Connecting Arduino to Processing a learn.sparkfun.com tutorial

Available online at: http://sfe.io/t69

Contents

Introduction

So, you've blinked some LEDs with Arduino, and maybe you've even drawn some pretty pictures with Processing - what's next? At this point you may be thinking, 'I wonder if there's a way to get Arduino and Processing to communicate to each other?'. Well, guess what - there is! - and this tutorial is going to show you how.

In this tutorial we will learn:

- How to send data from Arduino to Processing over the serial port
- How to receive data from Arduino in Processing
- How to send data from Processing to Arduino
- How to receive data from Processing in Arduino
- How to write a serial 'handshake' between Arduino and Processing to control data flow
- How to make a 'Pong' game that uses analog sensors to control the paddles

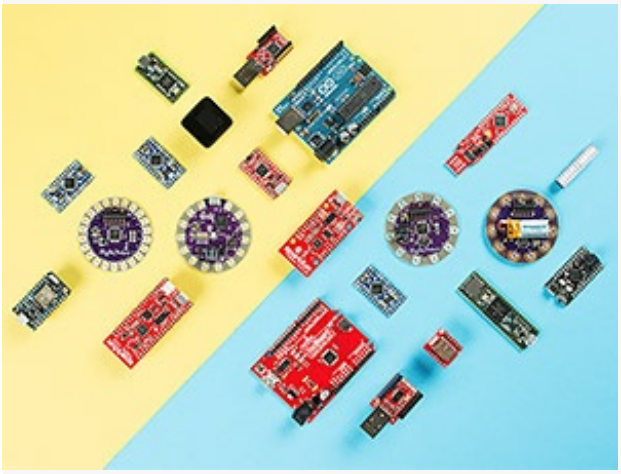
Before we get started, there are a few things you should be certain you're familiar with to get the most out of this tutorial:

- What's an Arduino?
- How to use a breadboard
- Working with wire
- What is serial communication?
- Some basic familiarity with Processing will be useful, but not strictly necessary.

Looking for the right Arduino?

Check out our **Arduino Comparison Guide**! We've compiled every Arduino development board we carry, so you can quickly compare them to find the perfect one for your needs.

Take me there!



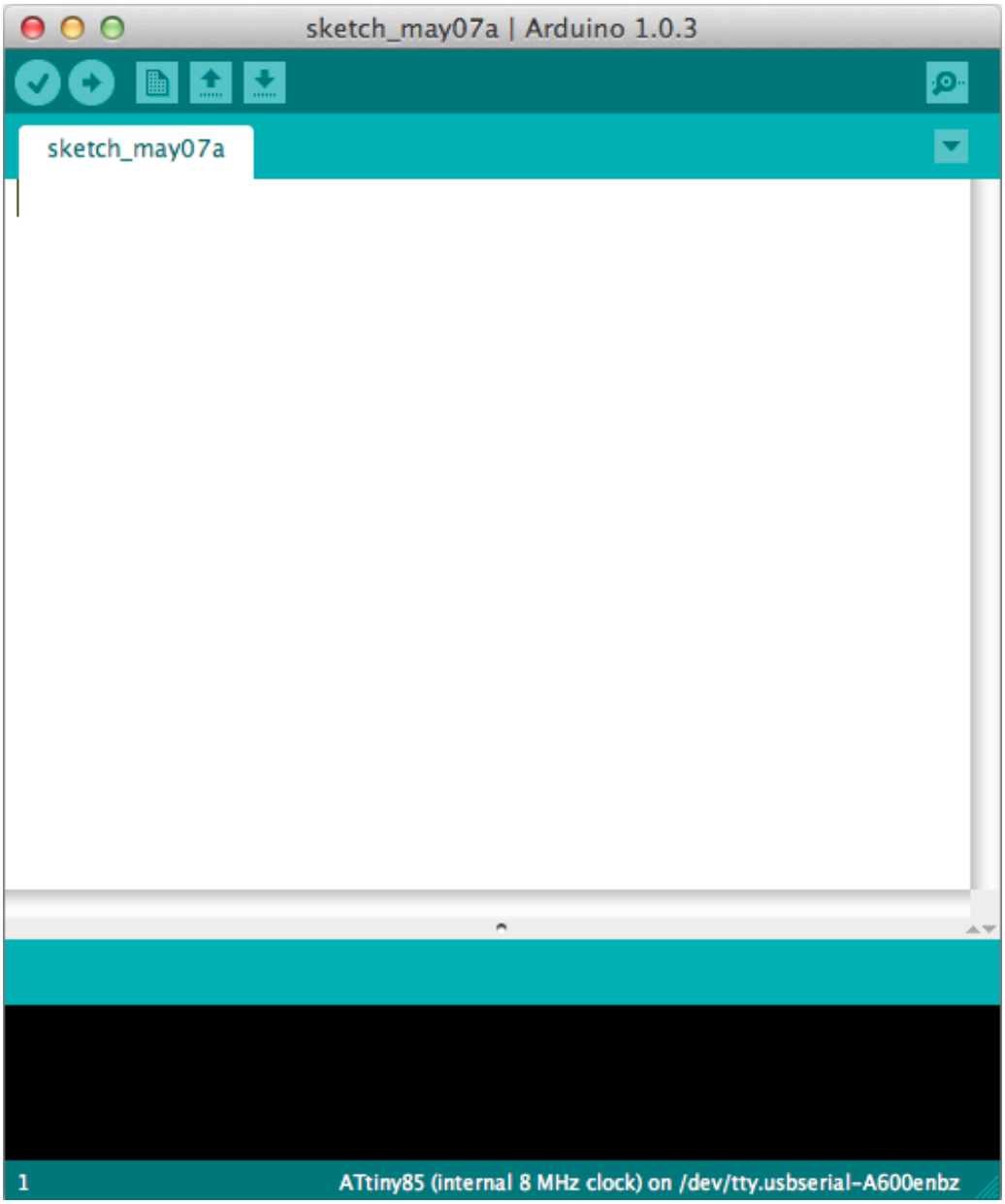
From Arduino...

Let's start with the Arduino side of things. We'll show you the basics of how to set up your Arduino sketch to send information over serial.

- First things first. If you haven't done so yet, download and install the Arduino software for your operating system. Here's a tutorial if you get stuck.
- You'll also need an Arduino-compatible microcontroller and an appropriate way to connect it to your computer (an A-to-B USB cable, micro USB, or FTDI breakout). Check this comparison guide if you're not sure what's right for you.

Ok. You should by this point have the Arduino software installed, an Arduino board of some kind, and a cable. Now for some coding! Don't worry, it's quite straightforward.

