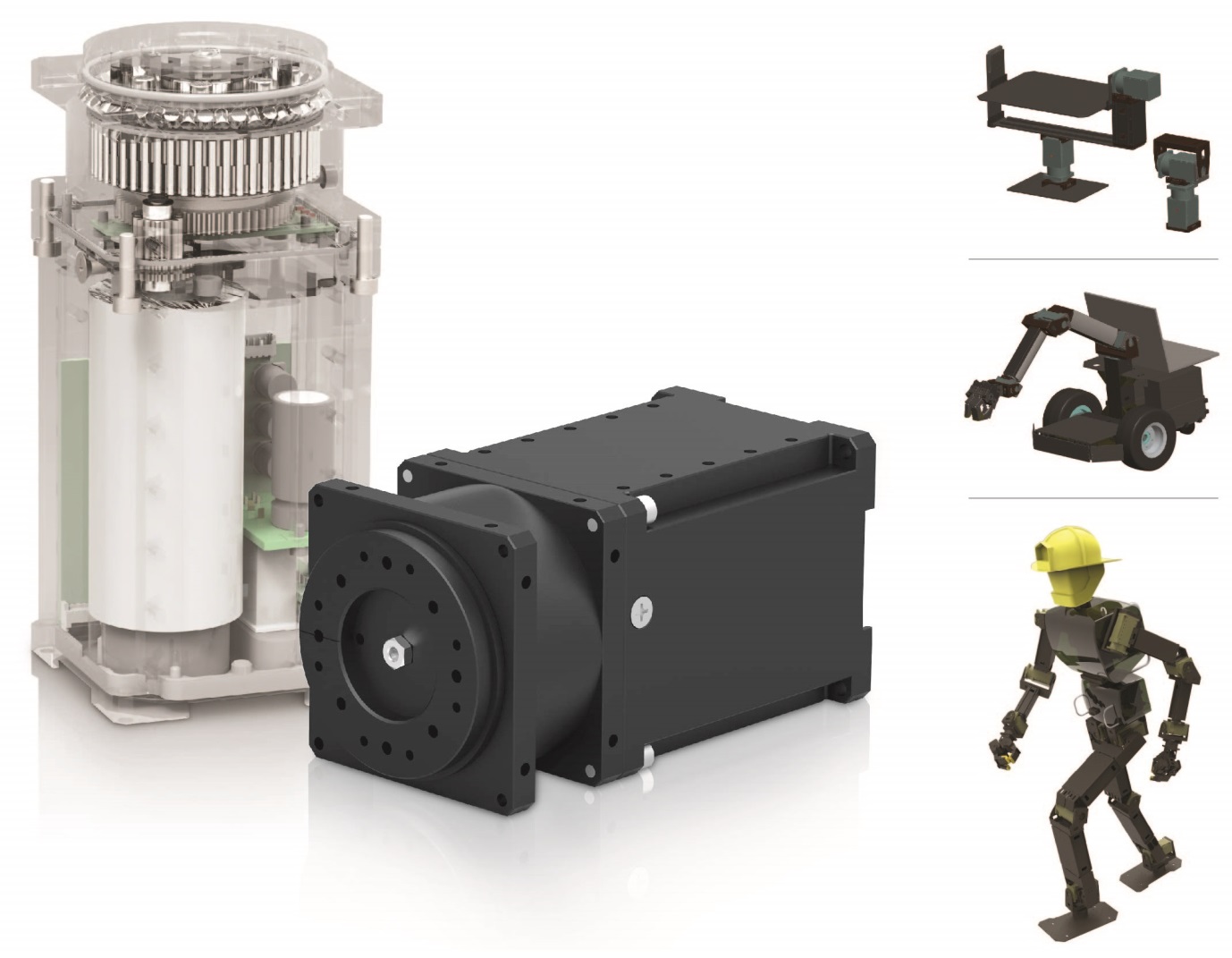
**Dynamixel SDK Manual**



http://www.robotis.com/img/img_ko/main/logo_top2.jpg

# **CONTENTS**

[**CONTENTS** 2](#_Toc444678350)

[1 Quick Overview 6](#_Toc444678351)

[1.1 Document Conventions 6](#_Toc444678352)

[1.2 Acronyms and Abbreviations 6](#_Toc444678353)

[2 Hardware Settings 6](#_Toc444678354)

[2.1 Dynamixel Controllers 6](#_Toc444678355)

[2.2 Dynamixels 6](#_Toc444678356)

[3 Software Configurations 7](#_Toc444678357)

[3.1 Source file 7](#_Toc444678358)

[3.2 Compiler 7](#_Toc444678359)

[3.3 Build Tool 7](#_Toc444678360)

[4 Dynamixel Example Codes 7](#_Toc444678361)

[4.1 Basic Codes 7](#_Toc444678362)

[4.2 Protocol1.0 10](#_Toc444678363)

[4.2.1 Ping 10](#_Toc444678364)

[4.2.2 ReadWrite 11](#_Toc444678365)

[4.2.3 MultiPort 12](#_Toc444678366)

[4.2.4 SyncWrite 14](#_Toc444678367)

[4.2.5 BulkRead 15](#_Toc444678368)

[4.2.6 Reset 16](#_Toc444678369)

[4.3 Protocol2.0 17](#_Toc444678370)

[4.3.1 Ping 17](#_Toc444678371)

[4.3.2 BroadcastPing 18](#_Toc444678372)

[4.3.3 ReadWrite 18](#_Toc444678373)

[4.3.4 MultiPort 19](#_Toc444678374)

[4.3.5 SyncReadWrite 21](#_Toc444678375)

[4.3.6 BulkReadWrite 22](#_Toc444678376)

[4.3.7 IndirectAddress 24](#_Toc444678377)

[4.3.8 Reboot 27](#_Toc444678378)

[4.3.9 FactoryReset 27](#_Toc444678379)

[4.4 ProtocolCombined 28](#_Toc444678380)

[4.5 DXLMonitor 30](#_Toc444678381)

[5 API Reference 30](#_Toc444678382)

[5.1 PortHandler 31](#_Toc444678383)

[5.2 PortHandlerLinux 32](#_Toc444678384)

[5.2.1 OpenPort 32](#_Toc444678385)

[5.2.2 ClosePort 32](#_Toc444678386)

[5.2.3 ClearPort 33](#_Toc444678387)

[5.2.4 SetPortName 33](#_Toc444678388)

[5.2.5 GetPortName 33](#_Toc444678389)

[5.2.6 SetBaudrate 34](#_Toc444678390)

[5.2.7 GetBaudrate 34](#_Toc444678391)

[5.2.8 GetBytesAvailable 34](#_Toc444678392)

[5.2.9 ReadPort 34](#_Toc444678393)

[5.2.10 WritePort 35](#_Toc444678394)

[5.2.11 SetPacketTimeout 35](#_Toc444678395)

[5.2.12 IsPacketTimeout 35](#_Toc444678396)

[5.3 PacketHandler 36](#_Toc444678397)

[5.4 Protocol1PacketHandler 37](#_Toc444678398)

[5.4.1 GetProtocolVersion 38](#_Toc444678399)

[5.4.2 TxPacket 39](#_Toc444678400)

[5.4.3 RxPacket 39](#_Toc444678401)

[5.4.4 TxRxPacket 39](#_Toc444678402)

[5.4.5 Ping 40](#_Toc444678403)

[5.4.6 BroadcastPing 40](#_Toc444678404)

[5.4.7 Action 41](#_Toc444678405)

[5.4.8 Reboot 41](#_Toc444678406)

[5.4.9 FactoryReset 41](#_Toc444678407)

[5.4.10 ReadTx 42](#_Toc444678408)

[5.4.11 ReadRx 42](#_Toc444678409)

[5.4.12 ReadTxRx 42](#_Toc444678410)

[5.4.13 Read1ByteTx 43](#_Toc444678411)

[5.4.14 Read1ByteRx 43](#_Toc444678412)

[5.4.15 Read1ByteTxRx 43](#_Toc444678413)

[5.4.16 Read2ByteTx 44](#_Toc444678414)

[5.4.17 Read2ByteRx 44](#_Toc444678415)

[5.4.18 Read2ByteTxRx 45](#_Toc444678416)

[5.4.19 Read4ByteTx 45](#_Toc444678417)

[5.4.20 Read4ByteRx 45](#_Toc444678418)

[5.4.21 Read4ByteTxRx 46](#_Toc444678419)

[5.4.22 WriteTxOnly 46](#_Toc444678420)

[5.4.23 WriteTxRx 46](#_Toc444678421)

[5.4.24 Write1ByteTxOnly 47](#_Toc444678422)

[5.4.25 Write1ByteTxRx 47](#_Toc444678423)

[5.4.26 Write2ByteTxOnly 48](#_Toc444678424)

[5.4.27 Write2ByteTxRx 48](#_Toc444678425)

[5.4.28 Write4ByteTxOnly 48](#_Toc444678426)

[5.4.29 Write4ByteTxRx 49](#_Toc444678427)

[5.4.30 RegWriteTxOnly 49](#_Toc444678428)

[5.4.31 RegWriteTxRx 49](#_Toc444678429)

[5.4.32 SyncReadTx 50](#_Toc444678430)

[5.4.33 SyncWriteTxOnly 50](#_Toc444678431)

[5.4.34 BulkReadTx 51](#_Toc444678432)

[5.4.35 BulkWriteTxOnly 51](#_Toc444678433)

[5.5 Protocol2PacketHandler 51](#_Toc444678434)

[5.5.1 GetProtocolVersion 53](#_Toc444678435)

[5.5.2 TxPacket 53](#_Toc444678436)

[5.5.3 RxPacket 53](#_Toc444678437)

[5.5.4 TxRxPacket 54](#_Toc444678438)

[5.5.5 Ping 54](#_Toc444678439)

[5.5.6 BroadcastPing 54](#_Toc444678440)

[5.5.7 Action 55](#_Toc444678441)

[5.5.8 Reboot 55](#_Toc444678442)

[5.5.9 FactoryReset 55](#_Toc444678443)

[5.5.10 ReadTx 56](#_Toc444678444)

[5.5.11 ReadRx 56](#_Toc444678445)

[5.5.12 ReadTxRx 57](#_Toc444678446)

[5.5.13 Read1ByteTx 57](#_Toc444678447)

[5.5.14 Read1ByteRx 57](#_Toc444678448)

[5.5.15 Read1ByteTxRx 58](#_Toc444678449)

[5.5.16 Read2ByteTx 58](#_Toc444678450)

[5.5.17 Read2ByteRx 58](#_Toc444678451)

[5.5.18 Read2ByteTxRx 59](#_Toc444678452)

[5.5.19 Read4ByteTx 59](#_Toc444678453)

[5.5.20 Read4ByteRx 59](#_Toc444678454)

[5.5.21 Read4ByteTxRx 60](#_Toc444678455)

[5.5.22 WriteTxOnly 60](#_Toc444678456)

[5.5.23 WriteTxRx 61](#_Toc444678457)

[5.5.24 Write1ByteTxOnly 61](#_Toc444678458)

[5.5.25 Write1ByteTxRx 61](#_Toc444678459)

[5.5.26 Write2ByteTxOnly 62](#_Toc444678460)

[5.5.27 Write2ByteTxRx 62](#_Toc444678461)

[5.5.28 Write4ByteTxOnly 62](#_Toc444678462)

[5.5.29 Write4ByteTxRx 63](#_Toc444678463)

[5.5.30 RegWriteTxOnly 63](#_Toc444678464)

[5.5.31 RegWriteTxRx 64](#_Toc444678465)

[5.5.32 SyncReadTx 64](#_Toc444678466)

[5.5.33 SyncWriteTxOnly 64](#_Toc444678467)

[5.5.34 BulkReadTx 65](#_Toc444678468)

[5.5.35 BulkWriteTxOnly 65](#_Toc444678469)

[5.6 GroupSyncRead 66](#_Toc444678470)

[5.6.1 AddParam 66](#_Toc444678471)

[5.6.2 RemoveParam 66](#_Toc444678472)

[5.6.3 ClearParam 67](#_Toc444678473)

[5.6.4 TxPacket 67](#_Toc444678474)

[5.6.5 RxPacket 67](#_Toc444678475)

[5.6.6 TxRxPacket 68](#_Toc444678476)

[5.6.7 GetData 68](#_Toc444678477)

[5.7 GroupSyncWrite 68](#_Toc444678478)

[5.7.1 AddParam 69](#_Toc444678479)

[5.7.2 RemoveParam 69](#_Toc444678480)

[5.7.3 ChangeParam 69](#_Toc444678481)

[5.7.4 ClearParam 70](#_Toc444678482)

[5.7.5 TxPacket 70](#_Toc444678483)

[5.8 GroupBulkRead 70](#_Toc444678484)

[5.8.1 AddParam 71](#_Toc444678485)

[5.8.2 RemoveParam 71](#_Toc444678486)

[5.8.3 ClearParam 71](#_Toc444678487)

[5.8.4 TxPacket 72](#_Toc444678488)

[5.8.5 RxPacket 72](#_Toc444678489)

[5.8.6 TxRxPacket 72](#_Toc444678490)

[5.8.7 GetData 73](#_Toc444678491)

[5.9 GroupBulkWrite 73](#_Toc444678492)

[5.9.1 AddParam 73](#_Toc444678493)

[5.9.2 RemoveParam 74](#_Toc444678494)

[5.9.3 ChangeParam 74](#_Toc444678495)

[5.9.4 ClearParam 74](#_Toc444678496)

[5.9.5 TxPacket 75](#_Toc444678497)

1. Quick Overview

ROBOTIS® Dynamixel SDK, or SDK, is a software development library that provides Dynamixel control functions for the packet communication. The API Library covers Dynamixel actuators and Dynamixel based platforms. It assumes that you are familiar with C++ programming. If you need to familiarize yourself with the Dynamixel, see e-manual uploaded onto the user support page ([support.robotis.com/](http://support.robotis.com/en/)). It provides various informations over all ROBOTIS Products and applications.

This document describes the SDK from actual applications to API. The chapter 2 and 3 shows how to set the environment for the development with Dynamixel. The chapter 4 may give a quick look on how Dynamixel operates by using SDK with examples. The chapter 5 exposes SDK API references.

* 1. Document Conventions

The document generally exposes the SDK API references in an itemized form. All code-related items appear in the Consolas typeface, such as OpenPort(), except the items in the table of contents.

* 1. Acronyms and Abbreviations

|  |  |
| --- | --- |
| API | Application Programming Interface |
| Controller | Desktop, Single Board Computer (SBC), etc. |
| Baudrate | Communication Baudrate |
| DXL | Dynamixel |
| MX | Dynamixel MX series |
| PRO | Dynamixel PRO series |

1. Hardware Settings
   1. Dynamixel Controllers

Dynamixels use either or both types of signal interface:

* TTL
* RS485.

(Please check which signal interface is applicated in your Dynamixel.)

Whatever the converter is used with, if it generates TTL or RS485 signal, the controller has a chance to handle the Dynamixel. The SDK and examples is coded based on using USB port with the ROBOTIS® USB2Dynamixel signal converter. Since the other signal ports, such as UART, I2C, etc., are in various conditions depending on the controller specification, modify SDK yourselves and adapt to your own controllers.

Although, SDK was tested using Dynamixel with common Desktops and SBCs, such as Raspberry Pi and ODROID on the Linux (Ubuntu 11.04 or later, lUbuntu, Raspbian, Ubuntu Mate, etc.) operating systems. The SDK for Windows would be supported soon.

[Note: Before using USB2Dynamixel, check FTDI driver compatibility on your controller.]

* 1. Dynamixels

SDK and examples were developed for Dynamixel MX-28 and Dynamixel PRO H54-200. They need the power supplement as follows:

* MX at 12V
* PRO at 24V.

To test the SDK examples, Dynamixel settings needs to be changed to:

* ID = 1 (and 2, when the example uses two Dynamixels)
* Baudrate = 1000000. (baudnum = 1 for MX, baudnum = 3 for Dynamixel PRO.)

To modify Dynamixel settings, use ROBOTIS® RoboPlus – Dynamixel Wizard. The instruction video is uploaded on the e-manual – Software Help – Dynamixel Wizard.

1. Software Configurations
   1. Source file

Download unified-dynamixel-sdk-vX.Y.zip from the Github:

http://github.com/ROBOTIS-GIT/

* 1. Compiler

Windows:

Dynamixel SDK for Windows would be supported soon.

Linux:

C : GNU gcc (Source is now being prepared)

C++: GNU g++

* 1. Build Tool

Windows:

Dynamixel SDK for Windows would be supported soon.

Linux:

Build-essential pkg → make

1. Dynamixel Example Codes
   1. Basic Codes
      * + 1. Description

These codes are used in all examples. Modify them on your needs, for instance, in case of using more than one Dynamixel, port, Protocol.

