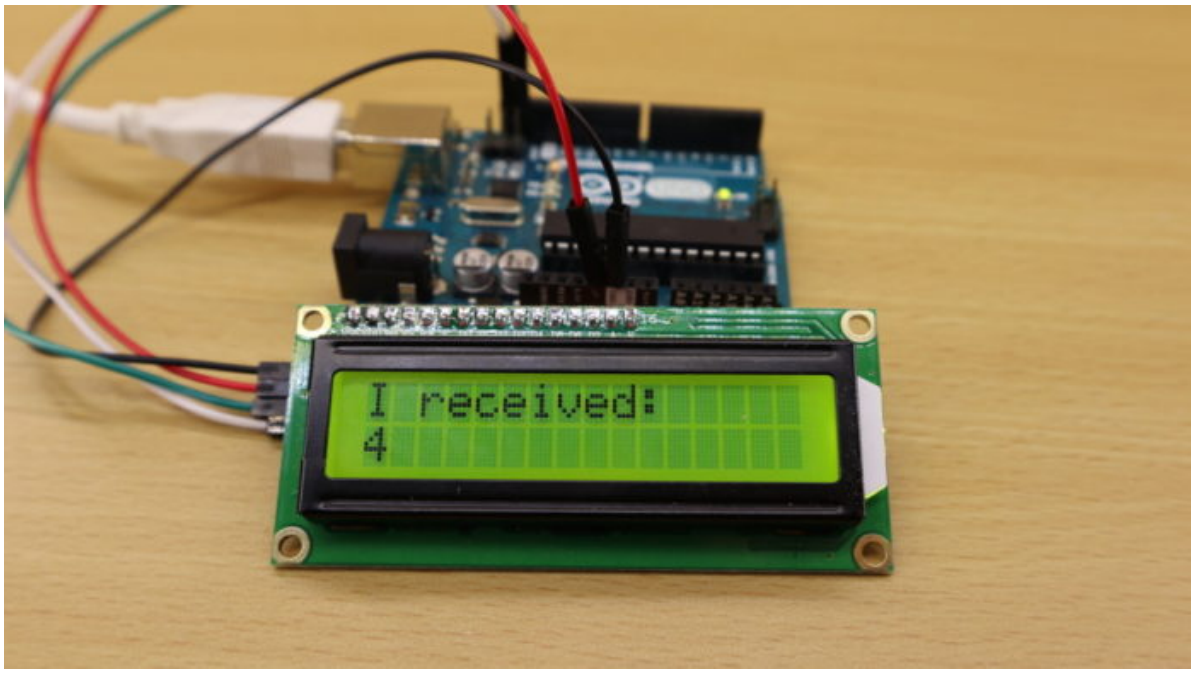
Tutorials | Arduino

Tutorial: Serial Connection between Java Application and Arduino Uno

December 29, 2017 6



In this tutorial, I demonstrate how to build up a serial connection between a Java Application and an Arduino Uno. The tutorial is divided into two parts: In the first part, it is explained how to send text (digits) from a Java Application to an Arduino. Moreover, the Arduino will print out the digits to an LCD module (LCM1602 IIC V1). In the second part, basically the same Java application is used to send digits to the Arduino – but this time with the help of a USB-to-TTL module. As a result, the Arduino's standard serial port can be used by the Arduino IDE to print the received digits to the serial monitor.

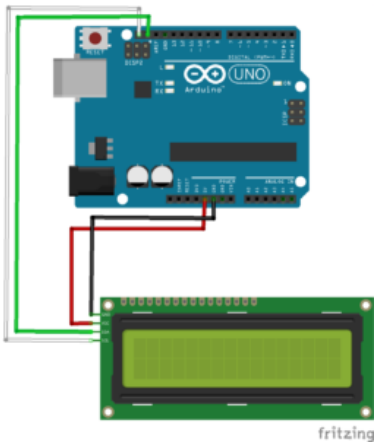
Notes:
– I added all source codes files to a github repository: https://github.com/mschoeffler/arduino-java-serial-communication
– I made also a video tutorial

List of materials:
– Arduino Uno (for Example Part 1&2) [Search on Aliexpress | Amazon | eBay.com]
– LCM1602 IIC V1 / LCD module (for Example Part 1) [Search on Aliexpress | Amazon | eBay.com]
– USB-to-TTL Serial Adapter (for Example Part 2) [Search on Aliexpress | Amazon | eBay.com]

Example Part 1

Setup
In this part, we have to wire an LCM1602 IIC V1 to the Arduino. The LCM1602 has four pins: VCC, GND, SDA, and SCL. The wiring is straightforward: VCC goes to the Arduino's 5V. The other three pins

have the exact same names on the Arduino: GND goes to GND, SDA to SDA, and SCL to SCL. Have a look at the fritzing file for more details:



Fritzing files that shows how to wire an LCM1602 IIC V1 module to an Arduino Uno.

