

Programming Your Home

Automate with Arduino,
Android, and Your Computer



Mike Riley

Edited by Jacquelyn Carter

Programming Your Home

Automate with Arduino, Android, and Your Computer

by Mike Riley

Version: P1.0 (February 2012) Copyright © 2012 The Pragmatic Programmers, LLC.

This book is licensed to the individual who purchased it. We don't copy-protect it because that would limit your ability to use it for your own purposes. Please don't break this trust—you can use this across all of your devices but please do not share this copy with other members of your team, with friends, or via file sharing services.

Thanks.

—Dave & Andy.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and The Pragmatic Programmers, LLC

was aware of a trademark claim, the designations have been printed in initial capital letters or in all capitals. The Pragmatic Starter Kit, The Pragmatic Programmer, Pragmatic Programming, Pragmatic Bookshelf and the linking

g device are trademarks of The Pragmatic Programmers,

LLC.

Every precaution was taken in the preparation of this

book. However, the publisher assumes no responsibility for errors or omissions, or for damages that may result from the use of information (including program listings) contained herein.

Our Pragmatic courses,

workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at

<http://pragprog.com>.

This book is dedicated to Bill, Eileen, and Josie.

Table of Contents

- [Acknowledgments](#)
- [Preface](#)
 - [Who Should Read This Book](#)
 - [What's in This Book](#)
 - [Arduinos, Androids, and iPhones, Oh My!](#)
 - [Code Examples and Conventions](#)
 - [Online Resources](#)
- [**Preparations**](#)
 - [Getting Started](#)
 - [What Is Home Automation?](#)
 - [Commercial Solutions](#)
 - [DIY Solutions](#)
 - [Justifying the Investment](#)
 - [Setting Up Your Workbench](#)
 - [Sketching Out Your Ideas](#)
 - [Writing, Wiring, and Testing](#)
 - [Documenting Your Work](#)
 - [Requirements](#)
 - [Knowing the Hardware](#)
 - [Knowing the Software](#)
 - [Be Safe, Have Fun!](#)
- [**Projects**](#)
 - [Water Level Notifier](#)
 - [What You Need](#)
 - [Building the Solution](#)
 - [Hooking It Up](#)
 - [Sketching Things Out](#)
 - [Writing the Web Mailer](#)
 - [Adding an Ethernet Shield](#)
 - [All Together Now](#)
 - [Next Steps](#)
 - [Electric Guard Dog](#)
 - [What You Need](#)
 - [Building the Solution](#)

- [Dog Assembly](#)
- [Dog Training](#)
- [Testing It Out](#)
- [Unleashing the Dog](#)
- [Next Steps](#)
- [Tweeting Bird Feeder](#)
 - [What You Need](#)
 - [Building the Solution](#)
 - [The Perch Sensor](#)
 - [The Seed Sensor](#)
 - [Going Wireless](#)
 - [Tweeting with Python](#)
 - [Putting It All Together](#)
 - [Next Steps](#)
- [Package Delivery Detector](#)
 - [What You Need](#)
 - [Building the Solution](#)
 - [Hardware Assembly](#)
 - [Writing the Code](#)
 - [The Package Delivery Sketch](#)
 - [Testing the Delivery Sketch](#)
 - [The Delivery Processor](#)
 - [Creating the Delivery Database](#)
 - [Installing the Package Dependencies](#)
 - [Writing the Script](#)
 - [Testing the Delivery Processor](#)
 - [Setting It Up](#)
 - [Next Steps](#)
- [WebEnabled Light Switch](#)
 - [What You Need](#)
 - [Building the Solution](#)
 - [Hooking It Up](#)
 - [Writing the Code for the Web Client](#)
 - [Testing Out the Web Client](#)
 - [Writing the Code for the Android Client](#)
 - [Testing Out the Android Client](#)
 - [Next Steps](#)
- [Curtain Automation](#)
 - [What You Need](#)