- Open up the Arduino software. You should see something like this:



The nice big white space is where we are going to write our code. Click in the white area and type the following (or copy and paste if you feel lazy):

```
language:cpp
void setup()
{
//initialize serial communications at a 9600 baud rate
Serial.begin(9600);
}
```

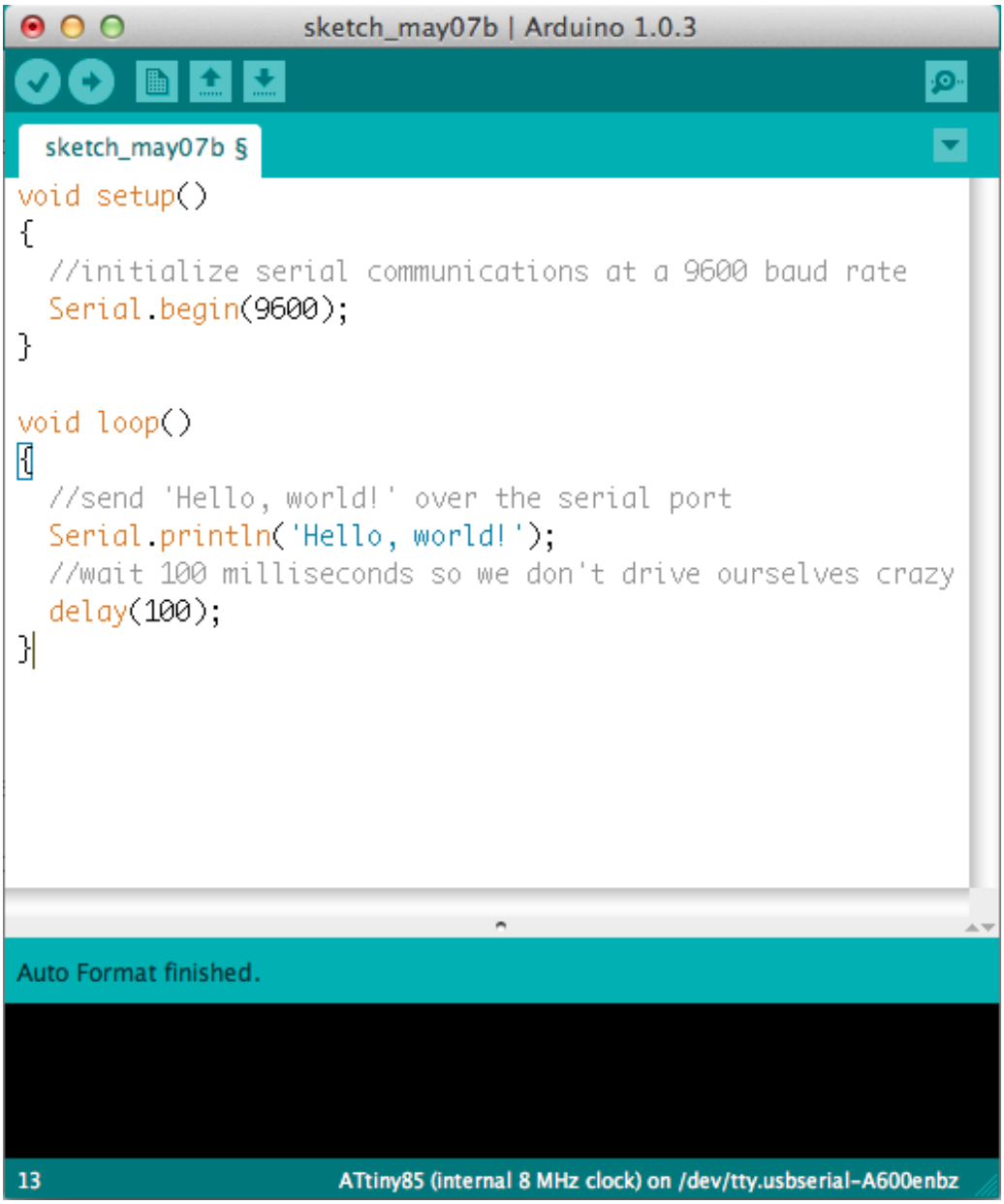
This is called our setup method. It's where we 'set up' our program. Here, we're using it to start serial communication from the Arduino to our computer at a baud rate of 9600. For now, all you need to now about baud rate is that (basically) it's the rate at which we're sending data to the computer, and if we're sending and receiving data at different rates, everything goes all gobbledy-gook and one side can't understand the other. This is bad.

After our `setup()` method, we need a method called `loop()`, which is going to repeat over and over as

long as our program is running. For our first example, we'll just send the string 'Hello, world!' over the serial port, over and over (and over). Type the following in your Arduino sketch, below the code we already wrote:

```
language:cpp
void loop()
{
//send 'Hello, world!' over the serial port
Serial.println("Hello, world!");
//wait 100 milliseconds so we don't drive ourselves crazy
delay(100);
}
```

That's all we need for the Arduino side of our first example. We're setting up serial communication from the Arduino and telling it to send data every 100 milliseconds. Your Arduino sketch should now look something like this:



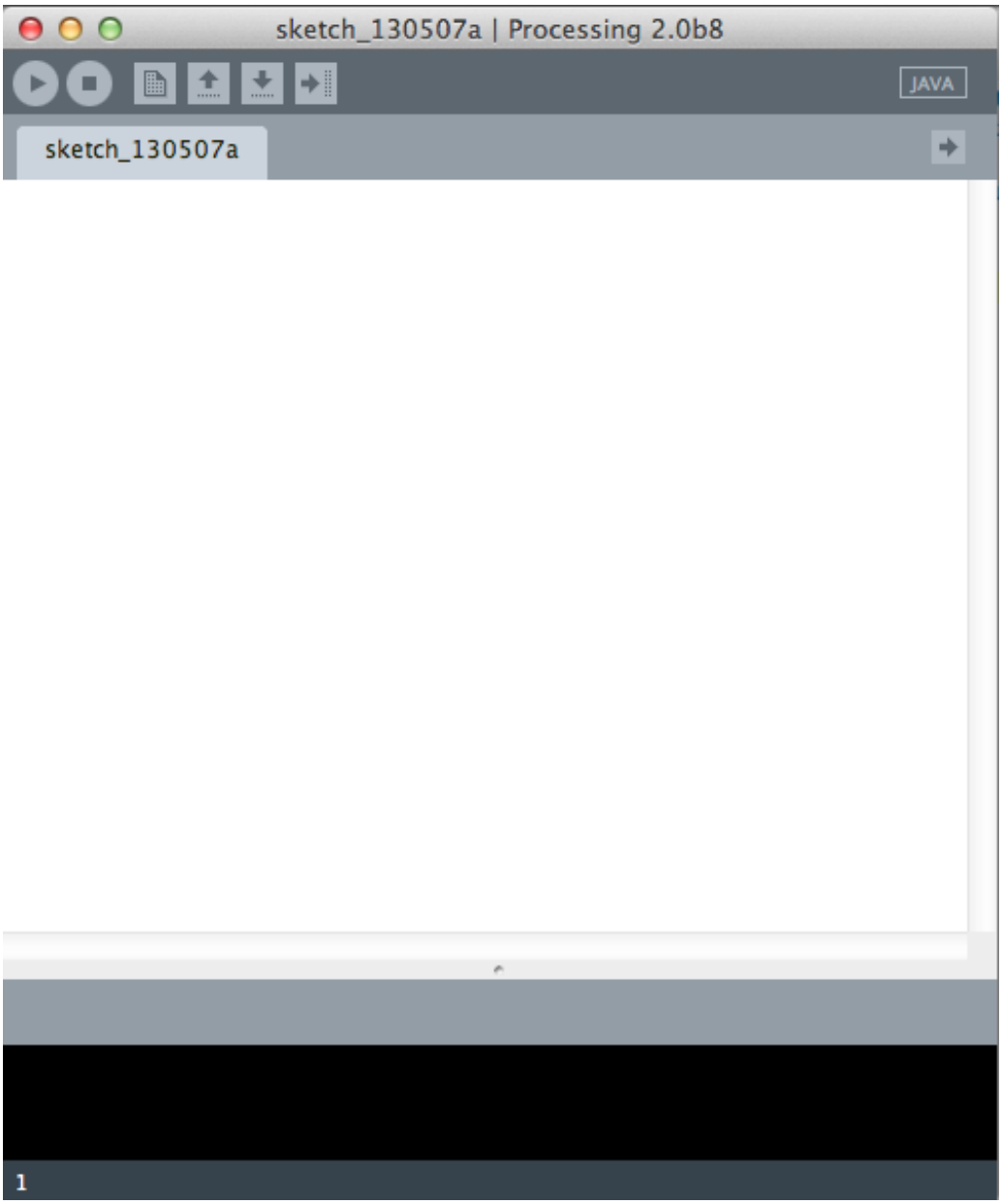
All that's left to do is to plug in your Arduino board, select your board type (under Tools -> Board

Type) and your Serial port (under Tools -> Serial Port) and hit the 'upload' button to load your code onto the Arduino.

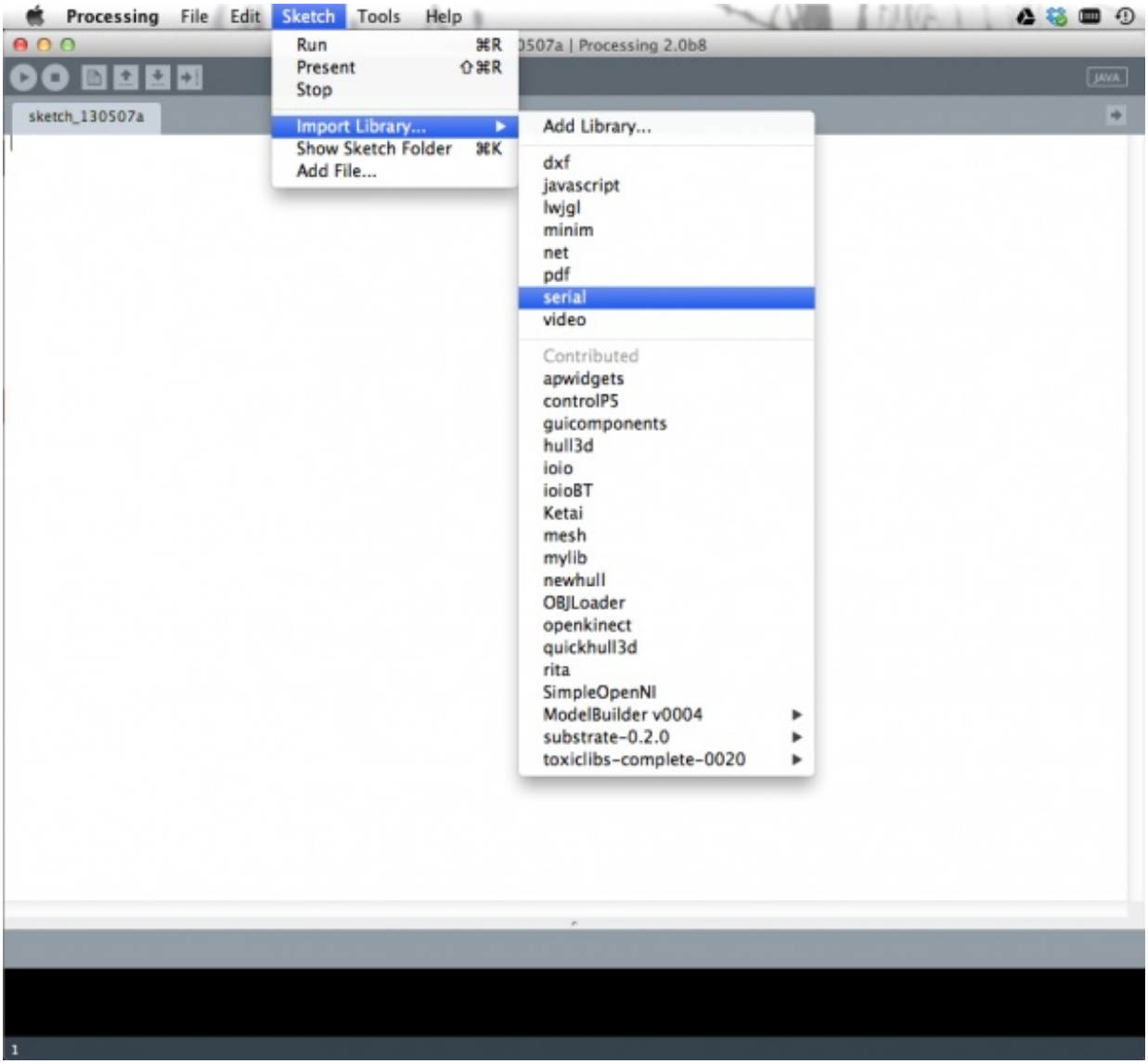
Now we're ready to see if we can magically (or through code) detect the 'Hello, world!' string we're sending from Processing.