* + - * 1. Simple code

|  |
| --- |
| Refers to all examples |
| /\*\* Put HEADERS here \*\*/  /\* Headers \*/  #include "DynamixelSDK.h "  /\*\* Put DEFINITIONS here \*\*/  /\* Default Setting \*/  #define DXL\_ID 1  #define BAUDRATE 1000000  /\* Protocol Version \*/  #define PROTOCOL\_VERSION 1.0  /\* Device Name of Port (uses USB2Dynamixel on Linux) \*/  #define DEVICE\_NAME “/dev/ttyUSB0”  /\*\* Put DEBUG FUNCTIONS here (if necessary) \*\*/  int main()  {  /\*\* Put VARIABLES here \*\*/  UINT8\_T dxl\_error = 0; /\* DXL error \*/    /\*\* Put HANDLERS here \*\*/  /\* gets methods and members of PortHandler Linux or Windows, and sets DEVICE\_NAME \*/  PortHandler \*portHandler = PortHandler::GetPortHandler(“DEVICE\_NAME”);  /\* packetHandler gets methods and members of **Protocol1 and Protocol2 PacketHandler** \*/  PacketHandler \*packetHandler = PacketHandler::GetPacketHandler(PROTOCOL\_VERSION);    /\* portHandler opens and gets the handle of selected port \*/  portHandler->OpenPort();  /\* portHandler retries to open the selected port with selected BAUDRATE \*/  portHandler->SetBaudrate(BAUDRATE);  /\*\* Put FUNCTIONS here \*\*/  /\* packetHandler closes selected port \*/  portHandler.ClosePort();  } |

To debug

In the example codes, there are some functions defined as a debug tool. The code shown below prints out messages about what kind of error is occurred.

|  |
| --- |
| PrintCommStatus function reference |
| void PrintCommStatus(int dxl\_comm\_result)  {  switch(dxl\_comm\_result)  {  case COMM\_PORT\_BUSY: // COMM\_PORT\_BUSY = -1000  printf(“COMM\_PORT\_BUSY: Port is in use!\n”);  break;  case COMM\_TX\_FAIL: // COMM\_TX\_FAIL = -1001  printf(“COMM\_TX\_FAIL: Failed transmit instruction packet!\n”);  break;  case COMM\_RX\_FAIL: // COMM\_RX\_FAIL = -1002  printf(“COMM\_RX\_FAIL: failed get status packet from device!\n”);  break;  case COMM\_TX\_ERROR: // COMM\_TX\_ERROR = -2000  printf(“COMM\_TX\_ERROR: Incorrect instruction packet!\n”);  break;  case COMM\_RX\_WAITING: // COMM\_RX\_WAITING = -3000  printf(“COMM\_RX\_WAITING: Now receiving status packet!\n);  break;  case COMM\_RX\_TIMEOUT: // COMM\_RX\_TIMEOUT = -3001  printf(“COMM\_TX\_TIMEOUT: There is no status packet!\n”);  break;  case COMM\_RX\_CORRUPT: // COMM\_RX\_CORRUPT = -3002  printf(“COMM\_RX\_CORRUPT: Incorrect status packet!\n”);  break;  case COMM\_NOT\_AVAILABLE: // COMM\_NOT\_AVAILABLE = -9000  printf(“COMM\_NOT\_AVAILABLE: Protocol does not support this  function!\n”);  break;  default:  printf(“This is unknown error code!\n”);  break;  }  } |

|  |
| --- |
| PrintErrorCode\_ForProtocol1 function reference |
| void PrintErrorCode\_ForMX(int dxl\_error)  {  switch(ErrorCode)  {  case ERRBIT\_PROTOCOL1\_VOLTAGE: // ERRBIT\_MX\_VOLTAGE = 1  printf("Input voltage error!\n");  break;  case ERRBIT\_PROTOCOL1\_ANGLE: // ERRBIT\_MX\_ANGLE = 2  printf("Angle limit error!\n");  break;  case ERRBIT\_PROTOCOL1\_OVERHEAT: // ERRBIT\_MX\_OVERHEAT = 4  printf("Overheat error!\n");  break;  case ERRBIT\_PROTOCOL1\_RANGE: // ERRBIT\_MX\_RANGE = 8  printf("Out of range error!\n");  break;  case ERRBIT\_PROTOCOL1\_CHECKSUM: // ERRBIT\_MX\_CHECKSUM = 16  printf("Checksum error!\n");  break;  case ERRBIT\_PROTOCOL1\_OVERLOAD: // ERRBIT\_MX\_OVERLOAD = 32  printf("Overload error!\n");  break;  case ERRBIT\_PROTOCOL1\_INSTRUCTION: // ERRBIT\_MX\_INSTRUCTION = 64  printf("Instruction code error!\n");  break;  default:  printf("Unknown error code!\n");  break;  }  } |

|  |
| --- |
| PrintErrorCode\_ForProtocol2 function reference |
| void PrintErrorCode\_ForPro(int dxl\_error)  {  if(ErrorCode & 0x80)  printf("Hardware error occurred. Check the error at Control Table ADDRESS: 892!\n");  switch(ErrorCode)  {  case ERRBIT\_PROTOCOL2\_RESULTFAIL: // ERRBIT\_PROTOCOL2\_RESULTFAIL = 1  printf("Failed to deal with instruction packet!\n");  break;  case ERRBIT\_PROTOCOL2\_INSTRUCTION: // ERRBIT\_PROTOCOL2\_INSTRUCTION = 2  printf("Undefined instruction!\n");  break;  case ERRBIT\_PROTOCOL2\_CRC: // ERRBIT\_PROTOCOL2\_CRC = 4  printf("CRC doesn't match!\n");  break;  case ERRBIT\_PROTOCOL2\_DATARANGE: // ERRBIT\_PROTOCOL2\_DATARANGE = 8  printf("Data is out of range!\n");  break;  case ERRBIT\_PROTOCOL2\_DATALENGTH: // ERRBIT\_PROTOCOL2\_DATALENGTH = 16  printf("Data is shorter than expected!\n");  break;  case ERRBIT\_PROTOCOL2\_DATALIMIT: // ERRBIT\_PROTOCOL2\_DATALIMIT = 32  printf("Data is too long!\n");  break;  case ERRBIT\_PROTOCOL2\_ADDRESS: // ERRBIT\_PROTOCOL2\_ADDRESS = 64  printf("Writing or Reading is prohibited on target address!\n");  break;  default:  printf("Unknown error code!\n");  break;  }  } |

To use debug functions, put each code parts on to the directed location in Basic Codes.

|  |
| --- |
| Debug code example |
| /\*\* Put below codes to debug functions \*\*/  /\* Device <-> Dynamixel communication result \*/  PortHandler \*portHandler = PortHandler::GetPortHandler(“DEVICE\_NAME”);  /\*\* Put below codes to additional VARIABLES \*\*/  /\* Device <-> Dynamixel communication result \*/  int dxl\_comm\_result  /\*\* Put below codes to DEBUG FUNCTIONS \*\*/  dxl\_comm\_result = packetHandler->Write1ByteTxRx(portHandler, DXL\_ID, ADDR\_MX\_TORQUE\_ENABLE  , dxl\_torque\_enable, &dxl\_error); // This is an example  if(dxl\_comm\_result == COMM\_SUCCESS)  {  if(dxl\_error != 0)  {  PrintErrorCode\_ForMX(dxl\_error); // or PrintErrorCode\_ForPRO(dxl\_error);  return 0;  }  }  else  {  PrintCommStatus(dxl\_comm\_result);  return 0;  } |

* 1. Protocol1.0
     1. Ping
        + 1. Description

This example tries to ping Dynamixel and shows the model number received from Dynamixel. The example is commonly used to check for Dynamixel connection or to get model number to refer its specification.

* + - * 1. Available Dynamixel

All series using protocol 1.0

* + - * 1. Code Parts

|  |
| --- |
| Refers to “/Protocol1.0/Ping/Ping.cpp” |
| /\*\* Put below codes to additional VARIABLES \*\*/  UINT16\_T dxl\_model\_number = 0; // DXL model number  /\*\* Put below codes to additional FUNCTIONS \*\*/  /\* Try to ping the DXL\_ID and gets dxl\_model\_number \*/  packetHandler->Ping(portHandler, DXL\_ID, &dxl\_model\_number, &dxl\_error); |

* + 1. ReadWrite
       - 1. Description

This example writes goal position of the Dynamixel and repeats read present position until it stops moving. The funtions that are related with the Read and the Write handle the number of items which are near each other in the Dynamixel control table, such as the goal position and the goal velocity.

* + - * 1. Available DXL

All series using protocol 1.0

* + - * 1. Code Parts

|  |
| --- |
| Refers to “/Protocol1.0/ReadWrite/ReadWrite.cpp” |
| /\*\* Put below codes to additional DEFINITIONS \*\*/  /\* Control table address \*/  #define ADDR\_MX\_TORQUE\_ENABLE 24  #define ADDR\_MX\_GOAL\_POSITION 30  #define ADDR\_MX\_PRESENT\_POSITION 36  #define ADDR\_MX\_MOVING 46  /\*\* Put below codes to additional VARIABLES \*\*/  int index = 0; // Index for rotation direction  int dxl\_goal\_position[2] = {100, 1000}; // Goal positions  UINT8\_T dxl\_torque\_enable = 1; // Value for torque enable  UINT8\_T dxl\_torque\_disable = 0; // Value for torque disable  UINT8\_T dxl\_moving = 0; // DXL moving status  UINT16\_T dxl\_present\_position = 0; // Present position  /\*\* Put below codes to additional FUNCTIONS \*\*/  /\* packetHandler does dxl\_torque\_enable to DXL\_ID, and receives dxl\_error \*/  packetHandler->Write1ByteTxRx(portHandler, DXL\_ID, ADDR\_MX\_TORQUE\_ENABLE  , dxl\_torque\_enable, &dxl\_error);  /\* Writes dxl\_goal\_position to DXL\_ID. The torque should be enabled previously \*/  packetHandler->Write2ByteTxRx(portHandler, DXL\_ID, ADDR\_MX\_GOAL\_POSITION  , dxl\_goal\_position[index], &dxl\_error);  /\* Reads dxl\_present\_position from DXL\_ID. It gives value after this is called \*/  packetHandler->Read2ByteTxRx(portHandler, DXL\_ID, ADDR\_MX\_PRESENT\_POSITION  , &dxl\_present\_position, &dxl\_error);  /\* Reads dxl\_moving status. If Dynamixel is moving, it returns 1, and if not, 0 \*/  packetHandler->Read1ByteTxRx(portHandler, DXL\_ID, ADDR\_MX\_MOVING  , &dxl\_moving, &dxl\_error);  /\* packetHandler does dxl\_torque\_disable to DXL\_ID \*/  packetHandler->Write1ByteTxRx(portHandler, DXL\_ID, ADDR\_MX\_TORQUE\_ENABLE  , dxl\_torque\_disable, &dxl\_error); |

* + 1. MultiPort
       - 1. Description

This example writes goal position of Dynamixels connected with two ports, and reads their present position until Dynamixels stop moving.

* + - * 1. Available Dynamixel

All series using protocol 1.0

* + - * 1. Basic code for MultiPort example

|  |
| --- |
| Refers to “/Protocol1.0/Multiport/Multiport.cpp” |
| /\*\* Put additional HEADERS here \*\*/  /\* Headers \*/  #include "dynamixel\_sdk/PortHandler.h"  #include "dynamixel\_sdk/PacketHandler.h"  /\* Default Setting \*/  #define DXL1\_ID 1  #define DXL2\_ID 2  #define BAUDRATE 1000000  /\* Protocol Version \*/  #define PROTOCOL\_VERSION 1.0  /\* Device Name of Port (uses USB2Dynamixel on Linux) \*/  #define DEVICE\_NAME\_1 “/dev/ttyUSB0”  #define DEVICE\_NAME\_2 “/dev/ttyUSB1”  /\*\* Put additional DEFINITIONS here \*\*/  /\* gets methods and members of PortHandler Linux and Windows, and sets DEVICE\_NAME \*/  PortHandler \*portHandler1 = (PortHandler\*)PortHandler::GetPortHandler(“DEVICE\_NAME\_1”);  PortHandler \*portHandler2 = (PortHandler\*)PortHandler::GetPortHandler(“DEVICE\_NAME\_2”);  /\*\* Put DEBUG FUNCTIONS here (if necessary) \*\*/  int main()  {  UINT8\_T dxl\_error = 0; /\* DXL error \*/    /\*\* Put additional VARIABLES here \*\*/  /\* portHandler opens and gets the handle of selected port \*/  portHandler1->OpenPort();  portHandler2->OpenPort();  /\* portHandler retries to open the selected port with selected BAUDRATE \*/  portHandler1->SetBaudrate(BAUDRATE);  portHandler2->SetBaudrate(BAUDRATE);  /\* packetHandler gets methods and members of **Protocol1 and Protocol2 PacketHandler** \*/  PacketHandler \*packetHandler = PacketHandler::GetPacketHandler(PROTOCOL\_VERSION);    /\*\* Put additional FUNCTIONS here \*\*/  /\* packetHandler closes selected port \*/  portHandler1.ClosePort();  portHandler2.ClosePort();  } |

* + - * 1. Simple code

|  |
| --- |
| Refers to “/Protocol1.0/Multiport/Multiport.cpp” |
| /\*\* Put below codes to additional DEFINITIONS \*\*/  /\* Control table address \*/  #define ADDR\_MX\_TORQUE\_ENABLE 24  #define ADDR\_MX\_GOAL\_POSITION 30  #define ADDR\_MX\_PRESENT\_POSITION 36  /\*\* Put below codes to additional VARIABLES \*\*/  int index = 0; // Index for rotation direction  int dxl\_goal\_position[2] = {100, 1000}; // Goal positions  UINT8\_T dxl\_torque\_enable = 1; // Value for torque enable  UINT8\_T dxl\_torque\_disable = 0; // Value for torque disable  UINT16\_T dxl\_present\_position1 = 0, dxl\_present\_position2 = 0;// Present position  /\*\* Put below codes to additional FUNCTIONS \*\*/  /\* packetHandler does dxl\_torque\_enable to DXL\_ID, and receives dxl\_error \*/  packetHandler->Write1ByteTxRx(portHandler1, DXL1\_ID, ADDR\_MX\_TORQUE\_ENABLE  , dxl\_torque\_enable, &dxl\_error);  packetHandler->Write1ByteTxRx(portHandler2, DXL2\_ID, ADDR\_MX\_TORQUE\_ENABLE  , dxl\_torque\_enable, &dxl\_error);  /\* Writes dxl\_goal\_position to DXL\_ID. The torque should be enabled previously \*/  packetHandler->Write2ByteTxRx(portHandler1, DXL1\_ID, ADDR\_MX\_GOAL\_POSITION  , dxl\_goal\_position[index], &dxl\_error);  packetHandler->Write2ByteTxRx(portHandler2, DXL2\_ID, ADDR\_MX\_GOAL\_POSITION  , dxl\_goal\_position[index], &dxl\_error);  /\* Reads dxl\_present\_position from DXLs. It gives value after this is called \*/  packetHandler->Read2ByteTxRx(portHandler1, DXL1\_ID, ADDR\_MX\_PRESENT\_POSITION  , &dxl\_present\_position1, &dxl\_error);  packetHandler->Read2ByteTxRx(portHandler2, DXL2\_ID, ADDR\_MX\_PRESENT\_POSITION  , &dxl\_present\_position2, &dxl\_error);  /\* packetHandler does dxl\_torque\_disable to DXLs \*/  packetHandler->Write1ByteTxRx(portHandler1, DXL1\_ID, ADDR\_MX\_TORQUE\_ENABLE  , dxl\_torque\_disable, &dxl\_error);  packetHandler->Write1ByteTxRx(portHandler2, DXL2\_ID, ADDR\_MX\_TORQUE\_ENABLE  , dxl\_torque\_disable, &dxl\_error); |

* + 1. SyncWrite
       - 1. Description

This example writes goal positions to two Dynamixels simultaneously and repeats to read each present positions until Dynamixels stop moving. The funtions that are related with the Syncwrite handle the number of items that are near to each other in the Dynamixel control table on multiple Dynamixels, such as the goal position and the goal velocity.

