optionswindow.cpp

3.8 dynamixel2::ping_data Struct Reference

Struct used to do a ping.

Public Attributes

- int iID
- int iModelNo
- int iFirmVer

3.8.1 Detailed Description

Struct used to do a ping.

3.8.2 Member Data Documentation

3.8.2.1 int dynamixel2::ping_data::iFirmVer

3.8.2.2 int dynamixel2::ping_data::iID

3.8.2.3 int dynamixel2::ping_data::iModelNo

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

- dxl/dynamixel.h

3.9 ServoThread::Servo Struct Reference

Struct for the AX12 servos.

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

Public Member Functions

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

  Default constructor.
- Servo (const Servo &s)

  Copy constructor.

Public Attributes

- int ID

  Contains the servo ID.
- double load

  Contains the servo load.
- double pos

  Contains the servo position.

3.9.1 Detailed Description

Struct for the AX12 servos.

3.9.2 Constructor & Destructor Documentation

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

Default constructor.

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

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

Copy constructor.

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

3.9.3 Member Data Documentation

3.9.3.1 int ServoThread::Servo::ID

Contains the servo ID.

3.9.3.2 double ServoThread::Servo::load

Contains the servo load.

3.9.3.3 double ServoThread::Servo::pos

Contains the servo position.

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

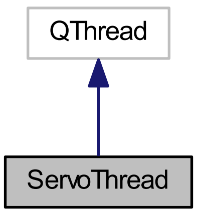
- servothread.h

3.10 ServoThread Class Reference

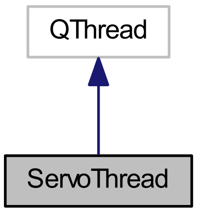
The ServoThread's class handles the comunication between the delta robot servos and the PC.

`#include <servothread.h>`

Inheritance diagram for ServoThread:



Collaboration diagram for ServoThread:



Classes

- struct Servo

  Struct for the AX12 servos.

Public Types

- enum Mode { controlled, manual }

  Contains the working mode.

Public Member Functions

- ServoThread ()

  Default constructor.
- ∼ServoThread ()

  Default destructor.
- void cont ()

  Continues program's execution.
- void end ()

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

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

  Pauses the execution.
- int getServoBaud ()

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

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

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

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

  Overloaded function to get the servo info.
- void setData (QVector< float > &aV, QVector< bool > &buts)

Adds the loaded data.
- void setServoBaud (unsigned int baud)

  Sets the servos port baud rate.
- void setServoPort (QString &port)

  Sets the servos port.
- void setSID (QVector< int > &V)

  Sets the servos ID.
- void write (QString &file)

  Writes data to the selected directory.

Private Types

- enum Version { v_1_0 }

  Enum containing all the save file versions.

Private Member Functions

- void run ()

  Used to create another thread.

Private Attributes

- QVector< float > _axis

  Contains the axis value.
- QVector< bool > _buts

  Contains the buttons value.
- int _cBaud

  Contains the baud rate used to 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.
- bool _end

  True when we must end executino.
- Mode _mod

  Contains the working mode.
- QMutex _mutex

  To prevent memory errors.
- bool _pause

  Pauses the execution of the thread.
- int _sBaud

  Contains the used baud rate to 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.

3.10.1 Detailed Description

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

3.10.2 Member Enumeration Documentation

3.10.2.1 enum ServoThread::Mode

Contains the working mode.

Enumerator

controlled

manual

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

3.10.2.2 enum ServoThread::Version [private]

Enum containing all the save file versions.

Enumerator

v_1_0

```
    {
        v_1_0
    };
```

3.10.3 Constructor & Destructor Documentation

3.10.3.1 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(3),
    _sPort("COM9"),
    _sPortChanged(false)
{

}
```

3.10.3.2 ServoThread::∼ServoThread ( )

Default destructor.

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

    wait();
}
```

3.10.4 Member Function Documentation

3.10.4.1 void ServoThread::cont ( ) [inline]

Continues program's execution.

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

3.10.4.2 void ServoThread::end ( ) [inline]

Ends the execution.

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

        wait();
    }
```

3.10.4.3 int ServoThread::getServoBaud ( ) [inline]

Returns the current servo Baud rate.

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

3.10.4.4 QString ServoThread::getServoPort ( ) [inline]

Returns the current servo Port.

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

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

Returns both servo Port and baud Rate.

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

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

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

Parameters

V	Servo vector to store information

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

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

Overloaded function to get the servo info.

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

3.10.4.8 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();
}
```

3.10.4.9 void ServoThread::pause ( ) [inline]

Pauses the execution.

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

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

    while (not _end) {

        msleep(10);
        _mutex.lock();
        if (not _end and _pause) {
            dxl.terminate();
            _cond.wait(&_mutex);
            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);
}
```

3.10.4.11 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();
}
```

3.10.4.12 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();
    }
```

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

Sets the servos port.

Parameters

port	String containing the port name

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

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

Sets the servos ID.

Parameters

V	Vector containing all the servos ID

```
{
    _mutex.lock();
    if (V.size() != _servos.size()) _servos.resize(V.size());

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

3.10.4.15 void ServoThread::write (QString & *file*)

Writes data to the selected directory.

