## [C Protocol 1.0](http://emanual.robotis.com/docs/en/software/dynamixel/dynamixel_sdk/sample_code/c_read_write_protocol_1_0/" \l "c-protocol-10)

### [C Read Write Protocol 1.0](http://emanual.robotis.com/docs/en/software/dynamixel/dynamixel_sdk/sample_code/c_read_write_protocol_1_0/" \l "c-read-write-protocol-10)

* Description

This example writes Goal Position to Dynamixel and reads Present Position of Dynamixel until it stops moving. The functions 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.

* Supported Dynamixels

Protocol 1.0 Dynamixels

#### Sample code

/\*

\* read\_write.c

\*

\* Created on: 2016. 5. 16.

\* Author: Leon Ryu Woon Jung

\*/

//

// \*\*\*\*\*\*\*\*\* Read and Write Example \*\*\*\*\*\*\*\*\*

//

//

// Available DXL model on this example : All models using Protocol 1.0

// This example is designed for using a Dynamixel MX-28, and an USB2DYNAMIXEL.

// To use another Dynamixel model, such as X series, see their details in E-Manual(support.robotis.com) and edit below "#define"d variables yourself.

// Be sure that Dynamixel MX properties are already set as %% ID : 1 / Baudnum : 1 (Baudrate : 1000000 [1M])

//

#ifdef \_\_linux\_\_

#include <unistd.h>

#include <fcntl.h>

#include <termios.h>

#elif defined(\_WIN32) || defined(\_WIN64)

#include <conio.h>

#endif

#include <stdlib.h>

#include <stdio.h>

#include "dynamixel\_sdk.h" // Uses Dynamixel SDK library

// Control table address

#define ADDR\_MX\_TORQUE\_ENABLE 24 // Control table address is different in Dynamixel model

#define ADDR\_MX\_GOAL\_POSITION 30

#define ADDR\_MX\_PRESENT\_POSITION 36

// Protocol version

#define PROTOCOL\_VERSION 1.0 // See which protocol version is used in the Dynamixel

// Default setting

#define DXL\_ID 1 // Dynamixel ID: 1

#define BAUDRATE 1000000

#define DEVICENAME "/dev/ttyUSB0" // Check which port is being used on your controller

// ex) Windows: "COM1" Linux: "/dev/ttyUSB0"

#define TORQUE\_ENABLE 1 // Value for enabling the torque

#define TORQUE\_DISABLE 0 // Value for disabling the torque

#define DXL\_MINIMUM\_POSITION\_VALUE 100 // Dynamixel will rotate between this value

#define DXL\_MAXIMUM\_POSITION\_VALUE 4000 // and this value (note that the Dynamixel would not move when the position value is out of movable range. Check e-manual about the range of the Dynamixel you use.)

#define DXL\_MOVING\_STATUS\_THRESHOLD 10 // Dynamixel moving status threshold

#define ESC\_ASCII\_VALUE 0x1b

int getch()

{

#ifdef \_\_linux\_\_

struct termios oldt, newt;

int ch;

tcgetattr(STDIN\_FILENO, &oldt);

newt = oldt;

newt.c\_lflag &= ~(ICANON | ECHO);

tcsetattr(STDIN\_FILENO, TCSANOW, &newt);

ch = getchar();

tcsetattr(STDIN\_FILENO, TCSANOW, &oldt);

return ch;

#elif defined(\_WIN32) || defined(\_WIN64)

return \_getch();

#endif

}

int kbhit(void)

{

#ifdef \_\_linux\_\_

struct termios oldt, newt;

int ch;

int oldf;

tcgetattr(STDIN\_FILENO, &oldt);

newt = oldt;

newt.c\_lflag &= ~(ICANON | ECHO);

tcsetattr(STDIN\_FILENO, TCSANOW, &newt);

oldf = fcntl(STDIN\_FILENO, F\_GETFL, 0);

fcntl(STDIN\_FILENO, F\_SETFL, oldf | O\_NONBLOCK);

ch = getchar();

tcsetattr(STDIN\_FILENO, TCSANOW, &oldt);

fcntl(STDIN\_FILENO, F\_SETFL, oldf);

if (ch != EOF)

{

ungetc(ch, stdin);

return 1;

}

return 0;

#elif defined(\_WIN32) || defined(\_WIN64)

return \_kbhit();

#endif

}

int main()

{

// Initialize PortHandler Structs

// Set the port path

// Get methods and members of PortHandlerLinux or PortHandlerWindows

int port\_num = portHandler(DEVICENAME);

// Initialize PacketHandler Structs

packetHandler();

int index = 0;

int dxl\_comm\_result = COMM\_TX\_FAIL; // Communication result

int dxl\_goal\_position[2] = { DXL\_MINIMUM\_POSITION\_VALUE, DXL\_MAXIMUM\_POSITION\_VALUE }; // Goal position

uint8\_t dxl\_error = 0; // Dynamixel error

uint16\_t dxl\_present\_position = 0; // Present position

// Open port

if (openPort(port\_num))

{

printf("Succeeded to open the port!\n");

}

else

{

printf("Failed to open the port!\n");

printf("Press any key to terminate...\n");

getch();

return 0;

}

// Set port baudrate

if (setBaudRate(port\_num, BAUDRATE))

{

printf("Succeeded to change the baudrate!\n");

}

else

{

printf("Failed to change the baudrate!\n");

printf("Press any key to terminate...\n");

getch();

return 0;

}

// Enable DXL Torque

write1ByteTxRx(port\_num, PROTOCOL\_VERSION, DXL\_ID, ADDR\_MX\_TORQUE\_ENABLE, TORQUE\_ENABLE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

else

{

printf("Dynamixel has been successfully connected \n");

}

while (1)

{

printf("Press any key to continue! (or press ESC to quit!)\n");

if (getch() == ESC\_ASCII\_VALUE)

break;

// Write goal position

write2ByteTxRx(port\_num, PROTOCOL\_VERSION, DXL\_ID, ADDR\_MX\_GOAL\_POSITION, dxl\_goal\_position[index]);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

do

{

// Read present position

dxl\_present\_position = read2ByteTxRx(port\_num, PROTOCOL\_VERSION, DXL\_ID, ADDR\_MX\_PRESENT\_POSITION);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

printf("[ID:%03d] GoalPos:%03d PresPos:%03d\n", DXL\_ID, dxl\_goal\_position[index], dxl\_present\_position);

} while ((abs(dxl\_goal\_position[index] - dxl\_present\_position) > DXL\_MOVING\_STATUS\_THRESHOLD));

// Change goal position

if (index == 0)

{

index = 1;

}

else

{

index = 0;

}

}

// Disable Dynamixel Torque

write1ByteTxRx(port\_num, PROTOCOL\_VERSION, DXL\_ID, ADDR\_MX\_TORQUE\_ENABLE, TORQUE\_DISABLE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

// Close port

closePort(port\_num);

return 0;

}

#### Details

#ifdef \_\_linux\_\_

#include <unistd.h>

#include <fcntl.h>

#include <termios.h>

#elif defined(\_WIN32) || defined(\_WIN64)

#include <conio.h>

#endif

This source includes above to get key input interruption while the example is running. Actual functions for getting the input is described in a little below.

#include <stdlib.h>

The function abs() is in the example code, and it needs stdlib.h to be included.

#include <stdio.h>

The example shows Dynamixel status in sequence by the function printf(). So here stdio.h is needed.

#include "dynamixel\_sdk.h" // Uses Dynamixel SDK library

All libraries of Dynamixel SDK are linked with the header file dynamixel\_sdk.h.

// Control table address

#define ADDR\_MX\_TORQUE\_ENABLE 24 // Control table address is different in Dynamixel model

#define ADDR\_MX\_GOAL\_POSITION 30

#define ADDR\_MX\_PRESENT\_POSITION 36

Dynamixel series have their own control tables: Addresses and Byte Length in each items. To control one of the items, its address (and length if necessary) is required. Find your requirements in <http://emanual.robotis.com/>.

// Protocol version

#define PROTOCOL\_VERSION 1.0 // See which protocol version is used in the Dynamixel

Dynamixel uses either or both protocols: Protocol 1.0 and Protocol 2.0. Choose one of the Protocol which is appropriate in the Dynamixel.

// Default setting

#define DXL\_ID 1 // Dynamixel ID: 1

#define BAUDRATE 1000000

#define DEVICENAME "/dev/ttyUSB0" // Check which port is being used on your controller

// ex) Windows: "COM1" Linux: "/dev/ttyUSB0"

#define TORQUE\_ENABLE 1 // Value for enabling the torque

#define TORQUE\_DISABLE 0 // Value for disabling the torque

#define DXL\_MINIMUM\_POSITION\_VALUE 100 // Dynamixel will rotate between this value

#define DXL\_MAXIMUM\_POSITION\_VALUE 4000 // and this value (note that the Dynamixel would not move when the position value is out of movable range. Check e-manual about the range of the Dynamixel you use.)

#define DXL\_MOVING\_STATUS\_THRESHOLD 10 // Dynamixel moving status threshold

#define ESC\_ASCII\_VALUE 0x1b

Here we set some variables to let you freely change them and use them to run the example code.

As the document already said in [previous chapter](http://emanual.robotis.com/docs/en/software/dynamixel/dynamixel_sdk/device_setup/" \l "dynamixel), customize Dynamixel control table items, such as DXL\_ID number, communication BAUDRATE, and the DEVICENAME, on your own terms of needs. In particular, BAUDRATE and DEVICENAME have systematical dependencies on your controller, so make clear what kind of communication method you will use.

Dynamixel basically needs the TORQUE\_ENABLE to be rotating or give you its internal information. On the other hand, it doesn’t need torque enabled if you get your goal, so finally do TORQUE\_DISABLE to prepare to the next sequence.

Since the Dynamixel has its own rotation range, it may shows malfunction if your request on your dynamixel is out of range. For example, Dynamixel MX-28 and Dynamixel PRO 54-200 has its rotatable range as 0 ~ 4028 and -250950 ~ 250950, each.

DXL\_MOVING\_STATUS\_THRESHOLD acts as a criteria for verifying its rotation stopped.

int getch()

{

#ifdef \_\_linux\_\_

struct termios oldt, newt;

int ch;

tcgetattr(STDIN\_FILENO, &oldt);

newt = oldt;

newt.c\_lflag &= ~(ICANON | ECHO);

tcsetattr(STDIN\_FILENO, TCSANOW, &newt);

ch = getchar();

tcsetattr(STDIN\_FILENO, TCSANOW, &oldt);

return ch;

#elif defined(\_WIN32) || defined(\_WIN64)

return \_getch();

#endif

}

int kbhit(void)

{

#ifdef \_\_linux\_\_

struct termios oldt, newt;

int ch;

int oldf;

tcgetattr(STDIN\_FILENO, &oldt);

newt = oldt;

newt.c\_lflag &= ~(ICANON | ECHO);

tcsetattr(STDIN\_FILENO, TCSANOW, &newt);

oldf = fcntl(STDIN\_FILENO, F\_GETFL, 0);

fcntl(STDIN\_FILENO, F\_SETFL, oldf | O\_NONBLOCK);

ch = getchar();

tcsetattr(STDIN\_FILENO, TCSANOW, &oldt);

fcntl(STDIN\_FILENO, F\_SETFL, oldf);

if (ch != EOF)

{

ungetc(ch, stdin);

return 1;

}

return 0;

#elif defined(\_WIN32) || defined(\_WIN64)

return \_kbhit();

#endif

}

These functions accept the key inputs in terms of example action. The example codes mainly apply the function getch() rather than the function kbhit() to get information which key has been pressed.

int main()

{

// Initialize PortHandler Structs

// Set the port path

// Get methods and members of PortHandlerLinux or PortHandlerWindows

int port\_num = portHandler(DEVICENAME);

// Initialize PacketHandler Structs

packetHandler();

int index = 0;

int dxl\_comm\_result = COMM\_TX\_FAIL; // Communication result

int dxl\_goal\_position[2] = { DXL\_MINIMUM\_POSITION\_VALUE, DXL\_MAXIMUM\_POSITION\_VALUE }; // Goal position

uint8\_t dxl\_error = 0; // Dynamixel error

uint16\_t dxl\_present\_position = 0; // Present position

// Open port

if (openPort(port\_num))

{

printf("Succeeded to open the port!\n");

}

else

{

printf("Failed to open the port!\n");

printf("Press any key to terminate...\n");

getch();

return 0;

}

// Set port baudrate

if (setBaudRate(port\_num, BAUDRATE))

{

printf("Succeeded to change the baudrate!\n");

}

else

{

printf("Failed to change the baudrate!\n");

printf("Press any key to terminate...\n");

getch();

return 0;

}

// Enable DXL Torque

write1ByteTxRx(port\_num, PROTOCOL\_VERSION, DXL\_ID, ADDR\_MX\_TORQUE\_ENABLE, TORQUE\_ENABLE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

else

{

printf("Dynamixel has been successfully connected \n");

}

while (1)

{

printf("Press any key to continue! (or press ESC to quit!)\n");

if (getch() == ESC\_ASCII\_VALUE)

break;

// Write goal position

write2ByteTxRx(port\_num, PROTOCOL\_VERSION, DXL\_ID, ADDR\_MX\_GOAL\_POSITION, dxl\_goal\_position[index]);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

do

{

// Read present position

dxl\_present\_position = read2ByteTxRx(port\_num, PROTOCOL\_VERSION, DXL\_ID, ADDR\_MX\_PRESENT\_POSITION);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

printf("[ID:%03d] GoalPos:%03d PresPos:%03d\n", DXL\_ID, dxl\_goal\_position[index], dxl\_present\_position);

} while ((abs(dxl\_goal\_position[index] - dxl\_present\_position) > DXL\_MOVING\_STATUS\_THRESHOLD));

// Change goal position

if (index == 0)

{

index = 1;

}

else

{

index = 0;

}

}

// Disable Dynamixel Torque

write1ByteTxRx(port\_num, PROTOCOL\_VERSION, DXL\_ID, ADDR\_MX\_TORQUE\_ENABLE, TORQUE\_DISABLE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

// Close port

closePort(port\_num);

return 0;

}

In main() function, the codes call actual functions for Dynamixel control.

// Initialize PortHandler Structs

// Set the port path

// Get methods and members of PortHandlerLinux or PortHandlerWindows

int port\_num = portHandler(DEVICENAME);

portHandler() function sets port path as DEVICENAME and get port\_num, and prepares an appropriate functions for port control in controller OS automatically. port\_num would be used in many functions in the body of the code to specify the port for use.

// Initialize PacketHandler Structs

packetHandler();

packetHandler() function initializes parameters used for packet construction and packet storing.

int index = 0;

int dxl\_comm\_result = COMM\_TX\_FAIL; // Communication result

int dxl\_goal\_position[2] = {DXL\_MINIMUM\_POSITION\_VALUE, DXL\_MAXIMUM\_POSITION\_VALUE}; // Goal position

uint8\_t dxl\_error = 0; // Dynamixel error

uint16\_t dxl\_present\_position = 0; // Present position

index variable points the direction to where the Dynamixel should be rotated.

dxl\_comm\_result indicates which error has been occurred during packet communication.

dxl\_goal\_position stores goal points of Dynamixel rotation.

dxl\_error shows the internal error in Dynamixel.

dxl\_present\_position views where now it points out.

// Open port

if (openPort(port\_num))

{

printf("Succeeded to open the port!\n");

}

else

{

printf("Failed to open the port!\n");

printf("Press any key to terminate...\n");

\_getch();

return 0;

}

First, controller opens #port\_num port to do serial communication with the Dynamixel. If it fails to open the port, the example will be terminated.

// Set port baudrate

if (setBaudRate(port\_num, BAUDRATE))

{

printf("Succeeded to change the baudrate!\n");

}

else

{

printf("Failed to change the baudrate!\n");

printf("Press any key to terminate...\n");

\_getch();

return 0;

}

Secondly, the controller sets the communication BAUDRATE at #port\_num port opened previously.

// Enable DXL Torque

write1ByteTxRx(port\_num, PROTOCOL\_VERSION, DXL\_ID, ADDR\_MX\_TORQUE\_ENABLE, TORQUE\_ENABLE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

else

{

printf("Dynamixel has been successfully connected \n");

}

As mentioned in the document, above code enables Dynamixel torque to set its status as being ready to move.

write1ByteTxRx() function orders to the #DXL\_ID Dynamixel in PROTOCOL\_VERSION communication protocol through #port\_num port, writing 1 byte of TORQUE\_ENABLE value to ADDR\_MX\_TORQUE\_ENABLE address. The function checks Tx/Rx result and receives Hardware error. getLastTxRxResult() function and getLastRxPacketError() function get either, and then printTxRxResult() function and printRxPacketError() function show results on the console window if any communication error or Hardware error has been occurred.

while (1)

{

printf("Press any key to continue! (or press ESC to quit!)\n");

if (getch() == ESC\_ASCII\_VALUE)

break;

// Write goal position

write2ByteTxRx(port\_num, PROTOCOL\_VERSION, DXL\_ID, ADDR\_MX\_GOAL\_POSITION, dxl\_goal\_position[index]);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

do

{

// Read present position

dxl\_present\_position = read2ByteTxRx(port\_num, PROTOCOL\_VERSION, DXL\_ID, ADDR\_MX\_PRESENT\_POSITION);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

printf("[ID:%03d] GoalPos:%03d PresPos:%03d\n", DXL\_ID, dxl\_goal\_position[index], dxl\_present\_position);

} while ((abs(dxl\_goal\_position[index] - dxl\_present\_position) > DXL\_MOVING\_STATUS\_THRESHOLD));

// Change goal position

if (index == 0)

{

index = 1;

}

else

{

index = 0;

}

}

During while() loop, the controller writes and reads the Dynamixel position through packet transmission/reception(Tx/Rx).