...to Processing

Our task now is to find a way to listen in on what our Arduino sketch is sending. Luckily, Processing comes with a Serial library designed for just this kind of thing! If you don't have a version of Processing, make sure you go to Processing.org and download the latest version for your operating system. Once Processing is installed, open it up. You should see something like this:



Looks a lot like Arduino, huh? The Arduino software was actually based in part off of Processing - that's the beauty of open-source projects. Once we have an open sketch, our first step is to import the Serial library. Go to Sketch->Import Library->Serial, as shown below:



You should now see a line like `import processing.serial.*;` at the top of your sketch. Magic! Underneath our import statement we need to declare some global variables. All this means is that these variables can used anywhere in our sketch. Add these two lines beneath the import statement:

```
language:java
Serial myPort;  // Create object from Serial class
String val;     // Data received from the serial port
```

In order to listen to any serial communication we have to get a Serial object (we call it `myPort` but you can it whatever you like), which lets us listen in on a serial port on our computer for any incoming data. We also need a variable to recieve the actual data coming in. In this case, since we're sending a String (the sequence of characters 'Hello, World!') from Arduino, we want to receive a String in Processing. Just like Arduino has `setup()` and `loop()`, Processing has `setup()` and `draw()` (instead of loop).

For our `setup()` method in Processing, we're going to find the serial port our Arduino is connected to and set up our Serial object to listen to that port.

```
language:java
void setup()
{
```

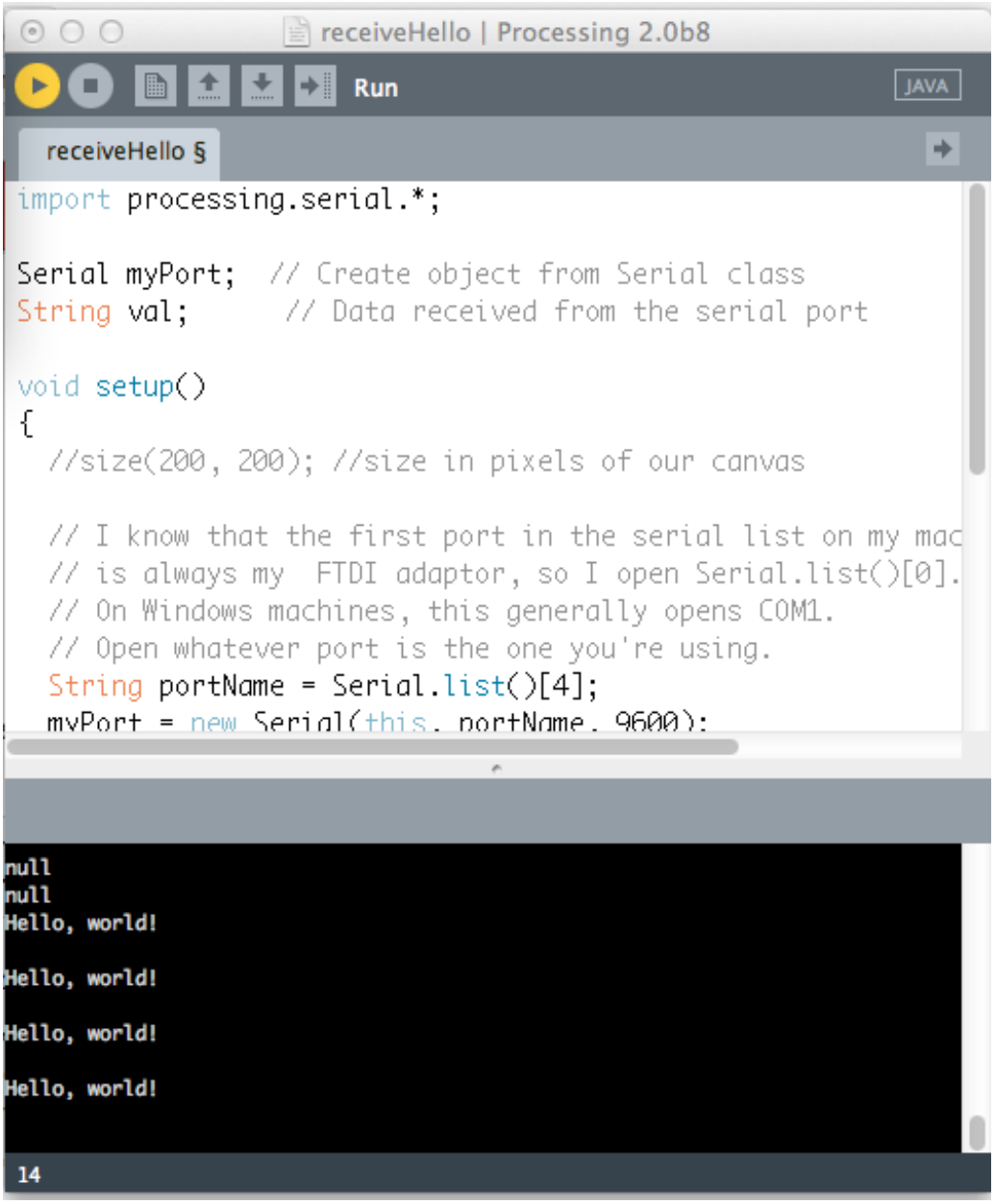
```
  // I know that the first port in the serial list on my mac
  // is Serial.list()[0].
  // On Windows machines, this generally opens COM1.
  // Open whatever port is the one you're using.
  String portName = Serial.list()[0]; //change the 0 to a 1 or 2 etc. to match your port
  myPort = new Serial(this, portName, 9600);
}
```

Remember how we set `Serial.begin(9600)` in Arduino? Well, if we don't want that gobbledy-gook I was talking about, we had better put 9600 as that last argument in our Serial object in Processing as well. This way Arduino and Processing are communicating at the same rate. Happy times!

In our `draw()` loop, we're going to listen in on our Serial port and we get something, stick that something in our `val` variable and print it to the console (that black area at the bottom of your Processing sketch).

```
language:java
void draw()
{
  if ( myPort.available() > 0)
  {  // If data is available,
  val = myPort.readStringUntil('\n');         // read it and store it in val
  }
println(val); //print it out in the console
}
```

Ta-Da! If you hit the 'run' button (and your Arduino is plugged in with the code on the previous page loaded up), you should see a little window pop-up, and after a sec you should see `Hello, World!' appear in the Processing console. Over and over. Like this:



Excellent! We've now conquered how to send data from Arduino to Processing. Our next step is figure out how go the opposite way - sending data from Processing to Arduino.

