Hands-on Lab

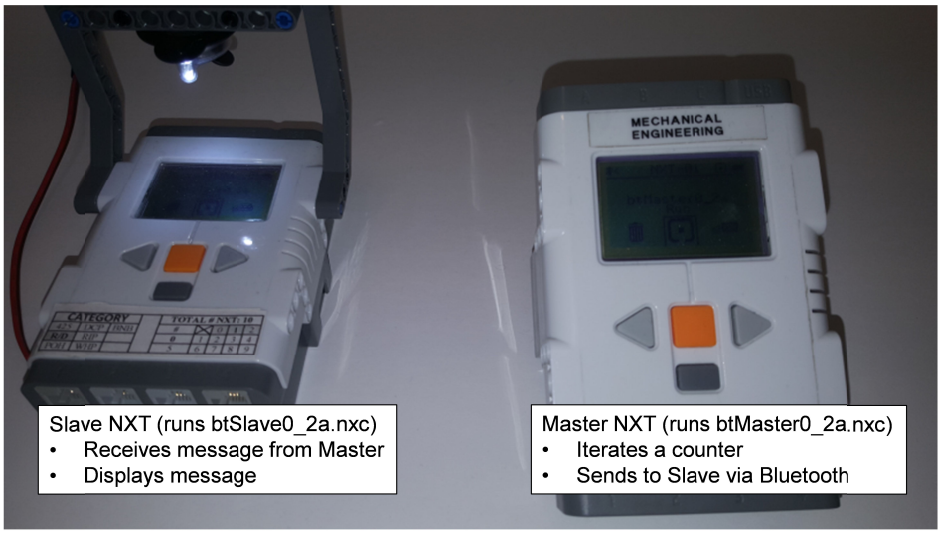
XL-320 NXC Programming – Bluetooth

NXC programs are introduced to have two NXT Bricks communicate via Bluetooth. This is useful because it enables distributed computing. For example, a motor could be attached to a Slave Brick that would receive messages. A sensor might be attached to a Master Brick that would then send messages containing desired motor speeds. Distributing task amongst multiple Bricks relieves computational expense.

Preliminary: Enable Bluetooth on the NXT Brick

A YouTube search yields many videos on establishing NXT Brick Bluetooth connections. One example is https://youtu.be/CN3iXGsK9YM. To customize the Brick's name, use BrixCC's `Tools – Diagnostics` and in the pop-up box, edits the `Name` field.

Concept 1: Master sending Bluetooth Messages to Slave



Concept: Two NXT Bricks; left is Slave and right is Master. YouTube demonstration: https://youtu.be/s9aWIpGIYZk

Step 1: Write, compile and download the Slave NXC program **`btSlave0_2a.nxc`**

Figure 1A shows the full NXC program to be run on the Slave NXT. The header file **`protocol0_2a.h`** was authored by Daniele Benedettelli, a famed Lego developer and author. This H-file has functions to send or receive messages between the Master and the Slave NXTs; these use NXC's Bluetooth functions like `BluetoothWrite` and `ReceiveMessage`. Also, the H-file has error checking and wait-states and employs NXC's functions like `btchannelcheck`, `btwaitfor`, and `BluetoothStatus`. The goal of this concept is simply to pass messages from the Master to the Slave. So, an in-depth discussion of the H-file will not be explored here.

The program begins by a call to `slavecheck()` to check on the Bluetooth connection. By design, the H-file defines the Slave and Master channels are 1 and 0 respectively and the mailbox for Bluetooth messages is set to 0.

Next an endless `for` loop is entered. Here, `receivefrommaster` is called. Any data in the mailbox is then stored in the string variable `stringFromMaster`. The number of characters in that string is also stored in variable `j`.

```
// FILE: btSlave0_2a.nxc - Works!
// DATE: 02/24/20 14:47
// AUTH: P.Oh
// DESC: Read message from Master and display it
//       Message contains a number (as string).  Perform math on that number
// REFS: Works with btMaster0_1a.nxc

#include "protocol0_2a.h"

task main() {

  string stringFromMaster; // store string from Master
  int j;                   // store length value of received string
  int intR, mathResult;    // int form of string and math performed on that number

  slavecheck();  // initialize NXT running this program as the Slave
  TextOut(0, LCD_LINE1, "Slave" );

  for(;;) {
    stringFromMaster = receivefrommaster();
    j = StrLen(stringFromMaster);

    // -- print to screen only if there is a message
    if(j!=0) {
      TextOut(0, LCD_LINE3, stringFromMaster);
    };

    intR = StrToNum(stringFromMaster); // Master's message contains a number, so convert it
    mathResult = 10*intR;              // Perform simple math to prove it's a number
    // TextOut(0, LCD_LINE4, FormatNum("math = %5d" , mathResult));
    NumOut(0, LCD_LINE4, mathResult);

    Wait(500); // min is 10 msec, but 500 msec makes easier to see on Brick
    ResetSleepTimer(); // don't time out and shut off Brick
  } // end for
} // end main
```

Figure 1A: Listing of **`btSlave0_2a.nxc`**

As will be shown in Step 2, the Master will send messages containing *numerical characters*. One observes the line `intR = StrToNum(stringFromMaster)`. The purpose is to convert the received string to *numerical values*. The Slave Brick will display the product of the number and 10. Before looping back, the program calls the `Wait` function. The value of 500 milliseconds helps to see what is displayed on the Brick before the next iteration.

