Lego NXT, Bluetooth and Linux

Jeremy Schultz
University of Lethbridge

January 23, 2008

What is the NXT?

The NXT is the ”brain“ behind Lego’s MINDSTORM robot.

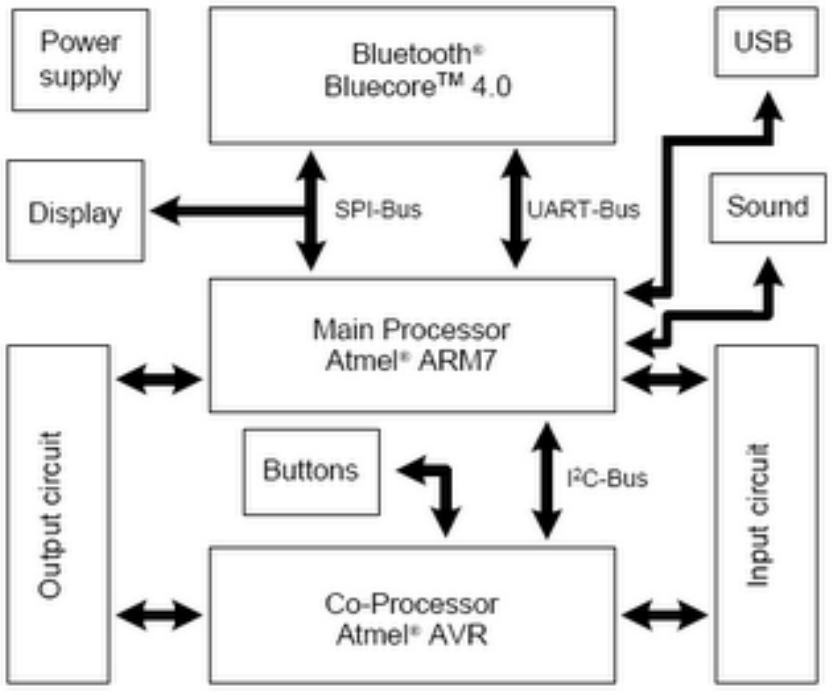
NXT Specs

Technical Specifications:

- 32-bit ARM7 microcontroller (48Mhz)
  with 256 Kbytes FLASH, 64 Kbytes RAM
- 8-bit AVR microcontroller (8Mhz)
  with 4 Kbytes FLASH, 512 Byte RAM
- Bluetooth (Bluetooth Class II V2.0 compliant)
- USB full speed port (12 Mbit/s)
- 4 input ports (1,2,3,4), 6-wire cable digital platform (RJ-12)
- 3 output ports (A,B,C), 6-wire cable digital platform (RJ-12)
- 100 x 64 pixel LCD graphical display
- Loudspeaker - 8 kHz sound quality.
- Power source: 6 AA batteries

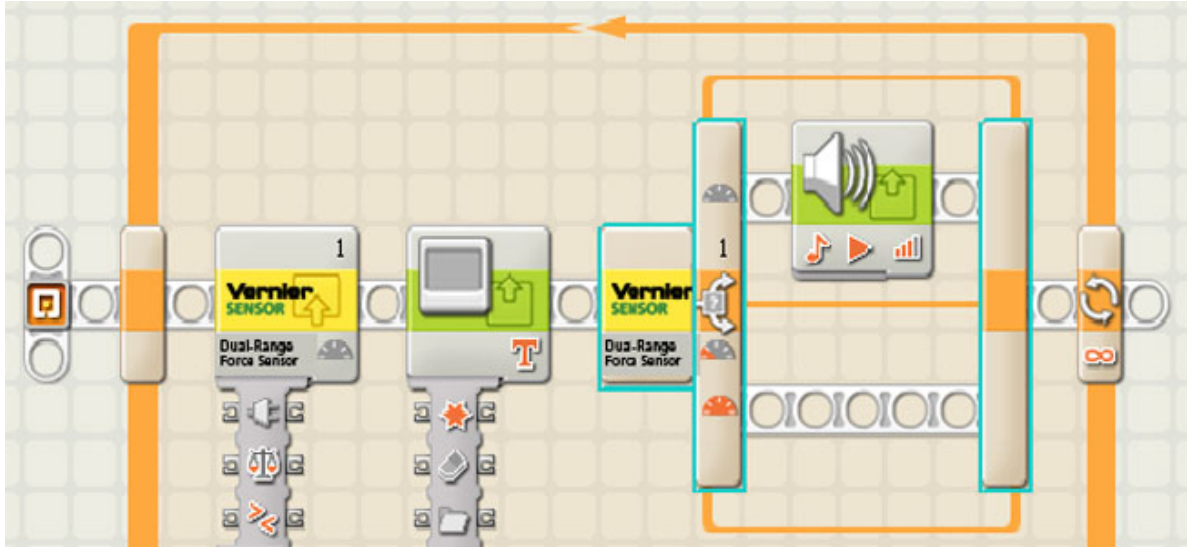
NXT Specs cont.