To continue its rotation, press any key except ESC.

write2ByteTxRx() function orders to the #DXL\_ID Dynamixel in PROTOCOL\_VERSION communication protocol through #port\_num port, writing 2 byte of dxl\_goal\_position[index] value to ADDR\_MX\_GOAL\_POSITION address. The function checks Tx/Rx result and receives Hardware error. getLastTxRxResult() function and getLastRxPacketError() function get either, and then printTxRxResult() function and printRxPacketError() function show results on the console window if any communication error or Hardware error has been occurred.

read2ByteTxRx() function orders to the #DXL\_ID Dynamixel in PROTOCOL\_VERSION communication protocol through #port\_num port, requesting 2 bytes of value in ADDR\_MX\_PRESENT\_POSITION address. The function checks Tx/Rx result and receives Hardware error. getLastTxRxResult() function and getLastRxPacketError() function get either, and then printTxRxResult() function and printRxPacketError() function show results on the console window if any communication error or Hardware error has been occurred.

Reading its present position will be ended when absolute value of (dxl\_goal\_position[index] - dxl\_present\_position) becomes smaller then DXL\_MOVING\_STATUS\_THRESHOLD.

At last, it changes its direction to the counter-wise and waits for extra key input.

// Disable Dynamixel Torque

write1ByteTxRx(port\_num, PROTOCOL\_VERSION, DXL\_ID, ADDR\_MX\_TORQUE\_ENABLE, TORQUE\_DISABLE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

The controller frees the Dynamixel to be idle.

write1ByteTxRx() function orders to the #DXL\_ID Dynamixel in PROTOCOL\_VERSION communication protocol through #port\_num port, writing 1 byte of TORQUE\_DISABLE value to ADDR\_MX\_TORQUE\_ENABLE address. The function checks Tx/Rx result and receives Hardware error. getLastTxRxResult() function and getLastRxPacketError() function get either, and then printTxRxResult() function and printRxPacketError() function show results on the console window if any communication error or Hardware error has been occurred.

// Close port

closePort(port\_num);

return 0;

Finally, port becomes disposed.

### [C Multi Port Protocol 1.0](http://emanual.robotis.com/docs/en/software/dynamixel/dynamixel_sdk/sample_code/c_multi_port_protocol_1_0/" \l "c-multi-port-protocol-10)

* Description

This example writes Goal Positions to two Dynamixels connected via two ports and reads their Present Positions until they stop moving.

* Supported Dynamixels

Protocol 1.0 Dynamixels

#### Sample code

/\*

\* multi\_port.c

\*

\* Created on: 2016. 5. 16.

\* Author: Leon Ryu Woon Jung

\*/

//

// \*\*\*\*\*\*\*\*\* Multi Port Example \*\*\*\*\*\*\*\*\*

//

//

// Available Dynamixel model on this example : All models using Protocol 1.0

// This example is designed for using two Dynamixel MX-28, and two USB2DYNAMIXEL.

// To use another Dynamixel model, such as X series, see their details in E-Manual(support.robotis.com) and edit below "#define"d variables yourself.

// Be sure that Dynamixel MX properties are already set as %% ID : 1 / Baudnum : 1 (Baudrate : 1000000 [1M])

//

#ifdef \_\_linux\_\_

#include <unistd.h>

#include <fcntl.h>

#include <termios.h>

#elif defined(\_WIN32) || defined(\_WIN64)

#include <conio.h>

#endif

#include <stdlib.h>

#include <stdio.h>

#include "dynamixel\_sdk.h" // Uses Dynamixel SDK library

// Control table address

#define ADDR\_MX\_TORQUE\_ENABLE 24 // Control table address is different in Dynamixel model

#define ADDR\_MX\_GOAL\_POSITION 30

#define ADDR\_MX\_PRESENT\_POSITION 36

// Protocol version

#define PROTOCOL\_VERSION 1.0 // See which protocol version is used in the Dynamixel

// Default setting

#define DXL1\_ID 1 // Dynamixel#1 ID: 1

#define DXL2\_ID 2 // Dynamixel#2 ID: 2

#define BAUDRATE 1000000

#define DEVICENAME1 "/dev/ttyUSB0" // Check which port is being used on your controller

#define DEVICENAME2 "/dev/ttyUSB1" // ex) Windows: "COM1" Linux: "/dev/ttyUSB0"

#define TORQUE\_ENABLE 1 // Value for enabling the torque

#define TORQUE\_DISABLE 0 // Value for disabling the torque

#define DXL\_MINIMUM\_POSITION\_VALUE 100 // Dynamixel will rotate between this value

#define DXL\_MAXIMUM\_POSITION\_VALUE 4000 // and this value (note that the Dynamixel would not move when the position value is out of movable range. Check e-manual about the range of the Dynamixel you use.)

#define DXL\_MOVING\_STATUS\_THRESHOLD 10 // Dynamixel moving status threshold

#define ESC\_ASCII\_VALUE 0x1b

int getch()

{

#ifdef \_\_linux\_\_

struct termios oldt, newt;

int ch;

tcgetattr(STDIN\_FILENO, &oldt);

newt = oldt;

newt.c\_lflag &= ~(ICANON | ECHO);

tcsetattr(STDIN\_FILENO, TCSANOW, &newt);

ch = getchar();

tcsetattr(STDIN\_FILENO, TCSANOW, &oldt);

return ch;

#elif defined(\_WIN32) || defined(\_WIN64)

return \_getch();

#endif

}

int kbhit(void)

{

#ifdef \_\_linux\_\_

struct termios oldt, newt;

int ch;

int oldf;

tcgetattr(STDIN\_FILENO, &oldt);

newt = oldt;

newt.c\_lflag &= ~(ICANON | ECHO);

tcsetattr(STDIN\_FILENO, TCSANOW, &newt);

oldf = fcntl(STDIN\_FILENO, F\_GETFL, 0);

fcntl(STDIN\_FILENO, F\_SETFL, oldf | O\_NONBLOCK);

ch = getchar();

tcsetattr(STDIN\_FILENO, TCSANOW, &oldt);

fcntl(STDIN\_FILENO, F\_SETFL, oldf);

if (ch != EOF)

{

ungetc(ch, stdin);

return 1;

}

return 0;

#elif defined(\_WIN32) || defined(\_WIN64)

return \_kbhit();

#endif

}

int main()

{

// Initialize PortHandler Structs

// Set the port path

// Get methods and members of PortHandlerLinux or PortHandlerWindows

int port\_num1 = portHandler(DEVICENAME1);

int port\_num2 = portHandler(DEVICENAME2);

// Initialize PacketHandler Structs

packetHandler();

int index = 0;

int dxl\_comm\_result = COMM\_TX\_FAIL; // Communication result

int dxl\_goal\_position[2] = { DXL\_MINIMUM\_POSITION\_VALUE, DXL\_MAXIMUM\_POSITION\_VALUE }; // Goal position

uint8\_t dxl\_error = 0; // Dynamixel error

uint16\_t dxl1\_present\_position = 0, dxl2\_present\_position = 0; // Present position

// Open port1

if (openPort(port\_num1))

{

printf("Succeeded to open the port1!\n");

}

else

{

printf("Failed to open the port1!\n");

printf("Press any key to terminate...\n");

getch();

return 0;

}

// Open port2

if (openPort(port\_num2))

{

printf("Succeeded to open the port2!\n");

}

else

{

printf("Failed to open the port2!\n");

printf("Press any key to terminate...\n");

getch();

return 0;

}

// Set port1 baudrate

if (setBaudRate(port\_num1, BAUDRATE))

{

printf("Succeed to change the baudrate port1!\n");

}

else

{

printf("Failed to change the baudrate port1!\n");

printf("Press any key to terminate...\n");

getch();

return 0;

}

// Set port2 baudrate

if (setBaudRate(port\_num2, BAUDRATE))

{

printf("Succeed to change the baudrate port2!\n");

}

else

{

printf("Failed to change the baudrate port2!\n");

printf("Press any key to terminate...\n");

getch();

return 0;

}

// Enable Dynamixel#1 Torque

write1ByteTxRx(port\_num1, PROTOCOL\_VERSION, DXL1\_ID, ADDR\_MX\_TORQUE\_ENABLE, TORQUE\_ENABLE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num1, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num1, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

else

{

printf("Dynamixel#%d has been successfully connected \n", DXL1\_ID);

}

// Enable Dynamixel#2 Torque

write1ByteTxRx(port\_num2, PROTOCOL\_VERSION, DXL2\_ID, ADDR\_MX\_TORQUE\_ENABLE, TORQUE\_ENABLE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num2, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num2, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

else

{

printf("Dynamixel#%d has been successfully connected \n", DXL2\_ID);

}

while (1)

{

printf("Press any key to continue! (or press ESC to quit!)\n");

if (getch() == ESC\_ASCII\_VALUE)

break;

// Write Dynamixel#1 goal position

write2ByteTxRx(port\_num1, PROTOCOL\_VERSION, DXL1\_ID, ADDR\_MX\_GOAL\_POSITION, dxl\_goal\_position[index]);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num1, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num1, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

// Write Dynamixel#2 goal position

write2ByteTxRx(port\_num2, PROTOCOL\_VERSION, DXL2\_ID, ADDR\_MX\_GOAL\_POSITION, dxl\_goal\_position[index]);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num2, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num2, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

do

{

// Read Dynamixel#1 present position

dxl1\_present\_position = read2ByteTxRx(port\_num1, PROTOCOL\_VERSION, DXL1\_ID, ADDR\_MX\_PRESENT\_POSITION);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num1, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num1, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

// Read Dynamixel#2 present position

dxl2\_present\_position = read2ByteTxRx(port\_num2, PROTOCOL\_VERSION, DXL2\_ID, ADDR\_MX\_PRESENT\_POSITION);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num2, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num2, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

printf("[ID:%03d] GoalPos:%03d PresPos:%03d\t[ID:%03d] GoalPos:%03d PresPos:%03d\n", DXL1\_ID, dxl\_goal\_position[index], dxl1\_present\_position, DXL2\_ID, dxl\_goal\_position[index], dxl2\_present\_position);

} while ((abs(dxl\_goal\_position[index] - dxl1\_present\_position) > DXL\_MOVING\_STATUS\_THRESHOLD) || (abs(dxl\_goal\_position[index] - dxl2\_present\_position) > DXL\_MOVING\_STATUS\_THRESHOLD));

// Change goal position

if (index == 0)

{

index = 1;

}

else

{

index = 0;

}

}

// Disable Dynamixel#1 Torque

write1ByteTxRx(port\_num1, PROTOCOL\_VERSION, DXL1\_ID, ADDR\_MX\_TORQUE\_ENABLE, TORQUE\_DISABLE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num1, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num1, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

// Disable Dynamixel#2 Torque

write1ByteTxRx(port\_num2, PROTOCOL\_VERSION, DXL2\_ID, ADDR\_MX\_TORQUE\_ENABLE, TORQUE\_DISABLE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num2, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num2, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

// Close port1

closePort(port\_num1);

// Close port2

closePort(port\_num2);

return 0;

}

#### Details

#ifdef \_\_linux\_\_

#include <unistd.h>

#include <fcntl.h>

#include <termios.h>

#elif defined(\_WIN32) || defined(\_WIN64)

#include <conio.h>

#endif

This source includes above to get key input interruption while the example is running. Actual functions for getting the input is described in a little below.

#include <stdlib.h>

The function abs() is in the example code, and it needs stdlib.h to be included.

#include <stdio.h>

The example shows Dynamixel status in sequence by the function printf(). So here stdio.h is needed.

#include "dynamixel\_sdk.h" // Uses Dynamixel SDK library

All libraries of Dynamixel SDK are linked with the header file dynamixel\_sdk.h.

// Control table address

#define ADDR\_MX\_TORQUE\_ENABLE 24 // Control table address is different in Dynamixel model

#define ADDR\_MX\_GOAL\_POSITION 30

#define ADDR\_MX\_PRESENT\_POSITION 36

Dynamixel series have their own control tables: Addresses and Byte Length in each items. To control one of the items, its address (and length if necessary) is required. Find your requirements in <http://emanual.robotis.com/>.

// Protocol version

#define PROTOCOL\_VERSION 1.0 // See which protocol version is used in the Dynamixel

Dynamixel uses either or both protocols: Protocol 1.0 and Protocol 2.0. Choose one of the Protocol which is appropriate in the Dynamixel.

// Default setting

#define DXL1\_ID 1 // Dynamixel#1 ID: 1

#define DXL2\_ID 2 // Dynamixel#2 ID: 2

#define BAUDRATE 1000000

#define DEVICENAME1 "/dev/ttyUSB0" // Check which port is being used on your controller

#define DEVICENAME2 "/dev/ttyUSB1" // ex) Windows: "COM1" Linux: "/dev/ttyUSB0"

#define TORQUE\_ENABLE 1 // Value for enabling the torque

#define TORQUE\_DISABLE 0 // Value for disabling the torque

#define DXL\_MINIMUM\_POSITION\_VALUE 100 // Dynamixel will rotate between this value

#define DXL\_MAXIMUM\_POSITION\_VALUE 4000 // and this value (note that the Dynamixel would not move when the position value is out of movable range. Check e-manual about the range of the Dynamixel you use.)

#define DXL\_MOVING\_STATUS\_THRESHOLD 10 // Dynamixel moving status threshold

#define ESC\_ASCII\_VALUE 0x1b

Here we set some variables to let you freely change them and use them to run the example code.

As the document previously said in [previous chapter](http://emanual.robotis.com/docs/en/software/dynamixel/dynamixel_sdk/device_setup/" \l "dynamixel), customize Dynamixel control table items, such as DXL\_ID number, communication BAUDRATE, and the DEVICENAME, on your own terms of needs. In particular, BAUDRATE and DEVICENAME have systematical dependencies on your controller, so make clear what kind of communication method you will use.

The example uses two Dynamixels DXL1\_ID, DXL2\_ID connected with each ports DEVICENAME1, DEVICENAME2

Dynamixel basically needs the TORQUE\_ENABLE to be rotating or give you its internal information. On the other hand, it doesn’t need torque enabled if you get your goal, so finally do TORQUE\_DISABLE to prepare to the next sequence.

Since the Dynamixel has its own rotation range, it may shows malfunction if your request on your dynamixel is out of range. For example, Dynamixel MX-28 and Dynamixel PRO 54-200 has its rotatable range as 0 ~ 4028 and -250950 ~ 250950, each.