Step 2: Write, compile and download the Master NXC program **`btMaster0_2a.nxc`**

Similar to the Slave program, **Figure 1B** shows the NXC code for the Master. After checking the Bluetooth connection with a call to `mastercheck`, an endless `for` loop is entered.

```
for(;;) {
    stringFromSlave = receivefromslave(); // read message (if any) from slave
    i++; // i will be the number Master wishes to send
    strI = NumToStr(i); // must convert numbers into string

    NumOut(0, LCD_LINE2, i); // Row 2 displays actual number
    TextOut(0, LCD_LINE3, strI); // Row 3 displays string version of number
    sendtoslave(strI); // Master sends string to Slave
```

```
    Wait(500); // min is 10 msec.  But wish to view the string on Brick
    ResetSleepTimer(); // keep Brick from sleeping and turning off Bluetooth connection

  } // end for
```

In this loop, a counter called `i` is incremented and then converted to a string. This string is displayed on the Master Brick and then `sendtoslave(strI)` sends this string via Bluetooth, to the Slave Brick.

```
// FILE: btMaster0_2a.nxc - Works!
// DATE: 02/24/20 14:01
// AUTH: P.Oh
// DESC: Master sends message to Slave; message displayed on Slave
// VERS: Clean up btMaster0_1a.nxc
// REFS: Works with btSlave0_2a.nxc

#include "protocol0_2a.h"
#define NAP 10  // milliseconds

task main() {

  string stringFromSlave;  // any messages from slave
  int i;                   // index
  string strI;             // string version of index

  TextOut(0, LCD_LINE1, "Master" );
  mastercheck(); // check Master bluetooth connection

  for(;;) {
    stringFromSlave = receivefromslave(); // read message (if any) from slave
    i++; // i will be the number Master wishes to send
    strI = NumToStr(i); // must convert numbers into string

    NumOut(0, LCD_LINE2, i); // Row 2 displays actual number
    TextOut(0, LCD_LINE3, strI); // Row 3 displays string version of number
    sendtoslave(strI); // Master sends string to Slave

    Wait(500); // min is 10 msec.  But wish to view the string on Brick
    ResetSleepTimer(); // keep Brick from sleeping and turning off Bluetooth connection

  } // end for
} // end main
```

Figure 1B continued: Listing for **btMaster0_2a.nxc**

Congratulations! Your Master NXT Brick can send strings via Bluetooth to a Slave NXC Brick.

Exercises

1-1. Write NXC programs to detect a Master's button push states as follows. Pushing the Master's left or right arrow buttons sends via Bluetooth, a 1 or 2 respectively. The Slave receives these numbers and displays on its LCD screen the messages "Left" or "Right" respectively.

Concept 2: Serial and Bluetooth Messages

Preamble: Reference Serial Communications Lab

Recall, the NXT Brick's Port 4 is cable of RS-485 serial communications. In a previous lab, a PC ran a terminal emulator. A USB-to-RS485 module and modified NXT cable physically connected the PC's USB port to the NXT's Port 4. With that setup, the PC transmitted messages and the NXT would receive and display them.

In this Concept, serial communications extends Concept 1: the Master NXT will receive serial messages from the PC and then wirelessly transmit them, via Bluetooth, to the Slave NXT (see **Figure 2A**).