Arduino Source Code

Next, we have to write some code for the Arduino Uno. Basically, the code just waits for bytes ready to be read by the serial port. If a byte was read, it is printed out to the LCM1602 IIC V1 module.

```
// (c) Michael Schoeffler 2017, http://www.mschoeffler.de
#include <Wire.h>
#include <LiquidCrystal_I2C.h> // LiquidCrystal_I2C library

LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE); // 0x27 is the i2c address of

void setup() {
  lcd.begin(16, 2); // begins connection to the LCD module
  lcd.backlight(); // turns on the backlight
  lcd.clear();

  Serial.begin(9600);
  while (!Serial) {
    ; // wait for serial port to connect.
  }

  lcd.setCursor(0, 0); // set cursor to first row
  lcd.print("Init done"); // print to lcd

}

void loop() {
  if (Serial.available() > 0) {
    byte incomingByte = 0;
    incomingByte = Serial.read(); // read the incoming byte:
    if (incomingByte != -1) { // -1 means no data is available
      lcd.setCursor(0, 0); // set cursor to first row
      lcd.print("I received: "); // print out to LCD
      lcd.setCursor(0, 1); // set cursor to secon row
      lcd.print(incomingByte); // print out the retrieved value to the second row
```

```
    }
  }
}
```

Java Source Code

The Java application uses the jSerialComm library to send text to an Arduino Uno via a standard USB connection. I made use of Maven to set up the dependency between my Java project and the jSerialComm library. If you also use Maven for your project, then my POM file might be of use to you:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchem
  <modelVersion>4.0.0</modelVersion>
  <groupId>de.mschoeffler.arduino.serialcomm</groupId>
  <artifactId>de.mschoeffler.arduino.serialcomm.example01</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>ArduinoBasicExample</name>
  <build>
    <sourceDirectory>src</sourceDirectory>
    <plugins>
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.5.1</version>
        <configuration>
          <source>1.8</source>
          <target>1.8</target>
        </configuration>
      </plugin>
    </plugins>
  </build>
  <dependencies>
    <dependency>
     <groupId>com.fazecast</groupId>
     <artifactId>jSerialComm</artifactId>
     <version>1.3.11</version>
  </dependency>
  </dependencies>
</project>
```

The actual Java source code is only a single class with a main method. A serial connection is established. Then, five digits (0-4) are written to the serial port. Finally, the serial connection is closed:

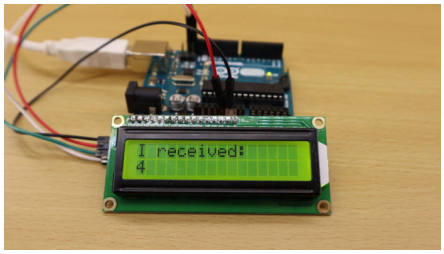
```
package de.mschoeffler.arduino.serialcomm.example01;

import java.io.IOException;
import com.fazecast.jSerialComm.SerialPort;

/**
 * Simple application that is part of an tutorial.
 * The tutorial shows how to establish a serial connection between a Java and Arduino program
 * @author Michael Schoeffler (www.mschoeffler.de)
```

```
 *
 */
public class Startup {

  public static void main(String[] args) throws IOException, InterruptedException {
    SerialPort sp = SerialPort.getCommPort("/dev/ttyACM1"); // device name TODO: must be chan
    sp.setComPortParameters(9600, 8, 1, 0); // default connection settings for Arduino
    sp.setComPortTimeouts(SerialPort.TIMEOUT_WRITE_BLOCKING, 0, 0); // block until bytes can

    if (sp.openPort()) {
      System.out.println("Port is open :)");
    } else {
      System.out.println("Failed to open port :(");
      return;
    }

    for (Integer i = 0; i < 5; ++i) {
      sp.getOutputStream().write(i.byteValue());
      sp.getOutputStream().flush();
      System.out.println("Sent number: " + i);
      Thread.sleep(1000);
    }

    if (sp.closePort()) {
      System.out.println("Port is closed :)");
    } else {
      System.out.println("Failed to close port :(");
      return;
    }


