Hands-on Lab

Lego File and Timer Handing - Review

We've explored the Lego NXT's input port and ADC to measure resistances and voltages. This lab will turn the NXT into a data logger. Also, the lab will explore using the output port as a voltage source.

Concept 1 – NXT File Saving Review:

As a quick review, the NXT's ability to save data will be explored.

The program `displaySquareAndSquareRoot1_0.nxc` displayed an integer, its square and square root on the Brick's LCD. This program used the for-loop to iterate the integer from 1 to 10. Building on this example, a program is written to save the values to a file. The file will then be imported into an Excel worksheet. Once one has a worksheet, the data can be manipulated and/or plotted.

Step 1: Click `File – Open` and load `displaySquareAndSquareRoot1_0.nxc.` Click `File – Save As` with the name "`displaySquareAndSquareRoot2_0.nxc`".

Step 2: Define global variables that serve for file handling. Add the following code to above your `task main` routine.

```
// File: displaySquareAndSquareRoot2_0.nxc
// Date: 10/01/12 15:43
// Desc: Display number, its square and square root save to file
// Vers: 2.0
// Refs: displaySquareAndSquareRoot1_0.nxc

// Global variables (for file writing)
unsigned int result; // flag returned when handling files
byte fileHandle; // handle to the data file
short bytesWritten; // number of bytes written to the file
string fileHeader; // column header for data in the file
int fileNumber, filePart; // integers to split up data file names
string fileName; // name of the file
string strFileNumber; // file number e.g myDataFile 1, 2, 3
string strFilePart; // file part e.g. myDataFile1-1, 1-2, 1-3
string text; // string to be written to file i.e. data values

task main ()
```

Step 3: Compose a function to initiate a file. Add the following code above `task main`:

```
string strFilePart; // file part e.g. myDataFile1-1, 1-2, 1-3
string text; // string to be written to file i.e. data values

// Create and initialize a file
void InitWriteToFile() {
  fileNumber = 0;  // set first data file to be zero
  filePart = 0; // set first part of first data file to zero
  fileName = "squareData.csv" ; // name of data file
  result=CreateFile(fileName, 1024, fileHandle);
  // NXT Guide Section 9.100 pg. 1812 and Section 6.59.2.2 pg. 535
  // returns file handle (unsigned int)

  // check if the file already exists
  while (result==LDR_FILEEXISTS) // LDR_FILEEXISTS returns if file pre-exists
     {
      CloseFile(fileHandle);
      fileNumber = fileNumber + 1; // create new file if already exists
      fileName=NumToStr(fileNumber);
      fileName=StrCat("squareData" , fileName, ".csv");
      result=CreateFile(fileName, 1024, fileHandle);
      } // end while

  // play a tone every time a file is created
  PlayTone(TONE_B7, 5);
  fileHeader = "x, x^2, sqrt(x)" ; // header for myData file
  WriteLnString(fileHandle, fileHeader, bytesWritten);
  // NXT Guide Section 6.59.2.43 pg. 554
  // Write string and new line to a file
  // bytesWritten is an unsigned int.  Its value is # of bytes written

} // end InitWriteToFile

task main ()
```

Step 4: Compose a function to write to file. Add the following code above `task main`:

```
} // end InitWriteToFile

void WriteToFile(string strTempText) {
  // strTempText stores the text (i.e. ticks and motorRpm to be written to file
  // write string to file
  result=WriteLnString(fileHandle, strTempText, bytesWritten);
  // if the end of file is reached, close the file and create a new part
  if (result==LDR_EOFEXPECTED) // LDR_EOFEXPECTED is flagged when end-of-file
    { // close the current file
      CloseFile(fileHandle); // NXT Guide Section 6.59.2.1 pg. 535
    // Closes file associated with file handle

    // create the next file name
    filePart = filePart + 1;
    strFileNumber = NumToStr(fileNumber);
    strFilePart = NumToStr(filePart);
    fileName = StrCat("squareData" , strFileNumber,"-", strFilePart ,".csv");

    // delete the file if it exists
    DeleteFile(fileName); // NXT Guide Section 6.59.2.5 pg. 537
    // Delete the file specified by the string input

    // create a new file
    CreateFile(fileName, 1024, fileHandle);
    // play a tone every time a file is created
    PlayTone(TONE_B7, 5);
    WriteLnString(fileHandle, strTempText, bytesWritten);
    } // end if
} // end WriteToFile

task main ()
```