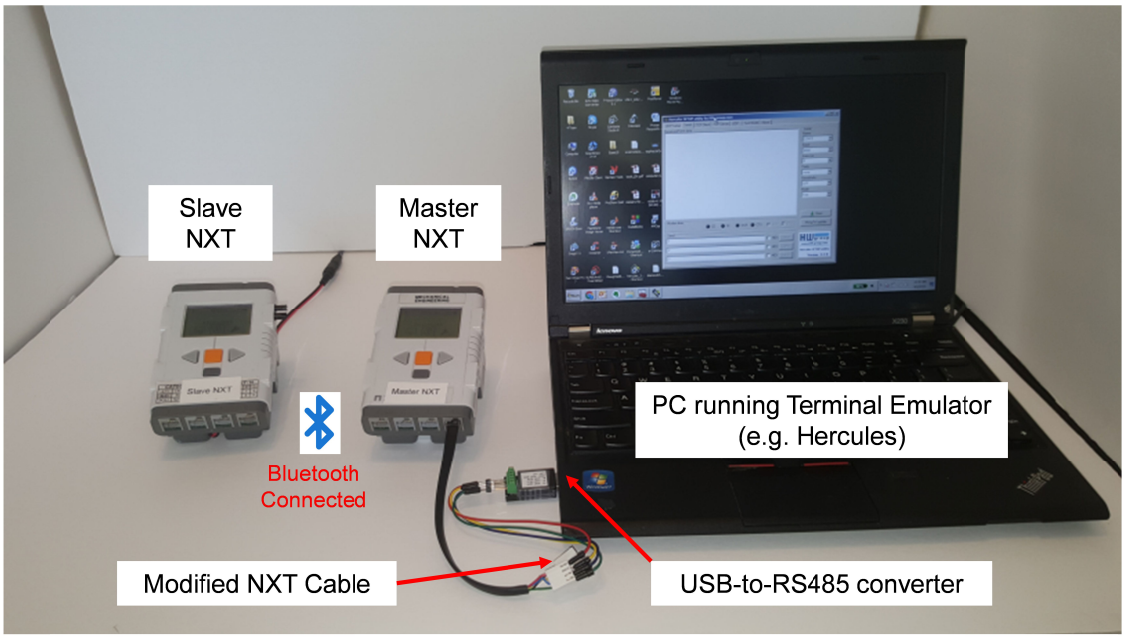


Figure 2A: Messages from PC are serially transmitted to the Master NXT. The Master NXT then wirelessly transmits them to the Slave NXT via Bluetooth. Video demonstration https://youtu.be/P3qDNDvtpu4

Step 1: Serially connect Master NXT to PC

Recall the Serial Communications Lab where one wrote `nxtReadFromPC1_0b.nxc`. Connect the USB-to-RS485 module into the PC. Use Windows Device Manager to identify the COM port. Set this port for 4800 baud. Verify that `nxtReadFromPC1_0b.nxc` works; messages from Hercules are displayed on the NXT Brick's LCD.

Step 2: Master: Combine Bluetooth and Serial Communication **`btAndSerialMaster0_1b.nxc`**

Figure 2B runs on the Master NXT. It is similar to prior code (e.g. **Figure 1B**) with the header-file for Bluetooth functions, Bluetooth related variables and `sendtoslave` function call.

Figure 2B also shares code from `nxtReadFromPC1_0b.nxc`. As yellow-highlighted, serial port related variables are defined and the serial port is enabled at 4800 baud.

The endless `while-loop` calls `RS485DataAvailable` to monitor the serial port for activity and then reads any (ASCII) messages. These are stored in string variable `charsRead` and displayed on the NXT LCD. Then, this string is send via Bluetooth with a call to `sendtoslave`.

```
// FILE: btAndSerialMaster0_1b.nxc - Works!
// DATE: 04/01/20 09:35
// AUTH: P.Oh
// DESC: Master receives serial message from PC.  Master creates Bluetooth
//       version of message and transmits to Slave
// VERS: 0_1a: prototyping
//       0_1b: Display string more nicely
// REFS: Works with btSlave0_2a.nxc.  btMaster0_2a.nxc and nxtReadFromPC1_0b.nxc

#include "protocol0_2a.h"

task main() {

  // Serial port related variables
  byte readBuffer[];          // array to store bytes received from PC
  string charsRead;           // string of ASCII characters read from PC

  // Bluetooth related variables
  string stringFromSlave;     // any messages from slave
  int i;                      // index
  string strI;                // string version of index

  // Set up Master NXT's Bluetooth
  TextOut(0, LCD_LINE1, "Master" );
  mastercheck(); // check Master bluetooth connection

  // Set up Master NXT's serial port
  UseRS485();                              // (1) Configure S4 for RS-485
  RS485Enable();                           // (2) Activate RS-485
  RS485Uart(HS_BAUD_4800, HS_MODE_DEFAULT); // (3) Baud 112500 and default parity
  Wait(MS_1);                              // (4) Wait briefly for port settings to be ready

  readBuffer = 0;
  while(true) { // keep reading and displaying strings received from PC until abort
    while(!RS485DataAvailable()) {
      // if no ASCII chars available, then do nothing
    };
    // Bytes ready, so now display and used them
    RS485Read(readBuffer);
    // ClearScreen();
    TextOut(0, LCD_LINE3, "PC's string" );
    TextOut(0, LCD_LINE4, ByteArrayToStr(readBuffer) );
    charsRead = ByteArrayToStr(readBuffer);
    // Clear buffer
    readBuffer = 0;

    // Send via Bluetooth, the string to Slave
    TextOut(0, LCD_LINE6, "BT message:" );
    TextOut(0, LCD_LINE7, charsRead);
    sendtoslave(charsRead);
    Wait(1000); // Wait 1 sec (same rate as PC)
    ClearLine(LCD_LINE4); // clear line displaying PC's message
    ClearLine(LCD_LINE7); // clear line displaying BT message
    ResetSleepTimer(); // keep Brick from sleeping and turning off Bluetooth connection
  }; // end while(true)

} // end main
```