From Processing...

So we've sent data from Arduino to Processing, but what if we want to send data the other way - from Processing to Arduino? Piece of cake!

Let's start with the Processing side of things. It starts out much like our last sketch: we import the Serial library and declare a global Serial object variable for our port up top, and in our setup() method we find our port and initialize Serial communication on that port with our Serial variable at 9600 baud. We're also going to use the size() command, to give us a little window to click in, which will trigger our sketch to send something over the Serial port to Arduino.

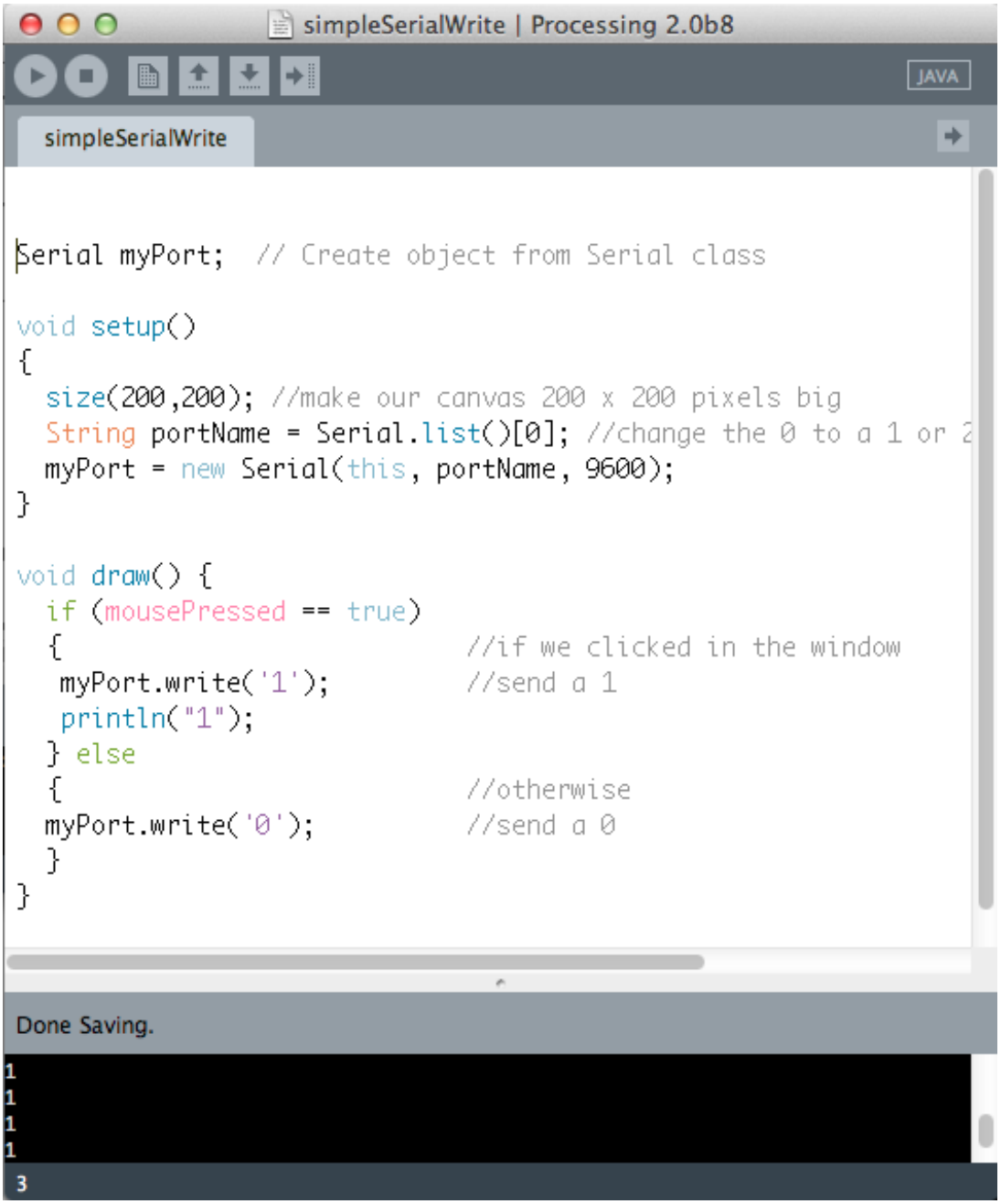
language:java
```
import processing.serial.*;
```

```
Serial myPort;  // Create object from Serial class

void setup()
{
  size(200,200); //make our canvas 200 x 200 pixels big
  String portName = Serial.list()[0]; //change the 0 to a 1 or 2 etc. to match your port
  myPort = new Serial(this, portName, 9600);
}
```

In our `draw()` loop, we send whatever we want over the serial port by using the `write` method from the Processing Serial library. For this sketch, we will send a '1' whenever we click our mouse in the Processing window. We'll also print it out on the console, just to see that we're actually sending something. If we aren't clicking we'll send a '0' instead.

```
language:java
void draw() {
  if (mousePressed == true)
  {                       //if we clicked in the window
    myPort.write('1');        //send a 1
    println("1");
  } else
  {                       //otherwise
  myPort.write('0');          //send a 0
  }
}
```

This is what your code should look like at this point:



If you run this code, you should see a bunch of 1's appear in the console area whenever you click your mouse in the window. Neat! But how do we look for these 1's from Arduino? And what can we do with them?

...to Arduino

Ok! On this page we're going to look for those 1's coming in from Processing, and, if we see them, we're going to turn on an LED on pin 13 (on some Arduinos, like the Uno, pin 13 is the on-board LED, so you don't need an external LED to see this work).