Parameters

file	Path to the file

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

    df << int(Version::v_1_0) << _cBaud << _cPort << _sBaud <<
 _sPort
       << _servos.size();
    for (const Servo &s : _servos) df << s.ID;

    _mutex.unlock();
}
```

3.10.5 Member Data Documentation

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

Contains the axis value.

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

Contains the buttons value.

3.10.5.3 int ServoThread::_cBaud [private]

Contains the baud rate used to comunicate with the clamp.

3.10.5.4 QWaitCondition ServoThread::_cond [private]

To start and pause the thread.

3.10.5.5 QString ServoThread::_cPort [private]

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

3.10.5.6 bool ServoThread::_dChanged [private]

True if the data changes.

3.10.5.7 bool ServoThread::_end [private]

True when we must end executino.

3.10.5.8 Mode ServoThread::_mod [private]

Contains the working mode.

3.10.5.9 QMutex ServoThread::_mutex [private]

To prevent memory errors.

3.10.5.10 bool ServoThread::_pause [private]

Pauses the execution of the thread.

3.10.5.11 int ServoThread::_sBaud [private]

Contains the used baud rate to comunicate with the servos.

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

Contains the servos information.

3.10.5.13 QString ServoThread::_sPort [private]

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

3.10.5.14 bool ServoThread::_sPortChanged [private]

True if the servos port changes.

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

- servothread.h
- servothread.cpp

Chapter 4

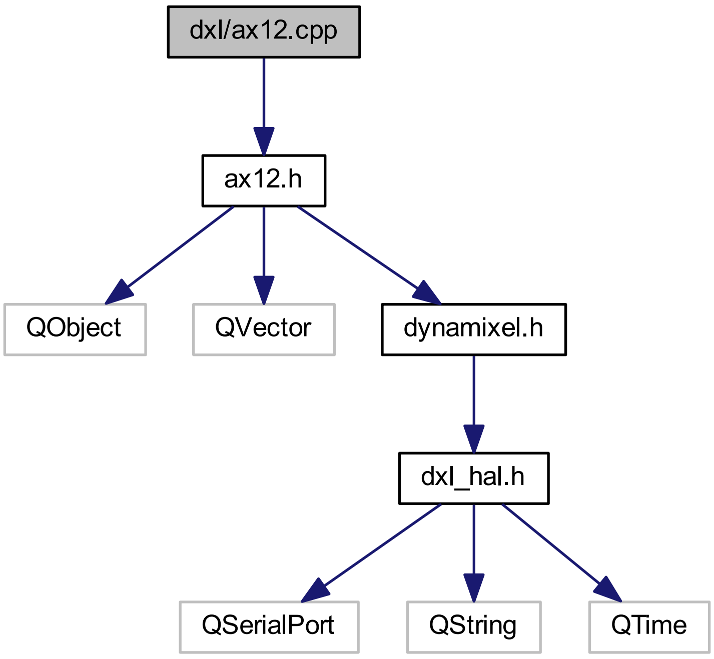
File Documentation

4.1 dxl/ax12.cpp File Reference

Contains the AX12 class implementation.

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

Include dependency graph for ax12.cpp:



4.1.1 Detailed Description

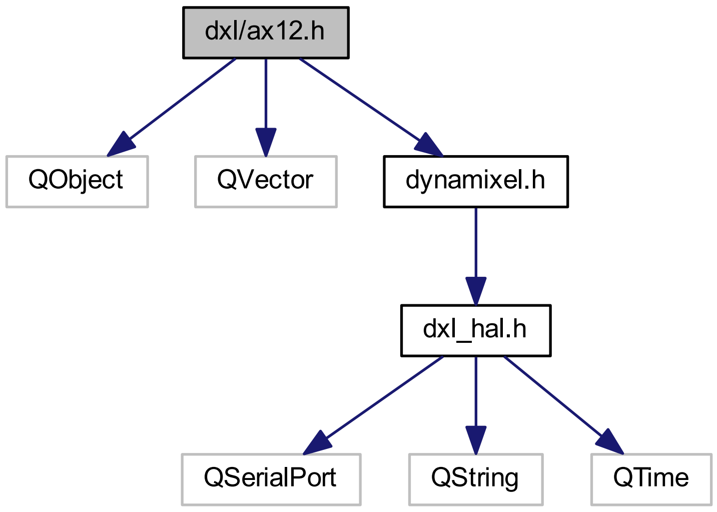
Contains the AX12 class implementation.

4.2 dxl/ax12.h File Reference

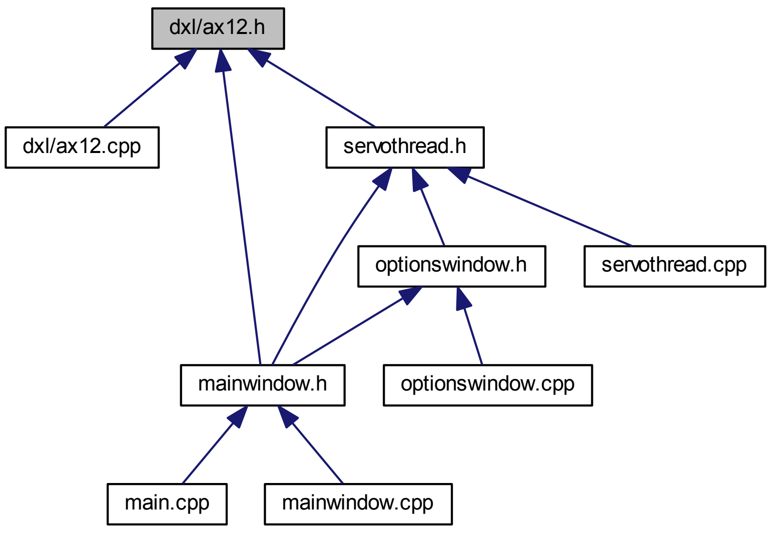
Contains the AX12 class declaration.