* + - * 1. Available Dynamixel

All series using protocol 1.0

* + - * 1. Code Parts

|  |
| --- |
| Refers to “/Protocol1.0/SyncWrite/SyncWrite.cpp” |
| /\*\* Put below codes to additional HEADERS \*\*/  /\* Header \*/  #include "dynamixel\_sdk/GroupSyncWrite.h"  /\*\* Put below codes to additional DEFINITIONS \*\*/  /\* Control table address \*/  #define ADDR\_MX\_TORQUE\_ENABLE 24  #define ADDR\_MX\_GOAL\_POSITION 30  #define ADDR\_MX\_PRESENT\_POSITION 36  /\* Data byte length \*/  #define LEN\_MX\_TORQUE\_ENABLE 1  #define LEN\_MX\_GOAL\_POSITION 2  #define LEN\_MX\_PRESENT\_POSITION 2  /\*\* Put below codes to additional VARIABLES \*\*/  int index = 0; // index for rotation direction  int dxl\_goal\_position[2] = {100, 1000}; // Goal positions  UINT8\_T dxl\_torque\_enable = 1; // Value for torque enable  UINT8\_T dxl\_torque\_disable = 0; // Value for torque disable  UINT8\_T param\_goal\_position[2];  UINT16\_T dxl\_present\_position1, dxl\_present\_position2;// Present position  /\*\* Put below codes to additional FUNCTIONS \*\*/  /\* Initializes groupSyncWrite instance using portHandler and packetHandler \*/  **GroupSyncWrite** groupSyncWrite(portHandler, packetHandler  , ADDR\_MX\_GOAL\_POSITION, LEN\_MX\_GOAL\_POSITION);  /\* packetHandler does dxl\_torque\_enable to DXLs, and receives dxl\_error \*/  packetHandler->Write1ByteTxRx(portHandler, DXL1\_ID, ADDR\_MX\_TORQUE\_ENABLE  , dxl\_torque\_enable, &dxl\_error);  packetHandler->Write1ByteTxRx(portHandler, DXL2\_ID, ADDR\_MX\_TORQUE\_ENABLE  , dxl\_torque\_enable, &dxl\_error);  /\* Allocates dxl\_goal\_position into byte array \*/  param\_goal\_position[0] = DXL\_LOBYTE(dxl\_goal\_position);  param\_goal\_position[1] = DXL\_HIBYTE(dxl\_goal\_position);  /\* AddParameter param\_goal\_position groupSyncWrite parameter Storage \*/  groupSyncWrite.AddParam(DXL1\_ID, param\_goal\_position);  groupSyncWrite.AddParam(DXL2\_ID, param\_goal\_position);  /\* Transmits the packet \*/  groupSyncWrite.TxPacket();  /\* ClearParameter of groupSyncWrite parameter storage \*/  groupSyncWrite.ClearParam();  /\* Reads dxl\_present\_position from DXLs. It gives value after this is called \*/  packetHandler->Read2ByteTxRx(portHandler, DXL1\_ID, ADDR\_MX\_PRESENT\_POSITION  , &dxl\_present\_position1, &dxl\_error);  packetHandler->Read2ByteTxRx(portHandler, DXL2\_ID, ADDR\_MX\_PRESENT\_POSITION  , &dxl\_present\_position2, &dxl\_error);  /\* packetHandler does dxl\_torque\_disable to DXLs, and receives dxl\_error \*/  packetHandler->Write1ByteTxRx(portHandler, DXL1\_ID, ADDR\_MX\_TORQUE\_ENABLE  , dxl\_torque\_disable, &dxl\_error);  packetHandler->Write1ByteTxRx(portHandler, DXL2\_ID, ADDR\_MX\_TORQUE\_ENABLE  , dxl\_torque\_disable, &dxl\_error); |

* + 1. BulkRead
       - 1. Description

This example writes goal position of two Dynamixels and simulateously reads their positions until Dynamixel stops moving. The functions that are related with the Bulkread function handle the number of items which are not near to each other in the Dynamixel control table on multiple Dynamixels.

* + - * 1. Available Dynamixel

MX or X series for protocol 1.0

* + - * 1. Code Parts

|  |
| --- |
| Refers to “/Protocol1.0/BulkRead/BulkRead.cpp” |
| /\*\* Put below codes to additional HEADERS \*\*/  /\* Header \*/  #include "dynamixel\_sdk/GroupBulkRead.h"  /\*\* Put below codes to additional DEFINITIONS \*\*/  /\* Control table address \*/  #define ADDR\_MX\_TORQUE\_ENABLE 24  #define ADDR\_MX\_GOAL\_POSITION 30  #define ADDR\_MX\_PRESENT\_POSITION 36  #define ADDR\_MX\_MOVING 46  /\* Data byte length \*/  #define LEN\_MX\_TORQUE\_ENABLE 1  #define LEN\_MX\_GOAL\_POSITION 2  #define LEN\_MX\_PRESENT\_POSITION 2  #define LEN\_MX\_MOVING 1  /\*\* Put below codes to additional VARIABLES \*\*/  int index = 0; // index for rotation direction  int dxl\_goal\_position[2] = {100, 1000}; // Goal positions  UINT8\_T dxl\_torque\_enable = 1; // Value for torque enable  UINT8\_T dxl\_torque\_disable = 0; // Value for torque disable  UINT8\_T dxl\_moving2; // DXL#2 moving status  UINT16\_T dxl\_present\_position1; // DXL#1 present position  /\*\* Put below codes to additional FUNCTIONS \*\*/  // Initialize GroupBulkRead instance using portHandler and packetHandler  **GroupBulkRead** groupBulkRead(portHandler, packetHandler);  /\* packetHandler does dxl\_torque\_enable to DXLs, and receives dxl\_error \*/  packetHandler->Write1ByteTxRx(portHandler, DXL1\_ID, ADDR\_MX\_TORQUE\_ENABLE  , dxl\_torque\_enable, &dxl\_error);  packetHandler->Write1ByteTxRx(portHandler, DXL2\_ID, ADDR\_MX\_TORQUE\_ENABLE  , dxl\_torque\_enable, &dxl\_error);  // AddParameter parameter storage for Dynamixel’s present\_goal\_position to groupBulkRead position value  groupBulkRead.AddParam(DXL1\_ID, param\_present\_position);  groupBulkRead.AddParam(DXL2\_ID, &dxl\_moving);    // Write goal position  packetHandler->Write2ByteTxRx(portHandler, DXL1\_ID, ADDR\_MX\_GOAL\_POSITION  , dxl\_goal\_position[index], &dxl\_error);  packetHandler->Write2ByteTxRx(portHandler, DXL2\_ID, ADDR\_MX\_GOAL\_POSITION  , dxl\_goal\_position[index], &dxl\_error);  // Bulkread present position  groupBulkRead.TxRxPacket();  // Clear bulkread parameter storage  groupBulkRead.ClearParam();  // Get DXL#1 present position value  groupBulkRead.GetData(DXL1\_ID, ADDR\_MX\_PRESENT\_POSITION, &dxl\_present\_position1);  // Get DXL#2 moving status value  groupBulkRead.GetData(DXL2\_ID, ADDR\_MX\_MOVING, &dxl\_moving2);  // Disable DXL torque  packetHandler->Write1ByteTxRx(portHandler, DXL1\_ID, ADDR\_MX\_TORQUE\_ENABLE  , dxl\_torque\_disable, &dxl\_error);  packetHandler->Write1ByteTxRx(portHandler, DXL2\_ID, ADDR\_MX\_TORQUE\_ENABLE  , dxl\_torque\_disable, &dxl\_error); |

* + 1. Reset
       - 1. Description

This example resets all settings of Dynamixel to default values as:

ID = 1

Baudrate=57600.

* + - * 1. Available Dynamixel

All series using protocol 1.0

* + - * 1. Simple code

|  |
| --- |
| Refers to “/Protocol1.0/Reset/Reset.cpp” |
| /\*\* Put below codes to additional DEFINITIONS \*\*/  /\* Control table address \*/  #define ADDR\_MX\_BAUDRATE 4  /\* Values needed for Reset \*/  #define RESET\_DEFAULTBAUDRATE 57600 // DXL baudrate set by factoryreset  #define NEW\_BAUDNUM 1 // New baudnum for recovering DXL baudrate  #define OPERATION\_MODE 0x00 // mode is unavailable in Protocol 1.0 Reset  /\*\* Put below codes to additional VARIABLES \*\*/  int dxl\_baudnum\_read = 0; // Read baudnum  /\*\* Put below codes to additional FUNCTIONS \*\*/  /\* Read present baudrate of the controller \*/  portHandler->GetBaudRate();  /\* Try to reset. This will reset Dynamixel settings to default values \*/  packetHandler->Reset(portHandler, DXL\_ID, OPERATION\_MODE, &dxl\_error);  /\* Wait for reset \*/  usleep(2000 \* 1000);  /\* portHandler retries to open the selected port with selected RESET\_DEFAULTBAUDRATE \*/  portHandler->SetBaudrate(RESET\_DEFAULTBAUDRATE);  /\* Reads dxl\_baudnum\_read from DXL\_ID. It gives value after this is called \*/  packetHandler->Read1ByteTxRx(portHandler, DXL\_ID, ADDR\_MX\_BAUDRATE  , &dxl\_baudnum\_read, &dxl\_error);  /\* Writes NEW\_BAUDNUM to DXL\_ID \*/  packetHandler->Write1ByteTxRx(portHandler, DXL\_ID, ADDR\_MX\_BAUDRATE  , NEW\_BAUDNUM, &dxl\_error);  /\* Set port baudrate to BAUDRATE \*/  portHandler->SetBaudrate(BAUDRATE);  /\* Reads dxl\_baudnum\_read from DXL\_ID. It gives value after this is called \*/  packetHandler->Read1ByteTxRx(portHandler, DXL\_ID, ADDR\_MX\_BAUDRATE  , &dxl\_baudnum\_read, &dxl\_error); |

* 1. Protocol2.0
     1. Ping
        + 1. Description

This example tries to ping Dynamixel and shows the model number responded from Dynamixel. The example is commonly used to check for Dynamixel’s connection or to get model number to refer its specification.

* + - * 1. Available Dynamixel

All series using protocol 2.0

* + - * 1. Simple code

|  |
| --- |
| Refers to “/Protocol2.0/Ping/Ping.cpp” |
| /\*\* Put below codes to additional VARIABLES \*\*/  UINT16\_T dxl\_model\_number = 0; // DXL model number  /\*\* Put below codes to additional FUNCTIONS \*\*/  /\* Try to Ping the DXL\_ID and gets dxl\_model\_number \*/  packetHandler->Ping(portHandler, DXL\_ID, &dxl\_model\_number, &dxl\_error); |

* + 1. BroadcastPing
       - 1. Description

This example tries to ping all Dynamixels that are connected to the controller, and shows which Dynamixel is successfully connected. The example is commonly used to check for Dynamixel’s connection at once.

* + - * 1. Available Dynamixel

All series using protocol 2.0

* + - * 1. Simple code

|  |
| --- |
| Refers to “/Protocol2.0/BroadcastPing/BroadcastPing.cpp” |
| /\* Put below codes to additional HEADERS \*/  #include <vector>  /\*\* Put below codes to additional VARIABLES \*\*/  std::vector<UINT8\_T> vec;  /\*\* Put below codes to additional FUNCTIONS \*\*/  /\* Try to BroadcastPing all Dynamixels and gets the list of connected Dynamixels \*/  packetHandler->BroadcastPing(portHandler, vec); |

* + 1. ReadWrite
       - 1. Description

This example writes goal position to the Dynamixel and repeats to read present position until it stops moving. The funtions that are related with the Read and the Write handle the number of items which are near each other in the Dynamixel control table, such as the goal position and the goal velocity.

* + - * 1. Available Dynamixel

All series using protocol 2.0

* + - * 1. Simple code

|  |
| --- |
| Refers to “/Protocol2.0/ReadWrite/ReadWrite.cpp” |
| /\*\* Put below codes to additional DEFINITIONS \*\*/  /\* Control table address \*/  #define ADDR\_PRO\_TORQUE\_ENABLE 562  #define ADDR PRO\_GOAL\_POSITION 596  #define ADDR\_PRO\_PRESENT\_POSITION 611  #define ADDR\_PRO\_MOVING 610  /\*\* Put below codes to additional VARIABLES \*\*/  int index = 0; // Index for rotation direction  int dxl\_goal\_position[2] = {-150000, 150000}; // Goal positions  UINT8\_T dxl\_torque\_enable = 1; // Value for torque enable  UINT8\_T dxl\_torque\_disable = 0; // Value for torque disable  UINT8\_T dxl\_moving = 0; // DXL moving status  UINT32\_T dxl\_present\_position = 0; // Present position  /\*\* Put below codes to additional FUNCTIONS \*\*/  /\* packetHandler does dxl\_torque\_enable to DXL\_ID, and receives dxl\_error \*/  packetHandler->Write1ByteTxRx(portHandler, DXL\_ID, ADDR\_PRO\_TORQUE\_ENABLE  , dxl\_torque\_enable, &dxl\_error);  /\* Writes dxl\_goal\_position to DXL\_ID. The torque should be enabled previously \*/  packetHandler->Write4ByteTxRx(portHandler, DXL\_ID, ADDR\_PRO\_GOAL\_POSITION  , dxl\_goal\_position[index], &dxl\_error);  /\* Reads dxl\_present\_position from DXL\_ID. It gives value after this is called \*/  packetHandler->Read4ByteTxRx(portHandler, DXL\_ID, ADDR\_PRO\_PRESENT\_POSITION  , &dxl\_present\_position, &dxl\_error);  /\* Reads dxl\_moving status. If Dynamixel is moving, it returns 1, and if not, 0 \*/  packetHandler->Read1ByteTxRx(portHandler, DXL\_ID, ADDR\_PRO\_MOVING  , &dxl\_moving, &dxl\_error);  /\* packetHandler does dxl\_torque\_disable to DXL\_ID \*/  packetHandler->Write1ByteTxRx(portHandler, DXL\_ID, ADDR\_PRO\_TORQUE\_ENABLE  , dxl\_torque\_disable, &dxl\_error); |

* + 1. MultiPort
       - 1. Description

This example writes goal position to Dynamixels connected to two serial ports, and reads their present position until Dynamixel stops moving.

* + - * 1. Available Dynamixel

All series using protocol 2.0

* + - * 1. Basic code for MultiPort example

|  |
| --- |
| Refers to “/Protocol2.0/Multiport/Multiport.cpp” |
| /\*\* Put additional HEADERS here \*\*/  /\* Headers \*/  #include "dynamixel\_sdk/PortHandler.h"  #include "dynamixel\_sdk/PacketHandler.h"  /\* Default Setting \*/  #define DXL1\_ID 1  #define DXL2\_ID 2  #define BAUDRATE 1000000  /\* Protocol Version \*/  #define PROTOCOL\_VERSION 2.0  /\* Device Name of Port (uses USB2Dynamixel on Linux) \*/  #define DEVICE\_NAME\_1 “/dev/ttyUSB0”  #define DEVICE\_NAME\_2 “/dev/ttyUSB1”  /\*\* Put additional DEFINITIONS here \*\*/  /\* gets methods and members of PortHandler Linux and Windows, and sets DEVICE\_NAME \*/  PortHandler \*portHandler1 = (PortHandler\*)PortHandler::GetPortHandler(“DEVICE\_NAME\_1”);  PortHandler \*portHandler2 = (PortHandler\*)PortHandler::GetPortHandler(“DEVICE\_NAME\_2”);  /\*\* Put DEBUG FUNCTIONS here (if necessary) \*\*/  int main()  {  UINT8\_T dxl\_error = 0; /\* DXL error \*/    /\*\* Put additional VARIABLES here \*\*/  /\* portHandler opens and gets the handle of selected port \*/  portHandler1->OpenPort();  portHandler2->OpenPort();  /\* portHandler retries to open the selected port with selected BAUDRATE \*/  portHandler1->SetBaudrate(BAUDRATE);  portHandler2->SetBaudrate(BAUDRATE);  /\* packetHandler gets methods and members of **Protocol1 and Protocol2 PacketHandler** \*/  PacketHandler \*packetHandler = PacketHandler::GetPacketHandler(PROTOCOL\_VERSION);    /\*\* Put additional FUNCTIONS here \*\*/  /\* packetHandler closes selected port \*/  portHandler1.ClosePort();  portHandler2.ClosePort();  } |

* + - * 1. Simple code

|  |
| --- |
| Refers to “/Protocol2.0/Multiport/Multiport.cpp” |
| /\*\* Put below codes to additional DEFINITIONS \*\*/  /\* Control table address \*/  #define ADDR\_PRO\_TORQUE\_ENABLE 562  #define ADDR\_PRO\_GOAL\_POSITION 596  #define ADDR\_PRO\_PRESENT\_POSITION 611  /\*\* Put below codes to additional VARIABLES \*\*/  int index = 0; // Index for rotation direction  int dxl\_goal\_position[2] = {-150000, 150000}; // Goal positions  UINT8\_T dxl\_torque\_enable = 1; // Value for torque enable  UINT8\_T dxl\_torque\_disable = 0; // Value for torque disable  UINT32\_T dxl\_present\_position1 = 0, dxl\_present\_position2 = 0;// Present position  /\*\* Put below codes to additional FUNCTIONS \*\*/  /\* packetHandler does dxl\_torque\_enable to DXLs, and receives dxl\_error \*/  packetHandler->Write1ByteTxRx(portHandler1, DXL1\_ID, ADDR\_PRO\_TORQUE\_ENABLE  , dxl\_torque\_enable, &dxl\_error);  packetHandler->Write1ByteTxRx(portHandler2, DXL2\_ID, ADDR\_PRO\_TORQUE\_ENABLE  , dxl\_torque\_enable, &dxl\_error);  /\* Writes dxl\_goal\_position to DXLs. The torque should be enabled previously \*/  packetHandler->Write4ByteTxRx(portHandler1, DXL1\_ID, ADDR\_PRO\_GOAL\_POSITION  , dxl\_goal\_position[index], &dxl\_error);  packetHandler->Write4ByteTxRx(portHandler2, DXL2\_ID, ADDR\_PRO\_GOAL\_POSITION  , dxl\_goal\_position[index], &dxl\_error);  /\* Reads dxl\_present\_position from DXLs. It gives value after this is called \*/  packetHandler->Read4ByteTxRx(portHandler1, DXL1\_ID, ADDR\_PRO\_PRESENT\_POSITION  , &dxl\_present\_position1, &dxl\_error);  packetHandler->Read4ByteTxRx(portHandler2, DXL2\_ID, ADDR\_PRO\_PRESENT\_POSITION  , &dxl\_present\_position2, &dxl\_error);  /\* packetHandler does dxl\_torque\_disable to DXLs \*/  packetHandler->Write1ByteTxRx(portHandler1, DXL1\_ID, ADDR\_PRO\_TORQUE\_ENABLE  , dxl\_torque\_disable, &dxl\_error);  packetHandler->Write1ByteTxRx(portHandler2, DXL2\_ID, ADDR\_PRO\_TORQUE\_ENABLE  , dxl\_torque\_disable, &dxl\_error); |

* + 1. SyncReadWrite
       - 1. Description

This example writes goal positions to two Dynamixels and repeats to read present positions simultaneously until Dynamixels stop moving. The funtions that are related with the Syncread and Syncwrite handle the number of items which are near each other in the Dynamixel control table on multiple Dynamixels, such as the goal position and the goal velocity.