DXL\_MOVING\_STATUS\_THRESHOLD acts as a criteria for verifying its rotation stopped.

int getch()

{

#ifdef \_\_linux\_\_

struct termios oldt, newt;

int ch;

tcgetattr(STDIN\_FILENO, &oldt);

newt = oldt;

newt.c\_lflag &= ~(ICANON | ECHO);

tcsetattr(STDIN\_FILENO, TCSANOW, &newt);

ch = getchar();

tcsetattr(STDIN\_FILENO, TCSANOW, &oldt);

return ch;

#elif defined(\_WIN32) || defined(\_WIN64)

return \_getch();

#endif

}

int kbhit(void)

{

#ifdef \_\_linux\_\_

struct termios oldt, newt;

int ch;

int oldf;

tcgetattr(STDIN\_FILENO, &oldt);

newt = oldt;

newt.c\_lflag &= ~(ICANON | ECHO);

tcsetattr(STDIN\_FILENO, TCSANOW, &newt);

oldf = fcntl(STDIN\_FILENO, F\_GETFL, 0);

fcntl(STDIN\_FILENO, F\_SETFL, oldf | O\_NONBLOCK);

ch = getchar();

tcsetattr(STDIN\_FILENO, TCSANOW, &oldt);

fcntl(STDIN\_FILENO, F\_SETFL, oldf);

if (ch != EOF)

{

ungetc(ch, stdin);

return 1;

}

return 0;

#elif defined(\_WIN32) || defined(\_WIN64)

return \_kbhit();

#endif

}

These functions accept the key inputs in terms of example action. The example codes mainly apply the function getch() rather than the function kbhit() to get information which key has been pressed.

int main()

{

// Initialize PortHandler Structs

// Set the port path

// Get methods and members of PortHandlerLinux or PortHandlerWindows

int port\_num1 = portHandler(DEVICENAME1);

int port\_num2 = portHandler(DEVICENAME2);

// Initialize PacketHandler Structs

packetHandler();

int index = 0;

int dxl\_comm\_result = COMM\_TX\_FAIL; // Communication result

int dxl\_goal\_position[2] = { DXL\_MINIMUM\_POSITION\_VALUE, DXL\_MAXIMUM\_POSITION\_VALUE }; // Goal position

uint8\_t dxl\_error = 0; // Dynamixel error

uint16\_t dxl1\_present\_position = 0, dxl2\_present\_position = 0; // Present position

// Open port1

if (openPort(port\_num1))

{

printf("Succeeded to open the port1!\n");

}

else

{

printf("Failed to open the port1!\n");

printf("Press any key to terminate...\n");

getch();

return 0;

}

// Open port2

if (openPort(port\_num2))

{

printf("Succeeded to open the port2!\n");

}

else

{

printf("Failed to open the port2!\n");

printf("Press any key to terminate...\n");

getch();

return 0;

}

// Set port1 baudrate

if (setBaudRate(port\_num1, BAUDRATE))

{

printf("Succeed to change the baudrate port1!\n");

}

else

{

printf("Failed to change the baudrate port1!\n");

printf("Press any key to terminate...\n");

getch();

return 0;

}

// Set port2 baudrate

if (setBaudRate(port\_num2, BAUDRATE))

{

printf("Succeed to change the baudrate port2!\n");

}

else

{

printf("Failed to change the baudrate port2!\n");

printf("Press any key to terminate...\n");

getch();

return 0;

}

// Enable Dynamixel#1 Torque

write1ByteTxRx(port\_num1, PROTOCOL\_VERSION, DXL1\_ID, ADDR\_MX\_TORQUE\_ENABLE, TORQUE\_ENABLE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num1, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num1, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

else

{

printf("Dynamixel#%d has been successfully connected \n", DXL1\_ID);

}

// Enable Dynamixel#2 Torque

write1ByteTxRx(port\_num2, PROTOCOL\_VERSION, DXL2\_ID, ADDR\_MX\_TORQUE\_ENABLE, TORQUE\_ENABLE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num2, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num2, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

else

{

printf("Dynamixel#%d has been successfully connected \n", DXL2\_ID);

}

while (1)

{

printf("Press any key to continue! (or press ESC to quit!)\n");

if (getch() == ESC\_ASCII\_VALUE)

break;

// Write Dynamixel#1 goal position

write2ByteTxRx(port\_num1, PROTOCOL\_VERSION, DXL1\_ID, ADDR\_MX\_GOAL\_POSITION, dxl\_goal\_position[index]);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num1, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num1, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

// Write Dynamixel#2 goal position

write2ByteTxRx(port\_num2, PROTOCOL\_VERSION, DXL2\_ID, ADDR\_MX\_GOAL\_POSITION, dxl\_goal\_position[index]);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num2, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num2, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

do

{

// Read Dynamixel#1 present position

dxl1\_present\_position = read2ByteTxRx(port\_num1, PROTOCOL\_VERSION, DXL1\_ID, ADDR\_MX\_PRESENT\_POSITION);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num1, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num1, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

// Read Dynamixel#2 present position

dxl2\_present\_position = read2ByteTxRx(port\_num2, PROTOCOL\_VERSION, DXL2\_ID, ADDR\_MX\_PRESENT\_POSITION);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num2, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num2, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

printf("[ID:%03d] GoalPos:%03d PresPos:%03d\t[ID:%03d] GoalPos:%03d PresPos:%03d\n", DXL1\_ID, dxl\_goal\_position[index], dxl1\_present\_position, DXL2\_ID, dxl\_goal\_position[index], dxl2\_present\_position);

} while ((abs(dxl\_goal\_position[index] - dxl1\_present\_position) > DXL\_MOVING\_STATUS\_THRESHOLD) || (abs(dxl\_goal\_position[index] - dxl2\_present\_position) > DXL\_MOVING\_STATUS\_THRESHOLD));

// Change goal position

if (index == 0)

{

index = 1;

}

else

{

index = 0;

}

}

// Disable Dynamixel#1 Torque

write1ByteTxRx(port\_num1, PROTOCOL\_VERSION, DXL1\_ID, ADDR\_MX\_TORQUE\_ENABLE, TORQUE\_DISABLE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num1, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num1, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

// Disable Dynamixel#2 Torque

write1ByteTxRx(port\_num2, PROTOCOL\_VERSION, DXL2\_ID, ADDR\_MX\_TORQUE\_ENABLE, TORQUE\_DISABLE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num2, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num2, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

// Close port1

closePort(port\_num1);

// Close port2

closePort(port\_num2);

return 0;

}

In main() function, the codes call actual functions for Dynamixel control.

// Initialize PortHandler Structs

// Set the port path

// Get methods and members of PortHandlerLinux or PortHandlerWindows

int port\_num1 = portHandler(DEVICENAME1);

int port\_num2 = portHandler(DEVICENAME2);

portHandler() function sets port path as DEVICENAME1 and DEVICENAME2 and get port\_num1 and port\_num2 each, and prepares an appropriate functions for port control in controller OS automatically. port\_num1 and port\_num2 would be used in many functions in the body of the code to specify the port for use.

// Initialize PacketHandler Structs

packetHandler();

packetHandler() function initializes parameters used for packet construction and packet storing.

int index = 0;

int dxl\_comm\_result = COMM\_TX\_FAIL; // Communication result

int dxl\_goal\_position[2] = {DXL\_MINIMUM\_POSITION\_VALUE, DXL\_MAXIMUM\_POSITION\_VALUE}; // Goal position

uint8\_t dxl\_error = 0; // Dynamixel error

uint16\_t dxl1\_present\_position = 0, dxl2\_present\_position = 0; // Present position

index variable points the direction to where the Dynamixel should be rotated.

dxl\_comm\_result indicates which error has been occurred during packet communication.

dxl\_goal\_position stores goal points of Dynamixel rotation.

dxl\_error shows the internal error in Dynamixel.

dxl1\_present\_position and dxl2\_present\_position view where now each Dynamixel points out.

// Open port1

if (openPort(port\_num1))

{

printf("Succeeded to open the port1!\n");

}

else

{

printf("Failed to open the port1!\n");

printf("Press any key to terminate...\n");

\_getch();

return 0;

}

// Open port2

if (openPort(port\_num2))

{

printf("Succeeded to open the port2!\n");

}

else

{

printf("Failed to open the port2!\n");

printf("Press any key to terminate...\n");

\_getch();

return 0;

}

First, controller opens #port\_num1 and #port\_num2 port to do serial communication with the Dynamixel. If it fails to open the port, the example will be terminated.

// Set port1 baudrate

if (setBaudRate(port\_num1, BAUDRATE))

{

printf("Succeed to change the baudrate port1!\n");

}

else

{

printf("Failed to change the baudrate port1!\n");

printf("Press any key to terminate...\n");

\_getch();

return 0;

}

// Set port2 baudrate

if (setBaudRate(port\_num2, BAUDRATE))

{

printf("Succeed to change the baudrate port2!\n");

}

else

{

printf("Failed to change the baudrate port2!\n");

printf("Press any key to terminate...\n");

\_getch();

return 0;

}

Secondly, the controller sets the communication BAUDRATE at #port\_num1 and #port\_num2 port opened previously.

// Enable Dynamixel#1 Torque

write1ByteTxRx(port\_num1, PROTOCOL\_VERSION, DXL1\_ID, ADDR\_MX\_TORQUE\_ENABLE, TORQUE\_ENABLE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num1, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num1, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

else

{

printf("Dynamixel#%d has been successfully connected \n", DXL1\_ID);

}

// Enable Dynamixel#2 Torque

write1ByteTxRx(port\_num2, PROTOCOL\_VERSION, DXL2\_ID, ADDR\_MX\_TORQUE\_ENABLE, TORQUE\_ENABLE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num2, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num2, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

else

{

printf("Dynamixel#%d has been successfully connected \n", DXL2\_ID);

}

As mentioned in the document, above code enables each Dynamixel`s torque to set their status as being ready to move.

write1ByteTxRx() function orders to the #DXL1\_ID and #DXL2\_ID Dynamixels in PROTOCOL\_VERSION communication protocol through #port\_num1 and #port\_num2 ports, writing 1 byte of TORQUE\_ENABLE value to ADDR\_MX\_TORQUE\_ENABLE address. The function checks Tx/Rx result and receives Hardware error. getLastTxRxResult() function and getLastRxPacketError() function get either, and then printTxRxResult() function and printRxPacketError() function show results on the console window if any communication error or Hardware error has been occurred.

while (1)

{

printf("Press any key to continue! (or press ESC to quit!)\n");

if (getch() == ESC\_ASCII\_VALUE)

break;

// Write Dynamixel#1 goal position

write2ByteTxRx(port\_num1, PROTOCOL\_VERSION, DXL1\_ID, ADDR\_MX\_GOAL\_POSITION, dxl\_goal\_position[index]);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num1, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num1, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

// Write Dynamixel#2 goal position

write2ByteTxRx(port\_num2, PROTOCOL\_VERSION, DXL2\_ID, ADDR\_MX\_GOAL\_POSITION, dxl\_goal\_position[index]);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num2, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num2, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

do

{

// Read Dynamixel#1 present position

dxl1\_present\_position = read2ByteTxRx(port\_num1, PROTOCOL\_VERSION, DXL1\_ID, ADDR\_MX\_PRESENT\_POSITION);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num1, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num1, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

// Read Dynamixel#2 present position

dxl2\_present\_position = read2ByteTxRx(port\_num2, PROTOCOL\_VERSION, DXL2\_ID, ADDR\_MX\_PRESENT\_POSITION);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num2, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num2, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

printf("[ID:%03d] GoalPos:%03d PresPos:%03d\t[ID:%03d] GoalPos:%03d PresPos:%03d\n", DXL1\_ID, dxl\_goal\_position[index], dxl1\_present\_position, DXL2\_ID, dxl\_goal\_position[index], dxl2\_present\_position);

} while ((abs(dxl\_goal\_position[index] - dxl1\_present\_position) > DXL\_MOVING\_STATUS\_THRESHOLD) || (abs(dxl\_goal\_position[index] - dxl2\_present\_position) > DXL\_MOVING\_STATUS\_THRESHOLD));

// Change goal position

if (index == 0)

{

index = 1;

}

else

{

index = 0;

}

}

During while() loop, the controller writes and reads each Dynamixel position through packet transmission/reception(Tx/Rx).

To continue their rotation, press any key except ESC.

write2ByteTxRx() function orders to the #DXL1\_ID and #DXL2\_ID Dynamixels in PROTOCOL\_VERSION communication protocol through #port\_num1 and #port\_num2 ports, writing 2 byte of dxl\_goal\_position[index] value to ADDR\_MX\_GOAL\_POSITION address. The function checks Tx/Rx result and receives Hardware error. getLastTxRxResult() function and getLastRxPacketError() function get either, and then printTxRxResult() function and printRxPacketError() function show results on the console window if any communication error or Hardware error has been occurred.

read2ByteTxRx() function orders to the #DXL1\_ID and #DXL2\_ID Dynamixels in PROTOCOL\_VERSION communication protocol through #port\_num1 and #port\_num2 ports, requesting 2 bytes of value in ADDR\_MX\_PRESENT\_POSITION address. The function checks Tx/Rx result and receives Hardware error. getLastTxRxResult() function and getLastRxPacketError() function get either, and then printTxRxResult() function and printRxPacketError() function show results on the console window if any communication error or Hardware error has been occurred.

Reading their present position will be ended when absolute value of (dxl1\_goal\_position[index] - dxl1\_present\_position) or (dxl2\_goal\_position[index] - dxl2\_present\_position) becomes smaller then DXL\_MOVING\_STATUS\_THRESHOLD.

At last, it changes their direction to the counter-wise and waits for extra key input.

// Disable Dynamixel#1 Torque

write1ByteTxRx(port\_num1, PROTOCOL\_VERSION, DXL1\_ID, ADDR\_MX\_TORQUE\_ENABLE, TORQUE\_DISABLE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num1, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num1, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

// Disable Dynamixel#2 Torque

write1ByteTxRx(port\_num2, PROTOCOL\_VERSION, DXL2\_ID, ADDR\_MX\_TORQUE\_ENABLE, TORQUE\_DISABLE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num2, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num2, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

The controller frees the Dynamixels to be idle.

write1ByteTxRx() function orders to the #DXL1\_ID and #DXL2\_ID Dynamixels in PROTOCOL\_VERSION communication protocol through #port\_num1 and #port\_num2 ports, writing 1 byte of TORQUE\_DISABLE value to ADDR\_MX\_TORQUE\_ENABLE address. The function checks Tx/Rx result and receives Hardware error. getLastTxRxResult() function and getLastRxPacketError() function get either, and then printTxRxResult() function and printRxPacketError() function show results on the console window if any communication error or Hardware error has been occurred.

// Close port1

closePort(port\_num1);

// Close port2

closePort(port\_num2);

return 0;

Finally, ports become disposed.

[Edit on GitHub](https://github.com/ROBOTIS-GIT/emanual/blob/master/docs/en/software/dynamixel/dynamixel_sdk/sample_code/c/c_multi_port_protocol_1_0.md)

### [C Multi Port Protocol 1.0](http://emanual.robotis.com/docs/en/software/dynamixel/dynamixel_sdk/sample_code/c_multi_port_protocol_1_0/" \l "c-multi-port-protocol-10)

* Description

This example writes Goal Positions to two Dynamixels connected via two ports and reads their Present Positions until they stop moving.

* Supported Dynamixels

Protocol 1.0 Dynamixels

#### Sample code

/\*

\* multi\_port.c

\*

\* Created on: 2016. 5. 16.

\* Author: Leon Ryu Woon Jung

\*/

//

// \*\*\*\*\*\*\*\*\* Multi Port Example \*\*\*\*\*\*\*\*\*

//

//

// Available Dynamixel model on this example : All models using Protocol 1.0

// This example is designed for using two Dynamixel MX-28, and two USB2DYNAMIXEL.

// To use another Dynamixel model, such as X series, see their details in E-Manual(support.robotis.com) and edit below "#define"d variables yourself.

// Be sure that Dynamixel MX properties are already set as %% ID : 1 / Baudnum : 1 (Baudrate : 1000000 [1M])

//

#ifdef \_\_linux\_\_

#include <unistd.h>

#include <fcntl.h>

#include <termios.h>

#elif defined(\_WIN32) || defined(\_WIN64)

#include <conio.h>

#endif

#include <stdlib.h>

#include <stdio.h>

#include "dynamixel\_sdk.h" // Uses Dynamixel SDK library

// Control table address

#define ADDR\_MX\_TORQUE\_ENABLE 24 // Control table address is different in Dynamixel model

#define ADDR\_MX\_GOAL\_POSITION 30

#define ADDR\_MX\_PRESENT\_POSITION 36

// Protocol version

#define PROTOCOL\_VERSION 1.0 // See which protocol version is used in the Dynamixel

// Default setting

#define DXL1\_ID 1 // Dynamixel#1 ID: 1

#define DXL2\_ID 2 // Dynamixel#2 ID: 2

#define BAUDRATE 1000000

#define DEVICENAME1 "/dev/ttyUSB0" // Check which port is being used on your controller

#define DEVICENAME2 "/dev/ttyUSB1" // ex) Windows: "COM1" Linux: "/dev/ttyUSB0"

#define TORQUE\_ENABLE 1 // Value for enabling the torque

#define TORQUE\_DISABLE 0 // Value for disabling the torque

#define DXL\_MINIMUM\_POSITION\_VALUE 100 // Dynamixel will rotate between this value

#define DXL\_MAXIMUM\_POSITION\_VALUE 4000 // and this value (note that the Dynamixel would not move when the position value is out of movable range. Check e-manual about the range of the Dynamixel you use.)