```
#include <QObject>
#include <QVector>
#include "dynamixel.h"
```

Include dependency graph for ax12.h:



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



Classes

- class AX12

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

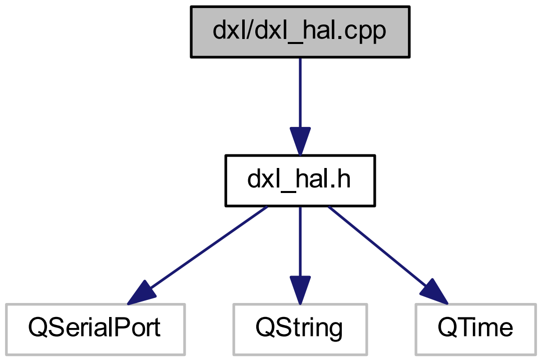
4.2.1 Detailed Description

Contains the AX12 class declaration.

4.3 dxl/dxl_hal.cpp File Reference

Contains the Dynamixel SDK platform dependent header source.

```
#include "dxl_hal.h"
```
Include dependency graph for dxl_hal.cpp:



4.3.1 Detailed Description

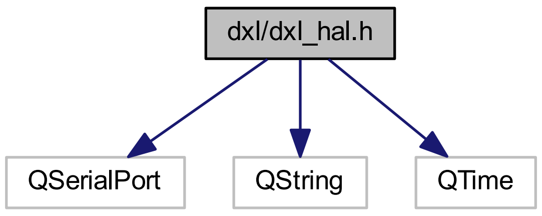
Contains the Dynamixel SDK platform dependent header source.

4.4 dxl/dxl_hal.h File Reference

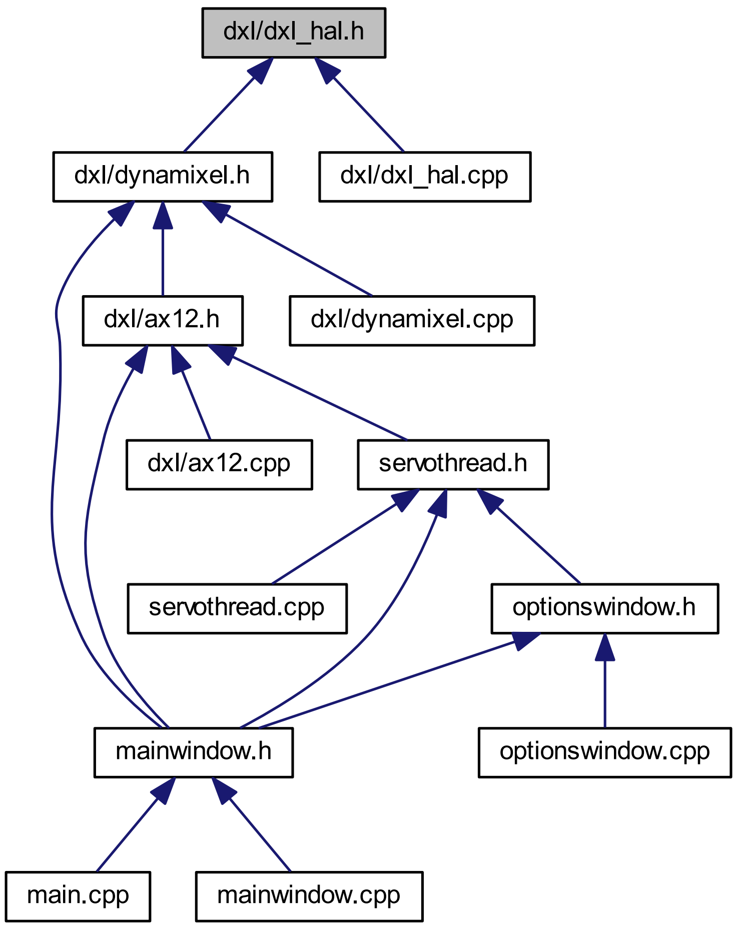
Contains the Dynamixel SDK platform dependent header declaration.

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

Include dependency graph for dxl_hal.h:



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



Classes

- class dxl_hal

  Dynamixel SDK platform dependent.

Macros

- #define MAXNUM_TXPACKET (10000)
- #define MAXNUM_RXPACKET (10000)

4.4.1 Detailed Description

Contains the Dynamixel SDK platform dependent header declaration.

4.4.2 Macro Definition Documentation

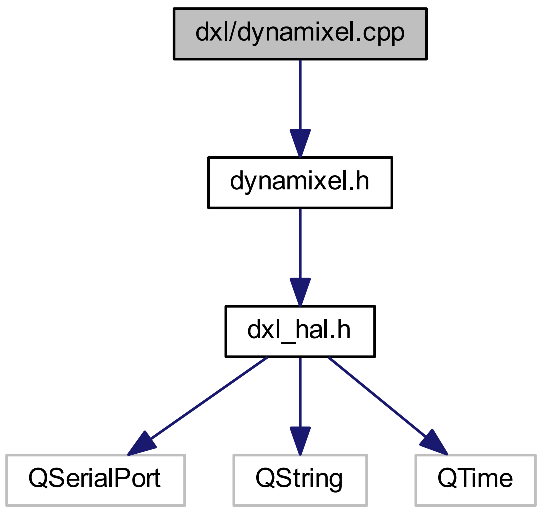
4.4.2.1 #define MAXNUM_RXPACKET (10000)

4.4.2.2 #define MAXNUM_TXPACKET (10000)

4.5 dxl/dynamixel.cpp File Reference

Contains the dynamixel and dynamixel2 classes implementation.