Step 5: Next, compose a function that closes the file. Add the following code above `task main`:

```
} // end WriteToFile

// Close the file
void StopWriteToFile() {
  // close the file
  CloseFile(fileHandle);
} // end StopWriteToFile

task main ()
```

Add this function

At this point, save your NxC program. To recap, Step 2 declared the variables needed for file handling and Steps 3 to 5 created functions to respectively initialize (i.e. create), write string data and close a file.

Step 6: File data is stored as strings. As such, strings must be declared for each integer and float. Also, to create a file, one must initialize one. Add the following within `task main`:

```
task main ()
{
  int x; // integers from 1 to 10
  int xSquared; // square of x
  float xSquareRoot; // square root of x

  string strX;
  string strXSquared;
  string strXSquareRoot;

  // Create a new file that captures time and motor speed
  InitWriteToFile();

  for (x = 1; x <=10; x++) {
    xSquared = x*x;
    xSquareRoot = sqrt(x);
```

Declare string versions of integers and floats. Also, create a file.

Step 7: In the for-loop, the program iterates from 1 to 10, calculating the square and square root. We can use the FormatNum function to create a string version of numbers (i.e. integers and floats). Add the following within the `for-loop`:

```
    TextOut (10, LCD_LINE4, FormatNum("x = %d" , x));
    TextOut (10, LCD_LINE5, FormatNum("xSquared = %d" , xSquared));
    TextOut (10, LCD_LINE6, FormatNum("sqrt(x) = %3.3f" , xSquareRoot));
    Wait (SEC_2);

    // Create string version of calculated values
    strX = FormatNum("%d" , x);
    strXSquared = FormatNum("%d" , xSquared);
    strXSquareRoot = FormatNum("%3.3f" , xSquareRoot);

  } // end of for loop


} // end of main
```

FormatNum is akin to ANSI-C's sprintf() function. It creates strings from numbers.

Step 8: Finally, one should write the 3 strings (strX, strXSquared and strXSquareRoot) to the file. To do so efficiently, one can employ the ANSI-C strcat function which concatenates multiple strings into a single one. Finally, write the string to file. Add the following code within the for-loop

```
// Create string version of calculated values
    strX = FormatNum("%d" , x);
    strXSquared = FormatNum("%d" , xSquared);
    strXSquareRoot = FormatNum("%3.3f" , xSquareRoot);

    // Concatenate the 3 strings into a single one.
    // Write resulting string to file. The text will be end with a EOL
    text=StrCat(strX, "," , strXSquared, "," , strXSquareRoot, "," );
    WriteToFile(text);

  } // end of for loop

} // end of main
```

Use strcat to combine strings. Write resulting string to file

Step 9: After the program has generated the data (i.e. completed the for-loop), one terminates the program gracefully by closing the file. One can also add an LCD message and beep to let the user know the program is done. Add the following after the for-loop and before the end of main.

```
  // Concatenate the 3 strings into a single one.
    // Write resulting string to file. The text will be end with a EOL
    text=StrCat(strX, "," , strXSquared, "," , strXSquareRoot, "," );
    WriteToFile(text);
  } // end of for loop

  // Finished computing square and square root, so clean up and quit
  ClearScreen();
  TextOut(0, LCD_LINE2, "Quitting", false);
  StopWriteToFile();
  PlaySound(SOUND_LOW_BEEP);  // Beep to signal quitting
  Wait(SEC_2);

} // end of main
```

Add this alert user of termination and close file

Step 10: Save, compile and execute the resulting program. The program should iterate from 1 to 10, displaying the integers, its square and square root. Additionally, in the background, the Brick stores the data to file named: `squareData.csv`.