#define DXL\_MOVING\_STATUS\_THRESHOLD 10 // Dynamixel moving status threshold

#define ESC\_ASCII\_VALUE 0x1b

int getch()

{

#ifdef \_\_linux\_\_

struct termios oldt, newt;

int ch;

tcgetattr(STDIN\_FILENO, &oldt);

newt = oldt;

newt.c\_lflag &= ~(ICANON | ECHO);

tcsetattr(STDIN\_FILENO, TCSANOW, &newt);

ch = getchar();

tcsetattr(STDIN\_FILENO, TCSANOW, &oldt);

return ch;

#elif defined(\_WIN32) || defined(\_WIN64)

return \_getch();

#endif

}

int kbhit(void)

{

#ifdef \_\_linux\_\_

struct termios oldt, newt;

int ch;

int oldf;

tcgetattr(STDIN\_FILENO, &oldt);

newt = oldt;

newt.c\_lflag &= ~(ICANON | ECHO);

tcsetattr(STDIN\_FILENO, TCSANOW, &newt);

oldf = fcntl(STDIN\_FILENO, F\_GETFL, 0);

fcntl(STDIN\_FILENO, F\_SETFL, oldf | O\_NONBLOCK);

ch = getchar();

tcsetattr(STDIN\_FILENO, TCSANOW, &oldt);

fcntl(STDIN\_FILENO, F\_SETFL, oldf);

if (ch != EOF)

{

ungetc(ch, stdin);

return 1;

}

return 0;

#elif defined(\_WIN32) || defined(\_WIN64)

return \_kbhit();

#endif

}

int main()

{

// Initialize PortHandler Structs

// Set the port path

// Get methods and members of PortHandlerLinux or PortHandlerWindows

int port\_num1 = portHandler(DEVICENAME1);

int port\_num2 = portHandler(DEVICENAME2);

// Initialize PacketHandler Structs

packetHandler();

int index = 0;

int dxl\_comm\_result = COMM\_TX\_FAIL; // Communication result

int dxl\_goal\_position[2] = { DXL\_MINIMUM\_POSITION\_VALUE, DXL\_MAXIMUM\_POSITION\_VALUE }; // Goal position

uint8\_t dxl\_error = 0; // Dynamixel error

uint16\_t dxl1\_present\_position = 0, dxl2\_present\_position = 0; // Present position

// Open port1

if (openPort(port\_num1))

{

printf("Succeeded to open the port1!\n");

}

else

{

printf("Failed to open the port1!\n");

printf("Press any key to terminate...\n");

getch();

return 0;

}

// Open port2

if (openPort(port\_num2))

{

printf("Succeeded to open the port2!\n");

}

else

{

printf("Failed to open the port2!\n");

printf("Press any key to terminate...\n");

getch();

return 0;

}

// Set port1 baudrate

if (setBaudRate(port\_num1, BAUDRATE))

{

printf("Succeed to change the baudrate port1!\n");

}

else

{

printf("Failed to change the baudrate port1!\n");

printf("Press any key to terminate...\n");

getch();

return 0;

}

// Set port2 baudrate

if (setBaudRate(port\_num2, BAUDRATE))

{

printf("Succeed to change the baudrate port2!\n");

}

else

{

printf("Failed to change the baudrate port2!\n");

printf("Press any key to terminate...\n");

getch();

return 0;

}

// Enable Dynamixel#1 Torque

write1ByteTxRx(port\_num1, PROTOCOL\_VERSION, DXL1\_ID, ADDR\_MX\_TORQUE\_ENABLE, TORQUE\_ENABLE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num1, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num1, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

else

{

printf("Dynamixel#%d has been successfully connected \n", DXL1\_ID);

}

// Enable Dynamixel#2 Torque

write1ByteTxRx(port\_num2, PROTOCOL\_VERSION, DXL2\_ID, ADDR\_MX\_TORQUE\_ENABLE, TORQUE\_ENABLE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num2, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num2, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

else

{

printf("Dynamixel#%d has been successfully connected \n", DXL2\_ID);

}

while (1)

{

printf("Press any key to continue! (or press ESC to quit!)\n");

if (getch() == ESC\_ASCII\_VALUE)

break;

// Write Dynamixel#1 goal position

write2ByteTxRx(port\_num1, PROTOCOL\_VERSION, DXL1\_ID, ADDR\_MX\_GOAL\_POSITION, dxl\_goal\_position[index]);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num1, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num1, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

// Write Dynamixel#2 goal position

write2ByteTxRx(port\_num2, PROTOCOL\_VERSION, DXL2\_ID, ADDR\_MX\_GOAL\_POSITION, dxl\_goal\_position[index]);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num2, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num2, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

do

{

// Read Dynamixel#1 present position

dxl1\_present\_position = read2ByteTxRx(port\_num1, PROTOCOL\_VERSION, DXL1\_ID, ADDR\_MX\_PRESENT\_POSITION);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num1, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num1, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

// Read Dynamixel#2 present position

dxl2\_present\_position = read2ByteTxRx(port\_num2, PROTOCOL\_VERSION, DXL2\_ID, ADDR\_MX\_PRESENT\_POSITION);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num2, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num2, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

printf("[ID:%03d] GoalPos:%03d PresPos:%03d\t[ID:%03d] GoalPos:%03d PresPos:%03d\n", DXL1\_ID, dxl\_goal\_position[index], dxl1\_present\_position, DXL2\_ID, dxl\_goal\_position[index], dxl2\_present\_position);

} while ((abs(dxl\_goal\_position[index] - dxl1\_present\_position) > DXL\_MOVING\_STATUS\_THRESHOLD) || (abs(dxl\_goal\_position[index] - dxl2\_present\_position) > DXL\_MOVING\_STATUS\_THRESHOLD));

// Change goal position

if (index == 0)

{

index = 1;

}

else

{

index = 0;

}

}

// Disable Dynamixel#1 Torque

write1ByteTxRx(port\_num1, PROTOCOL\_VERSION, DXL1\_ID, ADDR\_MX\_TORQUE\_ENABLE, TORQUE\_DISABLE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num1, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num1, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

// Disable Dynamixel#2 Torque

write1ByteTxRx(port\_num2, PROTOCOL\_VERSION, DXL2\_ID, ADDR\_MX\_TORQUE\_ENABLE, TORQUE\_DISABLE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num2, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num2, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

// Close port1

closePort(port\_num1);

// Close port2

closePort(port\_num2);

return 0;

}

#### Details

#ifdef \_\_linux\_\_

#include <unistd.h>

#include <fcntl.h>

#include <termios.h>

#elif defined(\_WIN32) || defined(\_WIN64)

#include <conio.h>

#endif

This source includes above to get key input interruption while the example is running. Actual functions for getting the input is described in a little below.

#include <stdlib.h>

The function abs() is in the example code, and it needs stdlib.h to be included.

#include <stdio.h>

The example shows Dynamixel status in sequence by the function printf(). So here stdio.h is needed.

#include "dynamixel\_sdk.h" // Uses Dynamixel SDK library

All libraries of Dynamixel SDK are linked with the header file dynamixel\_sdk.h.

// Control table address

#define ADDR\_MX\_TORQUE\_ENABLE 24 // Control table address is different in Dynamixel model

#define ADDR\_MX\_GOAL\_POSITION 30

#define ADDR\_MX\_PRESENT\_POSITION 36

Dynamixel series have their own control tables: Addresses and Byte Length in each items. To control one of the items, its address (and length if necessary) is required. Find your requirements in http://emanual.robotis.com/.

// Protocol version

#define PROTOCOL\_VERSION 1.0 // See which protocol version is used in the Dynamixel

Dynamixel uses either or both protocols: Protocol 1.0 and Protocol 2.0. Choose one of the Protocol which is appropriate in the Dynamixel.

// Default setting

#define DXL1\_ID 1 // Dynamixel#1 ID: 1

#define DXL2\_ID 2 // Dynamixel#2 ID: 2

#define BAUDRATE 1000000

#define DEVICENAME1 "/dev/ttyUSB0" // Check which port is being used on your controller

#define DEVICENAME2 "/dev/ttyUSB1" // ex) Windows: "COM1" Linux: "/dev/ttyUSB0"

#define TORQUE\_ENABLE 1 // Value for enabling the torque

#define TORQUE\_DISABLE 0 // Value for disabling the torque

#define DXL\_MINIMUM\_POSITION\_VALUE 100 // Dynamixel will rotate between this value

#define DXL\_MAXIMUM\_POSITION\_VALUE 4000 // and this value (note that the Dynamixel would not move when the position value is out of movable range. Check e-manual about the range of the Dynamixel you use.)

#define DXL\_MOVING\_STATUS\_THRESHOLD 10 // Dynamixel moving status threshold

#define ESC\_ASCII\_VALUE 0x1b

Here we set some variables to let you freely change them and use them to run the example code.

As the document previously said in [previous chapter](http://emanual.robotis.com/docs/en/software/dynamixel/dynamixel_sdk/device_setup/" \l "dynamixel), customize Dynamixel control table items, such as DXL\_ID number, communication BAUDRATE, and the DEVICENAME, on your own terms of needs. In particular, BAUDRATE and DEVICENAME have systematical dependencies on your controller, so make clear what kind of communication method you will use.

The example uses two Dynamixels DXL1\_ID, DXL2\_ID connected with each ports DEVICENAME1, DEVICENAME2

Dynamixel basically needs the TORQUE\_ENABLE to be rotating or give you its internal information. On the other hand, it doesn’t need torque enabled if you get your goal, so finally do TORQUE\_DISABLE to prepare to the next sequence.

Since the Dynamixel has its own rotation range, it may shows malfunction if your request on your dynamixel is out of range. For example, Dynamixel MX-28 and Dynamixel PRO 54-200 has its rotatable range as 0 ~ 4028 and -250950 ~ 250950, each.

DXL\_MOVING\_STATUS\_THRESHOLD acts as a criteria for verifying its rotation stopped.

int getch()

{

#ifdef \_\_linux\_\_

struct termios oldt, newt;

int ch;

tcgetattr(STDIN\_FILENO, &oldt);

newt = oldt;

newt.c\_lflag &= ~(ICANON | ECHO);

tcsetattr(STDIN\_FILENO, TCSANOW, &newt);

ch = getchar();

tcsetattr(STDIN\_FILENO, TCSANOW, &oldt);

return ch;

#elif defined(\_WIN32) || defined(\_WIN64)

return \_getch();

#endif

}

int kbhit(void)

{

#ifdef \_\_linux\_\_

struct termios oldt, newt;

int ch;

int oldf;

tcgetattr(STDIN\_FILENO, &oldt);

newt = oldt;

newt.c\_lflag &= ~(ICANON | ECHO);

tcsetattr(STDIN\_FILENO, TCSANOW, &newt);

oldf = fcntl(STDIN\_FILENO, F\_GETFL, 0);

fcntl(STDIN\_FILENO, F\_SETFL, oldf | O\_NONBLOCK);

ch = getchar();

tcsetattr(STDIN\_FILENO, TCSANOW, &oldt);

fcntl(STDIN\_FILENO, F\_SETFL, oldf);

if (ch != EOF)

{

ungetc(ch, stdin);

return 1;

}

return 0;

#elif defined(\_WIN32) || defined(\_WIN64)

return \_kbhit();

#endif

}

These functions accept the key inputs in terms of example action. The example codes mainly apply the function getch() rather than the function kbhit() to get information which key has been pressed.

int main()

{

// Initialize PortHandler Structs

// Set the port path

// Get methods and members of PortHandlerLinux or PortHandlerWindows

int port\_num1 = portHandler(DEVICENAME1);

int port\_num2 = portHandler(DEVICENAME2);

// Initialize PacketHandler Structs

packetHandler();

int index = 0;

int dxl\_comm\_result = COMM\_TX\_FAIL; // Communication result

int dxl\_goal\_position[2] = { DXL\_MINIMUM\_POSITION\_VALUE, DXL\_MAXIMUM\_POSITION\_VALUE }; // Goal position

uint8\_t dxl\_error = 0; // Dynamixel error

uint16\_t dxl1\_present\_position = 0, dxl2\_present\_position = 0; // Present position

// Open port1

if (openPort(port\_num1))

{

printf("Succeeded to open the port1!\n");

}

else

{

printf("Failed to open the port1!\n");

printf("Press any key to terminate...\n");

getch();

return 0;

}

// Open port2

if (openPort(port\_num2))

{

printf("Succeeded to open the port2!\n");

}

else

{

printf("Failed to open the port2!\n");

printf("Press any key to terminate...\n");

getch();

return 0;

}

// Set port1 baudrate

if (setBaudRate(port\_num1, BAUDRATE))

{

printf("Succeed to change the baudrate port1!\n");

}

else

{

printf("Failed to change the baudrate port1!\n");

printf("Press any key to terminate...\n");

getch();

return 0;

}

// Set port2 baudrate

if (setBaudRate(port\_num2, BAUDRATE))

{

printf("Succeed to change the baudrate port2!\n");

}

else

{

printf("Failed to change the baudrate port2!\n");

printf("Press any key to terminate...\n");

getch();

return 0;

}

// Enable Dynamixel#1 Torque

write1ByteTxRx(port\_num1, PROTOCOL\_VERSION, DXL1\_ID, ADDR\_MX\_TORQUE\_ENABLE, TORQUE\_ENABLE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num1, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num1, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

else

{

printf("Dynamixel#%d has been successfully connected \n", DXL1\_ID);

}

// Enable Dynamixel#2 Torque

write1ByteTxRx(port\_num2, PROTOCOL\_VERSION, DXL2\_ID, ADDR\_MX\_TORQUE\_ENABLE, TORQUE\_ENABLE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num2, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num2, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

else

{

printf("Dynamixel#%d has been successfully connected \n", DXL2\_ID);

}

while (1)

{

printf("Press any key to continue! (or press ESC to quit!)\n");

if (getch() == ESC\_ASCII\_VALUE)

break;

// Write Dynamixel#1 goal position

write2ByteTxRx(port\_num1, PROTOCOL\_VERSION, DXL1\_ID, ADDR\_MX\_GOAL\_POSITION, dxl\_goal\_position[index]);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num1, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num1, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

// Write Dynamixel#2 goal position

write2ByteTxRx(port\_num2, PROTOCOL\_VERSION, DXL2\_ID, ADDR\_MX\_GOAL\_POSITION, dxl\_goal\_position[index]);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num2, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num2, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

do

{

// Read Dynamixel#1 present position

dxl1\_present\_position = read2ByteTxRx(port\_num1, PROTOCOL\_VERSION, DXL1\_ID, ADDR\_MX\_PRESENT\_POSITION);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num1, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num1, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

// Read Dynamixel#2 present position

dxl2\_present\_position = read2ByteTxRx(port\_num2, PROTOCOL\_VERSION, DXL2\_ID, ADDR\_MX\_PRESENT\_POSITION);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num2, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num2, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

printf("[ID:%03d] GoalPos:%03d PresPos:%03d\t[ID:%03d] GoalPos:%03d PresPos:%03d\n", DXL1\_ID, dxl\_goal\_position[index], dxl1\_present\_position, DXL2\_ID, dxl\_goal\_position[index], dxl2\_present\_position);

} while ((abs(dxl\_goal\_position[index] - dxl1\_present\_position) > DXL\_MOVING\_STATUS\_THRESHOLD) || (abs(dxl\_goal\_position[index] - dxl2\_present\_position) > DXL\_MOVING\_STATUS\_THRESHOLD));

// Change goal position

if (index == 0)

{

index = 1;

}

else

{

index = 0;

}

}

// Disable Dynamixel#1 Torque

write1ByteTxRx(port\_num1, PROTOCOL\_VERSION, DXL1\_ID, ADDR\_MX\_TORQUE\_ENABLE, TORQUE\_DISABLE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num1, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num1, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

// Disable Dynamixel#2 Torque

write1ByteTxRx(port\_num2, PROTOCOL\_VERSION, DXL2\_ID, ADDR\_MX\_TORQUE\_ENABLE, TORQUE\_DISABLE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num2, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num2, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

// Close port1

closePort(port\_num1);

// Close port2

closePort(port\_num2);

return 0;

}

In main() function, the codes call actual functions for Dynamixel control.

// Initialize PortHandler Structs

// Set the port path

// Get methods and members of PortHandlerLinux or PortHandlerWindows

int port\_num1 = portHandler(DEVICENAME1);

int port\_num2 = portHandler(DEVICENAME2);

portHandler() function sets port path as DEVICENAME1 and DEVICENAME2 and get port\_num1 and port\_num2 each, and prepares an appropriate functions for port control in controller OS automatically. port\_num1 and port\_num2 would be used in many functions in the body of the code to specify the port for use.

// Initialize PacketHandler Structs

packetHandler();

packetHandler() function initializes parameters used for packet construction and packet storing.

int index = 0;

int dxl\_comm\_result = COMM\_TX\_FAIL; // Communication result

int dxl\_goal\_position[2] = {DXL\_MINIMUM\_POSITION\_VALUE, DXL\_MAXIMUM\_POSITION\_VALUE}; // Goal position

uint8\_t dxl\_error = 0; // Dynamixel error

uint16\_t dxl1\_present\_position = 0, dxl2\_present\_position = 0; // Present position

index variable points the direction to where the Dynamixel should be rotated.

dxl\_comm\_result indicates which error has been occurred during packet communication.

dxl\_goal\_position stores goal points of Dynamixel rotation.

dxl\_error shows the internal error in Dynamixel.

dxl1\_present\_position and dxl2\_present\_position view where now each Dynamixel points out.

// Open port1

if (openPort(port\_num1))

{

printf("Succeeded to open the port1!\n");

}

else

{

printf("Failed to open the port1!\n");

printf("Press any key to terminate...\n");

\_getch();

return 0;

}

// Open port2

if (openPort(port\_num2))

{

printf("Succeeded to open the port2!\n");

}

else

{

printf("Failed to open the port2!\n");

printf("Press any key to terminate...\n");

\_getch();

return 0;

}

First, controller opens #port\_num1 and #port\_num2 port to do serial communication with the Dynamixel. If it fails to open the port, the example will be terminated.

// Set port1 baudrate

if (setBaudRate(port\_num1, BAUDRATE))

{

printf("Succeed to change the baudrate port1!\n");

}

else

{

printf("Failed to change the baudrate port1!\n");

printf("Press any key to terminate...\n");

\_getch();

return 0;

}

// Set port2 baudrate

if (setBaudRate(port\_num2, BAUDRATE))

{

printf("Succeed to change the baudrate port2!\n");

}

else

{

printf("Failed to change the baudrate port2!\n");

printf("Press any key to terminate...\n");

\_getch();

return 0;

}

Secondly, the controller sets the communication BAUDRATE at #port\_num1 and #port\_num2 port opened previously.