```
#include "dynamixel.h"
```
Include dependency graph for dynamixel.cpp:



Macros

- #define LATENCY_TIME (16)

- #define PING_STATUS_LENGTH (14)

Functions

- unsigned short update_crc (unsigned short crc_accum, unsigned char ∗data_blk_ptr, unsigned short data↩ _blk_size)

4.5.1 Detailed Description

Contains the dynamixel and dynamixel2 classes implementation.

4.5.2 Macro Definition Documentation

4.5.2.1 #define LATENCY_TIME (16)

4.5.2.2 #define PING_STATUS_LENGTH (14)

4.5.3 Function Documentation

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

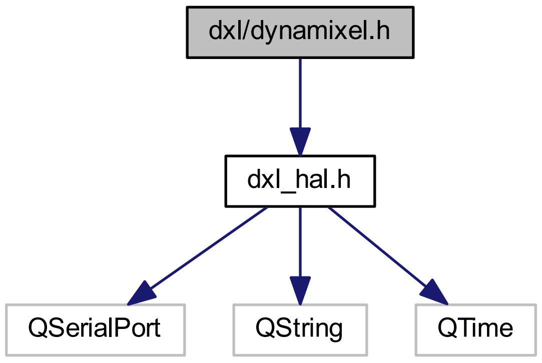
```
{
    unsigned short i, j;
    unsigned short crc_table[256] = {0x0000,
                                    0x8005, 0x800F, 0x000A, 0x801B, 0x001E, 0x0014, 0x8011,
                                    0x8033, 0x0036, 0x003C, 0x8039, 0x0028, 0x802D, 0x8027,
                                    0x0022, 0x8063, 0x0066, 0x006C, 0x8069, 0x0078, 0x807D,
                                    0x8077, 0x0072, 0x0050, 0x8055, 0x805F, 0x005A, 0x804B,
                                    0x004E, 0x0044, 0x8041, 0x80C3, 0x00C6, 0x00CC, 0x80C9,
                                    0x00D8, 0x80DD, 0x80D7, 0x00D2, 0x00F0, 0x80F5, 0x80FF,
                                    0x00FA, 0x80EB, 0x00EE, 0x00E4, 0x80E1, 0x00A0, 0x80A5,
                                    0x80AF, 0x00AA, 0x80BB, 0x00BE, 0x00B4, 0x80B1, 0x8093,
                                    0x0096, 0x009C, 0x8099, 0x0088, 0x808D, 0x8087, 0x0082,
                                    0x8183, 0x0186, 0x018C, 0x8189, 0x0198, 0x819D, 0x8197,
                                    0x0192, 0x01B0, 0x81B5, 0x81BF, 0x01BA, 0x81AB, 0x01AE,
                                    0x01A4, 0x81A1, 0x01E0, 0x81E5, 0x81EF, 0x01EA, 0x81FB,
                                    0x01FE, 0x01F4, 0x81F1, 0x81D3, 0x01D6, 0x01DC, 0x81D9,
                                    0x01C8, 0x81CD, 0x81C7, 0x01C2, 0x0140, 0x8145, 0x814F,
                                    0x014A, 0x815B, 0x015E, 0x0154, 0x8151, 0x8173, 0x0176,
                                    0x017C, 0x8179, 0x0168, 0x816D, 0x8167, 0x0162, 0x8123,
                                    0x0126, 0x012C, 0x8129, 0x0138, 0x813D, 0x8137, 0x0132,
                                    0x0110, 0x8115, 0x811F, 0x011A, 0x810B, 0x010E, 0x0104,
                                    0x8101, 0x8303, 0x0306, 0x030C, 0x8309, 0x0318, 0x831D,
                                    0x8317, 0x0312, 0x0330, 0x8335, 0x833F, 0x033A, 0x832B,
                                    0x032E, 0x0324, 0x8321, 0x0360, 0x8365, 0x836F, 0x036A,
                                    0x837B, 0x037E, 0x0374, 0x8371, 0x8353, 0x0356, 0x035C,
                                    0x8359, 0x0348, 0x834D, 0x8347, 0x0342, 0x03C0, 0x83C5,
                                    0x83CF, 0x03CA, 0x83DB, 0x03DE, 0x03D4, 0x83D1, 0x83F3,
                                    0x03F6, 0x03FC, 0x83F9, 0x03E8, 0x83ED, 0x83E7, 0x03E2,
                                    0x83A3, 0x03A6, 0x03AC, 0x83A9, 0x03B8, 0x83BD, 0x83B7,
                                    0x03B2, 0x0390, 0x8395, 0x839F, 0x039A, 0x838B, 0x038E,
                                    0x0384, 0x8381, 0x0280, 0x8285, 0x828F, 0x028A, 0x829B,
                                    0x029E, 0x0294, 0x8291, 0x82B3, 0x02B6, 0x02BC, 0x82B9,
                                    0x02A8, 0x82AD, 0x82A7, 0x02A2, 0x82E3, 0x02E6, 0x02EC,
                                    0x82E9, 0x02F8, 0x82FD, 0x82F7, 0x02F2, 0x02D0, 0x82D5,
                                    0x82DF, 0x02DA, 0x82CB, 0x02CE, 0x02C4, 0x82C1, 0x8243,
                                    0x0246, 0x024C, 0x8249, 0x0258, 0x825D, 0x8257, 0x0252,
                                    0x0270, 0x8275, 0x827F, 0x027A, 0x826B, 0x026E, 0x0264,
                                    0x8261, 0x0220, 0x8225, 0x822F, 0x022A, 0x823B, 0x023E,
                                    0x0234, 0x8231, 0x8213, 0x0216, 0x021C, 0x8219, 0x0208,
                                    0x820D, 0x8207, 0x0202 };