- [Building the Solution](#)
 - [Using the Stepper Motor](#)
 - [Programming the Stepper Motor](#)
 - [Adding the Sensors](#)
 - [Writing the Sketch](#)
 - [Installing the Hardware](#)
 - [Next Steps](#)
- [Android Door Lock](#)
 - [What You Need](#)
 - [Building the Solution](#)
 - [Controlling the Android Door Lock](#)
 - [Writing the Android Server](#)
 - [Writing the Android Client](#)
 - [Test and Install](#)
 - [Next Steps](#)
- [Giving Your Home a Voice](#)
 - [What You Need](#)
 - [Speaker Setup](#)
 - [Giving Lion a Voice](#)
 - [Wireless Mic Calibration](#)
 - [Programming a Talking Lion](#)
 - [Conversing with Your Home](#)
 - [Next Steps](#)
- [Predictions](#)
 - [Future Designs](#)
 - [Living in the Near](#)
 - [The Long View](#)
 - [The Home of the Future](#)
 - [More Project Ideas](#)
 - [Clutter Detector](#)
 - [Electricity Usage Monitor](#)
 - [Electric Scarecrow](#)
 - [Entertainment System Remote](#)
 - [Home Sleep Timer](#)
 - [Humidity Sensor-Driven Sprinkler System](#)
 - [Networked Smoke Detectors](#)
 - [Proximity Garage Door Opener](#)
 - [Smart HVAC Controller](#)
 - [Smart Mailbox](#)

- [Smart Lighting](#)
- [Solar and Wind Power Monitors](#)
- **[Appendices](#)**
 - [Installing Arduino Libraries](#)
 - [Apple OSX](#)
 - [Linux](#)
 - [Windows](#)
 - [Bibliography](#)

Copyright © 2012, The Pragmatic Bookshelf

Praise for *Programming Your Home*

Mike has a broad technology experience base that puts all the pieces of some remarkable projects together. It's amazing that he makes it all so easy and affordable. Don't miss all that can be learned from this gem.

→ Michael Bengtson, Consultant

The Web-Enabled Light Switch project gave my family convenience and security options and enhanced my knowledge of RS-232 communications. It is nice to be able to switch on lights from my favorite chair. And the Tweeting Bird Feeder project has opened my eyes to the uses of radio communications around the home for things besides Wi-Fi, and it will help in my work to contribute to the preservation of bird species that are struggling for food and habitat.

→ Bob Cochran, Information Technology Specialist

With this book, Mike Riley celebrates the Arduino microcontroller in a way that both beginning and advanced home automation hobbyists will enjoy.

→ Sven Davies, Vice President of Applications

This is an outstanding reference that should be on the desk of every DIYer. In much the same way that software engineers mention "The Gang of Four Patterns Book," I predict this text will eventually be referred to as "The Riley Book of Home Automation."

→ Jon Kurz, President, Dycet, LLC

Every technology is only as exciting as the things you do with it. Mike takes a few cheap electronics parts, an Arduino, and a bit of code and turns your home into a much more exciting and enjoyable place. His easy-to-follow instructions make every single one of these projects both fun and useful.

→ Maik Schmidt, Software Developer, Author of *Arduino: A Quick-Start Guide*

I've had more fun learning new languages, systems, and gadgets with this book than any other book I've read!

→ James Schultz, Software Developer

Home automation is great fun, and *Programming Your Home* by Mike Riley will get you started right away. By leveraging this book and the easily available free/inexpensive hardware and software, anyone can tackle some great projects.

→ Tony Williamitis, Senior Embedded Systems Engineer

This is a fun and enthusiastic survey of electronic devices that can interact with the real world and that starts in your own home!

→ John Winans, Chief Software Architect

Acknowledgments

I have been a lifelong tinkerer. My earliest recollection of dissecting my father's broken tape recorder instilled an appreciation for the technology that drove it. From there, erector sets, model railroads, and programmable calculators led to personal computers, mobile devices, and microcontrollers. Over the years, this passion for learning not only how stuff works but also how technical concepts can be remixed with surprising, often highly satisfying results has been liberating. That's why this book was such a joy for me to write.

Helping others to see what's possible by observing their surroundings and having the desire to take an active role in making their lives easier with technology while having fun is this book's primary goal. Yet without others helping me distill my ideas into what you are reading now, this book would not have been possible. It is to them that I wish to express my deepest gratitude for their support.

A boatload of thanks goes to the book's editor, Jackie Carter, who spent countless hours ensuring that my words were constructed with clarity and precision. Copy editor Molly McBeath did a fantastic job catching hidden (from my view anyway) typos and grammatical misconstructions. Big thanks to Susannah Pfalzer for her infectious enthusiasm and boundless boosts of encouragement and to Arduino expert and fellow Pragmatic author Maik Schmidt, whose own success helped pave the way for a book like this.

