Imagine if you could not only display information from the Arduino on an external screen, but also send it out over the Internet for the world to see? What if you could control your Arduino remotely? Getting your Arduino on the Internet and remotely talking to your computer are just two of the many communication channels possible with the Arduino. In this article, a companion to the WiFi section of chapter 8 in *Arduino in Action*, which shows you how to use the official Arduino Wi-Fi shield, the authors discuss WiFi network and Bluetooth communication with the Arduino.

To save 35% on your next purchase use Promotional Code `mevans0835` when you check out at [www.manning.com](http://www.manning.com).

You may also be interested in…

**WiFi Communication and Arduino**

The Ethernet shield will get your Arduino online in no time, but surely there are times where being wireless would be useful. Perhaps you need to retrieve real-time data from a roaming rover-bot you’ve made or your project needs to get online but just isn’t near a wired router or connection. Enter the WiFly breakout, an elegant solution that will get your Arduino up and running on WiFi networks.

**WiFly Breakout**

The WiFly breakout by Sparkfun Electronics (figure 1) will enable your Arduino to connect to any 802.11b/g wireless network. It uses the Roving Network’s RN-131C wireless module and only requires a few wires to get up and running. Additionally the breakout possesses additional functionality such as a “pigtail” terminal to hook up external antennas when extended range is desired.
The RN-131C WiFi chip is highly configurable with an extensive list of commands and, once online, can even update its own firmware!

**WiFlySerial Library & Functions**

In order to begin using your WiFly breakout, you are going to need to install the WiFlySerial library at http://arduinology.tumblr.com/. The library handles all of the low-level networking communication for you, and also provides support for many of the commands and functionality provided by the WiFly break. It should be noted that the library also requires two additional libraries called Streaming and PString, which you can download from http://arduiniana.org/. Please download and install all three libraries, and then refer to table 1 for the main functions you will be using from the WiFlySerial library.

Table 8.5 Main WiFlySerial Library Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>begin()</td>
<td>Initialize the WiFlySerial library and begin communication with the WiFly device</td>
</tr>
<tr>
<td>join()</td>
<td>Join a network, either using the SSID (network name) set with setSSID() (top) or by providing it when joining (bottom)</td>
</tr>
<tr>
<td>leave()</td>
<td>Disconnect from current SSID</td>
</tr>
<tr>
<td>setSSID( const char* ssid)</td>
<td>Set the SSID (network name)</td>
</tr>
<tr>
<td>setIP(const char* ipAddress)</td>
<td>Set the IP address manually</td>
</tr>
<tr>
<td>setNetMask(const char* netmask)</td>
<td>Set the network mask manually</td>
</tr>
<tr>
<td>setDNS(const char* dns)</td>
<td>Set the DNS manually</td>
</tr>
<tr>
<td>setAuthMode(int authMode)</td>
<td>Manually set authorization (passphrase) encryption mode. Can be set with a number 0-6 or the key as per below</td>
</tr>
<tr>
<td></td>
<td>Open: 0 or WIFLY_AUTH_OPEN</td>
</tr>
<tr>
<td></td>
<td>WEP: 1 or WIFLY_AUTH_WEP</td>
</tr>
<tr>
<td></td>
<td>WPA1: 2 or WIFLY_AUTH_WPA1</td>
</tr>
<tr>
<td></td>
<td>Mixed-mode: 3 or WIFLY_AUTH_WPA1_2</td>
</tr>
<tr>
<td></td>
<td>WPA2-PSK: 4 or WIFLY_AUTH_WPA2_PSK</td>
</tr>
<tr>
<td></td>
<td>Ad-hoc: 6 or WIFLY_AUTH_ADHOC</td>
</tr>
<tr>
<td>setDHCPMode(const int dhcpMode)</td>
<td>Set DHCP on/off</td>
</tr>
<tr>
<td>openConnection(const char* serverIp)</td>
<td>Open or close a client connection to a server</td>
</tr>
<tr>
<td>closeConnection()</td>
<td>Start a server connection for clients</td>
</tr>
<tr>
<td>serveConnection()</td>
<td></td>
</tr>
</tbody>
</table>