    for(j = 0; j < data_blk_size; j++)
    {
        i = ((unsigned short)(crc_accum >> 8) ^ *data_blk_ptr++) & 0xFF;
        crc_accum = (crc_accum << 8) ^ crc_table[i];
    }

    return crc_accum;
}
```

4.6 dxl/dynamixel.h File Reference

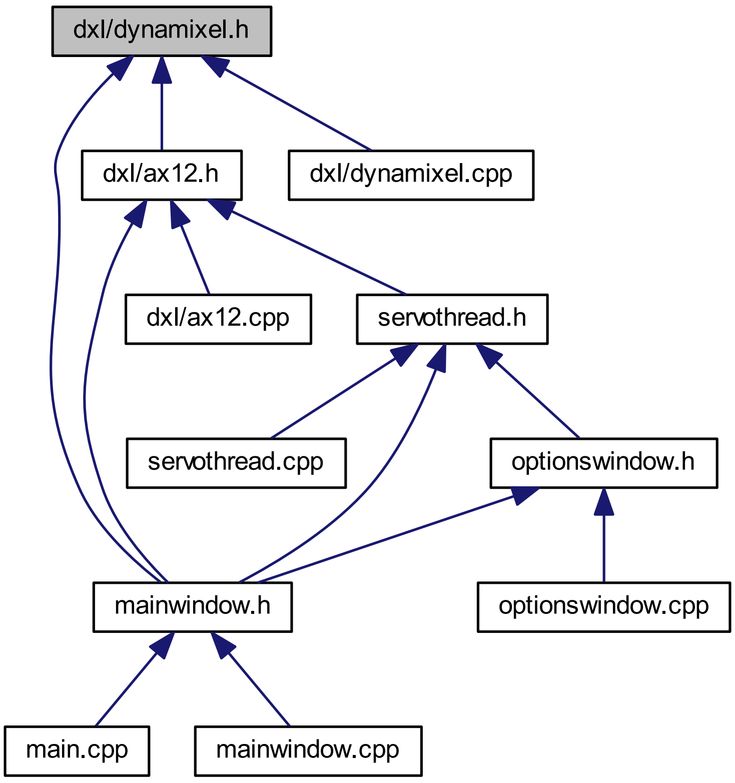
Contains the dynamixel and dynamixel2 classes declaration.

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

Include dependency graph for dynamixel.h:



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



Classes

- class dynamixel

  Dynamixel 1.0 protocol class.
- class dynamixel2

  Dynamixel 2.0 protocol class.
- struct dynamixel2::ping_data

  Struct used to do a ping.
- struct dynamixel2::data

  Struct used to handle dynamixel data.

Macros

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

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

4.6.1 Detailed Description

Contains the dynamixel and dynamixel2 classes declaration.

4.6.2 Macro Definition Documentation

4.6.2.1 #define BROADCAST_ID (254)

4.6.2.2 #define COMM_RXCORRUPT (7)

4.6.2.3 #define COMM_RXFAIL (3)

4.6.2.4 #define COMM_RXSUCCESS (1)

4.6.2.5 #define COMM_RXTIMEOUT (6)

4.6.2.6 #define COMM_RXWAITING (5)

4.6.2.7 #define COMM_TXERROR (4)

4.6.2.8 #define COMM_TXFAIL (2)

4.6.2.9 #define COMM_TXSUCCESS (0)

4.6.2.10 #define ERR_ACCESS (7)

4.6.2.11 #define ERR_CRC (3)

4.6.2.12 #define ERR_DATA_LENGTH (5)

4.6.2.13 #define ERR_DATA_LIMIT (6)

4.6.2.14 #define ERR_DATA_RANGE (4)

4.6.2.15 #define ERR_INSTRUCTION (2)

4.6.2.16 #define ERR_RESULT_FAIL (1)

4.6.2.17 #define ERRBIT_ALERT (128)

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

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

4.6.2.20 #define INST_ACTION (5)

4.6.2.21 #define INST_BULK_READ (146)

4.6.2.22 #define INST_BULK_WRITE (147)

4.6.2.23 #define INST_PING (1)

4.6.2.24 #define INST_READ (2)

4.6.2.25 #define INST_REBOOT (8)

4.6.2.26 #define INST_REG_WRITE (4)