Figure 2B: **`btAndSerialMaster0_1b.nxc`** is compiled and executed on the Master NXT

Step 3: Slave NXT – **btAndSerialSlave0_1b.nxc**

Figure 2C runs on the Slave NXT. `ClearLine` adds code to **Figure 1A** to display messages nicely on the Slave NXT Brick's LCD. Recall, the Slave NXT simply waits for Bluetooth messages from the Master.

```
// FILE: btAndSerialSlave0_1b.nxc - Works!
// DATE: 04/01/20 09:25
// AUTH: P.Oh
// DESC: Read Bluetooth message from Master and display it
//       NB: Original message will be from PC, sent to Master via Serial port
// VERS: 0_1a: prototyping
//       0_1b: Display received strings more nicely
// REFS: Works with btAndSerialMaster0_1a.nxc; btSlave0_2a.nxc

#include "protocol0_2a.h"

task main() {

  string stringFromMaster; // store string from Master
  int j;                   // store length value of received string
  int intR, mathResult;    // int form of string and math performed on that number

  slavecheck();  // initialize NXT running this program as the Slave
  TextOut(0, LCD_LINE1, "Slave" );
  TextOut(0, LCD_LINE3, "Master's String" );

  for(;;) {
    stringFromMaster = receivefrommaster();
    j = StrLen(stringFromMaster);

    // -- print to screen only if there is a message
    if(j!=0) {
      TextOut(0, LCD_LINE4, stringFromMaster);
    };

    intR = StrToNum(stringFromMaster); // Master's message contains a number, so convert
it
    mathResult = 10*intR;              // Perform simple math to prove it's a number
    TextOut(0, LCD_LINE6, "Math Result" );
    NumOut(0, LCD_LINE7, mathResult);
    Wait(1000); // 1 sec because same rate as Master which is same as PC
    ClearLine(LCD_LINE4); // clear line for string received from Master
    ClearLine(LCD_LINE7); // clear line for math result
    ResetSleepTimer(); // don't time out and shut off Brick
  } // end for
} // end main
```

Figure 2C: **btAndSerialSlave0_1b.nxc** runs on the Slave NXT

It's assumed that the alphanumeric (ASCII) message contains a number. This serves to demonstrate that a StrToNum call can converted that string into numeric form. The variable `mathResult` is used to demonstrate math can be performed on that number (e.g. multiply by 10).

Step 4: Hardware connections and software execution

Set up Master-Slave Bluetooth connections. Recall in Concept 1 that this set up assumes that Mailbox 1 is used as the Bluetooth channel. Once connected, run `btAndSerialSlave0_1b.nxc` on the Slave and `btAndSerialMaster0_1b.nxc` on the Master.

Next, execute a terminal emulator (e.g. Hercules) on the PC. Ensure that the COM and serial port settings (i.e. 4800 baud) are correctly set. Send a numeric character (e.g. 1) from the emulator. This should display on the Master as well as the Slave.

Note, one might have to transmit the numeric character multiple times. This is because `btAndSerialSlave0_1b.nxc` and `btAndSerialMaster0_1b.nxc` each contain 1 second `Wait` statements. This is a rather long period for microprocessors to wait. This delay was used to simply ensure the buffers are transmitted and do not overflow. A more proper way is to invoke serial and Bluetooth function calls for message checking. This process involves: counting the number of characters to be sent; verifying that the message received has these number of characters; and letting the sender know when the receiver is ready to accept the next message. For the purposes of this Concept, the long Wait statement avoids such message checking, albeit with delays.

Congratulations! Your Master NXT Brick can receive strings from a PC serially and send them to a Slave NXC Brick via Bluetooth.

Exercises

Use Concept 2 to have the PC serially transmit a number to the Master NXT. The Master NXT then sends this number via Bluetooth, to the Slave NXT.

2-1. Write NXC code for the Slave such that when the number received is a “1” then the Slave NXT plays a tone. If the number is “0”, the tone stops.

2-2. Connect an XL-320 to the Slave NXT. Write NXC code for the Slave such that when the number is “1” the XL-320 rotates back-and-forth from -90 to +90 degrees. If the number is “0”, the XL-320 stops.