Many thanks also go to John Winans, tech wiz extraordinaire, who refactored the state machine code used in several of the projects, as well as to Sven Davies, Mike Bengtson, Jon Bearscove, Kevin Gisi, Michael Hunter, Jerry Kuch, Preston Patton, and Tony Williamitis for helping to make this book as technically

accurate and complete as it is. Shout-outs also go to Jon Erikson and Jon Kurz for their enthusiastic encouragement. I also want to thank Bob Cochran and Jim Schultz for providing wonderfully helpful feedback during the book's beta period. Thanks also go to Philip Aaberg for filling my ears with music to code by. And to the makers of and contributors to the Arduino and Fritzing projects, you people have changed the world for the better.

I am most grateful to my wife, Marinette, and my family for allowing me to tunnel away for months in my mythical man cave to complete this book. And I can't gush enough over the wonderful pencil illustrations that my daughter drew for the book. I am so proud of you, Marielle!

Finally, I am sincerely thankful to Dave Thomas and Andy Hunt for their passion and vision. You're the best.

Mike Riley
<mailto:mike@mikeriley.com>
Naperville, IL, December 2011

Copyright © 2012 The Pragmatic Bookshelf.

Preface

Welcome to the exciting, empowering world of home automation! If you have ever wanted your home to do more than just protect you against the outside elements and want to interface it to the digital domain, this book will show you how. By demonstrating several easy-to-build projects, you will be able to take the skills you learned from this book and expand upon and apply them toward custom home automation projects of your own design.

The book's primary objective is to get you excited about the broader possibilities for home automation and instill the confidence you need to ultimately build upon these and your own ideas. The projects also make great parent-child learning activities, as the finished products instill a great sense of accomplishment. And who knows? Your nifty home automation creations may even change the world and become a huge new business opportunity for other homeowners actively seeking an automation solution that saves them time and money.

Who Should Read This Book

Programming Your Home is best suited to DIYers, programmers, and tinkerers who enjoy spending their leisure time building high-tech solutions to further automate their lives and impress their friends and family with their creations. Essentially, it is for those who generally enjoy creating custom technology and electronics solutions for their own personal living space.

A basic understanding of Arduino and programming languages like Ruby and Python are recommended but not required. You will learn how to combine these technologies in unique configurations to resolve homemaker annoyances and improve home management efficiencies.

In addition to the inclusion of Python scripts and Ruby on Rails-based web services, several of the projects call upon Google's Android platform to help enhance the data event collection, visualization, and instantiation of activities. A basic familiarity with the Android SDK will be beneficial so that the projects that make use of the Android OS can offer a more mobile reach.

If you're the type of person who prefers to build versus buy your home accessories, this book will further motivate you to use what you learned in the book as a starting point to expand upon and optimize them in various ways for their environment. Even though some of the topics deal with multiple software- and hardware-based solutions, they are easy to follow and inexpensive to build. Most of all, they show how a few simple ideas can transform a static analog environment into a smart digital one while having fun.

What's in This Book

After a basic introduction to home automation and the tools of the trade, this book will teach you how to construct and program eight unique projects that improve home utility and leisure-time efficiencies. Each project incorporates a variety of inexpensive sensors, actuators, and microcontrollers that have their own unique functions. You will assemble the hardware and codify the software that will perform a number of functions, such as turning on and off power switches from your phone, detecting package deliveries and transmitting emails announcing their arrival, posting tweets on Twitter when your bird feeder needs to be refilled, and opening and closing curtains depending on light and temperature, and more.

Because the recommended skill set for building these solutions includes some familiarity with programming, this book builds upon several previously published Pragmatic Bookshelf titles. If you would like to learn more about programming Arduinos or writing Ruby or Python scripts, I strongly recommend checking out the books listed in Appendix 2, [Bibliography](#).

Each project begins with a general introduction and is followed by a What You Need section that lists the hardware parts used. This is followed by a section called Building the Solution that provides step-by-step instructions on assembling the hardware. *Programming Your Home* will call upon the Arduino extensively for most (but not all) of the projects. Once the hardware is constructed, it can be programmed to perform the automation task we built it to do. Programs can range from code for Arduino microcontrollers to scripts that execute on a computer designed to control, capture, and process the data from the assembled hardware elements.

The book concludes with a chapter on future projections in home automation and a chapter filled with idea starters that reuse the hardware and software approaches demonstrated in the eight projects.

Arduinos, Androids, and iPhones, Oh My!