4.6.2.27 #define INST_RESET (6)

4.6.2.28 #define INST_STATUS (85)

4.6.2.29 #define INST_SYNC_READ (130)

4.6.2.30 #define INST_SYNC_WRITE (131)

4.6.2.31 #define INST_WRITE (3)

4.6.2.32 #define LOBYTE(*w*) ((unsigned char)(((unsigned long)(w)) & 0xff))

4.6.2.33 #define LOWORD(*l*) ((unsigned short)(((unsigned long)(l)) & 0xffff))

4.6.2.34 #define MAKEDWORD(*a,* *b*) ((unsigned int)(((unsigned short)(((unsigned long)(a)) & 0xffff)) | ((unsigned int)((unsigned short)(((unsigned long)(b)) & 0xffff))) << 16))

4.6.2.35 #define MAKEWORD(*a,* *b*) ((unsigned short)(((unsigned char)(((unsigned long)(a)) & 0xff)) | ((unsigned short)((unsigned char)(((unsigned long)(b)) & 0xff))) << 8))

4.6.2.36 #define MAX_ID (252)

4.6.2.37 #define PING_INFO_FIRM_VER (2)

4.6.2.38 #define PING_INFO_MODEL_NUM (1)

4.6.2.39 #define PRT1_PKT_ERRBIT (4)

4.6.2.40 #define PRT1_PKT_ID (2)

4.6.2.41 #define PRT1_PKT_INSTRUCTION (4)

4.6.2.42 #define PRT1_PKT_LENGTH (3)

4.6.2.43 #define PRT1_PKT_PARAMETER0 (5)

4.6.2.44 #define PRT2_INSTRUCTION_PKT_PARAMETER0 (8)

4.6.2.45 #define PRT2_PKT_ERRBIT (8)

4.6.2.46 #define PRT2_PKT_HEADER0 (0)

4.6.2.47 #define PRT2_PKT_HEADER1 (1)

4.6.2.48 #define PRT2_PKT_HEADER2 (2)

4.6.2.49 #define PRT2_PKT_ID (4)

4.6.2.50 #define PRT2_PKT_INSTRUCTION (7)

4.6.2.51 #define PRT2_PKT_LENGTH_H (6)

4.6.2.52 #define PRT2_PKT_LENGTH_L (5)

4.6.2.53 #define PRT2_PKT_RESERVED (3)

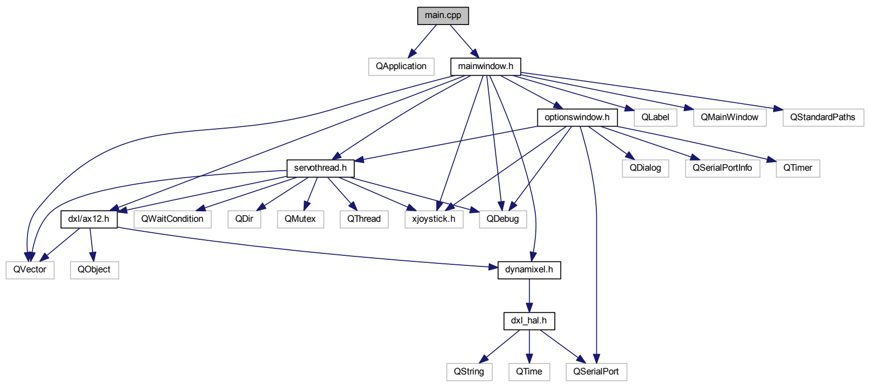
4.6.2.54 #define PRT2_STATUS_PKT_PARAMETER0 (9)

4.7 main.cpp File Reference

Contains the Main of the program.

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

Include dependency graph for main.cpp:



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[ ]*)

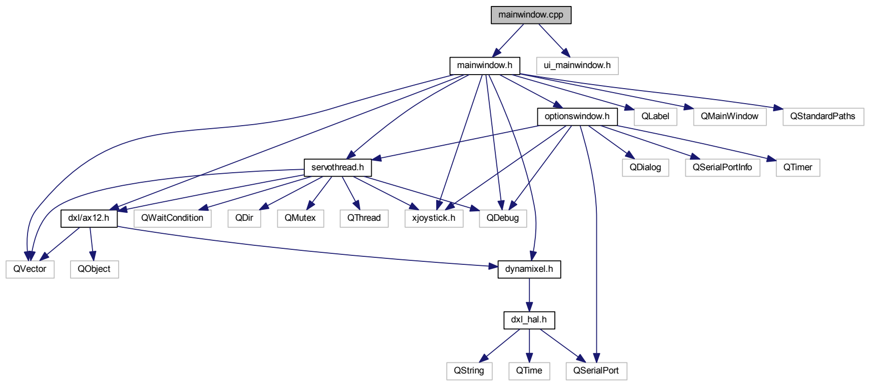
```
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}
```

4.8 mainwindow.cpp File Reference

Contains the MainWindow class implementation.

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
```