Once you’ve reviewed table 1, you are ready to move on to the example project where you will be sending out gestural sensor data over a wireless network. This is only a shortlist of the libraries commands, and it is highly recommended to take a peek into WiFlySerial.h (from the WiFlySerial library) to discover the other calls available (including querying all available networks, sending special commands to the WiFly chip, and so on).
**Additional library notes: streaming**

As previously mentioned, the WiFlySerial library requires two additional libraries from www.arduiniana.org in order to work. One of these are particularly useful, and used in the example we are about to see. As such, in this article we will briefly go over what the Streaming library is, and why we take advantage of it in our example.

If the Streaming operators `<<` look familiar to you from coding in other languages, you’re in good company. If you’ve never seen them before, don’t worry; they only serve to simplify things! To concatenate and print a number of messages, we might write something like

```cpp
Serial.print("value1 is: "); Serial.println(value1);
```

This would first print the text (string) “value1 is: ”, followed by the actual value of the variable `value1` and then a new line.

While this is just fine, it requires two print statements or two lines of code. Imagine if you wanted to print ten or twenty different variables! As you can see, while it is just fine to do, it is perhaps not the most elegant solution.

Many programming languages provide streaming operators that make this type of concatenation easier. Instead of having to repeat blocks of code like the print statement above, you can use the streaming operators to concatenate all the lines or variables in one go. The example above would turn into

```cpp
Serial << "value1 is: " << value1 << endl;
```

Now, everything is concisely in one line. First you begin the stream by writing `Serial`, followed by the stream operator `<<`, and then each message or variable you wish to append (separated by the stream operator). To achieve a new line like `println`, simply end the stream with `endl`, which tells it to end the line. What’s great is that not only is this vary similar to streaming in other languages, but it can be used with any classes that inherit from `Print`, and it also does not require any more memory resources than your regular sketch! Now that you’re familiar with the libraries and code, let’s move on to the sensor so we can jump in and get things going!

**Gestures: wireless accelerometers**

In this example, were going to use a WiFly-powered Arduino to send sensor data from an accelerometer wirelessly over a network. Accelerometers are fantastic sensors allowing you to do all kinds of exciting gestural interactions. Perhaps you are interested in investigating new game-playing scenarios (as made popular by the Nintendo Wii and in many modern mobile devices), or putting accelerometers on a dancer’s body to take their motion and control live visuals and sound, or using accelerometers in assistive aid for the physically impaired. As you can see, there are many practical uses of wireless accelerometers and we’re sure you can think of more. In this example, you’re going to need an Arduino, the Sparkfun WiFly breakout board, and at least one accelerometer (for example the ADXL335). We will use Processing to create a server in which we can stream and parse the data from our wireless accelerometers.

**Connecting the hardware**

The WiFly breakout has a number of pins but only four are required to communicate with your Arduino. First, connect the TX (transmit) pin on the WiFly breakout to digital pin 2. Next, connect the WiFly RX pin to digital pin 3. Ground (GND) on the WiFly breakout should connect to GND on the Arduino. Finally, connect VBatt to the Arduino’s 3.3V line.

Connecting the accelerometer to your Arduino will differ depending on the particular model you are using. If you are using the ADXL335 tri-axis accelerometer, which provides an independent analog output for each of its axis (X,Y,Z), please refer to figure 2 when making your connections. If you are using another accelerometer model, the connections may be similar or may use PWM pins instead of the analog inputs. Refer to your model’s datasheet and documentation for proper connection if you are using another type of accelerometer.
NOTE Figure 2 does not show all of the pins on the WiFly breakout, only the pins required to connect to the Arduino. Please read the pin labels on the WiFly breakout carefully.

With the WiFly breakout and accelerometer connected you are ready to rock n’ roll. Let’s use the WiFlySerial library to get your Arduino sending the accelerometer data wirelessly to a server running on your computer. Wait! Server on your computer? How do we do that? We’re going to use a programming environment called Processing, so if you don’t already have it on your computer, now would be a good time to go to processing.org and download the latest version. If you haven’t used Processing before, don’t worry; the code in listing 2 will look very familiar to the code you’ve written before. But first, let’s get the Arduino side of things moving along.