Basic schematic on how the NXT components work together



NXT Software

The MINDSTORM comes with development software to program the NXT. Currently, only Windows and Mac OSX are supported.



It is powered by NI LabView. A graphical programming software.

What is NBC/NCX?

NBC/NXC

- Next Byte Codes (NBC) is an assembly language for NXT
- Not eXactly C (NCX) is C-type Language built upon NBC compiler
- No need for a modified Firmware
- Available at `http://bricxcc.sourceforge.net/nbc/`

NBC/NCX Documentation

Download the NXC Guide, a 123-paged pdf.

Guide Overview:

- Section 2 gives the basics of the Language
  - Section 2.1.4, Identifiers and Keywords
  - Section 2.2.1, Tasks
    Provides support for multi-threading
    Needs at least one task named ”main“
  - Section 2.2.3, Variables
    eg. bool, int, string, mutex
- Section 3 describes the APIs
  - Section 3.1, General Features
    eg. Timing, String, Array, Numeric, Program Control
  - Section 3.2 - 3.15, Modules
    eg. Comm, Display, Input, Output most have High-level and Low-level functions
- Some functions require a enhanced firmware

Simple Program: hello_world.ncx

```
#define PIXELS_PER_CHAR 6

task main ()    {
  string line1 = "Hello";

  ClearScreen();
  TextOut(0, LCD_LINE1, line1);
  TextOut(strlen(line1) * PIXELS_PER_CHAR, LCD_LINE2, "World");

  LineOut(0, LCD_LINE2-2, 100,  LCD_LINE2-2);

  NumOut(0, LCD_LINE8, 8);
  Wait(3000);
}
```

Compile, Upload and Run a NXC program

Compile:

- Use ”nbc“ with a file ending in ”.nxc“.
- The default extension to use for compiled programs is ”.rxe“

```
# nbc hello_world.nbc -O=hello_world.rxe
```

- Note: The maximum NXT filename length is 20 characters or 15.3

Upload:

- Make sure the NXT is on and the USB cord is plugged in
- To upload the program to the NXT use ”linxt“.

```
# linxt --upload hello_world.rxe
```

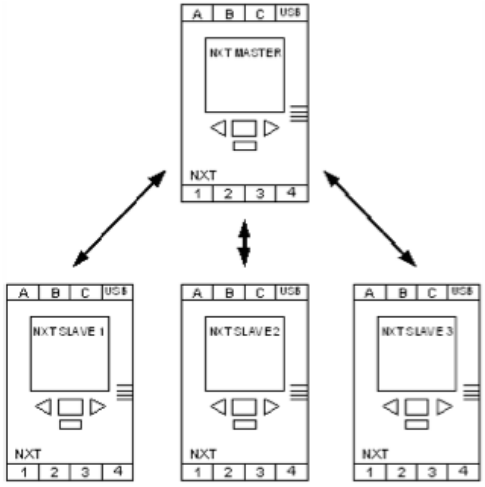
Run program:

- My Files $\rightarrow$ Software files
- Use the arrow buttons to find the ”program_name“ eg. hello_world

Bluetooth on NXT

Bluetooth Functionality:

- Master/slave communication channel
  - 1 master, use channel 0
  - 3 slaves, use channel 1-3



- Communicate to one device at the time
- NXT cannot be a master and slave at the same time
- Turn Bluetooth On/Off
- Search and connect to other Bluetooth devices
- Remember previous connections

Bluetooth on Linux

Basic information:

- Linux uses the Bluez protocol stack.
- A Bluetooth connection uses a socket.
- The Bluetooth socket programing is very similar to TCP/IP socket programming.