  }

}
```

The main important method call is `SerialPort.getCommPort(...)`. The method has only one argument which has to be the device name of your Arduino Uno. Therefore, it is very likely that you need to change this argument value. You can find out the device name of your Arduino Uno by having a look to the Arduino IDE. In Tools->Port you find all connected devices. In order to upload code to an Arduino, you have to select the correct device name of the corresponding Arduino. Luckily, the same device name is needed by the jSerialComm library. So simply copy the device name from your Arduino IDE to the Java source code.

Execution

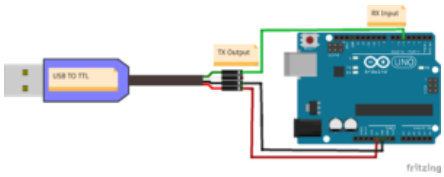
When you have uploaded the source code to the Arduino and started the java application, you should see that the digits 0-4 appear on the LCM1602 IIC V1 module. Sometimes, I have problems executing this code, if I use cheap Arduino clones. Luckily, it always executes perfectly on my original Arduino Uno from Italy ;)



Example 2

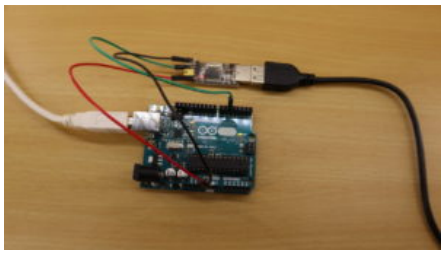
In the second part of this tutorial, the digits (coming from the Java application) are printed to the default serial connection of the Arduino. As a result, the received digits can be viewed from the Arduino IDE's serial monitor (Tools -> Serial Monitor). Unfortunately, as a consequence, the standard USB connection cannot be used by the Java application since the serial monitor will already catch the serial port as soon as we open it. Therefore, we make use of a USB-to-TTL serial adapter.

Setup
Typically, a USB-to-TTL adapter has at least four pins: VCC, GND, RX (receive data), and TX (transmit data). As we will use this adapter only to send data from the Java application, we can ignore the RX pin. VCC must be connected to the Arduino's 5V pin, GND must be connected to the Arduino's GND pin, and TX must be connected to the Arduino digital pin #5 (also other digital pins can be used).



Fritzing file that shows how to connect a USB-to-TTL serial adapter to an Arduino Uno.

Here you see how my setup looks like (including the serial adapter):



USB-to-TTL serial adapter connected to an Arduino Uno.

Arduino Source Code:
The Arduino program makes use of a so-called software serial. When a software serial object is initialized, it requires the pin numbers of the receive and transmit pin. As we do not plan to transmit text from the Arduino Uno, we can set the transmit pin to any number. Here, I simply set it to 6.

```
// (c) Michael Schoeffler 2017, http://www.mschoeffler.de
#include <SoftwareSerial.h>

SoftwareSerial sserial(5,6); // receive pin (used), transmit pin (unused)

void setup() {
```

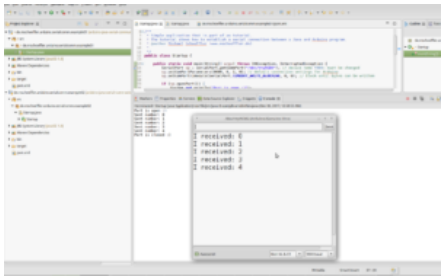
```
  Serial.begin(9600); // used for printing to serial monitor of the Arduino IDE
  sserial.begin(9600); // used to receive digits from the Java application
  while (!Serial) {
    ; // wait for serial port to connect.
  }
}

void loop() {
  if (sserial.available() > 0) {
    byte incomingByte = 0;
    incomingByte = sserial.read();
    if (incomingByte != -1) {
      Serial.print("I received: "); // print out to serial monitor
      Serial.println(incomingByte); // print out to serial monitor
    }
  }
}
```

Java source code:

The Java application is basically the same than the one used in part one of this tutorial. The only exception is the device name of the USB-to-TTL device. Conveniently, you can again make use of the Arduino IDE as it will show you the name of the serial adapter, too.

Execution:

If everything was successfully executed, the serial monitor of the Arduino IDE should show five received digits.



Shows the result of a serial connection between a Java application (Eclipse IDE) and an Arduino Uno (Arduino IDE / serial monitor).

Video Tutorial:

Serial Connection from Java to Arduino | UATS Advanced Programming #01