To view this data file, after the program completes, select Tools – NXT Explorer (see **Figure 1A**). A pop-up box should display the files stored within your NXT Brick (as shown **in Figure 1B**). Click-and-drag the file `squareData.csv` from the left pane (i.e. Brick's directory) to the right one (your PC's drive).
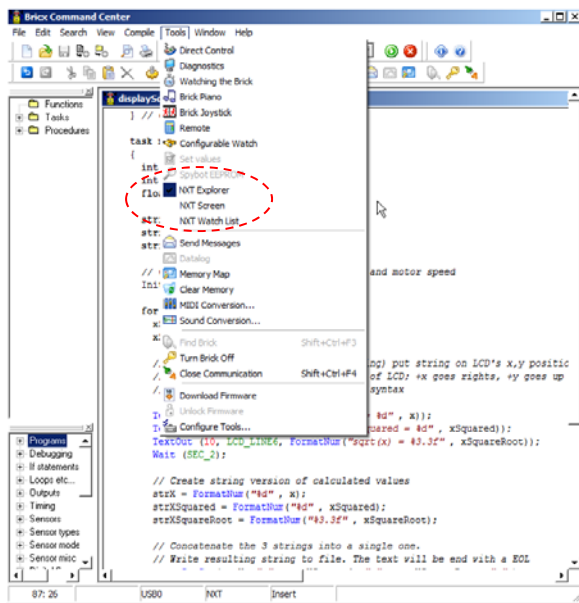

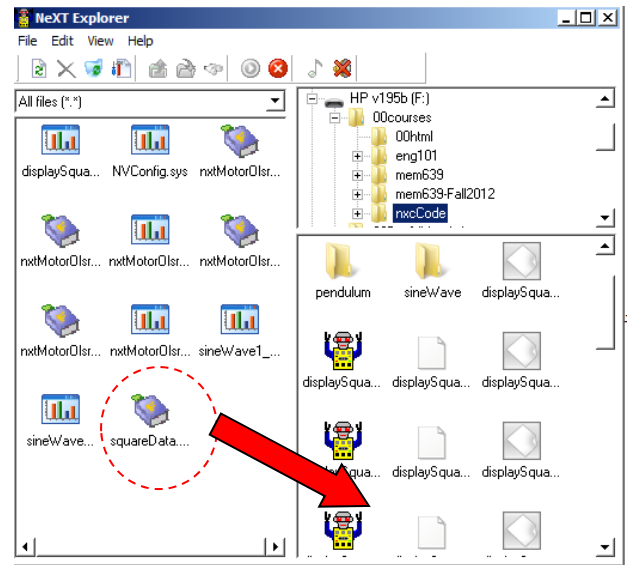**Figure 1A:** Launch the NXT Explorer to view Brick's files

Figure 1B: Click-and-drag the data file `squareData.csv` to your PC.

Step 11: Double-click on the version of `squareData.csv` that is saved on your PC. Excel should already be configured to open CSV (comma-separated files), resulting in **Figure 1C.** **Figure 1D** shows the resulting scatter plot of the first 2 columns.
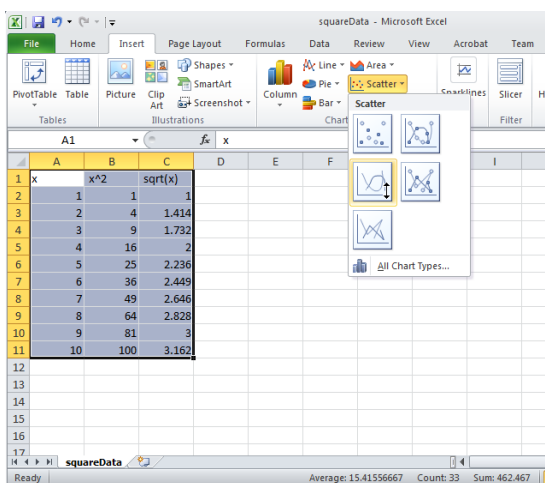