With the meteoric rise of mobile device proliferation, the post-PC moniker has made its way into the tech world's vocabulary. I am a big proponent of technology shifts, but I am also old enough to have lived through three major computing revolutions (the shift from mainframes to PCs, the rise of the Internet, and the shift from PCs to mobile devices) and know that change isn't as fast as people say it is. Until mobile applications can be developed on mobile devices the way PC applications can be developed on PCs, a Linux, Windows, or Mac computer will be a central requirement for developing mobile apps. The same holds true for Arduino programming.

That said, the times are indeed a-changing. Microsoft Research was one of the first major phone OS providers to attempt to create native mobile applications directly on the mobile device with their release of TouchStudio. Google engineer Damon Kohler created the Scripting Layer for Android (SL4A) that gives Android users the ability to write fairly sophisticated programs using a text editor on their phone. Coupled with Sparkfun's IOIO ("yo-yo") board, we're already seeing early glimpses of what could replace the PC for some of the scripts created for this book.

Since you will need a Mac, Linux, or Windows computer to program the Arduinos and mobile apps in this book, this computer will also be the machine that runs the server-side programs that interpret and extend information out to your mobile devices. Of course, if you only have one computer and it's a laptop that travels with you, consider purchasing a cheap Linux or Mac to run as your home server. Not only will you benefit from having a dedicated system to run the monitoring apps 24/7/365, but it can also serve as your home Network Attached Storage (NAS) server as well.

I am a believer in open source hardware and software. As such, the projects in the book depend upon these. I am also technology-agnostic and rarely have any overriding devotion to one hardware supplier or programming language. Code for this book could have been presented just as easily in Mono-based C# and

Perl, but I opted for Ruby and Python because of their portability and multiparty open source support. I could have used a Windows or Linux machine as the server and development system but chose Mac for the book because Ruby and Python are preinstalled with the OS, thereby eliminating the time and space required to install, configure, and troubleshoot the operating environment.

In accordance with this open source philosophy, I also opted to demonstrate the mobile application examples exclusively for the Android OS. While I personally prefer iOS devices as the platform of choice for my mobile lifestyle, the overhead associated with writing applications for iOS is a hassle. In addition to learning Objective-C and the various frameworks as well as dealing with the burden of memory management, deploying iOS apps requires either a jailbroken device or the legitimate purchase of an annual membership to Apple's iPhone developer network. Conversely, Android's SDK and application deployment is free and open. Android programs can also multitask better than iOS programs. Of course, these two advantages also bring on greater security and resource utilization risks. That said, I encourage readers who prefer the mobile demos to run on non-Android devices to port the simple client programs presented in this book to their favorite mobile OS and share these conversions with the Programming Your Home community.

Another term that is gaining a foothold in the tech press is the "Internet of Things." This phrase refers to the idea that with the proliferation of network-connected microcontrollers, Internet-based communication between such small devices will eventually outnumber people surfing the Web. While that may be the case for submitting data upstream, reaching such a device from the Internet is still a hassle. Besides the technical knowledge required to set up a dynamic DNS and securely configure port forwarding to easily reach the device, ISPs may block outbound ports to prevent end consumers from setting up dedicated servers on popular network ports like FTP, HTTP/S, and SMTP.

The projects in this book should work perfectly fine in a home local area network. However, obtaining sensor data outside of this local network is a challenge. How do you check on the status of something like a real-time temperature reading without going through the hassles of opening and

forwarding ports on your router (not to mention the potential security risks that entails)?

Fortunately, several companies have begun to aggressively offer platforms accessible via simple web service APIs to help overcome these hassles. Three of these gaining momentum are Pachube, Exosite, and Yaler.[\[1\]](#) Configuring and consuming their services is a fairly straightforward process. I encourage you to visit these sites to learn more about how to incorporate their messaging capabilities into your own projects.

Code Examples and Conventions

The code in this book consists of C/C++ for Arduino, Java for Android, Ruby for web middleware, and Python for desktop scripts. Most of the code examples are listed in full, except when burdened by external library overhead (such as in the case of Android and Ruby on Rails program listings). Syntax for each of these languages is highlighted appropriately, and much of the code is commented inline along with bullet markings to help bring attention to the big ideas in the listings.

Highlights and sidebars are used sparingly in the book in an effort to keep information moving at a quick yet manageable clip.

Online Resources

Visit the book's website at <http://pragprog.com/titles/mrhome>, where you can download the code for all the projects, participate in the book's discussion forum, ask questions, and post your own home automation ideas. Bugs, typos, omissions, and other errors in the book can be found on the book's errata web page.

Other popular website resources include the popular DIY websites Makezine, and Instructables,[2] where participants share a wide variety of home-brewed creations with their peers.