* + - * 1. Available Dynamixel

All series using protocol 2.0

* + - * 1. Code

|  |
| --- |
| Refers to “/Protocol2.0/SyncReadWrite/SyncReadWrite.cpp” |
| /\*\* Put below codes to additional HEADERS \*\*/  #include "dynamixel\_sdk/GroupSyncWrite.h"  #include "dynamixel\_sdk/GroupSyncRead.h"  /\*\* Put below codes to additional DEFINITIONS \*\*/  /\* Control table address \*/  #define ADDR\_PRO\_TORQUE\_ENABLE 562  #define ADDR\_PRO\_GOAL\_POSITION 596  #define ADDR\_PRO\_PRESENT\_POSITION 611  /\* Data byte length \*/  #define LEN\_PRO\_TORQUE\_ENABLE 1  #define LEN\_PRO\_GOAL\_POSITION 4  #define LEN\_PRO\_PRESENT\_POSITION 4  /\*\* Put below codes to additional VARIABLES \*\*/  int index = 0; // Index for rotation direction  int dxl\_goal\_position[2] = {-150000, 150000}; // Goal positions  UINT8\_T dxl\_torque\_enable = 1; // Value for torque enable  UINT8\_T dxl\_torque\_disable = 0; // Value for torque disable  UINT8\_T param\_goal\_position[4] = 0;  UINT32\_T dxl\_present\_position1, dxl\_present\_position2;// Present position  /\*\* Put below codes to additional FUNCTIONS \*\*/  /\* Initializes groupSyncWrite instance using portHandler and packetHandler \*/  GroupSyncWritegroupSyncWrite(portHandler, packetHandler  , ADDR\_PRO\_GOAL\_POSITION, LEN\_PRO\_GOAL\_POSITION);  // Initialize **GroupSyncRead** instance  GroupSyncReadgroupSyncRead(portHandler, packetHandler  , ADDR\_PRO\_PRESENT\_POSITION, LEN\_PRO\_PRESENT\_POSITION);  /\* packetHandler does dxl\_torque\_enable to DXLs, and receives dxl\_error \*/  packetHandler->Write1ByteTxRx(portHandler, DXL1\_ID, ADDR\_PRO\_TORQUE\_ENABLE  , dxl\_torque\_enable, &dxl\_error);  packetHandler->Write1ByteTxRx(portHandler, DXL2\_ID, ADDR\_PRO\_TORQUE\_ENABLE  , dxl\_torque\_enable, &dxl\_error);  // Add parameter storage for DXL present position value  groupSyncRead.AddParam(DXL1\_ID);  groupSyncRead.AddParam(DXL2\_ID);  /\* Allocates dxl\_goal\_position into byte array \*/  param\_goal\_position[0] = DXL\_LOBYTE(DXL\_LOWORD(dxl\_goal\_position[index]));  param\_goal\_position[1] = DXL\_HIBYTE(DXL\_LOWORD(dxl\_goal\_position[index]));  param\_goal\_position[2] = DXL\_LOBYTE(DXL\_HIWORD(dxl\_goal\_position[index]));  param\_goal\_position[3] = DXL\_HIBYTE(DXL\_HIWORD(dxl\_goal\_position[index]));  /\* AddParameter param\_goal\_position groupSyncWrite parameter Storage \*/  groupSyncWrite.AddParam(DXL1\_ID, param\_goal\_position);  groupSyncWrite.AddParam(DXL2\_ID, param\_goal\_position);  /\* Transmits the packet \*/  groupSyncWrite.TxPacket();  /\* ClearParameter of groupSyncWrite parameter storage \*/  groupSyncWrite.ClearParam();  // Syncread present position  groupSyncRead.TxRxPacket();  // Get DXL#1 present position value  groupSyncRead.GetData(DXL1\_ID, ADDR\_PRO\_PRESENT\_POSITION, &dxl\_present\_position1);  // Get DXL#2 moving status value  groupSyncRead.GetData(DXL2\_ID, ADDR\_PRO\_PRESENT\_POSITION , &dxl\_present\_position2);  // Clear syncread parameter storage  groupSyncRead.ClearParam();  // Disable DXL torque  packetHandler->Write1ByteTxRx(portHandler, DXL1\_ID, ADDR\_PRO\_TORQUE\_ENABLE  , dxl\_torque\_disable, &dxl\_error);  packetHandler->Write1ByteTxRx(portHandler, DXL2\_ID, ADDR\_PRO\_TORQUE\_ENABLE  , dxl\_torque\_disable, &dxl\_error); |

* + 1. BulkReadWrite
       - 1. Description

This example writes either of goal position or LED value of two Dynamixels and simulateously reads them until Dynamixel stops moving. The functions that are related with the Bulkwrite and the Bulkread function handle the number of items which are not near each other in the Dynamixel control table on multiple Dynamixels.

* + - * 1. Available Dynamixel

All series using protocol 2.0

* + - * 1. Simple code

|  |
| --- |
| Refers to “/Protocol2.0/BulkReadWrite/BulkReadWrite.cpp” |
| /\*\* Put below codes to additional HEADERS \*\*/  #include "dynamixel\_sdk/GroupBulkRead.h"  #include "dynamixel\_sdk/GroupBulkWrite.h"  /\*\* Put below codes to additional DEFINITIONS \*\*/  /\* Control table address \*/  #define ADDR\_PRO\_TORQUE\_ENABLE 562  #define ADDR\_PRO\_LED\_RED 563  #define ADDR\_PRO\_GOAL\_POSITION 596  #define ADDR\_PRO\_PRESENT\_POSITION 611  /\* Data byte length \*/  #define LEN\_PRO\_TORQUE\_ENABLE 1  #define LEN\_PRO\_LED\_RED 1  #define LEN\_PRO\_GOAL\_POSITION 4  #define LEN\_PRO\_PRESENT\_POSITION 4  /\*\* Put below codes to additional VARIABLES \*\*/  int index = 0; // Index for rotation direction  int dxl\_goal\_position[2] = {-150000, 150000}; // Goal positions  UINT8\_T dxl\_torque\_enable = 1; // Value for torque enable  UINT8\_T dxl\_torque\_disable = 0; // Value for torque disable  UINT8\_T dxl\_led\_value = 0xFF; // DXL LED value  UINT8\_T param\_goal\_position[4];  UINT8\_T dxl\_led\_value\_read2; // Read DXL#2 led value  UINT32\_T dxl\_present\_position1; // DXL#1 present position  /\*\* Put below codes to additional FUNCTIONS \*\*/  // Initialize **GroupBulkWrite** instance  GroupBulkWritegroupBulkWrite(portHandler, packetHandler);  // Initialize **GroupBulkRead** instance  GroupBulkReadgroupBulkRead(portHandler, packetHandler);  // Enable DXL torque  packetHandler->Write1ByteTxRx(portHandler, DXL1\_ID, ADDR\_PRO\_TORQUE\_ENABLE  , dxl\_torque\_enable, &dxl\_error);  packetHandler->Write1ByteTxRx(portHandler, DXL2\_ID, ADDR\_PRO\_TORQUE\_ENABLE  , dxl\_torque\_enable, &dxl\_error);  // Add parameter storage for DXL present position value  groupBulkRead.AddParam(DXL1\_ID, ADDR\_PRO\_PRESENT\_POSITION, LEN\_PRO\_PRESENT\_POSITION);  groupBulkRead.AddParam(DXL2\_ID, ADDR\_PRO\_LED\_RED, LEN\_PRO\_LED\_RED);    // Allocate goal position value into byte array  param\_goal\_position[0] = DXL\_LOBYTE(DXL\_LOWORD(dxl\_goal\_position[index]));  param\_goal\_position[1] = DXL\_HIBYTE(DXL\_LOWORD(dxl\_goal\_position[index]));  param\_goal\_position[2] = DXL\_LOBYTE(DXL\_HIWORD(dxl\_goal\_position[index]));  param\_goal\_position[3] = DXL\_HIBYTE(DXL\_HIWORD(dxl\_goal\_position[index]));  // Add parameter storage for DXL#1 goal position  groupBulkWrite.AddParam(DXL1\_ID, ADDR\_PRO\_PRESENT\_POSITION, LEN\_PRO\_PRESENT\_POSITION  , param\_goal\_position);  // Add parameter storage for DXL#2 LED value  groupBulkWrite.AddParam(DXL2\_ID, ADDR\_PRO\_LED\_RED, LEN\_PRO\_LED\_RED, dxl\_led\_value);  // Write goal position  packetHandler->Write2ByteTxRx(portHandler, DXL1\_ID, ADDR\_PRO\_GOAL\_POSITION  , dxl\_goal\_position, &dxl\_error);  packetHandler->Write2ByteTxRx(portHandler, DXL2\_ID, ADDR\_PRO\_GOAL\_POSITION  , dxl\_goal\_position, &dxl\_error);  // Bulkwrite goal position and LED value  groupBulkWrite.TxPacket();  // Clear bulkwrite parameter storage  groupBulkWrite.ClearParam();  // Bulkread present position and LED value  groupBulkRead.TxRxPacket();  // Get DXL#1 present position value  groupBulkRead.GetData(DXL1\_ID, ADDR\_PRO\_PRESENT\_POSITION, &dxl\_present\_position1);  // Get DXL#2 LED value  groupBulkRead.GetData(DXL2\_ID, ADDR\_PRO\_LED\_RED, &dxl\_led\_value\_read2);  // Clear bulkread parameter storage  groupBulkRead.ClearParam();  // Disable DXL torque  packetHandler->Write1ByteTxRx(portHandler, DXL1\_ID, ADDR\_PRO\_TORQUE\_ENABLE  , dxl\_torque\_disable, &dxl\_error);  packetHandler->Write1ByteTxRx(portHandler, DXL2\_ID, ADDR\_PRO\_TORQUE\_ENABLE  , dxl\_torque\_disable, &dxl\_error); |

* + 1. IndirectAddress
       - 1. Description

This example writes the goal position and LED value and repeats to read present position and moving status through the indirect data storage, rather than write directly to the their own data storages. The indirect address links between direct and indirect data storages. This makes the Syncread and the Syncwrite function accessible to the items which are far from each other’s address.

* + - * 1. Available Dynamixel

All series using protocol 2.0

* + - * 1. Simple code

|  |
| --- |
| Refers to “/Protocol2.0/IndirectAddress/IndirectAddress.cpp” |
| /\*\* Put below codes to additional HEADERS \*\*/  #include "dynamixel\_sdk/GroupSyncWrite.h"  #include "dynamixel\_sdk/GroupSyncRead.h"  /\*\* Put below codes to additional DEFINITIONS \*\*/  /\* Control table address \*/  #define ADDR\_PRO\_INDIRECTADDRESS\_FOR\_WRITE 49  #define ADDR\_PRO\_INDIRECTADDRESS\_FOR\_READ 59  #define ADDR\_PRO\_TORQUE\_ENABLE 562  #define ADDR\_PRO\_LED\_RED 563  #define ADDR\_PRO\_GOAL\_POSITION 596  #define ADDR\_PRO\_MOVING 610  #define ADDR\_PRO\_PRESENT\_POSITION 611  #define ADDR\_PRO\_INDIRECTDATA\_FOR\_WRITE 634  #define ADDR\_PRO\_INDIRECTDATA\_FOR\_READ 639  /\* Data byte length \*/  #define LEN\_PRO\_INDIRECTDATA\_FOR\_WRITE 5  #define LEN\_PRO\_INDIRECTDATA\_FOR\_READ 5  /\*\* Put below codes to additional VARIABLES \*\*/  int index = 0; // Index for rotation direction  int dxl\_goal\_position[2] = {-150000, 150000}; // Goal positions  UINT8\_T dxl\_torque\_enable = 1; // Value for torque enable  UINT8\_T dxl\_torque\_disable = 0; // Value for torque disable  UINT8\_T dxl\_moving; // DXL moving status  UINT8\_T dxl\_led\_value = 0xFF; // LED value  UINT8\_T param\_indirect\_data\_for\_write[LEN\_PRO\_INDIRECTDATA\_FOR\_WRITE];  UINT32\_T dxl\_present\_position; // Present position  /\*\* Put below codes to additional FUNCTIONS \*\*/  // Initialize **GroupSyncWrite** instance  GroupSyncWritegroupSyncWrite(portHandler, packetHandler  , ADDR\_PRO\_INDIRECTDATA\_FOR\_WRITE  , LEN\_PRO\_INDIRECTDATA\_FOR\_WRITE);  // Initialize **GroupSyncRead** instance  GroupSyncReadgroupSyncRead(portHandler, packetHandler  , ADDR\_PRO\_INDIRECTDATA\_FOR\_READ  , LEN\_PRO\_INDIRECTDATA\_FOR\_READ);  // Disable DXL torque  // Indirect address would not be accessible when the torque is already enabled  packetHandler->Write1ByteTxRx(portHandler, DXL\_ID, ADDR\_PRO\_TORQUE\_ENABLE  , dxl\_torque\_disable, &dxl\_error);  // INDIRECTDATA parameter storages replace LED, goal position, present position and moving status  packetHandler->Write2ByteTxRx(portHandler, DXL\_ID  , ADDR\_PRO\_INDIRECTADDRESS\_FOR\_WRITE + 0  , ADDR\_PRO\_GOAL\_POSITION + 0, &dxl\_error);  packetHandler->Write2ByteTxRx(portHandler, DXL\_ID  , ADDR\_PRO\_INDIRECTADDRESS\_FOR\_WRITE + 2  , ADDR\_PRO\_GOAL\_POSITION + 1, &dxl\_error);  packetHandler->Write2ByteTxRx(portHandler, DXL\_ID  , ADDR\_PRO\_INDIRECTADDRESS\_FOR\_WRITE + 4  , ADDR\_PRO\_GOAL\_POSITION + 2, &dxl\_error);  packetHandler->Write2ByteTxRx(portHandler, DXL\_ID  , ADDR\_PRO\_INDIRECTADDRESS\_FOR\_WRITE + 6  , ADDR\_PRO\_GOAL\_POSITION + 3, &dxl\_error);  packetHandler->Write2ByteTxRx(portHandler, DXL\_ID  , ADDR\_PRO\_INDIRECTADDRESS\_FOR\_WRITE + 8  , ADDR\_PRO\_LED\_RED, &dxl\_error);  packetHandler->Write2ByteTxRx(portHandler, DXL\_ID  , ADDR\_PRO\_INDIRECTADDRESS\_FOR\_READ + 0  , ADDR\_PRO\_PRESENT\_POSITION + 0, &dxl\_error);  packetHandler->Write2ByteTxRx(portHandler, DXL\_ID  , ADDR\_PRO\_INDIRECTADDRESS\_FOR\_READ + 2  , ADDR\_PRO\_PRESENT\_POSITION + 1, &dxl\_error);  packetHandler->Write2ByteTxRx(portHandler, DXL\_ID  , ADDR\_PRO\_INDIRECTADDRESS\_FOR\_READ + 4  , ADDR\_PRO\_PRESENT\_POSITION + 2, &dxl\_error);  packetHandler->Write2ByteTxRx(portHandler, DXL\_ID  , ADDR\_PRO\_INDIRECTADDRESS\_FOR\_READ + 6  , ADDR\_PRO\_PRESENT\_POSITION + 3, &dxl\_error);  packetHandler->Write2ByteTxRx(portHandler, DXL\_ID  , ADDR\_PRO\_INDIRECTADDRESS\_FOR\_READ + 8  , ADDR\_PRO\_MOVING, &dxl\_error);  // Enable DXL torque  packetHandler->Write1ByteTxRx(portHandler, DXL\_ID, ADDR\_PRO\_TORQUE\_ENABLE  , dxl\_torque\_enable, &dxl\_error);  // Add parameter storage for DXL present position value  groupSyncRead.AddParam(DXL\_ID);  // Allocate goal position value into byte array  param\_indirect\_data\_for\_write [0] = DXL\_LOBYTE(DXL\_LOWORD(dxl\_goal\_position[index]));  param\_indirect\_data\_for\_write [1] = DXL\_HIBYTE(DXL\_LOWORD(dxl\_goal\_position[index]));  param\_indirect\_data\_for\_write [2] = DXL\_LOBYTE(DXL\_HIWORD(dxl\_goal\_position[index]));  param\_indirect\_data\_for\_write [3] = DXL\_HIBYTE(DXL\_HIWORD(dxl\_goal\_position[index]));  param\_indirect\_data\_for\_write [4] = dxl\_led\_value;  // Add DXL goal position value to the Syncwrite storage  groupSyncWrite.AddParam(DXL\_ID, param\_indirect\_data\_for\_write);  // Syncwrite goal position and LED value  groupSyncWrite.TxPacket();  // Clear syncwrite parameter storage  groupSyncWrite.ClearParam();  // Syncread present position and moving status  groupSyncRead.TxRxPacket();  // Get present position value  groupSyncRead.GetData(DXL\_ID, ADDR\_PRO\_INDIRECTDATA\_FOR\_READ, &dxl\_present\_position);  // Get moving status value  groupSyncRead.GetData(DXL\_ID, ADDR\_PRO\_INDIRECTDATA\_FOR\_READ + 4, &dxl\_moving);  // Clear syncread parameter storage  groupSyncRead.ClearParam();  // Disable DXL torque  packetHandler->Write1ByteTxRx(portHandler, DXL\_ID, ADDR\_PRO\_TORQUE\_ENABLE  , dxl\_torque\_disable, &dxl\_error); |

* + 1. Reboot
       - 1. Description

This example makes Dynamixel reboot. The Reboot function can be used when the Dynamixel stops moving since the Dynamixel error occurred by, for example, overload.