// Enable Dynamixel#1 Torque

write1ByteTxRx(port\_num1, PROTOCOL\_VERSION, DXL1\_ID, ADDR\_MX\_TORQUE\_ENABLE, TORQUE\_ENABLE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num1, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num1, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

else

{

printf("Dynamixel#%d has been successfully connected \n", DXL1\_ID);

}

// Enable Dynamixel#2 Torque

write1ByteTxRx(port\_num2, PROTOCOL\_VERSION, DXL2\_ID, ADDR\_MX\_TORQUE\_ENABLE, TORQUE\_ENABLE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num2, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num2, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

else

{

printf("Dynamixel#%d has been successfully connected \n", DXL2\_ID);

}

As mentioned in the document, above code enables each Dynamixel`s torque to set their status as being ready to move.

write1ByteTxRx() function orders to the #DXL1\_ID and #DXL2\_ID Dynamixels in PROTOCOL\_VERSION communication protocol through #port\_num1 and #port\_num2 ports, writing 1 byte of TORQUE\_ENABLE value to ADDR\_MX\_TORQUE\_ENABLE address. The function checks Tx/Rx result and receives Hardware error. getLastTxRxResult() function and getLastRxPacketError() function get either, and then printTxRxResult() function and printRxPacketError() function show results on the console window if any communication error or Hardware error has been occurred.

while (1)

{

printf("Press any key to continue! (or press ESC to quit!)\n");

if (getch() == ESC\_ASCII\_VALUE)

break;

// Write Dynamixel#1 goal position

write2ByteTxRx(port\_num1, PROTOCOL\_VERSION, DXL1\_ID, ADDR\_MX\_GOAL\_POSITION, dxl\_goal\_position[index]);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num1, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num1, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

// Write Dynamixel#2 goal position

write2ByteTxRx(port\_num2, PROTOCOL\_VERSION, DXL2\_ID, ADDR\_MX\_GOAL\_POSITION, dxl\_goal\_position[index]);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num2, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num2, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

do

{

// Read Dynamixel#1 present position

dxl1\_present\_position = read2ByteTxRx(port\_num1, PROTOCOL\_VERSION, DXL1\_ID, ADDR\_MX\_PRESENT\_POSITION);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num1, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num1, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

// Read Dynamixel#2 present position

dxl2\_present\_position = read2ByteTxRx(port\_num2, PROTOCOL\_VERSION, DXL2\_ID, ADDR\_MX\_PRESENT\_POSITION);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num2, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num2, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

printf("[ID:%03d] GoalPos:%03d PresPos:%03d\t[ID:%03d] GoalPos:%03d PresPos:%03d\n", DXL1\_ID, dxl\_goal\_position[index], dxl1\_present\_position, DXL2\_ID, dxl\_goal\_position[index], dxl2\_present\_position);

} while ((abs(dxl\_goal\_position[index] - dxl1\_present\_position) > DXL\_MOVING\_STATUS\_THRESHOLD) || (abs(dxl\_goal\_position[index] - dxl2\_present\_position) > DXL\_MOVING\_STATUS\_THRESHOLD));

// Change goal position

if (index == 0)

{

index = 1;

}

else

{

index = 0;

}

}

During while() loop, the controller writes and reads each Dynamixel position through packet transmission/reception(Tx/Rx).

To continue their rotation, press any key except ESC.

write2ByteTxRx() function orders to the #DXL1\_ID and #DXL2\_ID Dynamixels in PROTOCOL\_VERSION communication protocol through #port\_num1 and #port\_num2 ports, writing 2 byte of dxl\_goal\_position[index] value to ADDR\_MX\_GOAL\_POSITION address. The function checks Tx/Rx result and receives Hardware error. getLastTxRxResult() function and getLastRxPacketError() function get either, and then printTxRxResult() function and printRxPacketError() function show results on the console window if any communication error or Hardware error has been occurred.

read2ByteTxRx() function orders to the #DXL1\_ID and #DXL2\_ID Dynamixels in PROTOCOL\_VERSION communication protocol through #port\_num1 and #port\_num2 ports, requesting 2 bytes of value in ADDR\_MX\_PRESENT\_POSITION address. The function checks Tx/Rx result and receives Hardware error. getLastTxRxResult() function and getLastRxPacketError() function get either, and then printTxRxResult() function and printRxPacketError() function show results on the console window if any communication error or Hardware error has been occurred.

Reading their present position will be ended when absolute value of (dxl1\_goal\_position[index] - dxl1\_present\_position) or (dxl2\_goal\_position[index] - dxl2\_present\_position) becomes smaller then DXL\_MOVING\_STATUS\_THRESHOLD.

At last, it changes their direction to the counter-wise and waits for extra key input.

// Disable Dynamixel#1 Torque

write1ByteTxRx(port\_num1, PROTOCOL\_VERSION, DXL1\_ID, ADDR\_MX\_TORQUE\_ENABLE, TORQUE\_DISABLE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num1, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num1, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

// Disable Dynamixel#2 Torque

write1ByteTxRx(port\_num2, PROTOCOL\_VERSION, DXL2\_ID, ADDR\_MX\_TORQUE\_ENABLE, TORQUE\_DISABLE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num2, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num2, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

The controller frees the Dynamixels to be idle.

write1ByteTxRx() function orders to the #DXL1\_ID and #DXL2\_ID Dynamixels in PROTOCOL\_VERSION communication protocol through #port\_num1 and #port\_num2 ports, writing 1 byte of TORQUE\_DISABLE value to ADDR\_MX\_TORQUE\_ENABLE address. The function checks Tx/Rx result and receives Hardware error. getLastTxRxResult() function and getLastRxPacketError() function get either, and then printTxRxResult() function and printRxPacketError() function show results on the console window if any communication error or Hardware error has been occurred.

// Close port1

closePort(port\_num1);

// Close port2

closePort(port\_num2);

return 0;

Finally, ports become disposed.

© 2019 ROBOTIS. Powered by [Jekyll](http://jekyllrb.com/) & [Minimal Mistakes](https://mademistakes.com/work/minimal-mistakes-jekyll-theme/).

[Edit on GitHub](https://github.com/ROBOTIS-GIT/emanual/blob/master/docs/en/software/dynamixel/dynamixel_sdk/sample_code/c/c_bulk_read_protocol_1_0.md)

### [C Bulk Read Protocol 1.0](http://emanual.robotis.com/docs/en/software/dynamixel/dynamixel_sdk/sample_code/c_bulk_read_protocol_1_0/" \l "c-bulk-read-protocol-10)

* 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.

* Supported Dynamixels

Protocol 1.0 Dynamixels

#### Sample code

/\*

\* bulk\_read.c

\*

\* Created on: 2016. 5. 16.

\* Author: Leon Ryu Woon Jung

\*/

//

// \*\*\*\*\*\*\*\*\* Bulk Read Example \*\*\*\*\*\*\*\*\*

//

//

// Available Dynamixel model on this example : MX or X series set to Protocol 1.0

// This example is designed for using two Dynamixel MX-28, and an USB2DYNAMIXEL.

// To use another Dynamixel model, such as X series, see their details in E-Manual(support.robotis.com) and edit below "#define"d variables yourself.

// Be sure that Dynamixel MX properties are already set as %% ID : 1 / Baudnum : 1 (Baudrate : 1000000)

//

#ifdef \_\_linux\_\_

#include <unistd.h>

#include <fcntl.h>

#include <termios.h>

#elif defined(\_WIN32) || defined(\_WIN64)

#include <conio.h>

#endif

#include <stdlib.h>

#include <stdio.h>

#include "dynamixel\_sdk.h" // Uses Dynamixel SDK library

// Control table address

#define ADDR\_MX\_TORQUE\_ENABLE 24 // Control table address is different in Dynamixel model

#define ADDR\_MX\_GOAL\_POSITION 30

#define ADDR\_MX\_PRESENT\_POSITION 36

#define ADDR\_MX\_MOVING 46

// Data Byte Length

#define LEN\_MX\_GOAL\_POSITION 2

#define LEN\_MX\_PRESENT\_POSITION 2

#define LEN\_MX\_MOVING 1

// Protocol version

#define PROTOCOL\_VERSION 1.0 // See which protocol version is used in the Dynamixel

// Default setting

#define DXL1\_ID 1 // Dynamixel#1 ID: 1

#define DXL2\_ID 2 // Dynamixel#2 ID: 2

#define BAUDRATE 1000000

#define DEVICENAME "/dev/ttyUSB0" // Check which port is being used on your controller

// ex) Windows: "COM1" Linux: "/dev/ttyUSB0"

#define TORQUE\_ENABLE 1 // Value for enabling the torque

#define TORQUE\_DISABLE 0 // Value for disabling the torque

#define DXL\_MINIMUM\_POSITION\_VALUE 100 // Dynamixel will rotate between this value

#define DXL\_MAXIMUM\_POSITION\_VALUE 4000 // and this value (note that the Dynamixel would not move when the position value is out of movable range. Check e-manual about the range of the Dynamixel you use.)

#define DXL\_MOVING\_STATUS\_THRESHOLD 10 // Dynamixel moving status threshold

#define ESC\_ASCII\_VALUE 0x1b

int getch()

{

#ifdef \_\_linux\_\_

struct termios oldt, newt;

int ch;

tcgetattr(STDIN\_FILENO, &oldt);

newt = oldt;

newt.c\_lflag &= ~(ICANON | ECHO);

tcsetattr(STDIN\_FILENO, TCSANOW, &newt);

ch = getchar();

tcsetattr(STDIN\_FILENO, TCSANOW, &oldt);

return ch;

#elif defined(\_WIN32) || defined(\_WIN64)

return \_getch();

#endif

}

int kbhit(void)

{

#ifdef \_\_linux\_\_

struct termios oldt, newt;

int ch;

int oldf;

tcgetattr(STDIN\_FILENO, &oldt);

newt = oldt;

newt.c\_lflag &= ~(ICANON | ECHO);

tcsetattr(STDIN\_FILENO, TCSANOW, &newt);

oldf = fcntl(STDIN\_FILENO, F\_GETFL, 0);

fcntl(STDIN\_FILENO, F\_SETFL, oldf | O\_NONBLOCK);

ch = getchar();

tcsetattr(STDIN\_FILENO, TCSANOW, &oldt);

fcntl(STDIN\_FILENO, F\_SETFL, oldf);

if (ch != EOF)

{

ungetc(ch, stdin);

return 1;

}

return 0;

#elif defined(\_WIN32) || defined(\_WIN64)

return \_kbhit();

#endif

}

int main()

{

// Initialize PortHandler Structs

// Set the port path

// Get methods and members of PortHandlerLinux or PortHandlerWindows

int port\_num = portHandler(DEVICENAME);

// Initialize PacketHandler Structs

packetHandler();

// Initialize Groupbulkread Structs

int group\_num = groupBulkRead(port\_num, PROTOCOL\_VERSION);

int index = 0;

int dxl\_comm\_result = COMM\_TX\_FAIL; // Communication result

uint8\_t dxl\_addparam\_result = False; // AddParam result

uint8\_t dxl\_getdata\_result = False; // GetParam result

int dxl\_goal\_position[2] = { DXL\_MINIMUM\_POSITION\_VALUE, DXL\_MAXIMUM\_POSITION\_VALUE }; // Goal position

uint8\_t dxl\_error = 0; // Dynamixel error

uint16\_t dxl1\_present\_position = 0; // Present position

uint8\_t dxl2\_moving = 0; // Dynamixel moving status

// Open port

if (openPort(port\_num))

{

printf("Succeeded to open the port!\n");

}

else

{

printf("Failed to open the port!\n");

printf("Press any key to terminate...\n");

getch();

return 0;

}

// Set port baudrate

if (setBaudRate(port\_num, BAUDRATE))

{

printf("Succeeded to change the baudrate!\n");

}

else

{

printf("Failed to change the baudrate!\n");

printf("Press any key to terminate...\n");

getch();

return 0;

}

// Enable Dynamixel#1 Torque

write1ByteTxRx(port\_num, PROTOCOL\_VERSION, DXL1\_ID, ADDR\_MX\_TORQUE\_ENABLE, TORQUE\_ENABLE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

else

{

printf("Dynamixel#%d has been successfully connected \n", DXL1\_ID);

}

// Enable Dynamixel#2 Torque

write1ByteTxRx(port\_num, PROTOCOL\_VERSION, DXL2\_ID, ADDR\_MX\_TORQUE\_ENABLE, TORQUE\_ENABLE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

else

{

printf("Dynamixel#%d has been successfully connected \n", DXL2\_ID);

}

// Add parameter storage for Dynamixel#1 present position value

dxl\_addparam\_result = groupBulkReadAddParam(group\_num, DXL1\_ID, ADDR\_MX\_PRESENT\_POSITION, LEN\_MX\_PRESENT\_POSITION);

if (dxl\_addparam\_result != True)

{

fprintf(stderr, "[ID:%03d] groupBulkRead addparam failed", DXL1\_ID);

return 0;

}

// Add parameter storage for Dynamixel#2 present moving value

dxl\_addparam\_result = groupBulkReadAddParam(group\_num, DXL2\_ID, ADDR\_MX\_MOVING, LEN\_MX\_MOVING);

if (dxl\_addparam\_result != True)

{

fprintf(stderr, "[ID:%03d] groupBulkRead addparam failed", DXL2\_ID);

return 0;

}

while (1)

{

printf("Press any key to continue! (or press ESC to quit!)\n");

if (getch() == ESC\_ASCII\_VALUE)

break;

// Write Dynamixel#1 goal position

write2ByteTxRx(port\_num, PROTOCOL\_VERSION, DXL1\_ID, ADDR\_MX\_GOAL\_POSITION, dxl\_goal\_position[index]);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

// Write Dynamixel#2 goal position

write2ByteTxRx(port\_num, PROTOCOL\_VERSION, DXL2\_ID, ADDR\_MX\_GOAL\_POSITION, dxl\_goal\_position[index]);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

do

{

// Bulkread present position and moving status

groupBulkReadTxRxPacket(group\_num);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

dxl\_getdata\_result = groupBulkReadIsAvailable(group\_num, DXL1\_ID, ADDR\_MX\_PRESENT\_POSITION, LEN\_MX\_PRESENT\_POSITION);

if (dxl\_getdata\_result != True)

{

fprintf(stderr, "[ID:%03d] groupBulkRead getdata failed", DXL1\_ID);

return 0;

}

dxl\_getdata\_result = groupBulkReadIsAvailable(group\_num, DXL2\_ID, ADDR\_MX\_MOVING, LEN\_MX\_MOVING);

if (dxl\_getdata\_result != True)

{

fprintf(stderr, "[ID:%03d] groupBulkRead getdata failed", DXL2\_ID);

return 0;

}

// Get Dynamixel#1 present position value

dxl1\_present\_position = groupBulkReadGetData(group\_num, DXL1\_ID, ADDR\_MX\_PRESENT\_POSITION, LEN\_MX\_PRESENT\_POSITION);

// Get Dynamixel#2 moving status value

dxl2\_moving = groupBulkReadGetData(group\_num, DXL2\_ID, ADDR\_MX\_MOVING, LEN\_MX\_MOVING);

printf("[ID:%03d] Present Position : %d \t [ID:%03d] Is Moving : %d\n", DXL1\_ID, dxl1\_present\_position, DXL2\_ID, dxl2\_moving);

} while (abs(dxl\_goal\_position[index] - dxl1\_present\_position) > DXL\_MOVING\_STATUS\_THRESHOLD);

// Change goal position

if (index == 0)

{

index = 1;

}

else

{

index = 0;

}

}

// Disable Dynamixel#1 Torque

write1ByteTxRx(port\_num, PROTOCOL\_VERSION, DXL1\_ID, ADDR\_MX\_TORQUE\_ENABLE, TORQUE\_DISABLE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

// Disable Dynamixel#2 Torque

write1ByteTxRx(port\_num, PROTOCOL\_VERSION, DXL2\_ID, ADDR\_MX\_TORQUE\_ENABLE, TORQUE\_DISABLE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

// Close port

closePort(port\_num);

return 0;

}

#### Details

#ifdef \_\_linux\_\_

#include <unistd.h>

#include <fcntl.h>

#include <termios.h>

#elif defined(\_WIN32) || defined(\_WIN64)

#include <conio.h>

#endif

This source includes above to get key input interruption while the example is running. Actual functions for getting the input is described in a little below.

#include <stdlib.h>

The function abs() is in the example code, and it needs stdlib.h to be included.

#include <stdio.h>

The example shows Dynamixel status in sequence by the function printf(). So here stdio.h is needed.

#include "dynamixel\_sdk.h" // Uses Dynamixel SDK library

All libraries of Dynamixel SDK are linked with the header file dynamixel\_sdk.h.

// Control table address

#define ADDR\_MX\_TORQUE\_ENABLE 24 // Control table address is different in Dynamixel model

#define ADDR\_MX\_GOAL\_POSITION 30

#define ADDR\_MX\_PRESENT\_POSITION 36

// Data Byte Length

#define LEN\_MX\_GOAL\_POSITION 2

#define LEN\_MX\_PRESENT\_POSITION 2

Dynamixel series have their own control tables: Addresses and Byte Length in each items. To control one of the items, its address (and length if necessary) is required. Find your requirements in <http://emanual.robotis.com/>.