There are also several IRC channels on freenode.net and SIG forums on Google Groups dedicated to the subject, with many focused on singular aspects of DIY gadget design, home automation, and hardware hacking.[3]

OK, enough with the preamble. Let's get ready to build something!

Footnotes

[1] <http://www.pachube.com>, <http://www.exosite.com>, and <http://www.yaler.org>, respectively.

[2] <http://www.makezine.com> and <http://www.instructables.com>, respectively.

[3] <http://groups.google.com/group/comp.home.automation/topics>

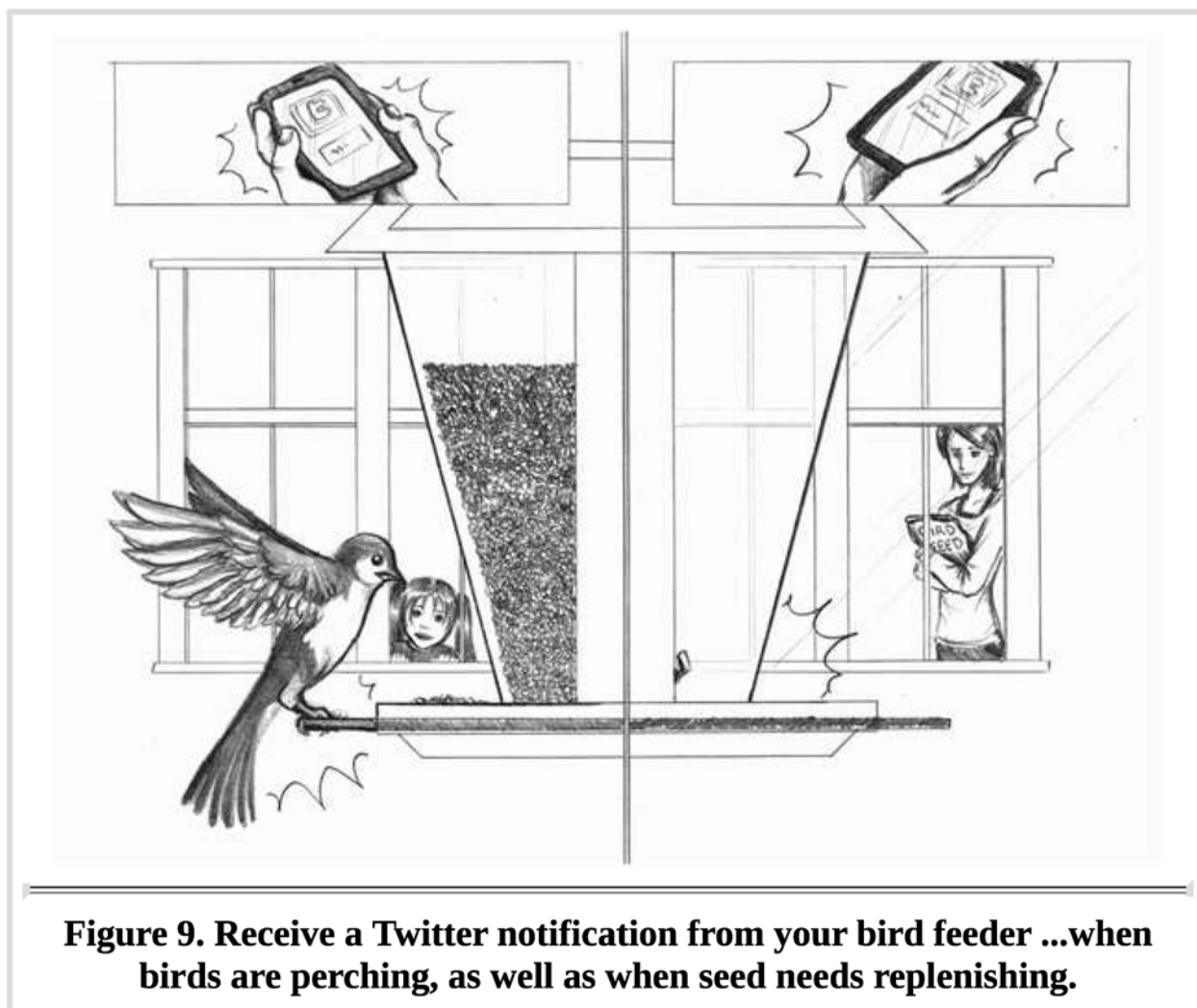
Chapter 5

Tweeting Bird Feeder

Both of my kids are bird lovers. They have had parakeets since they were toddlers and enjoy watching wild birds nest and feed outside their bedroom windows. But one of the chores that somehow always slips past us is refilling the feeders with birdseed. For a variety of reasons, there may be days, sometimes even weeks, that go by without a refill. Wouldn't it be so much easier for the feeder to tell us when it needed to be refilled?