* + - * 1. Available Dynamixel

All series using protocol 2.0

* + - * 1. Simple code

|  |
| --- |
| Refers to “/Protocol2.0/Reboot/Reboot.cpp” |
| /\*\* Put below codes to additional DEFINITIONS \*\*/  /\* Control table address \*/  #define ADDR\_MX\_BAUDRATE 4  /\*\* Put below codes to additional FUNCTIONS \*\*/  // Try reboot  // DXL Green LED will flicker while it reboots  packetHandler->Reboot(portHandler, DXL\_ID, &dxl\_error); |

* + 1. FactoryReset
       - 1. Description

This example resets settings of Dynamixel to default values. The Factoryreset function has three operation modes:   
 0xFF : reset all values

0x01 : reset all values except ID

0x02 : reset all values except ID and baudrate.

* + - * 1. Available Dynamixel

All series using protocol 2.0

* + - * 1. Simple code

|  |
| --- |
| Refers to “/Protocol2.0/FactoryReset/FacotoryReset.cpp” |
| /\*\* Put below codes to additional DEFINITIONS \*\*/  /\* Control table address \*/  #define ADDR\_PRO\_BAUDRATE 8  #define FACTORYRST\_DEFAULTBAUDRATE 57600 // DXL baudrate set by factoryreset  #define NEW\_BAUDNUM 3 // New baudnum to recover DXL baudrate  #define OPERATION\_MODE 0x01 // 0xFF : reset all values  // 0x01 : reset all values except ID  // 0x02 : reset all values except ID, baudrate  /\*\* Put below codes to additional VARIABLES \*\*/  int dxl\_baudnum\_read = 0; // Read baudnum  /\*\* Put below codes to additional FUNCTIONS \*\*/  // Read present baudrate of the controller  portHandler.GetBaudRate();  // Try factoryreset  packetHandler->FactoryReset(portHandler, DXL\_ID, OPERATION\_MODE, &dxl\_error);  // Wait for reset  usleep(2000 \* 1000);  // Set serial port baudrate to Factoryreset Default Baudrate  portHandler.SetBaudrate(FACTORYRST\_DEFAULTBAUDRATE);  // Read DXL baudrate  packetHandler->Read1ByteTxRx(portHandler, DXL\_ID, ADDR\_PRO\_BAUDRATE  , &dxl\_baudnum\_read, &dxl\_error);  // Write DXL baudrate  packetHandler->Write1ByteTxRx(portHandler, DXL\_ID, ADDR\_PRO\_BAUDRATE  , NEW\_BAUDNUM, &dxl\_error);  // Set serial port baudrate to BAUDRATE  portHandler.SetBaudrate(BAUDRATE);  // Read DXL baudrate  packetHandler->Read1ByteTxRx(portHandler, DXL\_ID, ADDR\_MX\_BAUDRATE  , &dxl\_baudnum\_read, &dxl\_error); |

* 1. ProtocolCombined
     + - 1. Description

This example, basically, writes the goal position of Dynamixel and repeats to read its present position, until it stops moving. Read and write function can be used even to handle other multiple items in nearby addresses on the control table lists, such as goal position and velocity.

* + - * 1. Available Dynamixel model

All models using protocol 1.0 and 2.0

* + - * 1. Basic code for ProtocolCombined example

|  |
| --- |
| Refers to “/ProtocolCombined/ProtocolCombined.cpp” |
| /\*\* Put additional HEADERS here \*\*/  /\* Headers \*/  #include "dynamixel\_sdk/PortHandler.h"  #include "dynamixel\_sdk/PacketHandler.h"  /\* Default Setting \*/  #define DXL1\_ID 1  #define DXL2\_ID 2  #define BAUDRATE 1000000  /\* Protocol Version \*/  #define PROTOCOL\_VERSION\_1 1.0  #define PROTOCOL\_VERSION\_2 2.0  /\* Device Name of Port (uses USB2Dynamixel on Linux) \*/  #define DEVICE\_NAME “/dev/ttyUSB0”  /\*\* Put additional DEFINITIONS here \*\*/  /\* gets methods and members of PortHandler Linux and Windows, and sets DEVICE\_NAME \*/  PortHandler \*portHandler = (PortHandler\*)PortHandler::GetPortHandler(“DEVICE\_NAME”);  /\*\* Put DEBUG FUNCTIONS here (if necessary) \*\*/  int main()  {  UINT8\_T dxl\_error = 0; /\* DXL error \*/    /\*\* Put additional VARIABLES here \*\*/  /\* portHandler opens and gets the handle of selected port \*/  portHandler->OpenPort();  /\* portHandler retries to open the selected port with selected BAUDRATE \*/  portHandler->SetBaudrate(BAUDRATE);  /\* packetHandler gets methods and members of **Protocol1 and Protocol2 PacketHandler** \*/  PacketHandler \*packetHandler1 = PacketHandler::GetPacketHandler(PROTOCOL\_VERSION\_1);  PacketHandler \*packetHandler2 = PacketHandler::GetPacketHandler(PROTOCOL\_VERSION\_2);    /\*\* Put additional FUNCTIONS here \*\*/  /\* packetHandler closes selected port \*/  portHandler.ClosePort();  } |

* + - * 1. Simple code

|  |
| --- |
| Refers to “/ProtocolCombined/ProtocolCombined.cpp” |
| /\*\* Put below codes to additional DEFINITIONS \*\*/  /\* Control table address for MX \*/  #define ADDR\_MX\_TORQUE\_ENABLE 562  #define ADDR\_MX\_GOAL\_POSITION 596  #define ADDR\_MX\_PRESENT\_POSITION 611  /\* Control table address for PRO \*/  #define ADDR\_PRO\_TORQUE\_ENABLE 562  #define ADDR\_PRO\_GOAL\_POSITION 596  #define ADDR\_PRO\_PRESENT\_POSITION 611  /\*\* Put below codes to additional VARIABLES \*\*/  int index = 0; // Index for rotation direction  int dxl\_goal\_position1[2] = {100, 1000}  , dxl\_goal\_position2[2] = {-150000, 150000}; // Goal position  UINT8\_T dxl\_torque\_enable = 1; // Value for torque enable  UINT8\_T dxl\_torque\_disable = 0; // Value for torque disable  UINT32\_T dxl\_present\_position1 = 0, dxl\_present\_position2 = 0;// Present position  /\*\* Put below codes to additional FUNCTIONS \*\*/  // Enable DXL torque  PacketHandler1->Write1ByteTxRx(portHandler1, DXL1\_ID, ADDR\_MX\_TORQUE\_ENABLE  , dxl\_torque\_enable, &dxl\_error);  PacketHandler2->Write1ByteTxRx(portHandler2, DXL2\_ID, ADDR\_PRO\_TORQUE\_ENABLE  , dxl\_torque\_enable, &dxl\_error);  // Write goal position  packetHandler1->Write2ByteTxRx(portHandler1, DXL1\_ID, ADDR\_MX\_GOAL\_POSITION  , dxl\_goal\_position1[index], &dxl\_error); // MX  packetHandler2->Write4ByteTxRx(portHandler2, DXL2\_ID, ADDR\_PRO\_GOAL\_POSITION  , dxl\_goal\_position2[index], &dxl\_error); // PRO  // Read present position  packetHandler1->Read2ByteTxRx(portHandler1, DXL1\_ID, ADDR\_MX\_PRESENT\_POSITION  , &dxl\_present\_position1, &dxl\_error); // MX  packetHandler2->Read4ByteTxRx(portHandler2, DXL2\_ID, ADDR\_PRO\_PRESENT\_POSITION  , &dxl\_present\_position2, &dxl\_error); // PRO  // Disable DXL torque  packetHandler1->Write1ByteTxRx(portHandler1, DXL1\_ID, ADDR\_MX\_TORQUE\_ENABLE  , dxl\_torque\_disable, &dxl\_error); // MX  packetHandler2->Write1ByteTxRx(portHandler2, DXL2\_ID, ADDR\_PRO\_TORQUE\_ENABLE  , dxl\_torque\_disable, &dxl\_error); // PRO |

* 1. DXLMonitor
     + - 1. Description

In this example, all fuctions except syncread, syncwrite, bulkread, bulkwrite appears on the command prompt or terminal. User may type commands to use the functions. For example:

Type below (use Dynamixel PRO of ID = 1)

[CMD] *wrb2 1 7 2* //*wrb2 [ID] [ADDRESS] [Value]*

and type enter. Then, the Dynamixel PRO’s ID would be changed to ID = 2. To see another commands, type “*help*” or ‘*h*’, or ‘*?*’ to show the help.

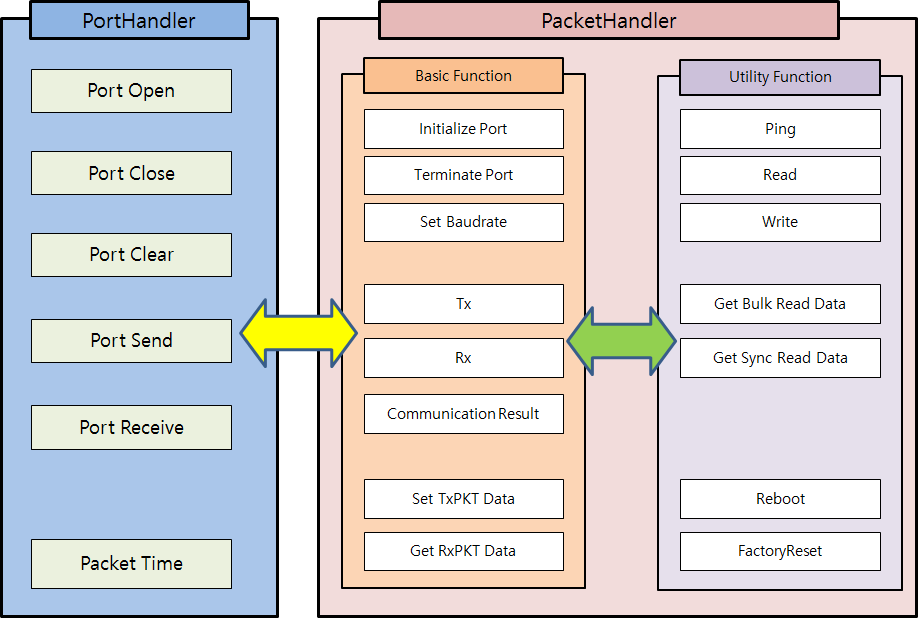
Each command is classified by numbering on the end of the command string. For instance, command “*wrb2*” will “write byte using Protocol 2.0”. On the other hand, command that doesn’t have number at the end, such as “*scan*”, will “scan and show list of connected Dynamixels and the model number of the Dynamixel, using both protocols”

* + - * 1. Available Dynamixel

All series using protocol 1.0 and 2.0

1. API Reference

Architecture



* 1. PortHandler
     + - 1. Description

This class gets instances of PortHandler from either of the OS, and handles the port for packet communication.

* + - * 1. Members

|  |  |
| --- | --- |
| DEFAULT\_BAUDRATE :=1000000 | Default Baudrate |
| is\_using | Shows whether the port is now in use |

* + - * 1. Methods

|  |  |
| --- | --- |
| GetPortHandler | gets PortHandler from either of the OS |
| ~PortHandler | None |
| OpenPort | opens the port |
| ClosePort | closes the port |
| ClearPort | clears the port |
| SetPortName | sets the device name |
| GetPortName | gets the device name |
| SetBaudrate | sets the baudrate of the port |
| GetBaudrate | gets the baudrate of the port |
| GetBytesAvailable | checks data bytes how many it can be read |
| ReadPort | reads the data from port buffer |
| WritePort | writes the data from port buffer |
| SetPacketTimeout | sets the time to decide communication result |
| IsPacketTimeout | checks communication result |

* + - * 1. Enumerator

None

* 1. PortHandlerLinux
     + - 1. Description

This class handles the port for packet communication on Linux OS.

* + - * 1. Members

None

* + - * 1. Methods

|  |  |
| --- | --- |
| PortHandlerLinux | initializes PortHandlerLinux instance |
| ~PortHandlerLinux | closes the port |
| OpenPort | opens the port |
| ClosePort | closes the port |
| ClearPort | clears the port |
| SetPortName | sets the device name |
| GetPortName | gets the device name |
| SetBaudrate | sets the baudrate of the port |
| GetBaudrate | gets the baudrate of the port |
| GetBytesAvailable | checks data bytes how many it can be read |
| ReadPort | reads the data from port buffer |
| WritePort | writes the data from port buffer |
| SetPacketTimeout | sets the time to decide communication result |
| IsPacketTimeout | checks communication result |

* + - * 1. Enumerator

None

* + 1. OpenPort
       - 1. Syntax

bool OpenPort()

* + - * 1. Parameters

None

* + - * 1. Description

This function opens the port by SetBaudRate function using DEFAULT\_BAUDRATE. If the baudrate is needed to be changed to another baudrate value, SetBaudRate function should be called after calling OpenPort function. When the port succeeds to be opened, this function will return true, and if not, then false.

* + 1. ClosePort
       - 1. Syntax

void ClosePort()

* + - * 1. Parameters

None

* + - * 1. Description

This function closes the port by closing the file descriptor.

* + 1. ClearPort
       - 1. Syntax

void ClearPort()

* + - * 1. Parameters

None

* + - * 1. Description

This function clears the port by flushing the file descriptor.

* + 1. SetPortName
       - 1. Syntax

void SetPortName(const char\* port\_name)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port\_name | Device name |

* + - * 1. Description

This function sets the device name as port\_name.

* + 1. GetPortName
       - 1. Syntax

char\* GetPortName()

* + - * 1. Parameters

None

* + - * 1. Description

This function returns the device name which the port is using.

* + 1. SetBaudrate
       - 1. Syntax

bool SetBaudRate(const int baudrate)

* + - * 1. Parameters

|  |  |
| --- | --- |
| baudrate | Target baudrate |

* + - * 1. Description

This function converts baudrate to baudrate type value at first. Secondly, it closes the port with ClosePort function, and opens the port with SetupPort function again. If the value of baudrate is not in the baudrate list shown in the GetCFlagBaud function, the SetCustomBaudrate function suggests the baudrate value which is closest of available baudrate value. Finally, it returns false.

* + 1. GetBaudrate
       - 1. Syntax

int GetBaudRate()

* + - * 1. Parameters

None

* + - * 1. Description

This function returns the baudrate value previously set.

* + - * 1. Return Status

None

* + 1. GetBytesAvailable
       - 1. Syntax

int GetBytesAvailable ()

* + - * 1. Parameters

None

* + - * 1. Description

This function checks how much the data can be read, and returns its length.

* + 1. ReadPort
       - 1. Syntax

int ReadPort(UINT8\_T\* packet, int length)

* + - * 1. Parameters

|  |  |
| --- | --- |
| packet | The number of data bytes read previously |
| length | Byte length for read |

* + - * 1. Description

This function gets length byte data from port buffer and returns a number of read data bytes. On end-of-file, 0 is returned, on error it returns -1.

* + 1. WritePort
       - 1. Syntax

int WritePort(UINT8\_T\* packet, int length)

* + - * 1. Parameters

|  |  |
| --- | --- |
| packet | The number of data bytes to write |
| length | Byte length for write |

* + - * 1. Description

This function transmits length byte, and returns how much the data was written. On error, it returns -1.

* + 1. SetPacketTimeout
       - 1. Syntax

void SetPacketTimeout (UINT16\_T packet\_length)

void SetPacketTimeout (double msec)

* + - * 1. Parameters

|  |  |
| --- | --- |
| Packet\_length | Target byte length for write |
| msec | Miliseconds |

* + - * 1. Description

This function sets the start time when it transmits the packet, and set the timeout of packet transmission to be ready for deciding communication result.