Figure 1C: Excel opens the resulting squareData.csv file. One can then select data for a scatter plot.
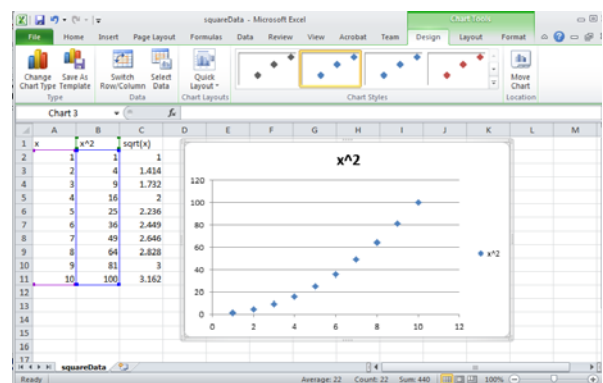

Figure 1D: Scatter plot of first 2 columns of data reveal the expected parabolic curve resulting from computing the square of values.

© Copyright Paul Oh

Code Explanation: `displaySquareAndSquareRoot2_0.nxc` iterates from 1 to 10 using a for-loop. Within this loop, the square and square root is also computed. To save any values to a file, one must first declare (Step 2) and initialize (Step 3) one. File data is stored as strings (i.e. a collection of alphanumeric characters). As such, string versions of any computation are needed and the strcat function is used (Steps 6 and 7) along with the file writing function created in Step 4. After computations are finished (i.e. for-loop terminates), the file should be closed (Step 9) using the function created in Step 5. Steps 10 and 11 show the instructions for using NXT Explorer within the BricxCC IDE to export any files saved on the Brick's memory, to one's PC.

Exercise 1: In NxC create programs for the following:

1-1: Iterate integers from -10 to +10 incrementally by 1. Compute the square and cube and save to a file named "squareAndCube.csv". Export the data file and plot the curves in Excel.

1-2: Capture all file handling functions into a header file named `fileSavingFunctions.h`. Rewrite a new program called `displaySquareAndSquareRoot3_0.nxc` that includes this header file. This new program should run like `displaySquareAndSquareRoot2_0.nxc` – and just serves as a sanity check that file saving works.

1-3: Prove that you know how to name data files and save data in desired formats, and can import these into Excel.

Concept 2 – NXT Timing Review:

As a quick review, the NXT's timer capabilities will be explored. Here, one defines a sampling frequency. A calculation is performed and the result and elapsed time will be recorded to a file.

Step 1: Open BricxCC, click `File - New` and save your program as `timer1_0.nxc`. Write the following code.

```
// FILE: timer1_0.nxc - Works!
// DATE: 08/26/16 12:42
// AUTH: P.Oh
// DESC: Basic timer and file saving demonstration
//       User specifies sampling time and desired maximum elapsed time;
//       calculate and write calculated value to file at desired sampling time

#include "fileSavingFunctionsForTimer1_0.h"

task main() {

  // Boolean related variables - Brick buttons to start/stop execution
  bool orangeButtonPushed, greyButtonPushed;

  // Timing related variables
  long prevTick, curTick, deltaTick; // previous, current and difference in ticks
  string strDeltaTick; // string form of deltaTick for file writing
  float deltaTickInSeconds;  // NB: deltaTick in [msec]
  float elapsedTimeInSeconds; // elapsed time in [sec]
  string strElapsedTimeInSeconds; // string form of elapsed time for file writing
  float maxElapsedTimeInSeconds; // total time in [sec] for data acquisition
  float samplingTimeInSeconds;
  float waitingTimeInSeconds; // for delay loop
  long  waitingTimeInMilliSeconds;
  float epsilonTime = 0.0001; // maxElapsedTimeInSeconds - elapsedTimeInSeconds will be almost zero

  // Calculation related variables
  int x;
  float xSquareRoot;
  string strXSquareRoot; // string form of xSquareRoot for file-writing

  // Create a new file to capture values... call the write-to-file function
  InitWriteToFile();
```

Code description: `task main` begins by declaring button and timer related variables. Also, time related variables are declared. As one will encounter later, the NxC function `CurrentTick()` will be used to poll the Brick's current clock (called a tick counter). Like a stopwatch, variables `curTick` and `prevTick` are used to calculate the time that has elapsed and store the resulting difference in the variable `elapsedTimeInSeconds`. File writing uses alphanumeric values so numeric variables are also declared strings like `strElapsedTimeInSeconds`.