That need was the genesis of inspiration for this project, and what better way to receive the notification than via a tweet on Twitter. Interested friends and family members can follow the feeder and know when birds are feeding from it, when it needs a refill, and when the refill chore has been satisfied. (See Figure 9, [*Receive a Twitter notification from your bird feeder*](#).) Since we will already be tracking the refilling patterns via Twitter, let's make the feeder broadcasts even more interesting by adding a homemade sensor on the feeder perch that will record when birds are enjoying a meal and for how long. Before posting the tweets, we will timestamp and record these events to a database so we can visualize feeding patterns over time.

Was April a more ravenous month for the birds compared to July? Are early mornings a busier time of day than late afternoons? What is the average time birds perch at the feeder? What are the time intervals between perches? How frequently does the feeder need to be refilled with seed? With the Tweeting Bird Feeder, you will be able to take on the role of field researcher to discover these and other feeding behavior questions. It's time to fly!



5.1 What You Need

Since this will be our first outdoor project, the equipment costs are more expensive for several reasons. First, unless you are willing to drill holes through your walls or leave a window or door open to run an Ethernet cable to the feeder, we will need an untethered way to broadcast sensor events. Fortunately, a low-power and relatively low-cost option exists in the form of XBee radios. While these take a little extra effort to configure initially, they are fairly reliable, easy to communicate with, and don't require much attention once operational.

Second, while we could use the standard size Arduino Uno (as shown in the wiring diagrams throughout this chapter), it might prove to be too long and/or too wide to conveniently fit into a typical bird feeder. Consequently, I recommend spending a few extra dollars on an Arduino Nano. The Nano is better suited to match the feeder's space constraints. The nice thing about the Nano is that the pin configurations and the hardware are nearly identical to that of its bigger brother, and the Nano delivers all that Arduino goodness in a package within a much smaller footprint.



Figure 10. Tweeting Bird Feeder parts

Third, while it is feasible to power the electronics via a long extension cord plugged into an outdoor outlet akin to a holiday lighting scenario, such a configuration is not a self-contained system. Besides, it would be appropriate to incorporate a greener energy option to be kinder to our environment.

Finally, due to the need to protect the electronics from the elements, we will need to do a good job weatherizing our assembly. Here's the shopping list (refer to the photo in Figure 10, [Tweeting Bird Feeder parts](#)):

1. An Arduino Uno or an Arduino Nano[\[43\]](#)
2. Two XBee radios with adapter kits and FTDI connector cable[\[44\]](#)
3. A photocell

4. A strip of aluminum foil
5. A piece of wire
6. A small solar panel with built-in rechargeable battery and USB connector, such as those provided by Solio[\[45\]](#)
7. One 10k ohm resistor and one 10M ohm resistor—verify that the color bands on the resistors are brown, black, orange, and gold for the 10k ohm and brown, black, blue, and gold for the 10M ohm resistors. Refer to Figure 11, [Tweeting Bird Feeder resistors](#). Also shown in the photo is the photocell (also referred to as a CdS photoresistor).



Figure 11. Tweeting Bird Feeder resistors

8. A bird feeder with a large enough seed cavity to house the weatherized Nano and XBee assembly
9. A computer (not pictured), preferably Linux or Mac-based, with Python 2.6 or higher installed to process incoming messages from the bird feeder

If you opt to use the Arduino Nano in place of the Arduino Uno, you will also need a standard A to Mini-B USB cable (not pictured) to connect the Arduino Nano to the computer. Additionally, since the Arduino Nano uses male pins for wiring connections instead of the female headers found on an Arduino Uno, you will need female jumper wires (not shown) instead of standard wires. This will allow you to more easily attach wires to the Nano's pins without having to solder

the wiring connections in place.

This project is a more complex than the Water Level Notifier, and getting the XBee radios working reliably is the trickiest part. Still, it's worth the effort since you will not only have a cool twenty-first-century high-tech bird feeder, but you will also be able to reuse the XBee radio setup in several other projects. Ready to roll up your sleeves? Then let's get to it!