* + 1. IsPacketTimeout
       - 1. Syntax

bool IsPacketTimeout ()

* + - * 1. Parameters

None

* + - * 1. Description

This function decides the timeover of packet communication. If the time limit is over, it returns false.

* 1. PacketHandler
     + - 1. Description

This class inherits the instances from the packet handler of either of the Protocol version, and handles the packet construction.

* + - * 1. Members

None

* + - * 1. Methods

|  |  |
| --- | --- |
| GetPacketHandler | gets PacketHandler from either of Protocols |
| ~PacketHandler | None |
| GetProtocolVersion | gets Protocol version |
| TxPacket | transmits the packet |
| RxPacket | receives the packet |
| TxRxPacket | transmits and receives the packet |
| Ping | ping a Dynamixel |
| BroadcastPing | ping all connected Dynamixels |
| Action | commands ‘Run’ the Regwritten |
| RegWrite | writes the packets  walit for the ‘Action’ command |
| Reboot | reboots Dynamixel |
| FactoryReset | resets all Dynamixel settings |
| ReadTx | transmits N byte read instruction packet |
| ReadRx | receives N byte read status packet |
| ReadTxRx | transmits and receives N byte packet |
| Read1ByteTx | transmits 1 byte read instruction packet |
| Read1ByteRx | receives 1 byte read status packet |
| Read1ByteTxRx | transmits and receives 1 byte packet |
| Read2ByteTx | transmits 2 byte read instruction packet |
| Read2ByteRx | receives 2 byte read status packet |
| Read2ByteTxRx | transmits and receives 2 byte packet |
| Read4ByteTx | transmits 4 byte read instruction packet |
| Read4ByteRx | receives 4 byte read status packet |
| Read4ByteTxRx | transmits and receives 4 byte packet |
| WriteTxOnly | transmits N byte write instruction packet |
| WriteTxRx | transmits and receives N byte packet |
| Write1ByteTxOnly | transmits 1 byte write instruction packet |
| Write1ByteTxRx | transmits and receives 1 byte packet |
| Write2ByteTxOnly | transmits 2 byte write instruction packet |
| Write2ByteTxRx | transmits and receives 2 byte packet |
| Write4ByteTxOnly | transmits 4 byte write instruction packet |
| Write4ByteTxRx | transmits and receives 4 byte packet et |
| RegWriteTxOnly | transmits register write instruction packet |
| RegWriteTxRx | transmits and receives register write packet |
| SyncReadTx | transmits N byte sync read Instruction packet |
| SyncWriteTxOnly | transmits N byte sync write Instruction packet |
| BulkReadTx | transmits N byte bulk read Instruction packet |
| BulkWriteTxOnly | transmits N byte bulk write Instruction packet |

* + - * 1. Enumerator

|  |  |
| --- | --- |
| DXL\_MAKEWORD(a, b) | makes value from a and b to word type |
| DXL\_MAKEDWORD(a, b) | makes value from a and b to dword type |
| DXL\_LOWORD(l) | gets lower word type value from l |
| DXL\_HIWORD(l) | gets higher word type value from l |
| DXL\_LOBYTE(w) | gets lower byte type value from w |
| DXL\_HIBYTE(w) | gets higher byte type value from w |
| BROADCAST\_ID := 0xFE | Broadcast ID |
| MAX\_ID := 0xFC | Maximum ID value |
| INST\_PING := 1 | Instruction value of Ping |
| INST\_READ := 2 | Instruction value of Read |
| INST\_WRITE := 3 | Instruction value of Write |
| INST\_REG\_WRITE := 4 | Instruction value of Register Write |
| INST\_ACTION := 5 | Instruction value of Action |
| INST\_FACTORY\_RESET := 6 | Instruction value of Factory Reset |
| INST\_SYNC\_WRITE := 131 | Instruction value of Sync Write |
| INST\_BULK\_READ := 146 | Instruction value of Bulk Read |
| INST\_REBOOT := 8 | Instruction value of Reboot |
| INST\_STATUS := 85 | Instruction value of Status |
| INST\_SYNC\_READ := 130 | Instruction value of Sync Read |
| INST\_BULK\_WRITE := 147 | Instruction value of Bulk Write |
| COMM\_SUCCESS := 0 | Status of Communication Success |
| COMM\_PORT\_BUSY := -1000 | Status of Port in use |
| COMM\_TX\_FAIL := -1001 | Status of Transmit packet failed |
| COMM\_RX\_FAIL := -1002 | Status of Receive packet failed |
| COMM\_TX\_ERROR := -2000 | Status of Transmit packet error |
| COMM\_RX\_WAITING := -3000 | Status of Receive packet waiting |
| COMM\_RX\_TIMEOUT := -3001 | Status of Receive packet timeout |
| COMM\_RX\_CORRUPT := -3002 | Status of Receive packet corrupt |
| COMM\_NOT\_AVAILABLE := -9000 | Status of Unavailable in protocol 1.0 |

* 1. Protocol1PacketHandler
     + - 1. Description

This class handles the packet construction using communication Protocol 1.0.

* + - * 1. Members

None

* + - * 1. Methods

|  |  |
| --- | --- |
| GetInstance | gets instance of this class |
| ~Protocol1PacketHandler | None |
| GetProtocolVersion | gets Protocol version |
| TxPacket | transmits the packet |
| RxPacket | receives the packet |
| TxRxPacket | transmits and receives the packet |
| Ping | ping a Dynamixel |
| BroadcastPing | Not available in Protocol 1.0 |
| Action | commands ‘Run’ the Regwritten |
| RegWrite | writes the packets  walit for the ‘Action’ command |
| Reboot | Not available in Protocol 1.0 |
| FactoryReset | resets all Dynamixel settings |
| ReadTx | transmits N byte read instruction packet |
| ReadRx | receives N byte read status packet |
| ReadTxRx | transmits and receives N byte packet |
| Read1ByteTx | transmits 1 byte read instruction packet |
| Read1ByteRx | receives 1 byte read status packet |
| Read1ByteTxRx | transmits and receives 1 byte packet |
| Read2ByteTx | transmits 2 byte read instruction packet |
| Read2ByteRx | receives 2 byte read status packet |
| Read2ByteTxRx | transmits and receives 2 byte packet |
| Read4ByteTx | Not available in Protocol 1.0 |
| Read4ByteRx | Not available in Protocol 1.0 |
| Read4ByteTxRx | Not available in Protocol 1.0 |
| WriteTxOnly | transmits N byte write instruction packet |
| WriteTxRx | transmits and receives N byte packet |
| Write1ByteTxOnly | transmits 1 byte write instruction packet |
| Write1ByteTxRx | transmits and receives 1 byte packet |
| Write2ByteTxOnly | transmits 2 byte write instruction packet |
| Write2ByteTxRx | transmits and receives 2 byte packet |
| Write4ByteTxOnly | Not available in Protocol 1.0 |
| Write4ByteTxRx | Not available in Protocol 1.0 |
| RegWriteTxOnly | transmits register write instruction packet |
| RegWriteTxRx | transmits and receives register write packet |
| SyncReadTx | Not available in Protocol 1.0 |
| SyncWriteTxOnly | transmits N byte sync write Instruction packet |
| BulkReadTx | transmits N byte bulk read Instruction packet |
| BulkWriteTxOnly | Not available in Protocol 1.0 |

* + - * 1. Enumerator

None

* + 1. GetProtocolVersion
       - 1. Syntax

float GetProtocolVersion()

* + - * 1. Parameters

None

* + - * 1. Description

This function returns the protocol version set in the PacketHandler as a float value.

* + - * 1. Return Status

None

* + 1. TxPacket
       - 1. Syntax

int TxPacket(PortHandler \*port, UINT8\_T \*txpacket)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| txpacket | packet for transmission |

* + - * 1. Description

This function transmits the packet. The function clears the port by ClearPort function at first, and passes the txpacket to the WritePort function. The function activates while the port is not busy, and when the packet is written on the port buffer.

* + 1. RxPacket
       - 1. Syntax

int RxPacket(PortHandler \*port, UINT8\_T \*rxpacket)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| rxpacket | packet for reception |

* + - * 1. Description

This function repeatedly tries to receive packets by ReadPort function until whole packets that it waits to get have arrived, or the packet wait time limit is over. After r, it returns communication status value.

* + 1. TxRxPacket
       - 1. Syntax

int TxRxPacket(PortHandler \*port, UINT8\_T\* txpacket

, UINT8\_T\* rxpacket, UINT8\_T\* error = 0)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| txpacket | packet for transmission |
| rxpacket | packet for reception |
| error | Dynamixel error |

* + - * 1. Description

This function transmits and receives packets by TxPacket and RxPacket functions. When TxPacket function succeeds to communicate, it will continue to RxPacket and finishes the process if the packet succeeds to be received. In case of using the group handler functions for write, such as SyncWrite, and BulkWrite, they don’t use RxPacket, so the function finishes its operation immediately after the TxPacket. Before the RxPacket, it sets packet timeout if the packet instruction is INST\_READ.

* + 1. Ping
       - 1. Syntax

int Ping (PortHandler \*port, UINT8\_T id, UINT8\_T\* error = 0)

int Ping (PortHandler \*port, UINT8\_T id, UINT16\_T\* model\_number

, UINT8\_T\* error = 0)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id | Dynamixel ID |
| error | Dynamixel error |

* + - * 1. Description

This function constructs the transmission packet with INST\_PING instruction, and TxRxPacket. Then, the function tries to get the model number of the Dynamixel by ReadTxRx function. When it succeeds to receive the packet, it makes the model number value by using DXL\_MAKEWORD to put two byte-type data together. Finally, it returns communication status value.

* + 1. BroadcastPing
       - 1. Syntax

int BroadcastPing(PortHandler \*port

, std::vector<UINT8\_T> &id\_list)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id\_list | Dynamixel ID list |

* + - * 1. Description

This function is not available with the Dynamixel Protocol 1.0. It returns COMM\_NOT\_AVAILABLE status.

* + 1. Action
       - 1. Syntax

int Action(PortHandler \*port, UINT8\_T id)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id | Dynamixel ID |

* + - * 1. Description

This function constructs the transmission packet with INST\_ACTION instruction, and TxRxPacket. Before using this function, the orders for the Dynamixels should be already written in the register in Dynamixel by RegWrite function. Finally, it returns communication status value.

* + 1. Reboot
       - 1. Syntax

int Reboot(PortHandler \*port, UINT8\_T id, UINT8\_T\* error)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id | Dynamixel ID |
| error | Dynamixel error |

* + - * 1. Description

This function is not available with the Dynamixel Protocol 1.0. It returns COMM\_NOT\_AVAILABLE status.

* + 1. FactoryReset
       - 1. Syntax

int FactoryReset(PortHandler \*port, UINT8\_T id, UINT8\_T option

, UINT8\_T\* error)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id | Dynamixel ID |
| option | Reset option |
| error | Dynamixel error |

* + - * 1. Description

This function constructs the transmission packet with INST\_FACTORY\_RESET instruction, and TxRxPacket. This resets all Dynamixel settings to the factory default settings. The option is not available in Dynamixel Protocol 1.0. Finally, it returns communication status.

* + 1. ReadTx
       - 1. Syntax

int ReadTx(PortHandler \*port, UINT8\_T id, UINT16\_T address

, UINT16\_T length)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id | Dynamixel ID |
| address | Address on the control table of Dynamixel |
| length | Packet length |

* + - * 1. Description

This function constructs the transmission packet with INST\_READ instruction, and TxPacket. Then the function calls SetPacketTimeout function when TxPacket succeeds. This function is not available for controlling more than one Dynamixel. Finally, it returns communication status.

* + 1. ReadRx
       - 1. Syntax

int ReadRx(PortHandler \*port, UINT16\_T length, UINT8\_T \*data

, UINT8\_T\* error)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| length | Packet length |
| data | Packet data |
| error | Dynamixel error |

* + - * 1. Description

This function calls RxPacket function and gets the reception packet if the communication succeeds. Finally, it returns communication status value.

* + 1. ReadTxRx
       - 1. Syntax

int ReadTxRx(PortHandler \*port, UINT8\_T id, UINT16\_T address

, UINT16\_T length, UINT8\_T \*data, UINT8\_T\* error)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id | Dynamixel ID |
| address | Address on the control table of Dynamixel |
| length | Packet length |
| data | Packet data |
| error | Dynamixel error |

* + - * 1. Description

This function calls ReadTx function and ReadRx function. ReadRx function will be called when ReadTx succeeds. Finally, it returns communication status value.

* + 1. Read1ByteTx
       - 1. Syntax

int Read1ByteTx(PortHandler \*port, UINT8\_T id, UINT16\_T address)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id | Dynamixel ID |
| address | Address on the control table of Dynamixel |

* + - * 1. Description

This function calls ReadTx function for 1 Byte packet transmission. Finally, it returns communication status.

* + 1. Read1ByteRx
       - 1. Syntax

int Read1ByteRx(PortHandler \*port, UINT8\_T\* data

, UINT8\_T\* error)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| data | Packet data |
| error | Dynamixel error |

* + - * 1. Description

This function calls ReadRx function for 1 Byte packet reception. Finally, it returns communication status.

* + 1. Read1ByteTxRx
       - 1. Syntax

int Read1ByteTxRx(PortHandler \*port, UINT8\_T id,

UINT16\_T address, UINT8\_T\* data, UINT8\_T\* error)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id | Dynamixel ID |
| address | Address on the control table of Dynamixel |
| data | Packet data |
| error | Dynamixel error |

* + - * 1. Description

This function calls Read1ByteTx function and Read1ByteRx function for 1 Byte packet transmission and reception. Read1ByteRx function will be called when Read1ByteTx succeeds. Finally, it returns communication status.

* + 1. Read2ByteTx
       - 1. Syntax

int Read2ByteTx(PortHandler \*port, UINT8\_T id, UINT16\_T address)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id | Dynamixel ID |
| address | Address on the control table of Dynamixel |

* + - * 1. Description

This function calls ReadTx function for 2 Byte packet transmission. Finally, it returns communication status.

* + 1. Read2ByteRx
       - 1. Syntax

int Read2ByteRx(PortHandler \*port, UINT16\_T\* data

, UINT8\_T\* error)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| data | Packet data |
| error | Dynamixel error |

* + - * 1. Description

This function calls ReadRx function for 2 Byte packet reception. Finally, it returns communication status.

* + 1. Read2ByteTxRx
       - 1. Syntax

int Read2ByteTxRx(PortHandler \*port, UINT8\_T id,

UINT16\_T address, UIN16\_T\* data, UINT8\_T\* error)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id | Dynamixel ID |
| address | Address on the control table of Dynamixel |
| data | Packet data |
| error | Dynamixel error |

* + - * 1. Description

This function calls Read2ByteTx function and Read2ByteRx function for 2 Byte packet transmission and reception. Read2ByteRx function will be called when Read2ByteTx succeeds. Finally, it returns communication status.

* + 1. Read4ByteTx
       - 1. Syntax

int Read4ByteTx(PortHandler \*port, UINT8\_T id, UINT16\_T address)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id | Dynamixel ID |
| address | Address on the control table of Dynamixel |

* + - * 1. Description

This function is not available with the Dynamixel Protocol 1.0. It returns COMM\_NOT\_AVAILABLE status.

* + 1. Read4ByteRx
       - 1. Syntax

int Read4ByteRx(PortHandler \*port, UINT32\_T\* data

, UINT8\_T\* error)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| data | Packet data |
| error | Dynamixel error |

* + - * 1. Description

This function is not available with the Dynamixel Protocol 1.0. It returns COMM\_NOT\_AVAILABLE status.

* + 1. Read4ByteTxRx
       - 1. Syntax

int Read4ByteTxRx(PortHandler \*port, UIN8\_T id

, UINT16\_T address, UINT32\_T\* data, UINT8\_T\* error)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id | Dynamixel ID |
| address | Address on the control table of Dynamixel |
| data | Data for write |
| error | Dynamixel error |

* + - * 1. Description

This function is not available with the Dynamixel Protocol 1.0. It returns COMM\_NOT\_AVAILABLE status.

* + 1. WriteTxOnly
       - 1. Syntax

int WriteTxOnly(PortHandler \*port, UINT8\_T id, UINT16\_T address

, UINT16\_T length, UINT8\_T\* data)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id | Dynamixel ID |
| address | Address on the control table of Dynamixel |
| length | Packet length |
| data | Data for write |

* + - * 1. Description

This function constructs the transmission packet with INST\_WRITE instruction, and TxPacket. Finally, it returns communication status.

* + 1. WriteTxRx
       - 1. Syntax

int WriteTxRx(PortHandler \*port, UINT8\_T id, UINT16\_T address

, UINT16\_T length, UINT8\_T\* data, UINT8\_T\* error)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id | Dynamixel ID |
| address | Address on the control table of Dynamixel |
| length | Packet length |
| data | Data for write |
| error | Dynamixel error |

* + - * 1. Description

This function constructs the transmission packet with INST\_WRITE instruction, and TxRxPacket. Finally, it returns communication status.