// Protocol version

#define PROTOCOL\_VERSION 1.0 // See which protocol version is used in the Dynamixel

Dynamixel uses either or both protocols: Protocol 1.0 and Protocol 2.0. Choose one of the Protocol which is appropriate in the Dynamixel.

// Default setting

#define DXL1\_ID 1 // Dynamixel#1 ID: 1

#define DXL2\_ID 2 // Dynamixel#2 ID: 2

#define BAUDRATE 1000000

#define DEVICENAME1 "/dev/ttyUSB0" // Check which port is being used on your controller

#define TORQUE\_ENABLE 1 // Value for enabling the torque

#define TORQUE\_DISABLE 0 // Value for disabling the torque

#define DXL\_MINIMUM\_POSITION\_VALUE 100 // Dynamixel will rotate between this value

#define DXL\_MAXIMUM\_POSITION\_VALUE 4000 // and this value (note that the Dynamixel would not move when the position value is out of movable range. Check e-manual about the range of the Dynamixel you use.)

#define DXL\_MOVING\_STATUS\_THRESHOLD 10 // Dynamixel moving status threshold

#define ESC\_ASCII\_VALUE 0x1b

Here we set some variables to let you freely change them and use them to run the example code.

As the document previously said in [previous chapter](http://emanual.robotis.com/docs/en/software/dynamixel/dynamixel_sdk/device_setup/" \l "dynamixel), customize Dynamixel control table items, such as DXL\_ID number, communication BAUDRATE, and the DEVICENAME, on your own terms of needs. In particular, BAUDRATE and DEVICENAME have systematical dependencies on your controller, so make clear what kind of communication method you will use.

The example uses two Dynamixels DXL1\_ID, DXL2\_ID connected with the port DEVICENAME.

Dynamixel basically needs the TORQUE\_ENABLE to be rotating or give you its internal information. On the other hand, it doesn’t need torque enabled if you get your goal, so finally do TORQUE\_DISABLE to prepare to the next sequence.

Since the Dynamixel has its own rotation range, it may shows malfunction if your request on your dynamixel is out of range. For example, Dynamixel MX-28 and Dynamixel PRO 54-200 has its rotatable range as 0 ~ 4028 and -250950 ~ 250950, each.

DXL\_MOVING\_STATUS\_THRESHOLD acts as a criteria for verifying its rotation stopped.

int getch()

{

#ifdef \_\_linux\_\_

struct termios oldt, newt;

int ch;

tcgetattr(STDIN\_FILENO, &oldt);

newt = oldt;

newt.c\_lflag &= ~(ICANON | ECHO);

tcsetattr(STDIN\_FILENO, TCSANOW, &newt);

ch = getchar();

tcsetattr(STDIN\_FILENO, TCSANOW, &oldt);

return ch;

#elif defined(\_WIN32) || defined(\_WIN64)

return \_getch();

#endif

}

int kbhit(void)

{

#ifdef \_\_linux\_\_

struct termios oldt, newt;

int ch;

int oldf;

tcgetattr(STDIN\_FILENO, &oldt);

newt = oldt;

newt.c\_lflag &= ~(ICANON | ECHO);

tcsetattr(STDIN\_FILENO, TCSANOW, &newt);

oldf = fcntl(STDIN\_FILENO, F\_GETFL, 0);

fcntl(STDIN\_FILENO, F\_SETFL, oldf | O\_NONBLOCK);

ch = getchar();

tcsetattr(STDIN\_FILENO, TCSANOW, &oldt);

fcntl(STDIN\_FILENO, F\_SETFL, oldf);

if (ch != EOF)

{

ungetc(ch, stdin);

return 1;

}

return 0;

#elif defined(\_WIN32) || defined(\_WIN64)

return \_kbhit();

#endif

}

These functions accept the key inputs in terms of example action. The example codes mainly apply the function getch() rather than the function kbhit() to get information which key has been pressed.

int main()

{

// Initialize PortHandler Structs

// Set the port path

// Get methods and members of PortHandlerLinux or PortHandlerWindows

int port\_num = portHandler(DEVICENAME);

// Initialize PacketHandler Structs

packetHandler();

// Initialize Groupbulkread Structs

int group\_num = groupBulkRead(port\_num, PROTOCOL\_VERSION);

int index = 0;

int dxl\_comm\_result = COMM\_TX\_FAIL; // Communication result

uint8\_t dxl\_addparam\_result = False; // AddParam result

uint8\_t dxl\_getdata\_result = False; // GetParam result

int dxl\_goal\_position[2] = { DXL\_MINIMUM\_POSITION\_VALUE, DXL\_MAXIMUM\_POSITION\_VALUE }; // Goal position

uint8\_t dxl\_error = 0; // Dynamixel error

uint16\_t dxl1\_present\_position = 0; // Present position

uint8\_t dxl2\_moving = 0; // Dynamixel moving status

// Open port

if (openPort(port\_num))

{

printf("Succeeded to open the port!\n");

}

else

{

printf("Failed to open the port!\n");

printf("Press any key to terminate...\n");

getch();

return 0;

}

// Set port baudrate

if (setBaudRate(port\_num, BAUDRATE))

{

printf("Succeeded to change the baudrate!\n");

}

else

{

printf("Failed to change the baudrate!\n");

printf("Press any key to terminate...\n");

getch();

return 0;

}

// Enable Dynamixel#1 Torque

write1ByteTxRx(port\_num, PROTOCOL\_VERSION, DXL1\_ID, ADDR\_MX\_TORQUE\_ENABLE, TORQUE\_ENABLE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

else

{

printf("Dynamixel#%d has been successfully connected \n", DXL1\_ID);

}

// Enable Dynamixel#2 Torque

write1ByteTxRx(port\_num, PROTOCOL\_VERSION, DXL2\_ID, ADDR\_MX\_TORQUE\_ENABLE, TORQUE\_ENABLE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

else

{

printf("Dynamixel#%d has been successfully connected \n", DXL2\_ID);

}

// Add parameter storage for Dynamixel#1 present position value

dxl\_addparam\_result = groupBulkReadAddParam(group\_num, DXL1\_ID, ADDR\_MX\_PRESENT\_POSITION, LEN\_MX\_PRESENT\_POSITION);

if (dxl\_addparam\_result != True)

{

fprintf(stderr, "[ID:%03d] groupBulkRead addparam failed", DXL1\_ID);

return 0;

}

// Add parameter storage for Dynamixel#2 present moving value

dxl\_addparam\_result = groupBulkReadAddParam(group\_num, DXL2\_ID, ADDR\_MX\_MOVING, LEN\_MX\_MOVING);

if (dxl\_addparam\_result != True)

{

fprintf(stderr, "[ID:%03d] groupBulkRead addparam failed", DXL2\_ID);

return 0;

}

while (1)

{

printf("Press any key to continue! (or press ESC to quit!)\n");

if (getch() == ESC\_ASCII\_VALUE)

break;

// Write Dynamixel#1 goal position

write2ByteTxRx(port\_num, PROTOCOL\_VERSION, DXL1\_ID, ADDR\_MX\_GOAL\_POSITION, dxl\_goal\_position[index]);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

// Write Dynamixel#2 goal position

write2ByteTxRx(port\_num, PROTOCOL\_VERSION, DXL2\_ID, ADDR\_MX\_GOAL\_POSITION, dxl\_goal\_position[index]);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

do

{

// Bulkread present position and moving status

groupBulkReadTxRxPacket(group\_num);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

dxl\_getdata\_result = groupBulkReadIsAvailable(group\_num, DXL1\_ID, ADDR\_MX\_PRESENT\_POSITION, LEN\_MX\_PRESENT\_POSITION);

if (dxl\_getdata\_result != True)

{

fprintf(stderr, "[ID:%03d] groupBulkRead getdata failed", DXL1\_ID);

return 0;

}

dxl\_getdata\_result = groupBulkReadIsAvailable(group\_num, DXL2\_ID, ADDR\_MX\_MOVING, LEN\_MX\_MOVING);

if (dxl\_getdata\_result != True)

{

fprintf(stderr, "[ID:%03d] groupBulkRead getdata failed", DXL2\_ID);

return 0;

}

// Get Dynamixel#1 present position value

dxl1\_present\_position = groupBulkReadGetData(group\_num, DXL1\_ID, ADDR\_MX\_PRESENT\_POSITION, LEN\_MX\_PRESENT\_POSITION);

// Get Dynamixel#2 moving status value

dxl2\_moving = groupBulkReadGetData(group\_num, DXL2\_ID, ADDR\_MX\_MOVING, LEN\_MX\_MOVING);

printf("[ID:%03d] Present Position : %d \t [ID:%03d] Is Moving : %d\n", DXL1\_ID, dxl1\_present\_position, DXL2\_ID, dxl2\_moving);

} while (abs(dxl\_goal\_position[index] - dxl1\_present\_position) > DXL\_MOVING\_STATUS\_THRESHOLD);

// Change goal position

if (index == 0)

{

index = 1;

}

else

{

index = 0;

}

}

// Disable Dynamixel#1 Torque

write1ByteTxRx(port\_num, PROTOCOL\_VERSION, DXL1\_ID, ADDR\_MX\_TORQUE\_ENABLE, TORQUE\_DISABLE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

// Disable Dynamixel#2 Torque

write1ByteTxRx(port\_num, PROTOCOL\_VERSION, DXL2\_ID, ADDR\_MX\_TORQUE\_ENABLE, TORQUE\_DISABLE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

// Close port

closePort(port\_num);

return 0;

}

In main() function, the codes call actual functions for Dynamixel control.

// Initialize PortHandler Structs

// Set the port path

// Get methods and members of PortHandlerLinux or PortHandlerWindows

int port\_num = portHandler(DEVICENAME);

portHandler() function sets port path as DEVICENAME and get port\_num, and prepares an appropriate functions for port control in controller OS automatically. port\_num would be used in many functions in the body of the code to specify the port for use.

// Initialize PacketHandler Structs

packetHandler();

packetHandler() function initializes parameters used for packet construction and packet storing.

// Initialize Groupbulkread Structs

int group\_num = groupBulkRead(port\_num, PROTOCOL\_VERSION);

groupBulkRead() function initializes grouped parameters used for packet construction and packet storing. The utility functions of bulk read deals simultaneously with more than one Dynamixel through #port\_num port, building packets by the function which uses PROTOCOL\_VERSION.

int index = 0;

int dxl\_comm\_result = COMM\_TX\_FAIL; // Communication result

bool dxl\_addparam\_result = false; // AddParam result

bool dxl\_getdata\_result = false; // GetParam result

int dxl\_goal\_position[2] = {DXL\_MINIMUM\_POSITION\_VALUE, DXL\_MAXIMUM\_POSITION\_VALUE}; // Goal position

uint8\_t dxl\_error = 0; // Dynamixel error

uint16\_t dxl1\_present\_position = 0; // Present position

uint8\_t dxl2\_moving = 0; // Dynamixel moving status

index variable points the direction to where the Dynamixel should be rotated.  
dxl\_comm\_result indicates which error has been occurred during packet communication.  
dxl\_addparam\_result indicates the result of parameter addition used for sync/bulk related functions  
dxl\_getdata\_result indicates the result of data reception used for sync/bulk related functions  
dxl\_goal\_position stores goal points of Dynamixel rotation.  
dxl\_error shows the internal error in Dynamixel.  
dxl1\_present\_position views where now Dynamixel DXL1\_ID points out.  
dxl2\_moving views whether the Dynamixel is stopped.

// Open port

if (openPort(port\_num))

{

printf("Succeeded to open the port!\n");

}

else

{

printf("Failed to open the port!\n");

printf("Press any key to terminate...\n");

\_getch();

return 0;

}

First, controller opens #port\_num port to do serial communication with the Dynamixel. If it fails to open the port, the example will be terminated.

// Set port baudrate

if (setBaudRate(port\_num, BAUDRATE))

{

printf("Succeeded to change the baudrate!\n");

}

else

{

printf("Failed to change the baudrate!\n");

printf("Press any key to terminate...\n");

\_getch();

return 0;

}

Secondly, the controller sets the communication BAUDRATE at #port\_num port opened previously.

// Enable Dynamixel#1 Torque

write1ByteTxRx(port\_num, PROTOCOL\_VERSION, DXL1\_ID, ADDR\_MX\_TORQUE\_ENABLE, TORQUE\_ENABLE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

else

{

printf("Dynamixel#%d has been successfully connected \n", DXL1\_ID);

}

// Enable Dynamixel#2 Torque

write1ByteTxRx(port\_num, PROTOCOL\_VERSION, DXL2\_ID, ADDR\_MX\_TORQUE\_ENABLE, TORQUE\_ENABLE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

else

{

printf("Dynamixel#%d has been successfully connected \n", DXL2\_ID);

}

As mentioned in the document, above code enables each Dynamixel`s torque to set their status as being ready to move.

write1ByteTxRx() function orders to the #DXL1\_ID and #DXL2\_ID Dynamixels in PROTOCOL\_VERSION communication protocol through #port\_num port, writing 1 byte of TORQUE\_ENABLE value to ADDR\_MX\_TORQUE\_ENABLE address. The function checks Tx/Rx result and receives Hardware error.

getLastTxRxResult() function and getLastRxPacketError() function get either, and then printTxRxResult() function and printRxPacketError() function show results on the console window if any communication error or Hardware error has been occurred.

// Add parameter storage for Dynamixel#1 present position value

dxl\_addparam\_result = groupBulkReadAddParam(group\_num, DXL1\_ID, ADDR\_MX\_PRESENT\_POSITION, LEN\_MX\_PRESENT\_POSITION);

if (dxl\_addparam\_result != True)

{

fprintf(stderr, "[ID:%03d] groupBulkRead addparam failed", DXL1\_ID);

return 0;

}

// Add parameter storage for Dynamixel#2 present moving value

dxl\_addparam\_result = groupBulkReadAddParam(group\_num, DXL2\_ID, ADDR\_MX\_MOVING, LEN\_MX\_MOVING);

if (dxl\_addparam\_result != True)

{

fprintf(stderr, "[ID:%03d] groupBulkRead addparam failed", DXL2\_ID);

return 0;

}

groupBulkReadAddParam() function stores the Dynamixel ID and address ADDR\_MX\_MOVING, byte length LEN\_MX\_MOVING of required data to the bulkread target Dynamixel list.

while (1)

{

printf("Press any key to continue! (or press ESC to quit!)\n");

if (getch() == ESC\_ASCII\_VALUE)

break;

// Write Dynamixel#1 goal position

write2ByteTxRx(port\_num, PROTOCOL\_VERSION, DXL1\_ID, ADDR\_MX\_GOAL\_POSITION, dxl\_goal\_position[index]);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

// Write Dynamixel#2 goal position

write2ByteTxRx(port\_num, PROTOCOL\_VERSION, DXL2\_ID, ADDR\_MX\_GOAL\_POSITION, dxl\_goal\_position[index]);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

do

{

// Bulkread present position and moving status

groupBulkReadTxRxPacket(group\_num);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

dxl\_getdata\_result = groupBulkReadIsAvailable(group\_num, DXL1\_ID, ADDR\_MX\_PRESENT\_POSITION, LEN\_MX\_PRESENT\_POSITION);

if (dxl\_getdata\_result != True)

{

fprintf(stderr, "[ID:%03d] groupBulkRead getdata failed", DXL1\_ID);

return 0;

}

dxl\_getdata\_result = groupBulkReadIsAvailable(group\_num, DXL2\_ID, ADDR\_MX\_MOVING, LEN\_MX\_MOVING);

if (dxl\_getdata\_result != True)

{

fprintf(stderr, "[ID:%03d] groupBulkRead getdata failed", DXL2\_ID);

return 0;

}

// Get Dynamixel#1 present position value

dxl1\_present\_position = groupBulkReadGetData(group\_num, DXL1\_ID, ADDR\_MX\_PRESENT\_POSITION, LEN\_MX\_PRESENT\_POSITION);

// Get Dynamixel#2 moving status value

dxl2\_moving = groupBulkReadGetData(group\_num, DXL2\_ID, ADDR\_MX\_MOVING, LEN\_MX\_MOVING);

printf("[ID:%03d] Present Position : %d \t [ID:%03d] Is Moving : %d\n", DXL1\_ID, dxl1\_present\_position, DXL2\_ID, dxl2\_moving);

} while (abs(dxl\_goal\_position[index] - dxl1\_present\_position) > DXL\_MOVING\_STATUS\_THRESHOLD);

// Change goal position

if (index == 0)

{

index = 1;

}

else

{

index = 0;

}

}

During while() loop, the controller writes and reads each Dynamixel position or mov ing status through packet transmission/reception(Tx/Rx).