At the top of our Arduino sketch, we need two global variables - one for holding the data coming from Processing, and another to tell Arduino which pin our LED is hooked up to.

language:cpp

```
 char val; // Data received from the serial port
 int ledPin = 13; // Set the pin to digital I/O 13
```

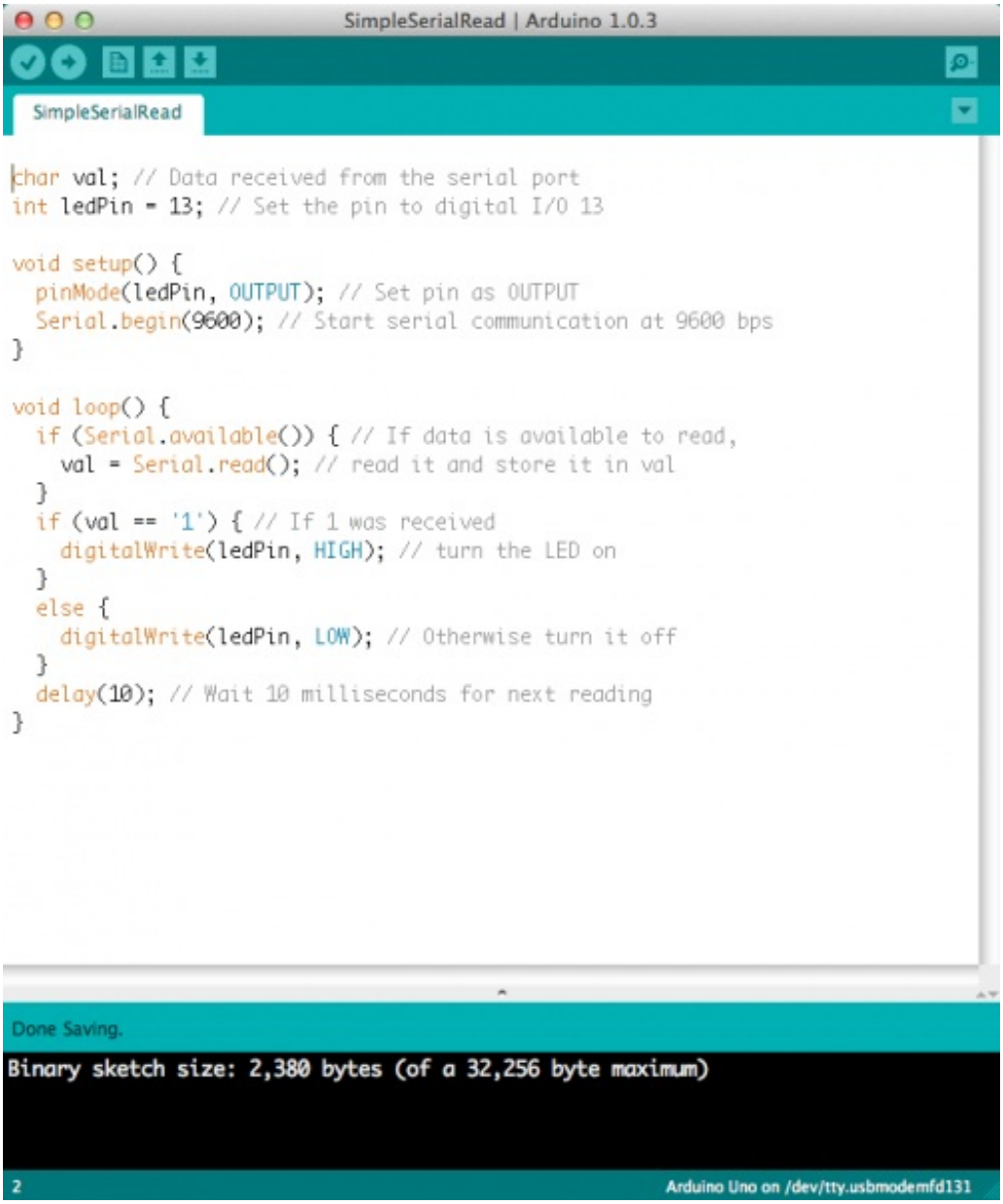
Next, in our `setup()` method, we'll set the LED pin to an output, since we're powering an LED, and we'll start Serial communication at 9600 baud.

```
language:cpp
 void setup() {
   pinMode(ledPin, OUTPUT); // Set pin as OUTPUT
   Serial.begin(9600); // Start serial communication at 9600 bps
 }
```

Finally, in the `loop()` method, we'll look at the incoming serial data. If we see a '1', we set the LED to HIGH (or on), and if we don't (e.g. we see a '0' instead), we turn the LED off. At the end of the loop, we put in a small delay to help the Arduino keep up with the serial stream.

```
language:cpp
 void loop() {
   if (Serial.available())
   { // If data is available to read,
     val = Serial.read(); // read it and store it in val
   }
   if (val == '1')
   { // If 1 was received
     digitalWrite(ledPin, HIGH); // turn the LED on
   } else {
     digitalWrite(ledPin, LOW); // otherwise turn it off
   }
   delay(10); // Wait 10 milliseconds for next reading
}
```

This is what your code should look like when you're done:



Voila! If we load up this code onto our Arduino, and run the Processing sketch from the previous page, you should be able to turn on an LED attached to pin 13 of your Arduino, simply by clicking within the Processing canvas.

Shaking Hands (Part 1)

So far we've shown that Arduino and Processing can communicate via serial when one is talking and the other is listening. Can we make a link that allows data to flow both ways, so that Arduino and Processing are both sending *and* receiving data? You bet! In the biz we call this a serial 'handshake', since both sides have to agree when to send and receive data.

On this page and the next, we're going to combine our two previous examples in such a way that Processing can both receive 'Hello, world!' from Arduino AND send a 1 back to Arduino to toggle an LED. Of course, this also means that Arduino has to be able to send 'Hello, world!' while listening for a 1 from Processing. Whew!

Let's start with the Arduino side of things. In order for this to run smoothly, both sides have to know what to listen for and what the other side is expecting to hear. We also want to minimize traffic over the serial port so we get more timely responses.

Just like in our Serial read example, we need a variable for our incoming data and a variable for the LED pin we want to light up:

```
language:cpp
char val; // Data received from the serial port
int ledPin = 13; // Set the pin to digital I/O 13
boolean ledState = LOW; //to toggle our LED
```

Since we're trying to be efficient, we're going to change our code so that we only listen for 1's, and each time we hear a '1' we toggle the LED on or off. To do this we added a boolean (true or false) variable for the HIGH or LOW state of our LED. This means we don't have to constantly send a 1 or 0 from Processing, which frees up our serial port quite a bit.

Our `setup()` method looks mostly the same, with the addition of an `establishContact()` function which we'll get to later - for now just type it in.

```
language:cpp
void setup()
{
  pinMode(ledPin, OUTPUT); // Set pin as OUTPUT
  //initialize serial communications at a 9600 baud rate
  Serial.begin(9600);
  establishContact();  // send a byte to establish contact until receiver responds
}
```

In our loop function, we've just combined and slimmed down the code from our two earlier sketches. Most importantly, we've changed our LED code to toggle based on our new boolean value. The '!' means every time we see a one, we set the boolean to the opposite of what it was before (so LOW becomes HIGH or vice-versa). We also put our 'Hello, world!' in an else statement, so that we're only sending it when we haven't seen a '1' come in.