5.2 Building the Solution

Assembling the hardware to fit snugly inside the feeder may require some ingenuity, especially if the bird feeder doesn't offer much space inside the seed container. Before we start cramming electronics into the feeder, we first need to make sure our components work as expected.

1. We will start with the easy part of connecting the aluminum foil capacitive sensor to the Arduino and writing a function that will send a message to the serial window (and eventually the serial port of the attached XBee radio) when the sensor is triggered.
2. Next, we will hook up the photocell to the Arduino and write the code for it to react to changes in light.
3. Then we will pair the two XBee radios and transmit these events from the XBee radio attached to the Arduino to the other XBee radio tethered to a computer via an FTDI USB cable.
4. And finally, we will write a Python script that will capture the data into an SQLite database and format and transmit the messages to be posted on Twitter.

Once everything is working, we will compact the Arduino (preferably the Nano)+XBee+perch resistor+photocell assembly into a weatherized package, house it in the feeder, fill the feeder with seed, and go outside for a live field test.

5.3 The Perch Sensor

Bird feeders come in different shapes and sizes. I opted to go with a low-tech solution for determining when a bird lands on the feeder perch. While it is certainly possible to construct an elaborate pressure switch mechanism, the time and expense required to implement it seems like a lot of work just to detect when something grips the perch. Instead, we can monitor fluctuations in electrical capacitance.

Cover the feeder perch with aluminum foil, attach a wire from the foil to a resistor connected to the digital pins on the Arduino. By measuring baseline values and fluctuations detected when a bird lands on this sensor, we can establish a threshold value to determine when a landing message should be transmitted.

Building the Sensor

Building and testing the perch sensor is the easiest part of this project. Take a piece of aluminum foil, flatten it to half the size of a gum stick wrapper, and wrap it across the bird perch. Then take a 10M ohm resistor and insert one end into the Arduino's digital pin 7 and the other end in digital pin 10. Then connect a wire from the foil to the resistor lead that is connected to the Arduino's digital pin 7. For the wiring diagram, refer to Figure 12, [*Wiring up a capacitive sensor*](#).

