If you are working on a multi-user project, you most likely want to run the servers and clients on separate computers. After all, this is the point of creating multi-user applications in the first place. However, for testing and development purposes, it is often convenient to run all the elements from one computer. In this case, the server IP address will be “localhost” or 127.0.0.1 (note this is the IP address used in this chapter’s examples).

As will be covered in Section 21.3, Processing’s “export to application” feature will allow you to export a stand-alone application for your server, which you can then run in the background while you develop your client in Processing. Details as to how “export to application” works can be found in Chapter 18. You can also run multiple copies of a stand-alone application to simulate an environment with more than one client. Figure 19.6 shows the server running with two client instances.

Exercise 19-4: Expand the whiteboard to allow for color. Each client should send a red, green, and blue value in addition to the XY location. You will not need to make any changes to the server for this to work.

Exercise 19-5: Create a two-player game of Pong played over the network. This is a complex assignment, so build it up slowly. For instance, you should get Pong to work first without networking (if you are stuck, an example is provided at the book’s web site). You will also need to make changes to the server; specifically, the server will need to assign the players a paddle as they connect (left or right).

19.8 Serial Communication

A nice reward for learning the ins and outs of networked communication is that it makes serial communication in Processing a breeze. Serial communication involves reading bytes from the computer’s serial port. These bytes might come from a piece of hardware you purchase (a serial joystick, for example) or one that you design yourself by building a circuit and programming a microcontroller.

This book does not cover the external hardware side of serial communication. However, if you are interested in learning more about physical computing, I recommend the book Physical Computing: Sensing and Controlling the Physical World with Computers (Course Technology PTR) by Dan O’Sullivan and Tom Igoe as well as Making Things Talk: Practical Methods for Connecting Physical Objects (Make Books) by Tom Igoe. The Arduino (http://www.arduino.cc/) and Wiring (http://wiring.org.co/) web sites are also excellent resources. Wiring and Arduino are two open-source physical computing platforms developed at the Interaction Design Institute Ivrea with a programming language modeled after Processing. I will provide some accompanying Arduino code for your reference, but the material in this book will only cover what to do once the data has already arrived in Processing.
Serial communication refers to the process of sending data in sequence, one byte at a time. This is how data was sent over the network in our client/server examples. The Processing serial library is designed for serial communication into the computer from a local device, most likely via a USB (Universal Serial Bus) port. The term “serial” refers to the serial port, designed to interface with modems, that is rarely found on newer computers.

The process of reading data from a serial port is virtually identical to that found in the networked client/server examples, with a few exceptions. First, instead of importing the network library, we import the serial library and create a Serial object.

```plaintext
import processing.serial.*;
Serial port = new Serial(this, "COM1", 9600);
```

The Serial constructor takes three arguments. The first one is always “this,” referring to this applet (see Chapter 16). Argument 2 is a String representing the communications port being used. Computers label ports with a name. On a PC, these will likely be “COM1,” “COM2,” “COM3,” and so on. On UNIX-based computers (such as MAC OS X), they will be labeled “/dev/tty.something” where “something” represents a terminal device. If you are using a USB device, you will probably need to install USB drivers before a working port will be available. Instructions for how to get this working with Arduino can be found at the Arduino guide: http://www.arduino.cc/en/Guide/HomePage.

You can also print out a list of available ports using the Serial library’s list() function, which returns an array of String objects.

```plaintext
String[] portList = Serial.list();
println(portList);
```

If the port you want to use is the first one in the list, for example, your call to the constructor would look like:

```plaintext
String[] portList = Serial.list();
Serial port = new Serial(this, portList[0], 9600);
```

The third argument is the rate at which the data is transmitted serially, typically 9,600 baud.

Bytes are sent out via the serial port using the write() function. The following data types can be sent: byte, char, int, byte array, and String. Remember, if you are sending a String, the actual data sent are raw ASCII byte values of each character.

```plaintext
port.write(65); // Sending the byte 65
```

Data can be read with the same functions found in clients and servers: read(), readString(), and readStringUntil(). A callback function, serialEvent(), is triggered whenever a serial event occurs, that is, whenever there is data available to be read.

```plaintext
void serialEvent(Serial port) {
  int input = port.read();
  println("Raw Input: " + input);
}
```
The `read()` function will return a -1 if there is nothing available to read, however, assuming you are writing the code inside `serialEvent()`, there will always be available data.

Following is an example that reads data from the serial port and uses it to color the sketch’s background.