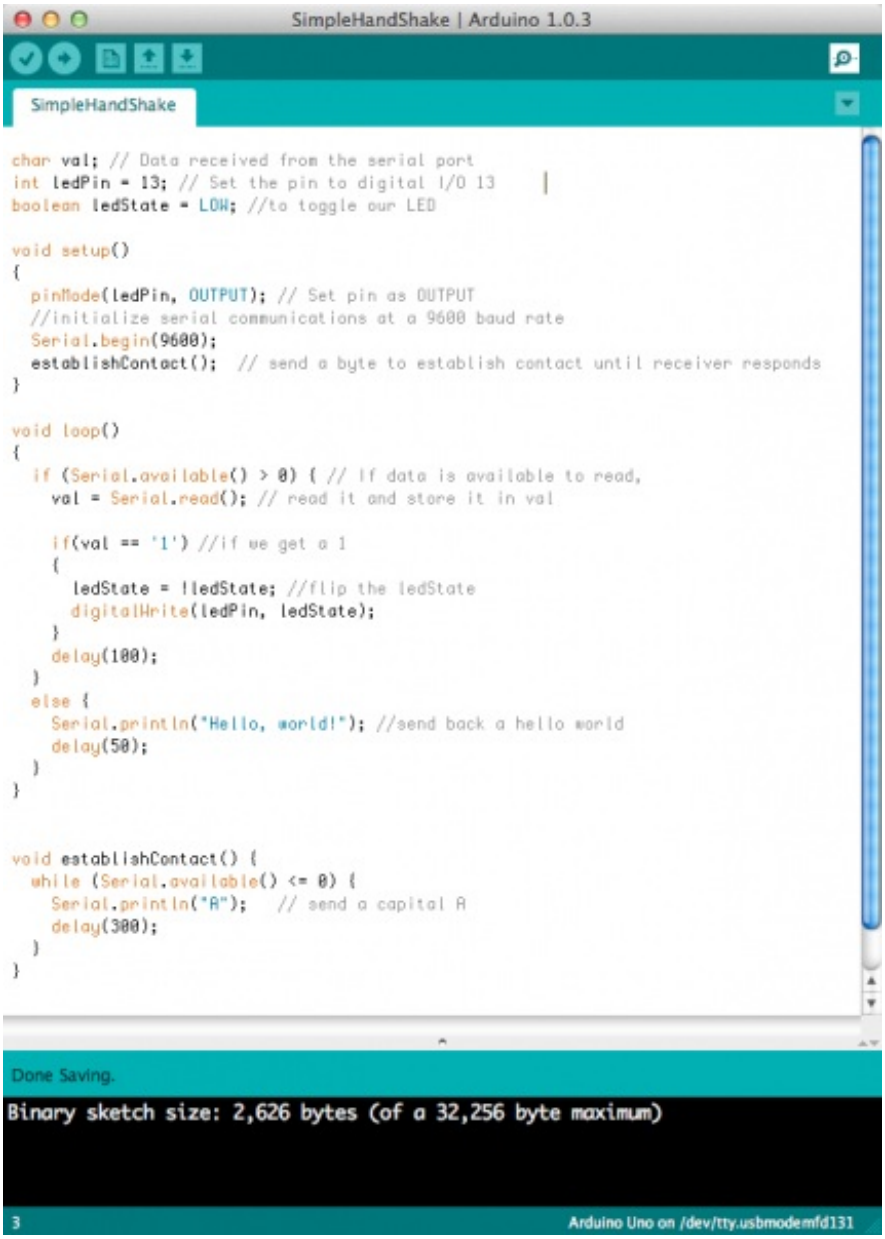
```
language:cpp
void loop()
{
  if (Serial.available() > 0) { // If data is available to read,
    val = Serial.read(); // read it and store it in val

    if(val == '1') //if we get a 1
    {
      ledState = !ledState; //flip the ledState
      digitalWrite(ledPin, ledState);
    }
    delay(100);
  }
    else {
    Serial.println("Hello, world!"); //send back a hello world
    delay(50);
    }
```

}

Now we get to that `establishContact()` function we put in our `setup()` method. This function just sends out a string (the same one we'll need to look for in Processing) to see if it hears anything back - indicating that Processing is ready to receive data. It's like saying 'Marco' over and over until you hear a 'Polo' back from somewhere.

```
language:cpp
void establishContact() {
  while (Serial.available() <= 0) {
  Serial.println("A");   // send a capital A
  delay(300);
  }
}
```

Your Arduino code should look like this:



That's it for the Arduino side, now on to Processing!

Shaking Hands (Part 2)

For the Processing side of things, we've got to make a few changes. We're going to make use of the serialEvent() method, which gets called every time we see a specific character in the serial buffer, which acts as our delimiter - basically it tells Processing that we're done with a specific 'chunk' of data - in our case, one 'Hello, world!'.

The beginning of our sketch is the same except for a new firstContact boolean, which let's us know when we've made a connection to Arduino.

```
language:java
import processing.serial.*; //import the Serial library
 Serial myPort;  //the Serial port object
 String val;
// since we're doing serial handshaking,
// we need to check if we've heard from the microcontroller
boolean firstContact = false;
```

Our setup() function is the same as it was for our serial write program, *except* we added the myPort.bufferUntil('\n'); line. This let's us store the incoming data into a buffer, until we see a specific character we're looking for. In this case, it's a carriage return (\n) because we sent a Serial.print*ln* from Arduino. The 'ln' at the end means the String is terminated with a carriage return, so we know that'll be the last thing we see.

```
language:java
void setup() {
  size(200, 200); //make our canvas 200 x 200 pixels big
  //  initialize your serial port and set the baud rate to 9600
  myPort = new Serial(this, Serial.list()[4], 9600);
  myPort.bufferUntil('\n');
}
```

Because we're continuously sending data, our serialEvent() method now acts as our new draw() loop, so we can leave it empty:

```
language:java
void draw() {
  //we can leave the draw method empty,
  //because all our programming happens in the serialEvent (see below)
}
```

Now for the big one: serialEvent(). Each time we see a carriage return this method gets called. We need to do a few things each time to keep things running smoothly:

- read the incoming data
- see if there's actually anything in it (i.e. it's not empty or 'null')
- trim whitespace and other unimportant stuff
- if it's our first time hearing the right thing, change our firstContact boolean and let Arduino know we're ready for more data
- if it's *not* our first run, print the data to the console and send back any valid mouse clicks (as