To continue their rotation, press any key except ESC.

write2ByteTxRx() function orders to the #DXL1\_ID and #DXL2\_ID Dynamixels in PROTOCOL\_VERSION communication protocol through #port\_num port, writing 2 byte of dxl\_goal\_position[index] value to ADDR\_MX\_GOAL\_POSITION address. The function checks Tx/Rx result and receives Hardware error. getLastTxRxResult() function and getLastRxPacketError() function get either, and then printTxRxResult() function and printRxPacketError() function show results on the console window if any communication error or Hardware error has been occurred.

groupBulkReadTxRxPacket() function orders to the Dynamixel #DXL1\_ID and #DXL2\_ID at the same time through #port\_num port, making it possible to require data bytes from different address. (In this example, LEN\_MX\_PRESENT\_POSITION bytes of the values to the address ADDR\_MX\_PRESENT\_POSITION and LEN\_MX\_MOVING bytes of the values to the address ADDR\_MX\_MOVING, each.) The function checks Tx/Rx result. getLastTxRxResult() function gets it, and then printTxRxResult() function shows result on the console window if any communication error has been occurred.

groupBulkReadIsAvailable() function checks if available data is in the groupbulkread data storage. The function returns false if no data is available in the storage.

groupBulkReadGetData() function pop the data received by groupBulkReadTxRxPacket() function out. In the example, it stores LEN\_MX\_PRESENT\_POSITION byte data got from ADDR\_MX\_PRESENT\_POSITION address of DXL1\_ID Dynamixel and LEN\_MX\_MOVING byte data got from ADDR\_MX\_MOVING address of DXL2\_ID Dynamixel, each.

groupBulkReadClearParam() function clears the Dynamixel list of groupbulkread.

Reading their present position will be ended when absolute value of (dxl\_goal\_position[index] - dxl1\_present\_position) becomes smaller then DXL\_MOVING\_STATUS\_THRESHOLD.

At last, it changes their direction to the counter-wise and waits for extra key input.

// Disable Dynamixel#1 Torque

write1ByteTxRx(port\_num, PROTOCOL\_VERSION, DXL1\_ID, ADDR\_MX\_TORQUE\_ENABLE, TORQUE\_DISABLE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

// Disable Dynamixel#2 Torque

write1ByteTxRx(port\_num, PROTOCOL\_VERSION, DXL2\_ID, ADDR\_MX\_TORQUE\_ENABLE, TORQUE\_DISABLE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

The controller frees the Dynamixels to be idle.

write1ByteTxRx() function orders to the #DXL1\_ID and #DXL2\_ID Dynamixels in PROTOCOL\_VERSION communication protocol through #port\_num port, writing 1 byte of TORQUE\_DISABLE value to ADDR\_MX\_TORQUE\_ENABLE address. The function checks Tx/Rx result and receives Hardware error.

getLastTxRxResult() function and getLastRxPacketError() function get either, and then printTxRxResult() function and printRxPacketError() function show results on the console window if any communication error or Hardware error has been occurred.

// Close port

closePort(port\_num);

return 0;

Finally, port becomes disposed.

© 2019 ROBOTIS. Powered by [Jekyll](http://jekyllrb.com/) & [Minimal Mistakes](https://mademistakes.com/work/minimal-mistakes-jekyll-theme/).

### [C Ping Protocol 1.0](http://emanual.robotis.com/docs/en/software/dynamixel/dynamixel_sdk/sample_code/c_ping_protocol_1_0/" \l "c-ping-protocol-10)

* Description

This example pings Dynamixel. The ping returns Dynamixel’s Model Number. Ping is commonly used to verify that Dynamixel is successfully connected.

* Supported Dynamixels

Protocol 1.0 Dynamixels

#### Sample code

/\*

\* ping.c

\*

\* Created on: 2016. 5. 16.

\* Author: Leon Ryu Woon Jung

\*/

//

// \*\*\*\*\*\*\*\*\* ping Example \*\*\*\*\*\*\*\*\*

//

//

// Available Dynamixel model on this example : All models using Protocol 1.0

// This example is designed for using a Dynamixel MX-28, and an USB2DYNAMIXEL.

// To use another Dynamixel model, such as X series, see their details in E-Manual(support.robotis.com) and edit below "#define"d variables yourself.

// Be sure that Dynamixel MX properties are already set as %% ID : 1 / Baudnum : 1 (Baudrate : 1000000 [1M])

//

#ifdef \_\_linux\_\_

#include <unistd.h>

#include <fcntl.h>

#include <termios.h>

#elif defined(\_WIN32) || defined(\_WIN64)

#include <conio.h>

#endif

#include <stdio.h>

#include "dynamixel\_sdk.h" // Uses Dynamixel SDK library

// Protocol version

#define PROTOCOL\_VERSION 1.0 // See which protocol version is used in the Dynamixel

// Default setting

#define DXL\_ID 1 // Dynamixel ID: 1

#define BAUDRATE 1000000

#define DEVICENAME "/dev/ttyUSB0" // Check which port is being used on your controller

// ex) Windows: "COM1" Linux: "/dev/ttyUSB0"

int getch()

{

#ifdef \_\_linux\_\_

struct termios oldt, newt;

int ch;

tcgetattr(STDIN\_FILENO, &oldt);

newt = oldt;

newt.c\_lflag &= ~(ICANON | ECHO);

tcsetattr(STDIN\_FILENO, TCSANOW, &newt);

ch = getchar();

tcsetattr(STDIN\_FILENO, TCSANOW, &oldt);

return ch;

#elif defined(\_WIN32) || defined(\_WIN64)

return \_getch();

#endif

}

int kbhit(void)

{

#ifdef \_\_linux\_\_

struct termios oldt, newt;

int ch;

int oldf;

tcgetattr(STDIN\_FILENO, &oldt);

newt = oldt;

newt.c\_lflag &= ~(ICANON | ECHO);

tcsetattr(STDIN\_FILENO, TCSANOW, &newt);

oldf = fcntl(STDIN\_FILENO, F\_GETFL, 0);

fcntl(STDIN\_FILENO, F\_SETFL, oldf | O\_NONBLOCK);

ch = getchar();

tcsetattr(STDIN\_FILENO, TCSANOW, &oldt);

fcntl(STDIN\_FILENO, F\_SETFL, oldf);

if (ch != EOF)

{

ungetc(ch, stdin);

return 1;

}

return 0;

#elif defined(\_WIN32) || defined(\_WIN64)

return \_kbhit();

#endif

}

int main()

{

// Initialize PortHandler Structs

// Set the port path

// Get methods and members of PortHandlerLinux or PortHandlerWindows

int port\_num = portHandler(DEVICENAME);

// Initialize PacketHandler Structs

packetHandler();

int dxl\_comm\_result = COMM\_TX\_FAIL; // Communication result

uint8\_t dxl\_error = 0; // Dynamixel error

uint16\_t dxl\_model\_number; // Dynamixel model number

// Open port

if (openPort(port\_num))

{

printf("Succeeded to open the port!\n");

}

else

{

printf("Failed to open the port!\n");

printf("Press any key to terminate...\n");

getch();

return 0;

}

// Set port baudrate

if (setBaudRate(port\_num, BAUDRATE))

{

printf("Succeeded to change the baudrate!\n");

}

else

{

printf("Failed to change the baudrate!\n");

printf("Press any key to terminate...\n");

getch();

return 0;

}

// Try to ping the Dynamixel

// Get Dynamixel model number

dxl\_model\_number = pingGetModelNum(port\_num, PROTOCOL\_VERSION, DXL\_ID);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

printf("[ID:%03d] ping Succeeded. Dynamixel model number : %d\n", DXL\_ID, dxl\_model\_number);

// Close port

closePort(port\_num);

return 0;

}

#### Details

#ifdef \_\_linux\_\_

#include <unistd.h>

#include <fcntl.h>

#include <termios.h>

#elif defined(\_WIN32) || defined(\_WIN64)

#include <conio.h>

#endif

This source includes above to get key input interruption while the example is running. Actual functions for getting the input is described in a little below.

#include <stdio.h>

The example shows Dynamixel status in sequence by the function printf(). So here stdio.h is needed.

#include "dynamixel\_sdk.h" // Uses Dynamixel SDK library

All libraries of Dynamixel SDK are linked with the header file dynamixel\_sdk.h.

// Protocol version

#define PROTOCOL\_VERSION 1.0 // See which protocol version is used in the Dynamixel

Dynamixel uses either or both protocols: Protocol 1.0 and Protocol 2.0. Choose one of the Protocol which is appropriate in the Dynamixel.

// Default setting

#define DXL\_ID 1 // Dynamixel ID: 1

#define BAUDRATE 1000000

#define DEVICENAME "/dev/ttyUSB0" // Check which port is being used on your controller

// ex) Windows: "COM1" Linux: "/dev/ttyUSB0"

Here we set some variables to let you freely change them and use them to run the example code.

As the document previously said in [previous chapter](http://emanual.robotis.com/docs/en/software/dynamixel/dynamixel_sdk/device_setup/" \l "dynamixel), customize Dynamixel control table items, such as DXL\_ID number, communication BAUDRATE, and the DEVICENAME, on your own terms of needs. In particular, BAUDRATE and DEVICENAME have systematical dependencies on your controller, so make clear what kind of communication method you will use.

int getch()

{

#ifdef \_\_linux\_\_

struct termios oldt, newt;

int ch;

tcgetattr(STDIN\_FILENO, &oldt);

newt = oldt;

newt.c\_lflag &= ~(ICANON | ECHO);

tcsetattr(STDIN\_FILENO, TCSANOW, &newt);

ch = getchar();

tcsetattr(STDIN\_FILENO, TCSANOW, &oldt);

return ch;

#elif defined(\_WIN32) || defined(\_WIN64)

return \_getch();

#endif

}

int kbhit(void)

{

#ifdef \_\_linux\_\_

struct termios oldt, newt;

int ch;

int oldf;

tcgetattr(STDIN\_FILENO, &oldt);

newt = oldt;

newt.c\_lflag &= ~(ICANON | ECHO);

tcsetattr(STDIN\_FILENO, TCSANOW, &newt);

oldf = fcntl(STDIN\_FILENO, F\_GETFL, 0);

fcntl(STDIN\_FILENO, F\_SETFL, oldf | O\_NONBLOCK);

ch = getchar();

tcsetattr(STDIN\_FILENO, TCSANOW, &oldt);

fcntl(STDIN\_FILENO, F\_SETFL, oldf);

if (ch != EOF)

{

ungetc(ch, stdin);

return 1;

}

return 0;

#elif defined(\_WIN32) || defined(\_WIN64)

return \_kbhit();

#endif

}

These functions accept the key inputs in terms of example action. The example codes mainly apply the function getch() rather than the function kbhit() to get information which key has been pressed.

int main()

{

// Initialize PortHandler Structs

// Set the port path

// Get methods and members of PortHandlerLinux or PortHandlerWindows

int port\_num = portHandler(DEVICENAME);

// Initialize PacketHandler Structs

packetHandler();

int dxl\_comm\_result = COMM\_TX\_FAIL; // Communication result

uint8\_t dxl\_error = 0; // Dynamixel error

uint16\_t dxl\_model\_number; // Dynamixel model number

// Open port

if (openPort(port\_num))

{

printf("Succeeded to open the port!\n");

}

else

{

printf("Failed to open the port!\n");

printf("Press any key to terminate...\n");

getch();

return 0;

}

// Set port baudrate

if (setBaudRate(port\_num, BAUDRATE))

{

printf("Succeeded to change the baudrate!\n");

}

else

{

printf("Failed to change the baudrate!\n");

printf("Press any key to terminate...\n");

getch();

return 0;

}

// Try to ping the Dynamixel

// Get Dynamixel model number

dxl\_model\_number = pingGetModelNum(port\_num, PROTOCOL\_VERSION, DXL\_ID);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

printf("[ID:%03d] ping Succeeded. Dynamixel model number : %d\n", DXL\_ID, dxl\_model\_number);

// Close port

closePort(port\_num);

return 0;

}

In main() function, the codes call actual functions for Dynamixel control.

// Initialize PortHandler Structs

// Set the port path

// Get methods and members of PortHandlerLinux or PortHandlerWindows

int port\_num = portHandler(DEVICENAME);

portHandler() function sets port path as DEVICENAME and get port\_num, and prepares an appropriate functions for port control in controller OS automatically. port\_num would be used in many functions in the body of the code to specify the port for use.

// Initialize PacketHandler Structs

packetHandler();

packetHandler() function initializes parameters used for packet construction and packet storing.

int dxl\_comm\_result = COMM\_TX\_FAIL; // Communication result

uint8\_t dxl\_error = 0; // Dynamixel error

uint16\_t dxl\_model\_number; // Dynamixel model number

dxl\_comm\_result indicates which error has been occurred during packet communication.

dxl\_error shows the internal error in Dynamixel.

dxl\_model\_number keeps Dynamixel model number.

// Open port

if (openPort(port\_num))

{

printf("Succeeded to open the port!\n");

}

else

{

printf("Failed to open the port!\n");

printf("Press any key to terminate...\n");

\_getch();

return 0;

}

First, controller opens #port\_num port to do serial communication with the Dynamixel. If it fails to open the port, the example will be terminated.

// Set port baudrate

if (setBaudRate(port\_num, BAUDRATE))

{

printf("Succeeded to change the baudrate!\n");

}

else

{

printf("Failed to change the baudrate!\n");

printf("Press any key to terminate...\n");

\_getch();

return 0;

}

Secondly, the controller sets the communication BAUDRATE at #port\_num port opened previously.

dxl\_model\_number = pingGetModelNum(port\_num, PROTOCOL\_VERSION, DXL\_ID);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

pingGetModelNum() function orders to the #DXL\_ID Dynamixel through #port\_num port. Then, it receives the dxl\_model\_number. The function checks Tx/Rx result and receives Hardware error. getLastTxRxResult() function and getLastRxPacketError() function get either, and then printTxRxResult() function and printRxPacketError() function show results on the console window if any communication error or Hardware error has been occurred.

// Close port

closePort(port\_num);

return 0;

Finally, port becomes disposed.

© 2019 ROBOTIS. Powered by [Jekyll](http://jekyllrb.com/) & [Minimal Mistakes](https://mademistakes.com/work/minimal-mistakes-jekyll-theme/).

### [C Reset Protocol 1.0](http://emanual.robotis.com/docs/en/software/dynamixel/dynamixel_sdk/sample_code/c_reset_protocol_1_0/" \l "c-reset-protocol-10)

* Description

This example resets all Dynamixel settings to factory defaults, including rate ID (1) and baud rate (57600 bps):

* Supported Dynamixels

Protocol 1.0 Dynamixels

#### Sample code

/\*

\* reset.c

\*

\* Created on: 2016. 5. 16.

\* Author: Leon Ryu Woon Jung

\*/

//

// \*\*\*\*\*\*\*\*\* Factory Reset Example \*\*\*\*\*\*\*\*\*

//

//

// Available Dynamixel model on this example : All models using Protocol 1.0

// This example is designed for using a Dynamixel MX-28, and an USB2DYNAMIXEL.

// To use another Dynamixel model, such as X series, see their details in E-Manual(support.robotis.com) and edit below "#define"d variables yourself.

