CSE 102 – Studio 6

LCD Display

Starting from the example application and support code here, author a program that receives strings from the serial port (of length up to 16 characters long, feel free to truncate if longer) and displays them on the LCD display. You can use `Serial.read()` or `Serial.readBytes()` to receive characters from the serial port.

The first received string should be displayed on the top row (row 0), the second received string should be displayed on the bottom row (row 1). When the third string is received, the display should be cleared and the cycled repeated (first displaying on the top row).

When using the serial monitor in the Arduino development environment to test the code above, the line that you type in is actually delivered to the Arduino when you hit the "return" key, which causes a `'\r'` character (carriage return) to be sent as well. Do not send this character to the LCD display, but do use it to indicate that the string is complete.

In addition to displaying the received strings, echo them back to the sender (including the trailing carriage return). This last capability you will likely want to turn on and off at various times in the testing below.

Enabling Java Communication

We will use the Java streams concept to communicate with the Arduino using the USB-based serial link. To support this, we need to add a pair of files to your eclipse environment. First, download the files RXTXcomm.jar and rxtxSerial.dll to your local filesystem. Start a "New -> Java Project" (I called mine cse102studio6). From within eclipse, "Import" these files (from the filesystem) into your project (not in the src directory, parallel to it).

Next, you need to alter the build path for Java. Right click on the project name, go down to Properties and pick Java Build Path. Then click libraries and add the RXTXcomm.jar file as a library. Once the jar file is listed as a library, expand it and double click on "Native library location", indicating the rxtxSerial.dll file in the workspace.

Download the `SerialComm` class file (and import it into your src directory) to enable `InputStream` and `OutputStream` access to the serial port, and use it to accomplish the tasks below.

Java Communication

Author an application in Java that sends a series of messages (strings of no more than 16 characters, terminated with `'\r'`) to the Arduino. Run this Java application in concert with the Arduino application

above to see that the strings being sent from Java are being displayed on the LCD Display. Then expand this Java application to receive strings from the Arduino and display them on the console.

Alter the Arduino application as appropriate for testing this Java application.

Communicating Larger Data Elements

Alter your Arduino echoing application to continue to echo bytes that have been received over the serial port; however, instead of displaying them as strings on the LCD display, show the individual bytes values in hexadecimal format. You may position bytes as you see fit on the LCD display, including overwriting previously received and displayed values.

Test this revised Arduino application using the serial monitor built into the Arduino IDE.

Next, alter your Java application to use a `DataOutputStream` object to send characters, short integers, long integers, and UTF-8 strings. Use a `DataInputStream` object to receive the echoed versions of these data elements, and print their value on the console. Check that the hex-displayed bytes showing on the LCD display are what you expect.