**Listing 1 Arduino accelerometer client**

```c
#include <Arduino.h>
#include <Streaming.h>
#include <WiFlySerial.h>
#include <SoftwareSerial.h>

char ssid[] = "network_name";
char passphrase[] = "network_key";
char server_address[] = "192.168.0.1";
int server_port = 10000;

WiFlySerial wifly(2,3); #1

void setup() {
  Serial << "Starting…" << endl;
  Serial.begin(9600); #2
  wifly.begin(); #2
  connectToNetwork(); #3
}

void connectToNetwork(){
  wifly.setAuthMode(WIFLY_AUTH_WPA2_PSK);
  wifly.setDHCPMode(WIFLY_DHCP_ON);
}
```

For Source Code, Sample Chapters, the Author Forum and other resources, go to www.manning.com/mevans/
After importing the necessary libraries and supplying a few of the network and server details, you must create a WiFlySerial object (providing the pins on the Arduino it is connected to [#1]). We then enter our setup routine in which we initialize serial and the WiFly (#2), before attempting to open a network connection (#3) using a method we create called connectToNetwork. After configuring the network and server properties we attempt to join the WiFly network (#4), and then refresh our status (#5) so we can query if we successfully connected. Next we enter the main loop, and if we successfully connected to the network (#6), but are not yet connected to the server (#7), we attempt to connect to the server. Otherwise, if we are already connected to the server, we make sure the WiFly is not in command mode, and then we read and send the current accelerometer values (#8).

Next, we must jump into Processing and create a server that will accept incoming connections from clients (the Arduino) and display the messages it is receiving on the screen.

### Listing 2 Processing sketch to request accelerometer data from Arduino server

```
import processing.net.*;

int direction = 1;
boolean serverRunning = false;
String currentData = ""

Server myServer;

void setup()
{
  size(400, 400);

```
textFont(createFont("SanSerif", 16));
myServer = new Server(this, 10000); #1
serverRunning = true;
printData();
}

void printData() {
  background(0);
  text("wireless accelerometer data: ", 15, 25);
  text(currentData, 15, 60);
}

void draw()
{
  Client thisClient = myServer.available(); #2
  if (thisClient != null) { #2
    if (thisClient.available() > 0) { #2
      currentData = "message from: " + thisClient.ip() + " : " + #3
        thisClient.readString(); #3
      printData(); #3
    }
  }
}

#1 Instantiate server
#2 Checks for incoming clients
#3 Unpacks and display data

After setting up our window and font for displaying our data in our setup function, we instantiate our server (#1) to start listening for incoming clients. The draw function is similar to our main loop in Arduino, and it continuously checks for incoming clients (#2). If a client connects, we simply unpack the message (thisClient.readString()), append it with additional information such as the client’s IP address, and display it on our window (#3).

**Upload and test**

That’s it! Click the play button on the Processing sketch and the server should be up and running. Upload the Arduino code to your Arduino, wire it up as discussed earlier, and, as long as your network is properly configured, you should begin to see the accelerometer values displaying on your Processing program. If you have hunger for more, perhaps a good place to start would be to unpack the individual accelerometer data from the String you received, and visualize the three (X,Y,Z) axis on your display.

**NOTE** You may notice a few lines in the Arduino code in listing 2 that look like Serial << “some message” << endl. These are extra messages that print to a Serial console to help with debugging, and can be commented out (or deleted) when your code is “production” ready.

**Summary**

We touched on the broad spectrum of communications made possible with the Arduino. Because communicating with the Arduino wirelessly is useful in so many applications, we zeroed in on communication with your Arduino both over WiFi and Bluetooth technology.
Here are some other Manning titles you might be interested in:

- **Open-Source ESBs in Action**  
  Tijs Rademakers and Jos Dirksen

- **Open Source SOA**  
  Jeff Davis

- **Mule in Action, Second Edition**  
  David Dossot, John D’Emic, and Victor Romero

Last updated: November 6, 2012