Include dependency graph for mainwindow.cpp:



4.8.1 Detailed Description

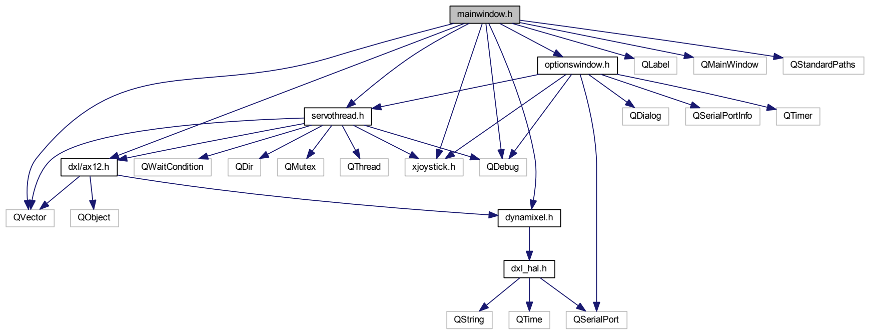
Contains the MainWindow class implementation.

4.9 mainwindow.h File Reference

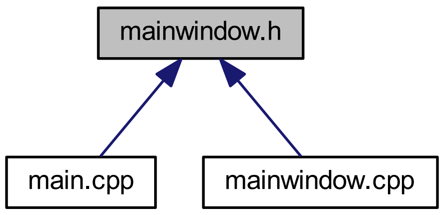
Contains the MainWindow class declaration.

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

Include dependency graph for mainwindow.h:



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



Classes

- class MainWindow

  Contains all the windows and other classes.

Namespaces

- Ui

  Namespace to work with a User Interface Qt Form.

4.9.1 Detailed Description

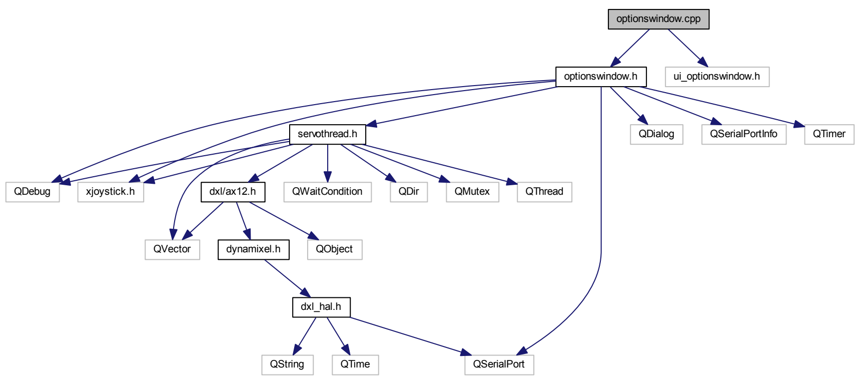
Contains the MainWindow class declaration.

4.10 optionswindow.cpp File Reference

Contains the OptionsWindow class implementation.

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

Include dependency graph for optionswindow.cpp:



4.10.1 Detailed Description

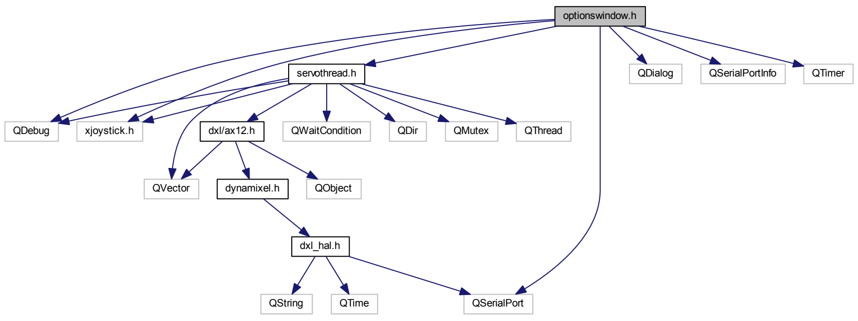
Contains the OptionsWindow class implementation.