* + 1. Write1ByteTxOnly
       - 1. Syntax

int Write1ByteTxOnly(PortHandler \*port, UINT8\_T id

, UINT16\_T address, UINT8\_T data)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id | Dynamixel ID |
| address | Address on the control table of Dynamixel |
| data | Data for write |

* + - * 1. Description

This function calls WriteTxOnly function for 1 Byte packet. Finally, it returns communication status.

* + 1. Write1ByteTxRx
       - 1. Syntax

int Write1ByteTxRx(PortHandler \*port, UINT8\_T id

, UINT16\_T address, UINT8\_T data, UINT8\_T\* error)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id | Dynamixel ID |
| address | Address on the control table of Dynamixel |
| data | Data for write |
| error | Dynamixel error |

* + - * 1. Description

This function calls WriteTxRx function for 1 Byte packet transmission and reception. Finally, it returns communication status.

* + 1. Write2ByteTxOnly
       - 1. Syntax

int Write2ByteTxOnly(PortHandler \*port, UINT8\_T id

, UINT16\_T address, UINT16\_T data)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id | Dynamixel ID |
| address | Address on the control table of Dynamixel |
| data | Data for write |

* + - * 1. Description

This function calls WriteTxOnly function for 2 Byte packet. Finally, it returns communication status.

* + 1. Write2ByteTxRx
       - 1. Syntax

int Write2ByteTxRx(PortHandler \*port, UINT8\_T id

, UINT16\_T address, UINT16\_T data, UINT8\_T\* error)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id | Dynamixel ID |
| address | Address on the control table of Dynamixel |
| data | Data for write |
| error | Dynamixel error |

* + - * 1. Description

This function calls WriteTxRx function for 2Byte packet transmission and reception. Finally, it returns communication status.

* + 1. Write4ByteTxOnly
       - 1. Syntax

int Write1ByteTxOnly(PortHandler \*port, UINT8\_T id

, UINT16\_T address, UINT32\_T data)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id | Dynamixel ID |
| address | Address on the control table of Dynamixel |
| data | Data for write |

* + - * 1. Description

This function is not available with the Dynamixel Protocol 1.0. It returns COMM\_NOT\_AVAILABLE status.

* + 1. Write4ByteTxRx
       - 1. Syntax

int Write4ByteTxRx(PortHandler \*port, UINT8\_T id

, UINT16\_T address, UINT32\_T data, UINT8\_T\* error)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id | Dynamixel ID |
| address | Address on the control table of Dynamixel |
| data | Data for write |
| error | Dynamixel error |

* + - * 1. Description

This function is not available with the Dynamixel Protocol 1.0. It returns COMM\_NOT\_AVAILABLE status.

* + 1. RegWriteTxOnly
       - 1. Syntax

int RegWriteTxOnly(PortHandler \*port, UINT8\_T id

, UINT16\_T address, UINT16\_T length, UINT8\_T\* data)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id | Dynamixel ID |
| address | Address on the control table of Dynamixel |
| length | Packet length |
| data | Data for write |

* + - * 1. Description

This function constructs the transmission packet with INST\_REG\_WRITE instruction, and TxPacket. Finally, it returns communication status.

* + 1. RegWriteTxRx
       - 1. Syntax

int RegWriteTxRx(PortHandler \*port, UINT8\_T id

, UINT16\_T address, UINT16\_T length

, UINT8\_T\* data, UINT8\_T\* error)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id | Dynamixel ID |
| address | Address on the control table of Dynamixel |
| length | Packet length |
| data | Data for write |
| error | Dynamixel error |

* + - * 1. Description

This function constructs the transmission packet with INST\_REG\_WRITE instruction, and TxRxPacket. Finally, it returns communication status.

* + 1. SyncReadTx
       - 1. Syntax

int SyndReadTx(PortHandler \*port, UINT16\_T address

, UINT16\_T data\_length, UINT8\_T\* param

, UINT16\_T param\_length)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| address | Address on the control table of Dynamixel |
| data\_length | Data length |
| param | Parameters |
| param\_length | Parameter length |

* + - * 1. Description

This function is not available with the Dynamixel Protocol 1.0. It returns COMM\_NOT\_AVAILABLE status.

* + 1. SyncWriteTxOnly
       - 1. Syntax

int SyncWriteTxOnly(PortHandler \*port, UINT16\_T start\_address

, UINT16\_T data\_length, UINT8\_T\* data

, UINT16\_T param\_length)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id | Dynamixel ID |
| address | Address on the control table of Dynamixel |
| data\_length | Data length |
| data | Data for write |
| param\_length | Parameter length |

* + - * 1. Description

This function constructs the transmission packet with INST\_SYNC\_WRITE instruction, and TxRxPacket. Finally, it returns communication status.

* + 1. BulkReadTx
       - 1. Syntax

int BulkReadTx(PortHandler \*port, UINT8\_T\* param

, UINT16\_T param\_length)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| param | Parameters |
| param\_length | Parameter length |

* + - * 1. Description

This function constructs the transmission packet with INST\_BULK\_READ instruction, and TxPacket. Then the function calls SetPacketTimeout function when TxPacket succeeds. Finally, it returns communication status.

* + 1. BulkWriteTxOnly
       - 1. Syntax

int BulkWriteTxOnly(PortHandler \*port, UINT8\_T\* param

, UINT16\_T param\_length)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| param | Parameters |
| param\_length | Parameter length |

* + - * 1. Description

This function is not available with the Dynamixel Protocol 1.0. It returns COMM\_NOT\_AVAILABLE status.

* 1. Protocol2PacketHandler
     + - 1. Description

This class handles the packet construction using communication Protocol 1.0.

* + - * 1. Members

None

* + - * 1. Methods

|  |  |
| --- | --- |
| GetInstance | gets instance of this class |
| ~Protocol2PacketHandler | None |
| GetProtocolVersion | gets Protocol version |
| TxPacket | transmits the packet |
| RxPacket | receives the packet |
| TxRxPacket | transmits and receives the packet |
| Ping | ping a Dynamixel |
| BroadcastPing | ping all connected Dynamixels |
| Action | commands ‘Run’ the Regwritten |
| RegWrite | writes the packets  walit for the ‘Action’ command |
| Reboot | reboots Dynamixel |
| FactoryReset | resets all Dynamixel settings |
| ReadTx | transmits N byte read instruction packet |
| ReadRx | receives N byte read status packet |
| ReadTxRx | transmits and receives N byte packet |
| Read1ByteTx | transmits 1 byte read instruction packet |
| Read1ByteRx | receives 1 byte read status packet |
| Read1ByteTxRx | transmits and receives 1 byte packet |
| Read2ByteTx | transmits 2 byte read instruction packet |
| Read2ByteRx | receives 2 byte read status packet |
| Read2ByteTxRx | transmits and receives 2 byte packet |
| Read4ByteTx | transmits 4 byte read instruction packet |
| Read4ByteRx | receives 4 byte read status packet |
| Read4ByteTxRx | transmits and receives 4 byte packet |
| WriteTxOnly | transmits N byte write instruction packet |
| WriteTxRx | transmits and receives N byte packet |
| Write1ByteTxOnly | transmits 1 byte write instruction packet |
| Write1ByteTxRx | transmits and receives 1 byte packet |
| Write2ByteTxOnly | transmits 2 byte write instruction packet |
| Write2ByteTxRx | transmits and receives 2 byte packet |
| Write4ByteTxOnly | transmits 4 byte write instruction packet |
| Write4ByteTxRx | transmits and receives 4 byte packet et |
| RegWriteTxOnly | transmits register write instruction packet |
| RegWriteTxRx | transmits and receives register write packet |
| SyncReadTx | transmits N byte sync read Instruction packet |
| SyncWriteTxOnly | transmits N byte sync write Instruction packet |
| BulkReadTx | transmits N byte bulk read Instruction packet |
| BulkWriteTxOnly | transmits N byte bulk write Instruction packet |

* + - * 1. Enumerator

None

* + 1. GetProtocolVersion
       - 1. Syntax

float GetProtocolVersion()

* + - * 1. Parameters

None

* + - * 1. Description

This function returns the protocol version set in the PacketHandler as a float value.

* + - * 1. Return Status

None

* + 1. TxPacket
       - 1. Syntax

int TxPacket(PortHandler \*port, UINT8\_T \*txpacket)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| txpacket | packet for transmission |

* + - * 1. Description

This function transmits the packet. The function clears the port by ClearPort function at first, and passes the txpacket to the WritePort function. The function activates while the port is not busy, and when the packet is written on the port buffer.

* + 1. RxPacket
       - 1. Syntax

int RxPacket(PortHandler \*port, UINT8\_T \*rxpacket)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| rxpacket | packet for reception |

* + - * 1. Description

This function repeatedly tries to receive packets by ReadPort function until whole packets that it waits to get have arrived, or the packet wait time limit is over. After r, it returns communication status value.

* + 1. TxRxPacket
       - 1. Syntax

int TxRxPacket(PortHandler \*port, UINT8\_T\* txpacket

, UINT8\_T\* rxpacket, UINT8\_T\* error = 0)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| txpacket | packet for transmission |
| rxpacket | packet for reception |
| error | Dynamixel error |

* + - * 1. Description

This function transmits and receives packets by TxPacket and RxPacket functions. When TxPacket function succeeds to communicate, it will continue to RxPacket and finishes the process if the packet succeeds to be received. In case of using the group handler functions for write, such as SyncWrite, and BulkWrite, they don’t use RxPacket, so the function finishes its operation immediately after the TxPacket. Before the RxPacket, it sets packet timeout if the packet instruction is INST\_READ.

* + 1. Ping
       - 1. Syntax

int Ping (PortHandler \*port, UINT8\_T id, UINT8\_T\* error = 0)

int Ping (PortHandler \*port, UINT8\_T id, UINT16\_T\* model\_number

, UINT8\_T\* error = 0)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id | Dynamixel ID |
| error | Dynamixel error |

* + - * 1. Description

This function constructs the transmission packet with INST\_PING instruction, and TxRxPacket. Then, the function tries to get the model number of the Dynamixel by ReadTxRx function. When it succeeds to receive the packet, it makes the model number value by using DXL\_MAKEWORD to put two byte-type data together. Finally, it returns communication status value.

* + 1. BroadcastPing
       - 1. Syntax

int BroadcastPing(PortHandler \*port

, std::vector<UINT8\_T> &id\_list)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id\_list | Dynamixel ID list |

* + - * 1. Description

This function.

* + 1. Action
       - 1. Syntax

int Action(PortHandler \*port, UINT8\_T id)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id | Dynamixel ID |

* + - * 1. Description

This function constructs the transmission packet with INST\_ACTION instruction, and TxRxPacket. Before using this function, the orders for the Dynamixels should be already written in the register in Dynamixel by RegWrite function. Finally, it returns communication status value.

* + 1. Reboot
       - 1. Syntax

int Reboot(PortHandler \*port, UINT8\_T id, UINT8\_T\* error)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id | Dynamixel ID |
| error | Dynamixel error |

* + - * 1. Description

This function constructs the transmission packet with INST\_REBOOT instruction, and TxRxPacket. Finally, it returns commnunication status.

* + 1. FactoryReset
       - 1. Syntax

int FactoryReset(PortHandler \*port, UINT8\_T id, UINT8\_T option

, UINT8\_T\* error)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id | Dynamixel ID |
| option | Reset option |
| error | Dynamixel error |

* + - * 1. Description

This function constructs the transmission packet with INST\_FACTORY\_RESET instruction, and TxRxPacket. This resets all Dynamixel settings to the factory default settings. The option changes the range of settings for reset:

0xFF : reset all values

0x01 : reset all values except ID

0x02 : reset all values except ID and Baudrate

Finally, it returns communication status.

* + 1. ReadTx
       - 1. Syntax

int ReadTx(PortHandler \*port, UINT8\_T id, UINT16\_T address

, UINT16\_T length)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id | Dynamixel ID |
| address | Address on the control table of Dynamixel |
| length | Packet length |

* + - * 1. Description

This function constructs the transmission packet with INST\_READ instruction, and TxPacket. Then the function calls SetPacketTimeout function when TxPacket succeeds. This function is not available for controlling more than one Dynamixel. Finally, it returns communication status.

* + 1. ReadRx
       - 1. Syntax

int ReadRx(PortHandler \*port, UINT16\_T length, UINT8\_T \*data

, UINT8\_T\* error)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| length | Packet length |
| data | Packet data |
| error | Dynamixel error |

* + - * 1. Description

This function calls RxPacket function and gets the reception packet if the communication succeeds. Finally, it returns communication status value.

* + 1. ReadTxRx
       - 1. Syntax

int ReadTxRx(PortHandler \*port, UINT8\_T id, UINT16\_T address

, UINT16\_T length, UINT8\_T \*data, UINT8\_T\* error)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id | Dynamixel ID |
| address | Address on the control table of Dynamixel |
| length | Packet length |
| data | Packet data |
| error | Dynamixel error |

* + - * 1. Description

This function calls ReadTx function and ReadRx function. ReadRx function will be called when ReadTx succeeds. Finally, it returns communication status value.

* + 1. Read1ByteTx
       - 1. Syntax

int Read1ByteTx(PortHandler \*port, UINT8\_T id, UINT16\_T address)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id | Dynamixel ID |
| address | Address on the control table of Dynamixel |

* + - * 1. Description

This function calls ReadTx function for 1 Byte packet transmission. Finally, it returns communication status.

* + 1. Read1ByteRx
       - 1. Syntax

int Read1ByteRx(PortHandler \*port, UINT8\_T\* data

, UINT8\_T\* error)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| data | Packet data |
| error | Dynamixel error |

* + - * 1. Description

This function calls ReadRx function for 1 Byte packet reception. Finally, it returns communication status.

* + 1. Read1ByteTxRx
       - 1. Syntax

int Read1ByteTxRx(PortHandler \*port, UINT8\_T id,

UINT16\_T address, UINT8\_T\* data, UINT8\_T\* error)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id | Dynamixel ID |
| address | Address on the control table of Dynamixel |
| data | Packet data |
| error | Dynamixel error |

* + - * 1. Description

This function calls Read1ByteTx function and Read1ByteRx function for 1 Byte packet transmission and reception. Read1ByteRx function will be called when Read1ByteTx succeeds. Finally, it returns communication status.

* + 1. Read2ByteTx
       - 1. Syntax

int Read2ByteTx(PortHandler \*port, UINT8\_T id, UINT16\_T address)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id | Dynamixel ID |
| address | Address on the control table of Dynamixel |

* + - * 1. Description

This function calls ReadTx function for 2 Byte packet transmission. Finally, it returns communication status.

* + 1. Read2ByteRx
       - 1. Syntax

int Read2ByteRx(PortHandler \*port, UINT16\_T\* data

, UINT8\_T\* error)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| data | Packet data |
| error | Dynamixel error |

* + - * 1. Description

This function calls ReadRx function for 2 Byte packet reception. Finally, it returns communication status.

* + 1. Read2ByteTxRx
       - 1. Syntax

int Read2ByteTxRx(PortHandler \*port, UINT8\_T id,

UINT16\_T address, UIN16\_T\* data, UINT8\_T\* error)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id | Dynamixel ID |
| address | Address on the control table of Dynamixel |
| data | Packet data |
| error | Dynamixel error |

* + - * 1. Description

This function calls Read2ByteTx function and Read2ByteRx function for 2 Byte packet transmission and reception. Read2ByteRx function will be called when Read2ByteTx succeeds. Finally, it returns communication status.

* + 1. Read4ByteTx
       - 1. Syntax

int Read4ByteTx(PortHandler \*port, UINT8\_T id, UINT16\_T address)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id | Dynamixel ID |
| address | Address on the control table of Dynamixel |

* + - * 1. Description

This function calls ReadTx function for 4 Byte packet transmission. Finally, it returns communication status.

* + 1. Read4ByteRx
       - 1. Syntax

int Read4ByteRx(PortHandler \*port, UINT32\_T\* data

, UINT8\_T\* error)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| data | Packet data |
| error | Dynamixel error |

* + - * 1. Description

This function calls ReadRx function for 4 Byte packet reception. Finally, it returns communication status.

* + 1. Read4ByteTxRx
       - 1. Syntax

int Read4ByteTxRx(PortHandler \*port, UIN8\_T id

, UINT16\_T address, UINT32\_T\* data, UINT8\_T\* error)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id | Dynamixel ID |
| address | Address on the control table of Dynamixel |
| data | Data for write |
| error | Dynamixel error |

* + - * 1. Description

This function calls Read4ByteTx function and Read4ByteRx function for 4 Byte packet transmission and reception. Read4ByteRx function will be called when Read4ByteTx succeeds. Finally, it returns communication status.

* + 1. WriteTxOnly
       - 1. Syntax

int WriteTxOnly(PortHandler \*port, UINT8\_T id, UINT16\_T address

, UINT16\_T length, UINT8\_T\* data)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id | Dynamixel ID |
| address | Address on the control table of Dynamixel |
| length | Packet length |
| data | Data for write |

* + - * 1. Description

This function constructs the transmission packet with INST\_WRITE instruction, and TxPacket. Finally, it returns communication status.