Linux Bluetooth Commands:

- hcitool - configure Bluetooth connections

```
# hcitool scan
Scanning ...
        00:16:53:04:B3:46       NXT
```

- sdptool - control and interrogate SDP servers
- rfcomm - RFCOMM configuration utility

Bluetooth Socket Programming: Initiate connection

On the Linux machine the program will:

- Connect to the NXT
- Display which address it connected to

Compiling:

- Use gcc
- Specify the libraries to use
  - -lm : Math library
  - -lbluetooth : Bluetooth library
- Complete command:

```
# gcc -lm -bluetooth nxt_bt_connect.c -o nxt_bt_connect
```

Bluetooth Socket Programming: Initiate connection

```
#include <stdio.h>
// Socket, used for Bluetooth socket
#include <sys/socket.h>
#include <sys/types.h>

// Bluetooth headers
#include <bluetooth/bluetooth.h>
#include <bluetooth/rfcomm.h>

// Global Variables
int nxtSocket;
int init_bluetooth(char *btAddress)     {
        struct sockaddr_rc addr={0};
        int status;

        /*-------------------------------------------------------
         * SOCK_STREAM
         *       Provides sequenced, reliable, two-way, connection-based
         *        byte streams. An out-of-band data transmission
         *        mechanism may be supported.
         *-------------------------------------------------------*/

        // Allocate a socket
        nxtSocket = socket(AF_BLUETOOTH, SOCK_STREAM, BTPROTO_RFCOMM);

        // Set what type and who to connect to
        addr.rc_family = AF_BLUETOOTH;
        addr.rc_channel = (uint8_t) 1;
        str2ba(btAddress, &addr.rc_bdaddr);

// Connect to the NXT
        status = connect(nxtSocket, (struct sockaddr *)&addr, sizeof(addr) );

        if (status < 0) {
                perror("Error connecting Bluetooth");
                return status;
        }
        return 0;
}
```

Bluetooth Socket Programming: main.c

```
int main (void) {
        // nxt brick alpha bluetooth address
        char btaddress[18] = "00:16:53:01:2c:84";

        // initiate bluetooth connection
        if (init_bluetooth(btaddress) < 0)      {
                close(nxtsocket);
                return 1;
        }
        printf("bluetooth connected to %s \n", btaddress);

        close(nxtsocket);
        return 0;
}
```

Linux to NXT via Bluetooth

Linux machine will act as the master and the NXT as the slave

Basic setup, pairing the devices:

- Turn on NXT
- Run program on Linux that connects via Bluetooth
- A bluez window should pop up, click on it
- Enter in the default code, 1234
- The NXT will beep and want the same code (passkey)

NXT Bluetooth Documentation

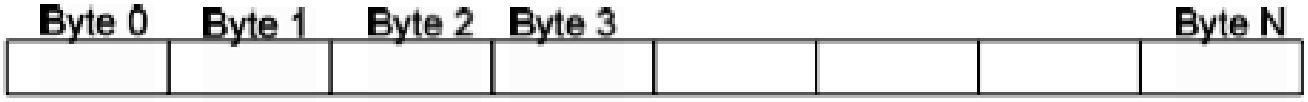
Get the documentation from:
`http://mindstorms.lego.com/Overview/NXTreme.aspx`

Download the ”Bluetooth Developer Kit“
It should contain 4 pdf documents:

- NXT Bluetooth Developer Kit
- Appendix 1 - Communication protocol
- Appendix 2 - Direct Commands
- ARM7 Bluetooth Interface specification

NXT Direct Commands

Direct Commands can be sent via USB or Bluetooth
A telegram is sent to the NXT



Byte 0, Telegram Type

- 0x00: Direct Command, response required
- 0x01: System Command, response required
- 0x02: Reply telegram
- 0x80: Direct Command, no response
- 0x81: System Command, no response

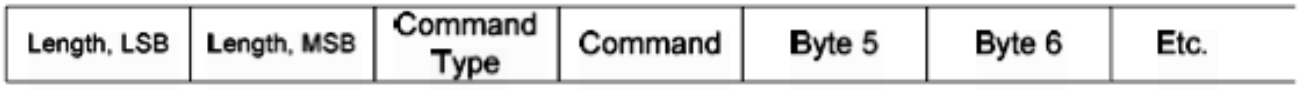
Byte 1-N, Command or Reply

- This depends on the telegram type

The maximum command telegram size is 64 bytes

NXT Direct Commands: Bluetooth

Bluetooth messages have 2 bytes in the front of the telegram



The Length is divide into LSB and MSB

Length, LSB

- This is the length of the telegram

Length, MSB

- This should always be 0x00, since max size is 64 bytes

Direct command: get battery level

On the Linux machine the program will:

- Connect to the NXT (use previously mentioned code)
- Send a direct command to the NXT for the battery level
- Display the battery Level percentage

Direct command: get battery level (part 1)

```
/*************************************************************
 * nxt get battery level
 *      this will get the battery level on the nxt
 * returns: the battery level as an integer
 *************************************************************/
int nxt_getbattery(void)        {
        /*-----------------------------------------------------
         * direct command format:
         *  {length/lsb, length/msb, byte 0, byte 1... byte n}
         *
         * for getbatterylevel (see direct commands):
         *      byte 0: 0x00
         *      byte 1: 0x0b
         *      length/lsb: 0x02, the command length is 2 bytes
         *-----------------------------------------------------*/
        char cmd[4]={0x02, 0x00, 0x00, 0x0b};
        char reply[max_message_size];
        int result;
        int blevel;
        int replylength;
        int error = 0;

        //- send request ---------------------------------------

        if ( (result = write(nxtsocket, cmd, 4)) < 0 )  {
                perror("error sending getbatterylevel command ");
                return result;
        }
```

Direct command: get battery level (part 2)

```
//- read reply ----------------------------------------
// get bluetooth message length
if ( (result = read(nxtsocket, reply, 2)) < 0)  {
        perror("error receiving getbatterylevel reply ");
        return result;
}
replylength = reply[0] + (reply[1] * 256);

// get return package
if ( (result = read(nxtsocket, reply, replylength)) < 0)        {
        perror("error receiving getbatterylevel reply ");
        return result;
}

// quick check to make sure we got everything
if (replylength != result)      {
        fprintf(stderr,
          "getbatterylevel : lengths do not match : %d != %d\n",
          replylength, result);
}
```

Direct command: get battery level (part 3)

```
        /*-------------------------------------------------------
         * return package format:
         *      {length/lsb, length/msb, byte0, byte1..., byten}
         * for getbatterylevel:
         *      byte0: 0x02
         *      byte1: 0x0b
         *      byte2: status byte
         *      byte3-4: voltage in millivolts (uword)
         *      length/lsb: 0x05
         *-------------------------------------------------------*/
        // byte 0
        if (reply[0] != 0x02)   {
                fprintf(stderr, "getbatterylevel : byte 0 : %hhx != 0x02\n", reply[0]);
                error = 1;
        }
        // byte 1
        if (reply[1] != 0x13)   {
                fprintf(stderr, "getbatterylevel : byte 1 : %hhx != 0x13\n", reply[1]);
                error = 1;
        }
        // byte 2
        if (reply[2] != 0x00)   {
                fprintf(stderr, "getbatterylevel : byte 2, status : %hhx \n", reply[2]);
                error = 1;
        }
        if (error)      {
                return -1;
        }

        // byte 3-4
        blevel = reply[5] + (reply[6] * 256);
        return blevel;
}
```

Direct command: main.c

```
int main (void) {
        // nxt brick alpha bluetooth address
        char btaddress[18] = "00:16:53:01:2c:84";

        // initiate bluetooth connection
        if (init_bluetooth(btaddress) < 0)      {
                close(nxtsocket);
                return 1;
        }
        printf("bluetooth connected to %s \n", btaddress);

        // get battery level (direct command)
        blevel = nxt_getbattery();
        if (blevel < 0) {
                close(nxtsocket);
                return 1;
        }
        printf("battery level: %.2f\n", blevel/100.00);

        close(nxtsocket);
        return 0;
}
```

NXT Mailboxes

Mailbox Information:

- NXT has 10 (0-9) mailboxes
  The NXC documentation may refer to them as queues
- NCX documentation says there are 5 mailboxes in a circular queue, each can hold 58 bytes

Master/Slave Communication:

- The master can send a message to a specific mailbox using direct commands
- The slave can read a messages from a specified mailbox
- The slave can write a message for the master to read
  This uses mailbox_number + 10
- The master can send a direct command to read this mailbox on the slave

Sample Mailbox Programs

Sample mailbox program on the Linux machine will:

- Connect to the NXT (use previously mentioned code)
- Send a message to the NXT mailbox 3 and verify the reply
- Send a message to read the NXT reply for mailbox 3
- Display the reply

Sample mailbox program on the NXT will:

- Display a start up message
- Start a loop
  - Display the loop number
  - Try and read mailbox 3, display the return value and message string

Mailbox: mboxdisplay.nxc

```
#define pixels_per_char 6

task main()     {
        int loopnum = 0;

        // initial display
        clearscreen();
        textout(0, lcd_line1, "display mailbox 3");
        wait(1000);

        // we will manually cancel the program
        while(1)        {
                string buffer, message="";
                char result;

                // display loop number
                clearscreen()
                textout(0, lcd_line1, "loop number:");
                numout( (strlen("loop number:") * pixels_per_char), lcd_line1, loopnum);

                // read mailbox 3
                result = receivemessage(3, true, buffer);
                numout(0, lcd_line2, result);
                if (result == 0)        {
                        message = buffer;

                        // send message back to master
                        sendmessage(3+10, message)
                }
                textout(0, lcd_line3, message);

                wait(500);
                loopnum++;
        }
}
```

Mailbox: nxt_sendmessage (part 1)

```
int nxt_sendmessage(int mbox, char *message)    {
        unsigned char btlength[2]={0x00,0x00};
        unsigned char cmd[max_message_size]={0x0};
        unsigned char reply[max_message_size]={0x0};
        int result, msgsize, replylength;
        int error = 0;

        //- send request ----------------------------------------
        /*-------------------------------------------------
         * direct command format:
         *  {length/lsb, length/msb, byte 0, byte 1 ... , byte n}
         *
         * for messagewrite (see direct commands):
         *      byte 0: 0x00 or 0x80
         *      byte 1: 0x09
         *      byte 2: inbox number (0-9)
         *      byte 3: message size
         *      byte 4-n: message data, where n = message size + 3
         *      length/lsb: message size + 4
         *
         *      max message size: 59 (max packet length is 64 bytes)
         *-------------------------------------------------*/

        //create the messagewrite command
        msgsize = strlen(message) + 1; // add one for null termination
        if (msgsize > (max_message_size - 4) )  {
                fprintf(stderr, "messagewrite : message is to long");
                return -1;
        }

        // nxt command
        cmd[0] = 0x00;
        cmd[1] = 0x09;
        cmd[2] = mbox;
        cmd[3] = msgsize;
        memcpy(&cmd[4], message, msgsize);

        // bluetooth length
        btlength[0]= 4 + msgsize;
```

Mailbox: nxt_sendmessage (part 2)

```
// send bluetooth length
if ( (result = write(nxtsocket, btlength, 2)) < 0)      {
        perror("error sending messagewrite command ");
        return result;
}

// send command
if ( (result = write(nxtsocket, cmd, btlength[0])) < 0) {
        perror("error sending messagewrite command ");
        return result;
}

//- read reply ----------------------------------------

// get bluetooth message length
if ( (result = read(nxtsocket, reply, 2)) < 0)  {
        perror("error receiving messagewrite reply ");
        return result;
}
replylength = reply[0] + (reply[1] * 256);

// get return package
if ( (result = read(nxtsocket, reply, replylength)) < 0)        {
        perror("error receiving messagewrite reply ");
        return result;
}

// quick check to make sure we got everything
if (replylength != result)      {
        fprintf(stderr,
                        "messagewrite : lengths do not match : %d != %d\n",
                        replylength, result);
}
```

Mailbox: nxt_sendmessage (part 3)

```
        /*-------------------------------------------------
        * return package for messagewrite
        * byte 0: 0x02
        * byte 1: 0x09
        * byte 2: status byte
        * byte 3: local inbox number (0-9), should match request
        * byte 4: message size
        * byte 5-63: message data (padded)
        *-------------------------------------------------*/
       // byte 0
       if (reply[0] != 0x02)   {
              fprintf(stderr, "messagewrite : byte 0 : %hhx != 0x02\n", reply[0]);
              error = 1;
       }
       // byte 1
       if (reply[1] != 0x09)   {
              fprintf(stderr, "messagewrite : byte 1 : %hhx != 0x13\n", reply[1]);
              error = 1;
       }
       // byte 2
       if (reply[2] != 0x00)   {
              fprintf(stderr, "messagewrite : byte 2, status : %hhx \n", reply[2]);
              error = 1;
       }

       if (error)      {
              return -1;
       }

       return 0;
}
```