4.11 optionswindow.h File Reference

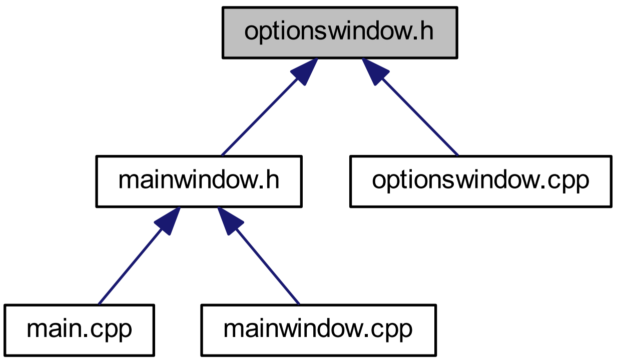
Contains the OptionsWindow class declaration.

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

Include dependency graph for optionswindow.h:



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



Classes

- class OptionsWindow

Class used to handle a Window to set the options.

Namespaces

- Ui

  Namespace to work with a User Interface Qt Form.

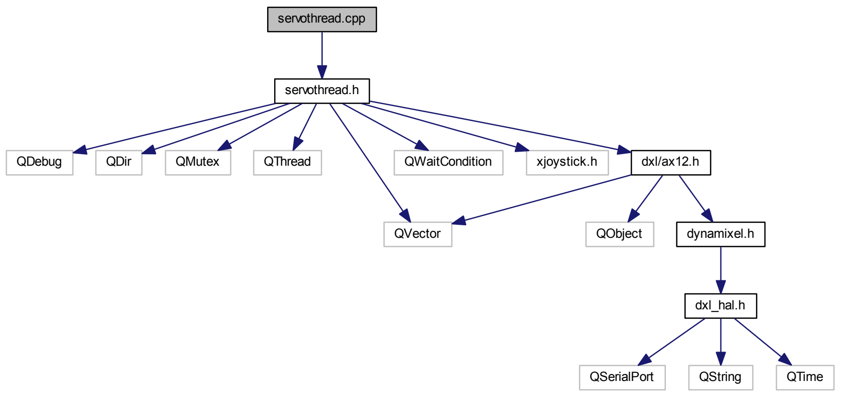
4.11.1 Detailed Description

Contains the OptionsWindow class declaration.

4.12 servothread.cpp File Reference

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

Include dependency graph for servothread.cpp:

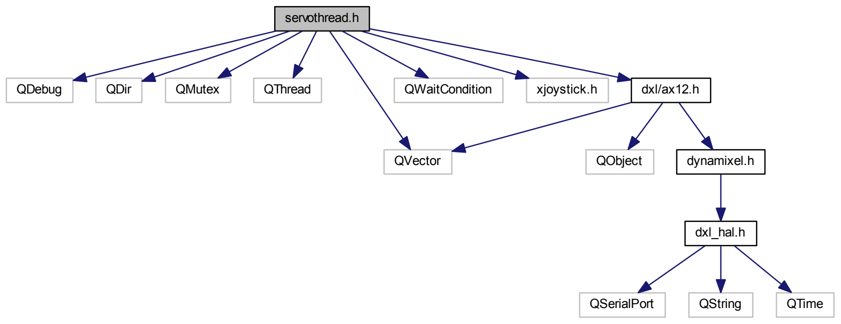


4.13 servothread.h File Reference

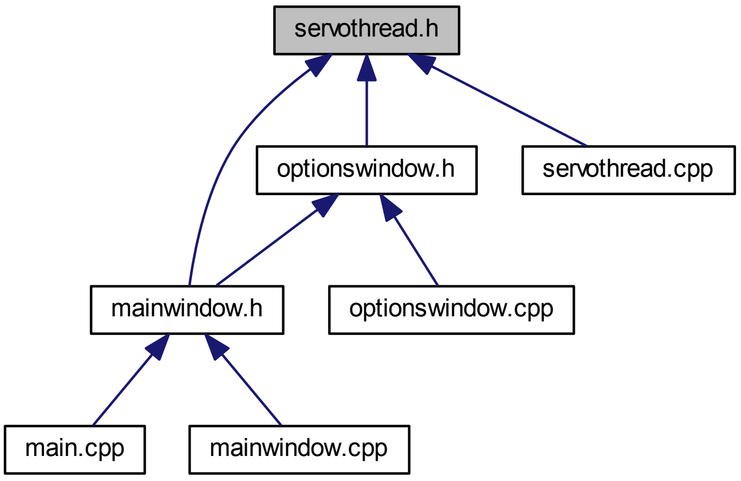
Contains the ServoThread class implementation.

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

Include dependency graph for servothread.h:



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



Classes

- class ServoThread

  The ServoThread's class handles the comunication between the delta robot servos and the PC.
- struct ServoThread::Servo

  Struct for the AX12 servos.

4.13.1 Detailed Description

Contains the ServoThread class implementation.

Contains the ServoThread class declaration.

Index