* + 1. WriteTxRx
       - 1. Syntax

int WriteTxRx(PortHandler \*port, UINT8\_T id, UINT16\_T address

, UINT16\_T length, UINT8\_T\* data, UINT8\_T\* error)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id | Dynamixel ID |
| address | Address on the control table of Dynamixel |
| length | Packet length |
| data | Data for write |
| error | Dynamixel error |

* + - * 1. Description

This function constructs the transmission packet with INST\_WRITE instruction, and TxRxPacket. Finally, it returns communication status.

* + 1. Write1ByteTxOnly
       - 1. Syntax

int Write1ByteTxOnly(PortHandler \*port, UINT8\_T id

, UINT16\_T address, UINT8\_T data)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id | Dynamixel ID |
| address | Address on the control table of Dynamixel |
| data | Data for write |

* + - * 1. Description

This function calls WriteTxOnly function for 1 Byte packet. Finally, it returns communication status.

* + 1. Write1ByteTxRx
       - 1. Syntax

int Write1ByteTxRx(PortHandler \*port, UINT8\_T id

, UINT16\_T address, UINT8\_T data, UINT8\_T\* error)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id | Dynamixel ID |
| address | Address on the control table of Dynamixel |
| data | Data for write |
| error | Dynamixel error |

* + - * 1. Description

This function calls WriteTxRx function for 1 Byte packet transmission and reception. Finally, it returns communication status.

* + 1. Write2ByteTxOnly
       - 1. Syntax

int Write2ByteTxOnly(PortHandler \*port, UINT8\_T id

, UINT16\_T address, UINT16\_T data)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id | Dynamixel ID |
| address | Address on the control table of Dynamixel |
| data | Data for write |

* + - * 1. Description

This function calls WriteTxOnly function for 2 Byte packet. Finally, it returns communication status.

* + 1. Write2ByteTxRx
       - 1. Syntax

int Write2ByteTxRx(PortHandler \*port, UINT8\_T id

, UINT16\_T address, UINT16\_T data, UINT8\_T\* error)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id | Dynamixel ID |
| address | Address on the control table of Dynamixel |
| data | Data for write |
| error | Dynamixel error |

* + - * 1. Description

This function calls WriteTxRx function for 2Byte packet transmission and reception. Finally, it returns communication status.

* + 1. Write4ByteTxOnly
       - 1. Syntax

int Write1ByteTxOnly(PortHandler \*port, UINT8\_T id

, UINT16\_T address, UINT32\_T data)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id | Dynamixel ID |
| address | Address on the control table of Dynamixel |
| data | Data for write |

* + - * 1. Description

This function calls WriteTxOnly function for 4 Byte packet. Finally, it returns communication status.

* + 1. Write4ByteTxRx
       - 1. Syntax

int Write4ByteTxRx(PortHandler \*port, UINT8\_T id

, UINT16\_T address, UINT32\_T data, UINT8\_T\* error)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id | Dynamixel ID |
| address | Address on the control table of Dynamixel |
| data | Data for write |
| error | Dynamixel error |

* + - * 1. Description

This function calls WriteTxRx function for 4Byte packet transmission and reception. Finally, it returns communication status.

* + 1. RegWriteTxOnly
       - 1. Syntax

int RegWriteTxOnly(PortHandler \*port, UINT8\_T id

, UINT16\_T address, UINT16\_T length, UINT8\_T\* data)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id | Dynamixel ID |
| address | Address on the control table of Dynamixel |
| length | Packet length |
| data | Data for write |

* + - * 1. Description

This function constructs the transmission packet with INST\_REG\_WRITE instruction, and TxPacket. Finally, it returns communication status.

* + 1. RegWriteTxRx
       - 1. Syntax

int RegWriteTxRx(PortHandler \*port, UINT8\_T id

, UINT16\_T address, UINT16\_T length

, UINT8\_T\* data, UINT8\_T\* error)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id | Dynamixel ID |
| address | Address on the control table of Dynamixel |
| length | Packet length |
| data | Data for write |
| error | Dynamixel error |

* + - * 1. Description

This function constructs the transmission packet with INST\_REG\_WRITE instruction, and TxRxPacket. Finally, it returns communication status.

* + 1. SyncReadTx
       - 1. Syntax

int SyndReadTx(PortHandler \*port, UINT16\_T address

, UINT16\_T data\_length, UINT8\_T\* param

, UINT16\_T param\_length)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| address | Address on the control table of Dynamixel |
| data\_length | Data length |
| param | Parameters |
| param\_length | Parameter length |

* + - * 1. Description

This function constructs the transmission packet with INST\_SYNC\_READ instruction, and TxPacket. Then the function calls SetPacketTimeout function when TxPacket succeeds. Finally, it returns communication status.

* + 1. SyncWriteTxOnly
       - 1. Syntax

int SyncWriteTxOnly(PortHandler \*port, UINT16\_T start\_address

, UINT16\_T data\_length, UINT8\_T\* data

, UINT16\_T param\_length)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| id | Dynamixel ID |
| address | Address on the control table of Dynamixel |
| data\_length | Data length |
| data | Data for write |
| param\_length | Parameter length |

* + - * 1. Description

This function constructs the transmission packet with INST\_SYNC\_WRITE instruction, and TxRxPacket. Finally, it returns communication status.

* + 1. BulkReadTx
       - 1. Syntax

int BulkReadTx(PortHandler \*port, UINT8\_T\* param

, UINT16\_T param\_length)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| param | Parameters |
| param\_length | Parameter length |

* + - * 1. Description

This function constructs the transmission packet with INST\_BULK\_READ instruction, and TxPacket. Then the function calls SetPacketTimeout function when TxPacket succeeds. Finally, it returns communication status.

* + 1. BulkWriteTxOnly
       - 1. Syntax

int BulkWriteTxOnly(PortHandler \*port, UINT8\_T\* param

, UINT16\_T param\_length)

* + - * 1. Parameters

|  |  |
| --- | --- |
| port | PortHandler instance |
| param | Parameters |
| param\_length | Parameter length |

* + - * 1. Description

This function constructs the transmission packet with INST\_BULK\_WRITE instruction, and TxRxPacket. Finally, it returns communication status.

* 1. GroupSyncRead
     + - 1. Description

This class handles a Dynamixel group to read data on same address simultaneously.

* + - * 1. Members

None

* + - * 1. Methods

|  |  |
| --- | --- |
| GroupSyncRead | Initializes groupSyncRead instance |
| ~GroupSyncRead | clears parameter storage |
| GetPortHandler | returns PortHandler instance |
| GetPacketHandler | returns PacketHandler instance |
| AddParam | adds parameter storage for read |
| RemoveParam | removes parameter on the storage |
| ClearParam | clears parameter storage |
| TxPacket | transmits packet to the number of Dynamixels |
| RxPacket | receives packet from the number of Dynamixels |
| TxRxPacket | transmits and receives packet on the number of Dynamixels |
| GetData | gets data from received packet |

* + - * 1. Enumerator

None

* + 1. AddParam
       - 1. Syntax

bool AddParam(UINT8\_T id)

* + - * 1. Parameters

|  |  |
| --- | --- |
| id | Dynamixel ID |

* + - * 1. Description

This function pushes id to the Dynamixel ID list, and initializes the parameter storage. It returns false when the class uses Protocol 1.0 or target ID exists already in the ID list, or returns true.

* + 1. RemoveParam
       - 1. Syntax

void RemoveParam(UINT8\_T id)

* + - * 1. Parameters

|  |  |
| --- | --- |
| id | Dynamixel ID |

* + - * 1. Description

This function removes target id in the Dynamixel ID list. It returns false when the class uses Protocol 1.0 or target ID does not exists in the ID list, or returns true.

* + 1. ClearParam
       - 1. Syntax

void ClearParam()

* + - * 1. Parameters

None

* + - * 1. Description

This function clears the Dynamixel ID list. It returns false when the class uses Protocol 1.0, or returns true.

* + 1. TxPacket
       - 1. Syntax

int TxPacket()

* + - * 1. Parameters

None

* + - * 1. Description

This function transmits the packet by using SyncReadTx function. It returns COMM\_NOT\_AVAILABLE when the class uses Protocol 1.0, or returns communication result.

* + 1. RxPacket
       - 1. Syntax

int RxPacket()

* + - * 1. Parameters

None

* + - * 1. Description

This function receives the packet by using ReadRx function. It returns COMM\_NOT\_AVAILABLE when the class uses Protocol 1.0 or there is no packet that had been received, or returns communication result.

* + 1. TxRxPacket
       - 1. Syntax

int TxRxPacket()

* + - * 1. Parameters

None

* + - * 1. Description

This function transmits and receives the packet by using TxPacket function and RxPacket function. It returns COMM\_NOT\_AVAILABLE when the class uses Protocol 1.0 or the packet transmission had not succeeded, or returns communication result.

* + 1. GetData
       - 1. Syntax

bool GetData(UINT8\_T id, UINT16\_T address, UINT8\_T\* data)

bool GetData(UINT8\_T id, UINT16\_T address, UINT16\_T\* data)

bool GetData(UINT8\_T id, UINT16\_T address, UINT32\_T\* data)

* + - * 1. Parameters

|  |  |
| --- | --- |
| id | Dynamixel ID |
| address | Address on the control table of Dynamixel |
| data | Packet data |

* + - * 1. Description

This function gets specific data from received packet. It returns false when the class uses Protocol 1.0 or there is no data from target address, or returns true.

* 1. GroupSyncWrite
     + - 1. Description

This class handles a Dynamixel group to write data into same address simultaneously.

* + - * 1. Members

None

* + - * 1. Methods

|  |  |
| --- | --- |
| GroupSyncWrite | Initializes groupSyncWrite instance |
| ~GroupSyncWrite | clears parameter storage |
| GetPortHandler | returns PortHandler instance |
| GetPacketHandler | returns PacketHandler instance |
| AddParam | adds parameter storage for read |
| RemoveParam | removes parameter on the storage |
| ChangeParam | changes parameter on the storage |
| ClearParam | clears parameter storage |
| TxPacket | transmits packet to the number of Dynamixels |

* + - * 1. Enumerator

None

* + 1. AddParam
       - 1. Syntax

bool AddParam(UINT8\_T id, UINT8\_T\* data)

* + - * 1. Parameters

|  |  |
| --- | --- |
| id | Dynamixel ID |
| data | data for write |

* + - * 1. Description

This function pushes id to the Dynamixel ID list, and initializes the parameter storage by adding data. It returns false when the target ID exists already in the ID list, or returns true.

* + 1. RemoveParam
       - 1. Syntax

void RemoveParam(UINT8\_T id)

* + - * 1. Parameters

|  |  |
| --- | --- |
| id | Dynamixel ID |

* + - * 1. Description

This function removes target id in the Dynamixel ID list. It returns false when the target ID does not exists in the ID list, or returns true.

* + 1. ChangeParam
       - 1. Syntax

bool ChangeParam(UINT8\_T id, UINT8\_T\* data)

* + - * 1. Parameters

|  |  |
| --- | --- |
| id | Dynamixel ID |
| data | data for write |

* + - * 1. Description

This function pushes new data to the parameter storage of same ID. It returns false when the target ID doesn’t exists in the ID list, or returns true.

* + 1. ClearParam
       - 1. Syntax

void ClearParam()

* + - * 1. Parameters

None

* + - * 1. Description

This function clears the Dynamixel ID list.

* + 1. TxPacket
       - 1. Syntax

int TxPacket()

* + - * 1. Parameters

None

* + - * 1. Description

This function transmits the packet by using SyncWriteTxOnly function. It returns COMM\_NOT\_AVAILABLE when there is no item on the Dynamixel ID list, or returns communication result.

* 1. GroupBulkRead
     + - 1. Description

This class handles a Dynamixel group to read data on different address simultaneously.

* + - * 1. Members

None

* + - * 1. Methods

|  |  |
| --- | --- |
| GroupBulkRead | Initializes groupBulkRead instance |
| ~GroupBulkRead | clears parameter storage |
| GetPortHandler | returns PortHandler instance |
| GetPacketHandler | returns PacketHandler instance |
| AddParam | adds parameter storage for read |
| RemoveParam | removes parameter on the storage |
| ClearParam | clears parameter storage |
| TxPacket | transmits packet to the number of Dynamixels |
| RxPacket | receives packet from the number of Dynamixels |
| TxRxPacket | transmits and receives packet on the number of Dynamixels |
| GetData | gets data from received packet |

* + - * 1. Enumerator

None

* + 1. AddParam
       - 1. Syntax

bool AddParam(UINT8\_T id, UINT16\_T start\_address

, UINT16\_T data\_length)

* + - * 1. Parameters

|  |  |
| --- | --- |
| id | Dynamixel ID |
| start\_address | Start address |
| data\_length | Data length |

* + - * 1. Description

This function pushes id to the Dynamixel ID list, and initializes the parameter storage by setting start\_address and data\_length. It returns false when the target ID exists already in the ID list, or returns true.

* + 1. RemoveParam
       - 1. Syntax

void RemoveParam(UINT8\_T id)

* + - * 1. Parameters

|  |  |
| --- | --- |
| id | Dynamixel ID |

* + - * 1. Description

This function removes target id in the Dynamixel ID list.

* + 1. ClearParam
       - 1. Syntax

void ClearParam()

* + - * 1. Parameters

None

* + - * 1. Description

This function clears the Dynamixel ID list..

* + 1. TxPacket
       - 1. Syntax

int TxPacket()

* + - * 1. Parameters

None

* + - * 1. Description

This function transmits the packet by using BulkReadTx function. It returns COMM\_NOT\_AVAILABLE when there is no item on the Dynamixel ID list, or returns communication result.

* + 1. RxPacket
       - 1. Syntax

int RxPacket()

* + - * 1. Parameters

None

* + - * 1. Description

This function receives the packet by using ReadRx function. It returns COMM\_RX\_FAIL when the packet reception had not succeeded, or returns COMM\_NOT\_AVAILABLE when there is no packet that had been received, or returns communication result.

* + 1. TxRxPacket
       - 1. Syntax

int TxRxPacket()

* + - * 1. Parameters

None

* + - * 1. Description

This function transmits and receives the packet by using TxPacket function and RxPacket function. It returns COMM\_TX\_FAIL when the packet transmission had not succeeded, or returns communication result.

* + 1. GetData
       - 1. Syntax

bool GetData(UINT8\_T id, UINT16\_T address, UINT8\_T\* data)

bool GetData(UINT8\_T id, UINT16\_T address, UINT16\_T\* data)

bool GetData(UINT8\_T id, UINT16\_T address, UINT32\_T\* data)

* + - * 1. Parameters

|  |  |
| --- | --- |
| id | Dynamixel ID |
| address | Address on the control table of Dynamixel |
| data | Packet data |

* + - * 1. Description

This function gets specific data from received packet. It returns false when there is no data from target address, or returns true.

* 1. GroupBulkWrite
     + - 1. Description

This class handles a Dynamixel group to write data into different address simultaneously.

* + - * 1. Members

None

* + - * 1. Methods

|  |  |
| --- | --- |
| GroupBulkWrite | Initializes groupBulkWrite instance |
| ~GroupBulkWrite | clears parameter storage |
| GetPortHandler | returns PortHandler instance |
| GetPacketHandler | returns PacketHandler instance |
| AddParam | adds parameter storage for read |
| RemoveParam | removes parameter on the storage |
| ChangeParam | changes parameter on the storage |
| ClearParam | clears parameter storage |
| TxPacket | transmits packet to the number of Dynamixels |

* + - * 1. Enumerator

None

* + 1. AddParam
       - 1. Syntax

bool AddParam(UINT8\_T id, UINT16\_T start\_address

, UINT16\_T data\_length, UINT8\_T\* data)

* + - * 1. Parameters

|  |  |
| --- | --- |
| id | Dynamixel ID |
| start\_address | Start address |
| data\_length | Data length |
| data | Data for write |

* + - * 1. Description

This function pushes id to the Dynamixel ID list, and initializes the parameter storage by setting start\_address and data\_length. It returns false when the target ID exists already in the ID list, or returns true.

* + 1. RemoveParam
       - 1. Syntax

void RemoveParam(UINT8\_T id)

* + - * 1. Parameters

|  |  |
| --- | --- |
| id | Dynamixel ID |

* + - * 1. Description

This function removes target id in the Dynamixel ID list.

* + 1. ChangeParam
       - 1. Syntax

bool ChangeParam(UINT8\_T id, UINT16\_T start\_address

, UINT16\_T data\_length, UINT8\_T\* data)

* + - * 1. Parameters

|  |  |
| --- | --- |
| id | Dynamixel ID |
| start\_address | Start address |
| data\_length | Data length |
| data | data for write |

* + - * 1. Description

This function pushes new data to the parameter storage of same ID. It returns false when the target ID doesn’t exists in the ID list, or returns true.

* + 1. ClearParam
       - 1. Syntax

void ClearParam()

* + - * 1. Parameters

None

* + - * 1. Description

This function clears the Dynamixel ID list..

* + 1. TxPacket
       - 1. Syntax

int TxPacket()

* + - * 1. Parameters

None

* + - * 1. Description

This function transmits the packet by using BulkWriteTxOnly function. It returns COMM\_NOT\_AVAILABLE when the class uses Protocol 1.0 or there is no item on the Dynamixel ID list, or returns communication result.