// Be sure that Dynamixel PRO properties are already set as %% ID : 1 / Baudnum : 1 (Baudrate : 1000000 [1M])

//

// Be aware that:

// This example resets all properties of Dynamixel to default values, such as %% ID : 1 / Baudnum : 34 (Baudrate : 57600)

//

#ifdef \_\_linux\_\_

#include <unistd.h>

#include <fcntl.h>

#include <termios.h>

#elif defined(\_WIN32) || defined(\_WIN64)

#include <conio.h>

#endif

#include <stdio.h>

#include "dynamixel\_sdk.h" // Uses Dynamixel SDK library

// Control table address

#define ADDR\_MX\_BAUDRATE 4 // Control table address is different in Dynamixel model

// Protocol version

#define PROTOCOL\_VERSION 1.0 // See which protocol version is used in the Dynamixel

// Default setting

#define DXL\_ID 1 // Dynamixel ID: 1

#define BAUDRATE 1000000

#define DEVICENAME "/dev/ttyUSB0" // Check which port is being used on your controller

// ex) Windows: "COM1" Linux: "/dev/ttyUSB0"

#define FACTORYRST\_DEFAULTBAUDRATE 57600 // Dynamixel baudrate set by factoryreset

#define NEW\_BAUDNUM 1 // New baudnum to recover Dynamixel baudrate as it was

#define OPERATION\_MODE 0x00 // Mode is unavailable in Protocol 1.0 Reset

#define TORQUE\_ENABLE 1 // Value for enabling the torque

#define TORQUE\_DISABLE 0 // Value for disabling the torque

int getch()

{

#ifdef \_\_linux\_\_

struct termios oldt, newt;

int ch;

tcgetattr(STDIN\_FILENO, &oldt);

newt = oldt;

newt.c\_lflag &= ~(ICANON | ECHO);

tcsetattr(STDIN\_FILENO, TCSANOW, &newt);

ch = getchar();

tcsetattr(STDIN\_FILENO, TCSANOW, &oldt);

return ch;

#elif defined(\_WIN32) || defined(\_WIN64)

return \_getch();

#endif

}

int kbhit(void)

{

#ifdef \_\_linux\_\_

struct termios oldt, newt;

int ch;

int oldf;

tcgetattr(STDIN\_FILENO, &oldt);

newt = oldt;

newt.c\_lflag &= ~(ICANON | ECHO);

tcsetattr(STDIN\_FILENO, TCSANOW, &newt);

oldf = fcntl(STDIN\_FILENO, F\_GETFL, 0);

fcntl(STDIN\_FILENO, F\_SETFL, oldf | O\_NONBLOCK);

ch = getchar();

tcsetattr(STDIN\_FILENO, TCSANOW, &oldt);

fcntl(STDIN\_FILENO, F\_SETFL, oldf);

if (ch != EOF)

{

ungetc(ch, stdin);

return 1;

}

return 0;

#elif defined(\_WIN32) || defined(\_WIN64)

return \_kbhit();

#endif

}

void msecSleep(int waitTime)

{

#ifdef \_\_linux\_\_

usleep(waitTime \* 1000);

#elif defined(\_WIN32) || defined(\_WIN64)

Sleep(waitTime);

#endif

}

int main()

{

// Initialize PortHandler Structs

// Set the port path

// Get methods and members of PortHandlerLinux or PortHandlerWindows

int port\_num = portHandler(DEVICENAME);

// Initialize PacketHandler Structs

packetHandler();

int dxl\_comm\_result = COMM\_TX\_FAIL; // Communication result

uint8\_t dxl\_error = 0; // Dynamixel error

uint8\_t dxl\_baudnum\_read; // Read baudnum

// Open port

if (openPort(port\_num))

{

printf("Succeeded to open the port!\n");

}

else

{

printf("Failed to open the port!\n");

printf("Press any key to terminate...\n");

getch();

return 0;

}

// Set port baudrate

if (setBaudRate(port\_num, BAUDRATE))

{

printf("Succeeded to change the baudrate!\n");

}

else

{

printf("Failed to change the baudrate!\n");

printf("Press any key to terminate...\n");

getch();

return 0;

}

// Read present baudrate of the controller

printf("Now the controller baudrate is : %d\n", getBaudRate(port\_num));

// Try factoryreset

printf("[ID:%03d] Try factoryreset : ", DXL\_ID);

factoryReset(port\_num, PROTOCOL\_VERSION, DXL\_ID, OPERATION\_MODE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printf("Aborted\n");

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

return 0;

}

else if ((dxl\_error = getLastRxPacketError(port\_num, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

// Wait for reset

printf("Wait for reset...\n");

msecSleep(2000);

printf("[ID:%03d] factoryReset Success!\n", DXL\_ID);

// Set controller baudrate to dxl default baudrate

if (setBaudRate(port\_num, FACTORYRST\_DEFAULTBAUDRATE))

{

printf("Succeed to change the controller baudrate to : %d\n", FACTORYRST\_DEFAULTBAUDRATE);

}

else

{

printf("Failed to change the controller baudrate\n");

printf("Press any key to terminate...\n");

getch();

return 0;

}

// Read Dynamixel baudnum

dxl\_baudnum\_read = read1ByteTxRx(port\_num, PROTOCOL\_VERSION, DXL\_ID, ADDR\_MX\_BAUDRATE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

else

{

printf("[ID:%03d] Dynamixel baudnum is now : %d\n", DXL\_ID, dxl\_baudnum\_read);

}

// Write new baudnum

write1ByteTxRx(port\_num, PROTOCOL\_VERSION, DXL\_ID, ADDR\_MX\_BAUDRATE, NEW\_BAUDNUM);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

else

{

printf("[ID:%03d] Set Dynamixel baudnum to : %d\n", DXL\_ID, NEW\_BAUDNUM);

}

// Set port baudrate to BAUDRATE

if (setBaudRate(port\_num, BAUDRATE))

{

printf("Succeed to change the controller baudrate to : %d\n", BAUDRATE);

}

else

{

printf("Failed to change the controller baudrate\n");

printf("Press any key to terminate...\n");

getch();

return 0;

}

msecSleep(200);

// Read Dynamixel baudnum

dxl\_baudnum\_read = read1ByteTxRx(port\_num, PROTOCOL\_VERSION, DXL\_ID, ADDR\_MX\_BAUDRATE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

else

{

printf("[ID:%03d] Dynamixel Baudnum is now : %d\n", DXL\_ID, dxl\_baudnum\_read);

}

// Close port

closePort(port\_num);

return 0;

}

#### Details

#ifdef \_\_linux\_\_

#include <unistd.h>

#include <fcntl.h>

#include <termios.h>

#elif defined(\_WIN32) || defined(\_WIN64)

#include <conio.h>

#endif

This source includes above to get key input interruption while the example is running. Actual functions for getting the input is described in a little below.

#include <stdio.h>

The example shows Dynamixel status in sequence by the function printf(). So here stdio.h is needed.

#include "dynamixel\_sdk.h" // Uses Dynamixel SDK library

All libraries of Dynamixel SDK are linked with the header file dynamixel\_sdk.h.

// Control table address

#define ADDR\_MX\_BAUDRATE 4 // Control table address is different in Dynamixel model

Dynamixel series have their own control tables: Addresses and Byte Length in each items. To control one of the items, its address (and length if necessary) is required. Find your requirements in <http://emanual.robotis.com/>.

// Protocol version

#define PROTOCOL\_VERSION 1.0 // See which protocol version is used in the Dynamixel

Dynamixel uses either or both protocols: Protocol 1.0 and Protocol 2.0. Choose one of the Protocol which is appropriate in the Dynamixel.

// Default setting

#define DXL\_ID 1 // Dynamixel ID: 1

#define BAUDRATE 1000000

#define DEVICENAME "/dev/ttyUSB0" // Check which port is being used on your controller

// ex) Windows: "COM1" Linux: "/dev/ttyUSB0"

#define FACTORYRST\_DEFAULTBAUDRATE 57600 // Dynamixel baudrate set by factoryreset

#define NEW\_BAUDNUM 1 // New baudnum to recover Dynamixel baudrate as it was

#define OPERATION\_MODE 0x00 // Mode is unavailable in Protocol 1.0 Reset

Here we set some variables to let you freely change them and use them to run the example code.

As the document previously said in [previous chapter](http://emanual.robotis.com/docs/en/software/dynamixel/dynamixel_sdk/device_setup/" \l "dynamixel), customize Dynamixel control table items, such as DXL\_ID number, communication BAUDRATE, and the DEVICENAME, on your own terms of needs. In particular, BAUDRATE and DEVICENAME have systematical dependencies on your controller, so make clear what kind of communication method you will use.

Since the factory reset recovers the Dynamixel control table items to the original state, the baudrate of controller serial port should be set to its changed baudrate(FACTORYRST\_DEFAULTBAUDRATE : 57600 bps) as well. After that, controller sets its baudrate to the value (1000000 bps : NEW\_BAUDNUM = 1) before factory reset.

In Protocol 1.0, only one mode that resets whole items is avaiable.

int getch()

{

#ifdef \_\_linux\_\_

struct termios oldt, newt;

int ch;

tcgetattr(STDIN\_FILENO, &oldt);

newt = oldt;

newt.c\_lflag &= ~(ICANON | ECHO);

tcsetattr(STDIN\_FILENO, TCSANOW, &newt);

ch = getchar();

tcsetattr(STDIN\_FILENO, TCSANOW, &oldt);

return ch;

#elif defined(\_WIN32) || defined(\_WIN64)

return \_getch();

#endif

}

int kbhit(void)

{

#ifdef \_\_linux\_\_

struct termios oldt, newt;

int ch;

int oldf;

tcgetattr(STDIN\_FILENO, &oldt);

newt = oldt;

newt.c\_lflag &= ~(ICANON | ECHO);

tcsetattr(STDIN\_FILENO, TCSANOW, &newt);

oldf = fcntl(STDIN\_FILENO, F\_GETFL, 0);

fcntl(STDIN\_FILENO, F\_SETFL, oldf | O\_NONBLOCK);

ch = getchar();

tcsetattr(STDIN\_FILENO, TCSANOW, &oldt);

fcntl(STDIN\_FILENO, F\_SETFL, oldf);

if (ch != EOF)

{

ungetc(ch, stdin);

return 1;

}

return 0;

#elif defined(\_WIN32) || defined(\_WIN64)

return \_kbhit();

#endif

}

These functions accept the key inputs in terms of example action. The example codes mainly apply the function getch() rather than the function kbhit() to get information which key has been pressed.

void msecSleep(int waitTime)

{

#ifdef \_\_linux\_\_

usleep(waitTime \* 1000);

#elif defined(\_WIN32) || defined(\_WIN64)

Sleep(waitTime);

#endif

}

msecSleep() function halt the computational process in which this function is used.

int main()

{

// Initialize PortHandler Structs

// Set the port path

// Get methods and members of PortHandlerLinux or PortHandlerWindows

int port\_num = portHandler(DEVICENAME);

// Initialize PacketHandler Structs

packetHandler();

int dxl\_comm\_result = COMM\_TX\_FAIL; // Communication result

uint8\_t dxl\_error = 0; // Dynamixel error

uint8\_t dxl\_baudnum\_read; // Read baudnum

// Open port

if (openPort(port\_num))

{

printf("Succeeded to open the port!\n");

}

else

{

printf("Failed to open the port!\n");

printf("Press any key to terminate...\n");

getch();

return 0;

}

// Set port baudrate

if (setBaudRate(port\_num, BAUDRATE))

{

printf("Succeeded to change the baudrate!\n");

}

else

{

printf("Failed to change the baudrate!\n");

printf("Press any key to terminate...\n");

getch();

return 0;

}

// Read present baudrate of the controller

printf("Now the controller baudrate is : %d\n", getBaudRate(port\_num));

// Try factoryreset

printf("[ID:%03d] Try factoryreset : ", DXL\_ID);

factoryReset(port\_num, PROTOCOL\_VERSION, DXL\_ID, OPERATION\_MODE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printf("Aborted\n");

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

return 0;

}

else if ((dxl\_error = getLastRxPacketError(port\_num, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

// Wait for reset

printf("Wait for reset...\n");

msecSleep(2000);

printf("[ID:%03d] factoryReset Success!\n", DXL\_ID);

// Set controller baudrate to dxl default baudrate

if (setBaudRate(port\_num, FACTORYRST\_DEFAULTBAUDRATE))

{

printf("Succeed to change the controller baudrate to : %d\n", FACTORYRST\_DEFAULTBAUDRATE);

}

else

{

printf("Failed to change the controller baudrate\n");

printf("Press any key to terminate...\n");

getch();

return 0;

}

// Read Dynamixel baudnum

dxl\_baudnum\_read = read1ByteTxRx(port\_num, PROTOCOL\_VERSION, DXL\_ID, ADDR\_MX\_BAUDRATE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

else

{

printf("[ID:%03d] Dynamixel baudnum is now : %d\n", DXL\_ID, dxl\_baudnum\_read);

}

// Write new baudnum

write1ByteTxRx(port\_num, PROTOCOL\_VERSION, DXL\_ID, ADDR\_MX\_BAUDRATE, NEW\_BAUDNUM);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

else

{

printf("[ID:%03d] Set Dynamixel baudnum to : %d\n", DXL\_ID, NEW\_BAUDNUM);

}

// Set port baudrate to BAUDRATE

if (setBaudRate(port\_num, BAUDRATE))

{

printf("Succeed to change the controller baudrate to : %d\n", BAUDRATE);

}

else

{

printf("Failed to change the controller baudrate\n");

printf("Press any key to terminate...\n");

getch();

return 0;

}

msecSleep(200);

// Read Dynamixel baudnum

dxl\_baudnum\_read = read1ByteTxRx(port\_num, PROTOCOL\_VERSION, DXL\_ID, ADDR\_MX\_BAUDRATE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

else

{

printf("[ID:%03d] Dynamixel Baudnum is now : %d\n", DXL\_ID, dxl\_baudnum\_read);

}

// Close port

closePort(port\_num);

return 0;

}

In main() function, the codes call actual functions for Dynamixel control.

// Initialize PortHandler Structs

// Set the port path

// Get methods and members of PortHandlerLinux or PortHandlerWindows

int port\_num = portHandler(DEVICENAME);

portHandler() function sets port path as DEVICENAME and get port\_num, and prepares an appropriate functions for port control in controller OS automatically. port\_num would be used in many functions in the body of the code to specify the port for use.

// Initialize PacketHandler Structs

packetHandler();

packetHandler() function initializes parameters used for packet construction and packet storing.

int dxl\_comm\_result = COMM\_TX\_FAIL; // Communication result

uint8\_t dxl\_error = 0; // Dynamixel error

uint8\_t dxl\_baudnum\_read; // Read baudnum

dxl\_comm\_result indicates which error has been occurred during packet communication.

dxl\_error shows the internal error in Dynamixel.

dxl\_baudnum\_read keeps Dynamixel baudrate.

// Open port

if (openPort(port\_num))

{

printf("Succeeded to open the port!\n");

}

else

{

printf("Failed to open the port!\n");

printf("Press any key to terminate...\n");

\_getch();

return 0;

}

First, controller opens #port\_num port to do serial communication with the Dynamixel. If it fails to open the port, the example will be terminated.

// Set port baudrate

if (setBaudRate(port\_num, BAUDRATE))

{

printf("Succeeded to change the baudrate!\n");

}

else

{

printf("Failed to change the baudrate!\n");

printf("Press any key to terminate...\n");

\_getch();

return 0;

}

Secondly, the controller sets the communication BAUDRATE at #port\_num port opened previously.

// Read present baudrate of the controller

printf("Now the controller baudrate is : %d\n", getBaudRate(port\_num));

getBaudRate() function shows which baudrate is used in #port\_num port of the controller.

factoryReset(port\_num, PROTOCOL\_VERSION, DXL\_ID, OPERATION\_MODE);

factoryReset() function orders to the #DXL\_ID Dynamixel through #port\_num port, executing it to be reset as OPERATION\_MODE format. The function checks Tx/Rx result and receives Hardware error. getLastTxRxResult() function and getLastRxPacketError() function get either, and then printTxRxResult() function and printRxPacketError() function show results on the console window if any communication error or Hardware error has been occurred.

// Wait for reset

printf("Wait for reset...\n");

msecSleep(2000);

Factory reset takes few seconds.

// Set controller baudrate to dxl default baudrate

if (setBaudRate(port\_num, FACTORYRST\_DEFAULTBAUDRATE))

{

printf("Succeed to change the controller baudrate to : %d\n", FACTORYRST\_DEFAULTBAUDRATE);

}

else

{

printf("Failed to change the controller baudrate\n");

printf("Press any key to terminate...\n");

\_getch();

return 0;

}

Controller should change its baudrate itself to do the communication with initialized Dynamixel.

// Read Dynamixel baudnum

dxl\_baudnum\_read = read1ByteTxRx(port\_num, PROTOCOL\_VERSION, DXL\_ID, ADDR\_MX\_BAUDRATE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

else

{

printf("[ID:%03d] Dynamixel baudnum is now : %d\n", DXL\_ID, dxl\_baudnum\_read);

}

This shows that reconnection between controller and Dynamixel is happened by adjusting the baudrate.

// Write new baudnum

write1ByteTxRx(port\_num, PROTOCOL\_VERSION, DXL\_ID, ADDR\_MX\_BAUDRATE, NEW\_BAUDNUM);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

else

{

printf("[ID:%03d] Set Dynamixel baudnum to : %d\n", DXL\_ID, NEW\_BAUDNUM);

}

To make the Dynamixel into previous condition, write1ByteTxRx() function orders to the #DXL\_ID Dynamixel in PROTOCOL\_VERSION communication protocol through #port\_num port, writing 1 byte of TORQUE\_ENABLE value to ADDR\_MX\_TORQUE\_ENABLE address. The function checks Tx/Rx result and receives Hardware error. getLastTxRxResult() function and getLastRxPacketError() function get either, and then printTxRxResult() function and printRxPacketError() function show results on the console window if any communication error or Hardware error has been occurred.

// Set port baudrate to BAUDRATE

if (setBaudRate(port\_num, BAUDRATE))

{

printf("Succeed to change the controller baudrate to : %d\n", BAUDRATE);

}

else

{

printf("Failed to change the controller baudrate\n");

printf("Press any key to terminate...\n");

getch();

return 0;

}

msecSleep(200);

// Read Dynamixel baudnum

dxl\_baudnum\_read = read1ByteTxRx(port\_num, PROTOCOL\_VERSION, DXL\_ID, ADDR\_MX\_BAUDRATE);

if ((dxl\_comm\_result = getLastTxRxResult(port\_num, PROTOCOL\_VERSION)) != COMM\_SUCCESS)

{

printTxRxResult(PROTOCOL\_VERSION, dxl\_comm\_result);

}

else if ((dxl\_error = getLastRxPacketError(port\_num, PROTOCOL\_VERSION)) != 0)

{

printRxPacketError(PROTOCOL\_VERSION, dxl\_error);

}

else

{

printf("[ID:%03d] Dynamixel Baudnum is now : %d\n", DXL\_ID, dxl\_baudnum\_read);

}

These changes controller baudrate and verify that the Dynamixel has been successfully set into previous state.

// Close port

closePort(port\_num);

return 0;

Finally, port becomes disposed.

© 2019 ROBOTIS. Powered by [Jekyll](http://jekyllrb.com/) & [Minimal Mistakes](https://mademistakes.com/work/minimal-mistakes-jekyll-theme/).