1's) we got in our window
- finally, tell Arduino we're ready for more data

That's a lot of steps, but luckily for us Processing has functions that make most of these tasks pretty easy. Let's take a look at how it all breaks down:

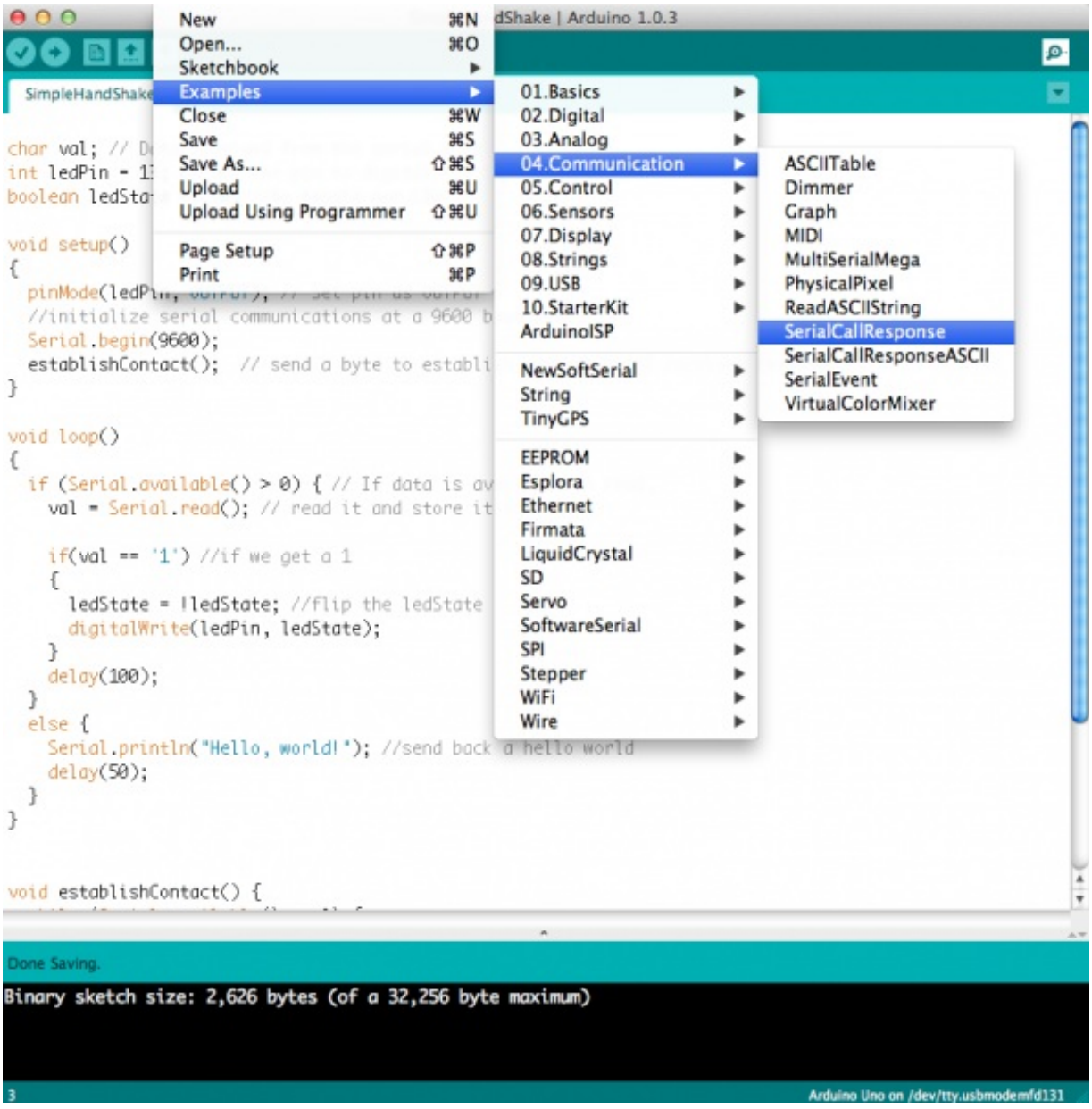
```
language:java
void serialEvent( Serial myPort) {
//put the incoming data into a String -
//the '\n' is our end delimiter indicating the end of a complete packet
val = myPort.readStringUntil('\n');
//make sure our data isn't empty before continuing
if (val != null) {
  //trim whitespace and formatting characters (like carriage return)
  val = trim(val);
  println(val);

  //look for our 'A' string to start the handshake
  //if it's there, clear the buffer, and send a request for data
  if (firstContact == false) {
    if (val.equals("A")) {
      myPort.clear();
      firstContact = true;
      myPort.write("A");
      println("contact");
    }
  }
  else { //if we've already established contact, keep getting and parsing data
    println(val);

    if (mousePressed == true)
    {                           //if we clicked in the window
      myPort.write('1');        //send a 1
      println("1");
    }

    // when you've parsed the data you have, ask for more:
    myPort.write("A");
    }
  }
}
```

Oof. That's a lot to chew on, but if you read carefully line by line (especially the comments), it'll start to make sense. If you've got your Arduino code finished and loaded onto your board, try running this sketch. You should see 'Hello, world!' coming in on the console, and when you click in the Processing window, you should see the LED on pin 13 turn on and off. Success! You are now a serial handshake expert.

Tips and Tricks

In developing your own projects with Arduino and Processing, there are a few 'gotchas' that are helpful to keep in mind in case you get stuck.

- make sure your baud rates match
- make sure you're reading off the right port in Processing - there's a Serial.list() command that will show you all the available ports you can connect to.
- if you're using the serialEvent() method, make sure to include the port.bufferUntil() function in your setup() method.
- also, make sure that whatever character you're buffering until (e.g., '\n') is a character that you're actually sending from Arduino.
- If you want to send over a number of sensor values, it's a good idea to count how many bytes you're expecting so you know how to properly parse out the sensor data. (the example (shown below) that comes with Arduino gives a great example of this:



This is the example to select for some good sensor parsing code

Resources and Going Further

Now that you know how to send data from Arduino to Processing and back again (even simultaneously!), you're ready for some seriously cool projects. By hooking together Arduino and Processing, you can do things like visualize sensor data in real-time, or make a glove with flex sensors in the fingers that makes penguins appear on the screen, or a command console from Processing that controls a giant array of LEDs.

Here are a few useful links that you may find useful going forward:

- Derek Runberg's Processing Curriculum
- Arduino & Processing ATLAS Curriculum
- Processing the Danger Shield
- Arduino, Processing, & MaxMSP