Step 2: Continue adding code to initiate the timer in `timer1_0.nxc`

```
   // Prompt user to begin
   TextOut (0, LCD_LINE1, "Orange Btn starts");
   do {
      orangeButtonPushed = ButtonPressed(BTNCENTER, FALSE);
   } while(!orangeButtonPushed);

   ClearScreen();
   TextOut(0, LCD_LINE1, "Grey Btn Stops");

   // Initialize timing- and calculation-related variables
   samplingTimeInSeconds = 0.1; // sampling time in [sec]
   maxElapsedTimeInSeconds = 10.0; // total time of data acquisition
   elapsedTimeInSeconds = 0.0; // set elapsed time to zero
   prevTick = CurrentTick();
   x = 0;
```

Code description: This code segment prompts the user to push the NXT's orange button. Timing is defined with sampling time (0.1 sec) with `samplingTimeInSeconds` and the maximum time for the program to run (10 sec) with maxElapsedTimeInSeconds. Before computing (in Step 3), the elapsed time is set to 0.0 sec with `elapsedTimeInSeconds`. The timer is then initiated with a call to `CurrentTick()`.

```
  do {
    greyButtonPushed = ButtonPressed(BTNEXIT, FALSE);
    TextOut(0, LCD_LINE6, FormatNum("Time = %5.3f s" , elapsedTimeInSeconds));

    // perform desired work i.e. square root of values
    x = x + 1;
    xSquareRoot = sqrt(x);
    strXSquareRoot = FormatNum("%5.2f" , xSquareRoot);

    // check how much time to wait for sampling interval to elapse
    curTick = CurrentTick(); // get current tick count
    deltaTick = curTick - prevTick; // measure elapsed ticks [msec]
    deltaTickInSeconds = deltaTick/1000.0; // in [sec]
    waitingTimeInSeconds = samplingTimeInSeconds - deltaTickInSeconds;
    waitingTimeInMilliSeconds = waitingTimeInSeconds * 1000;
    Wait(waitingTimeInMilliSeconds);

    // sampling time interval has past, so capture time and write to file
    curTick = CurrentTick();
    deltaTick = curTick - prevTick; // measure elapsed ticks [msec]
    deltaTickInSeconds = deltaTick/1000.0; // in [sec]
    elapsedTimeInSeconds = elapsedTimeInSeconds + deltaTickInSeconds; // in [sec]
    strElapsedTimeInSeconds = FormatNum("%5.3f" , elapsedTimeInSeconds);
    text=StrCat(strElapsedTimeInSeconds, "," , strXSquareRoot);
    WriteToFile(text);

    // Update current tic value
    prevTick = curTick;

  } while( ((maxElapsedTimeInSeconds - elapsedTimeInSeconds) >= epsilonTime) && !greyButtonPushed);
  PlayTone(400,200); TextOut(0, LCD_LINE4, "Done!");

  StopAllTasks(); StopWriteToFile();

} // end main
```

Code description: The program begins a do-while loop which exits when the user pushed the Brick's grey button or when the total elapsed time has been reached.

A simple square root calculation is performed. The timer is polled again with a curTick = CurrentTick() statement. The difference between ticks is computed between the current and previous ticks using deltaTick= curTick – prevTick statement. This difference is compared to the sampling time to determine how long the Brick should idle. This idle time is given by a Wait(waitingTimeInMilliSeconds) statement.

After the Brick has reached the appropriate sampling time, the elapsed time since the user hit the orange button is computed. This elapsed time and square root computation is then written to file using a WriteToFile(text) statement.

Before looping back, the tick variables are updated using a prevTick = curTick statement.

Once the total elapsed time exceeds the user-defined maximum (i.e. 10 sec), the loop exits, plays a tone, and gracefully exits.