**Example 19-8: Reading from serial port**

```java
import processing.serial.*;

int val = 0; // To store data from serial port, used to color background
Serial port; // The serial port object

void setup() {
    size(200, 200);

    // In case you want to see the list of available ports
    // println(Serial.list());

    // Using the first available port (might be different on your computer)
    port = new Serial(this, Serial.list()[0], 9600);
}

void draw() {
    // Set the background
    background(val);
}

// Called whenever there is something available to read
void serialEvent(Serial port) {
    // Read the data
    val = port.read();
    // For debugging
    // println("Raw Input: " + input);
}
```

For reference, if you are using Arduino, here is some corresponding code:

```java
int val;

void setup() {
    beginSerial(9600);
    pinMode(3, INPUT);
}

void loop() {
    val = analogRead(0);
    Serial.print(val, BYTE);
}
```

This is not Processing code! It is Arduino code. For more about Arduino, visit: http://www.arduino.cc/.

### 19.9 Serial communication with handshaking

It is often advantageous to add a *handshaking* component to serial communication code. If a hardware device sends bytes faster than a Processing sketch can read, for example, there can sometimes be a logjam
of information, causing the sketch to lag. The sensor values may arrive late, making the interaction confusing or misleading to the user. The process of sending information only when requested, known as “handshaking,” alleviates this lag.

When the sketch starts up, it will send a byte to the hardware device asking for data.

---

**Example 19-9: Handshaking**

```cpp
void setup() {
  size(200, 200);

  // In case you want to see the list of available ports
  // println(Serial.list());

  // Using the first available port (might be different on your computer)
  port = new Serial(this, Serial.list()[0], 9600);

  // Request values from the hardware device
  port.write(65);
}
```

After the sketch finishes processing a byte inside of `serialEvent()`, it asks again for a new value.

```cpp
void serialEvent(Serial port) {

  // Read the data
  val = port.read();

  // For debugging
  // println("Raw Input: "+input);

  // Request a new value
  port.write(65);
}
```

As long as the hardware device is designed to only send the sensor values when requested, any possible lag will be eliminated. Here is the revised Arduino code. This example does not care what the request byte is, only that there is a byte request. A more advanced version might have different replies for different requests.

```cpp
int val;

void setup() {
  beginSerial(9600);
  pinMode(3, INPUT);
}

void loop() {
  // Only send out if something has come in
  if (Serial.available() > 0) {
    Serial.read();
    val = analogRead(0);
    Serial.print(val, BYTE);
  }
}
```

This is not *Processing* code! It is Arduino code. For more about Arduino, visit: [http://www.arduino.cc/](http://www.arduino.cc/).
19.10 Serial Communication with Strings

In cases where you need to retrieve multiple values from the serial port (or numbers greater than 255), the `readStringUntil()` function is handy. For example, let’s assume you want to read from three sensors, using the values for the red, green, and blue components of your sketch’s background color. Here, we will use the same protocol designed in the multi-user whiteboard example. We will ask the hardware device (where the sensors live) to send the data as follows:

Sensor Value 1 COMMA Sensor Value 2 COMMA Sensor Value 3 ASTERISK

For example:

104,5,76*

Example 19-10: Serial communication with Strings

```java
import processing.serial.*;

int r, g, b; // Used to color background
Serial port; // The serial port object

void setup() {
    size(200, 200);

    // In case you want to see the list of available ports
    // println(Serial.list());
    // Using the first available port (might be different on your computer)
    port = new Serial(this, Serial.list()[0], 9600);

    // Request values right off the bat
    port.write(65);
}

void draw() {
    // Set the background
    background(r, g, b);
}

// Called whenever there is something available to read
void serialEvent(Serial port) {
    // Read the data
    String input = port.readStringUntil('*');
    if (input != null) {
        // Print message received
        println("Receiving: " + input);
        // Split up the String into an array of integers
        int[] vals = int(splitTokens(input, ',', '*'));
        // Fill r, g, b variables
        r = vals[0];
g = vals[1];
b = vals[2];
    }

    // When finished ask for values again
    port.write(65);
}
```

Data from the Serial port is read in `serialEvent()` using the `readStringUntil()` function with "*" as the end character.

The data is split into an array of Strings with a comma or asterisk as a delimiter and converted into an array of integers.

Three global variables are filled using the input data.
Corresponding Arduino code:

```c
int sensor1 = 0;
int sensor2 = 0;
int sensor3 = 0;

void setup()
{
  beginSerial(9600);
  pinMode(3, INPUT);
}

void loop()
{
  if (Serial.available() > 0) //only send if you have hear back
  {
    Serial.read();
    sensor1 = analogRead(0);
    sensor2 = analogRead(1);
    sensor3 = analogRead(2);

    // Send the integer out as a String using "DEC"
    Serial.print(sensor1,DEC);
    Serial.print("", ',', BYTE);
    Serial.print(sensor2,DEC);
    Serial.print("", ',', BYTE);
    Serial.print(sensor3,DEC);
    // Send an asterisk -- ASCII code 42
    Serial.print("*", BYTE);
  }
}
```

Exercise 19-6: If you have an Arduino board, build your own interface to control a Processing sketch you have already made. (Before you attempt this, you should make sure you can successfully run the simple examples provided in this chapter.)