Mailbox: nxt_readmessage (part 1)

```
int nxt_readmessage(int mbox, char **message)   {
        unsigned char btlength[2]={0x00,0x00};
        unsigned char cmd[5]={0x0};
        unsigned char reply[max_message_size];
        int result, cmdlength, msgsize;
        int error = 0;

        //- send request ----------------------------------------
        /*-------------------------------------------------------
         * direct command format:
         *  {length/lsb, length/msb, byte0, byte1 ... , byten}
         *
         * for messageread (see direct commands):
         *      byte 0: 0x00 or 0x80
         *      byte 1: 0x13
         *      byte 2: remote inbox number (0-9)
         *      byte 3: local inbox number (0-9)
         *      byte 4: remove?
         *      length/lsb: 5
         *-------------------------------------------------------*/

        // nxt command
        cmd[0] = 0x00;
        cmd[1] = 0x13;
        cmd[2] = mbox+10;
        cmd[3] = 0x00;
        cmd[4] = 0x01;

        // bluetooth message length
        btlength[0] = 5;

        // send bluetooth length
        if ( (result = write(nxtsocket, btlength, 2)) < 0)      {
                perror("error sending messageread command ");
                return result;
        }

        // send command
        if ( (result = write(nxtsocket, cmd, 5)) < 0)   {
                perror("error sending messageread command ");
                return result;
        }
```

Mailbox: nxt_readmessage (part 2)

```
//- read reply ---------------------------------------
message = null;

// get bluetooth message length
if ( (result = read(nxtsocket, reply, 2)) < 0)  {
        perror("error receiving messageread reply ");
        return result;
}
cmdlength = reply[0] + (reply[1] * 256);

// get return package
if ( (result = read(nxtsocket, reply, cmdlength)) < 0)  {
        perror("error recieveing messageread reply ");
        return result;
}

// quick check to make sure we got everything
if (cmdlength != result)        {
        fprintf(stderr,
            "messageread : lengths do not match : %d != %d\n",
            cmdlength, result);
}
```

Mailbox: nxt_readmessage (part 3)

```
        /*-------------------------------------------------------
         * return package for messageread
         * byte 0: 0x02
         * byte 1: 0x13
         * byte 2: status byte
         * byte 3: local inbox number (0-9), should match request
         * byte 4: message size
         * byte 5-63: message data (padded)
         *-------------------------------------------------------*/
        if (reply[0] != 0x02)   {       // byte 0
                fprintf(stderr, "messageread : byte 0 : %hhx != 0x02\n", reply[0]);
                error = 1;
        }
        if (reply[1] != 0x13)   {       // byte 1
                fprintf(stderr, "messageread : byte 1 : %hhx != 0x13\n", reply[1]);
                error = 1;
        }
        if (reply[2] == 0x40)   {       // byte 2
                printf("mailbox empty");
                return reply[2];
        }
        if (reply[2] != 0x00)   {
                fprintf(stderr, "messageread : byte 2, status : %hhx \n", reply[2]);
                error = 1;
        }
        if (reply[3] != 0x00)   {       // byte 3
                fprintf(stderr, "messageread : byte 3, mbox: %hhx != 0x00\n", reply[3]);
                error = 1;
        }
        if (error)      { return -1; }
        // byte 4
        msgsize = reply[4];
        *message = (char *)malloc(sizeof(char) * msgsize + 2);
        printf("reply[5-%d]: %s\n", msgsize+6, &reply[5]);

        // byte 5-63: message data
        memcpy(*message, &reply[5], msgsize);
        printf("message: %s\n", *message);

        return 0;
}
```

Mailbox: main.c

```
int main (void) {
        // nxt brick alpha bluetooth address
        char btaddress[18] = "00:16:53:01:2c:84";

        // initiate bluetooth connection
        if (init_bluetooth(btaddress) < 0)      {
                close(nxtsocket);
                return 1;
        }
        printf("bluetooth connected to %s \n", btaddress);

        // Send Message
        nxt_sendMessage(3, "world");

        // Read Message
        while ( (status = nxt_readMessage(3, &reply)) == 0X40);
        if (status < 0) {
                return 1;
        }
        printf("Reply: %s\n", reply);
        free(reply);

        close(nxtsocket);
        return 0;
}
```