Step 5: Lastly, save your header file `fileSavingFunctions.h` as `fileSavingFunctionsForTimer1_0.h` and make the following modifications.

```
void InitWriteToFile() {
  fileNumber = 0;  // set first data file to be zero
  filePart = 0; // set first part of first data file to zero
  fileName = "timer.csv" ; // name of data file
  result=CreateFile(fileName, 1024, fileHandle);
  // NXT Guide Section 9.100 pg. 1812 and Section 6.59.2.2 pg. 535
  // returns file handle (unsigned int)

// check if the file already exists
  while (result==LDR_FILEEXISTS) // LDR_FILEEXISTS returns if file pre-exists
     {
      CloseFile(fileHandle);
      fileNumber = fileNumber + 1; // create new file if already exists
      fileName=NumToStr(fileNumber);
      fileName=StrCat("timer" , fileName, ".csv");
      result=CreateFile(fileName, 1024, fileHandle);
      } // end while

  // play a tone every time a file is created
  PlayTone(TONE_B7, 5);
  fileHeader = "time [s], Sqrt" ; // header
  WriteLnString(fileHandle, fileHeader, bytesWritten);
  // NXT Guide Section 6.59.2.43 pg. 554
  // Write string and new line to a file
  // bytesWritten is an unsigned int.  Its value is # of bytes written

} // end InitWriteToFile
```

In your `WriteToFile()` function, edit code so that data is saved to the desired file

```
void WriteToFile(string strTempText) {
  // strTempText stores the text (i.e. ticks and motorRpm to be written to file
:
:
:
// create the next file name
    filePart = filePart + 1;
    strFileNumber = NumToStr(fileNumber);
    strFilePart = NumToStr(filePart);
    fileName = StrCat("timer" , strFileNumber,"-", strFilePart ,".csv");
:
    } // end if
} // end WriteToFile
```

Step 6: Save both `timer1_0.nxc` and `fileSavingFunctionsForTimer1_0.h` files. Compile and execute. Once the program runs, hit the orange button to start the program. After 10 seconds, the program should play a tone and exit. Open NXT Explorer to retrieve your data file (called timer.csv) and plot in Excel which should resemble **Figure 2A**.
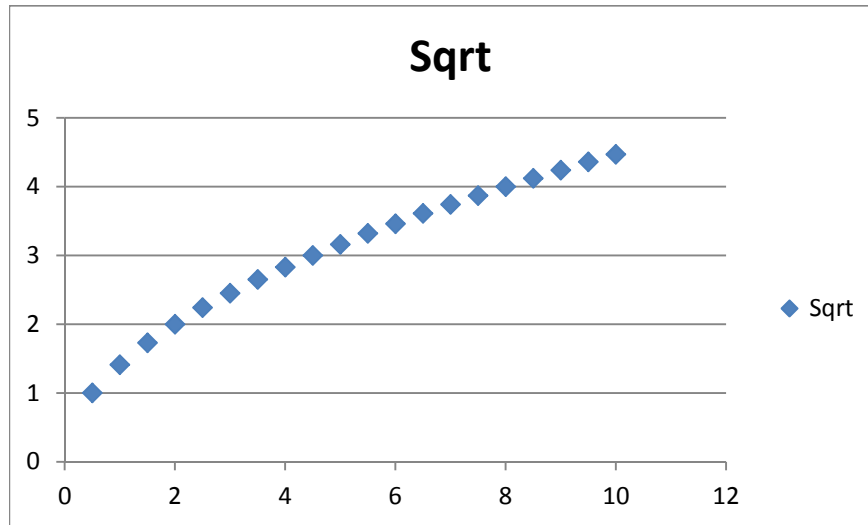

Figure 2A: Excel plot of `timer.csv`

Exercise 2: In NxC create programs for the following:

2-1: Modify timer1_0.nxc for a maximum run time of 20 seconds and computes the square root at a 0.1 second sampling time. Your saved files may likely look like timer.csv, timer-1.csv, timer-2.csv, etc. Stitch these data files into a single one and plot the curve.

2-2: Prove that you can capture data for desired duration and at a desired sampling time.