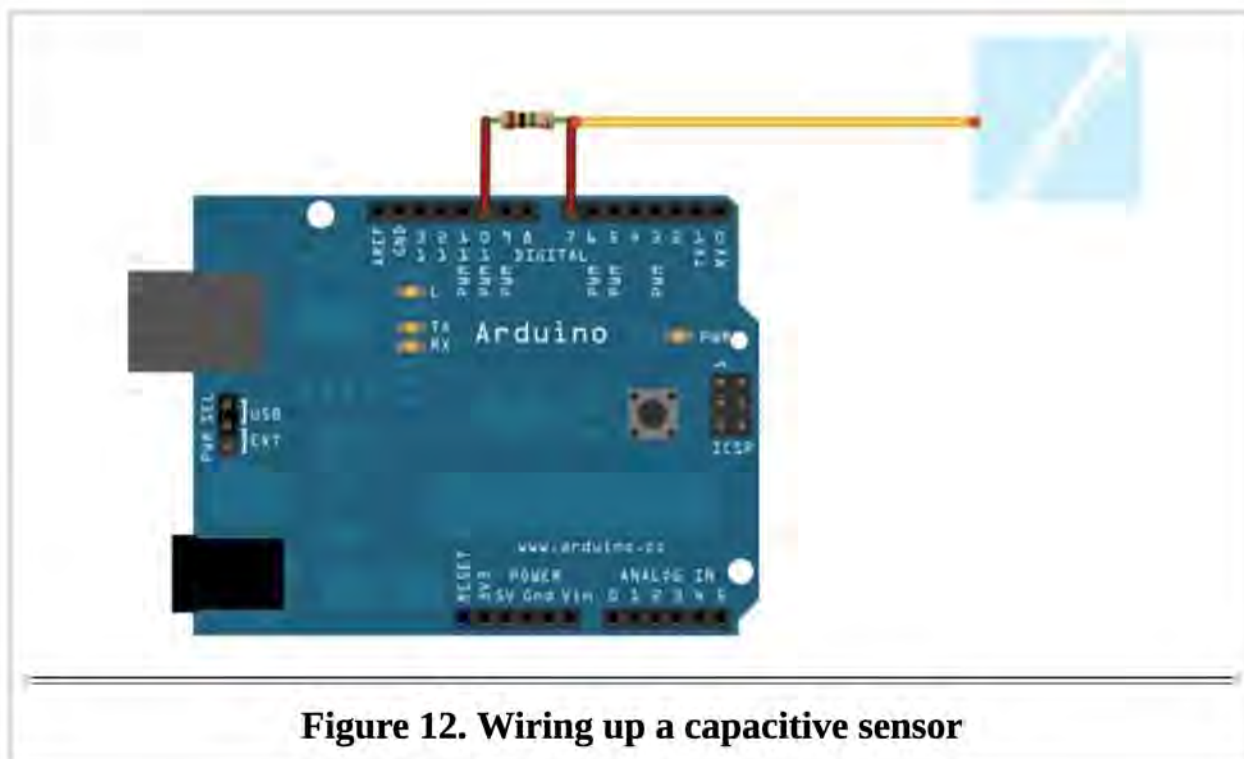


Figure 12. Wiring up a capacitive sensor

Programming the Sensor

Connect the Arduino to your computer and fire up the Arduino IDE to write the sensing code. Like the Water Level Notifier project, we will write code that we will use to do the following:

1. Display the capacitive sensor's electrical values to the Arduino IDE's serial monitor window.
2. Identify the baseline electrical value of the sensor.
3. Alter the flow of current when placing a finger on the sensor.
4. Record the new value to use for our alert trigger condition.

In order to more easily and programmatically detect the electrical changes that occur when something like a finger or a bird lands on the foil, we will call upon the help of Arduino enthusiast Paul Badger. Paul wrote an Arduino library that makes measuring changes in capacitive sensors like the foil one we constructed for this project a breeze. Called the Capacitive Sensing library,[\[46\]](#) the library

gives Arduino programmers the ability to turn two or more Arduino pins into a capacitive sensor that can be used to sense the electrical capacitance of a body. A human body is considerably larger than a bird and will therefore create a much larger deflected value. Nevertheless, a bird also has measurable electrical capacitance, and that is the threshold value we will attempt to refine in our program.

Download this library, uncompress its contents, and copy it into your Arduino's `libraries` folder. For more details, refer to Appendix 1, [Installing Arduino Libraries](#).

Create a new Arduino project and use the `#include CapSense.h`;

Due to the difference in size and surface area touching the foil, a bird will have a very different value than that of a person. If possible, measure the value difference with a bird. Fortunately my kids have pet parakeets, and these birds were all too eager to be test subjects in exchange for the seed supplied by the feeder. My test measurements concluded that the baseline value varied between 900 and 1400, and the bird's capacitance increased that value to more than 1500. Using these values, we can use the same type of conditional code from the Water Level Notifier project to raise and reset the bird landing and departure notifications.

We will write the code that will load the CapSense library and capture and display the capacitive values to the serial monitor window.

`TweakingBirdFeeder/BirdPerchTest.gd`

```
#include <CapSense.h>

#define ON_PERCH 1500
#define CAP_SENSE 30
#define ONBOARD_LED 13

CapSense foil_sensor = CapSense(10,7); // capacitive sensor
                                   // resistor bridging digital pins 10 and 7,
                                   // wire attached to pin 7 side of resistor

int perch_value = 0;
byte perch_state = 0;
```

```

void setup()
{
    // for serial window debugging
    Serial.begin(9600);

    // set pin for onboard led
    pinMode(ONBOARD_LED, OUTPUT);
}

void SendPerchAlert(int perch_value, int perch_state)
{
    digitalWrite(ONBOARD_LED, perch_state ? HIGH : LOW);
    if (perch_state)
        Serial.print("Perch arrival event, perch_value=");
    else
        Serial.print("Perch departure event, perch_value=");
    Serial.println(perch_value);
}

void loop() {
    // wait a second each loop iteration
    delay(1000);

    // poll foil perch value
    perch_value = foil_sensor.capSense(CAP_SENSE);

    switch (perch_state)
    {
    case 0: // no bird currently on the perch
        if (perch_value >= ON_PERCH)
        {
            perch_state = 1;
            SendPerchAlert(perch_value, perch_state);
        }
        break;

    case 1: // bird currently on the perch
        if (perch_value < ON_PERCH)
        {
            perch_state = 0;
            SendPerchAlert(perch_value, perch_state);
        }
        break;
    }
}

```

```
}
```

Note the defined `ON_PERCH` value of 1500 to compare against the recorded `perch_value`. Due to variations in the conductivity and surface area of your foil sensor, you may need to tweak the `ON_PERCH` threshold value just like you did for the Water Level Notifier project so that it works best for your configuration. Also note the value of 30 assigned to the `CAP_SENSE` constant. This is the number of samples to poll during the capacitive measurement cycle.

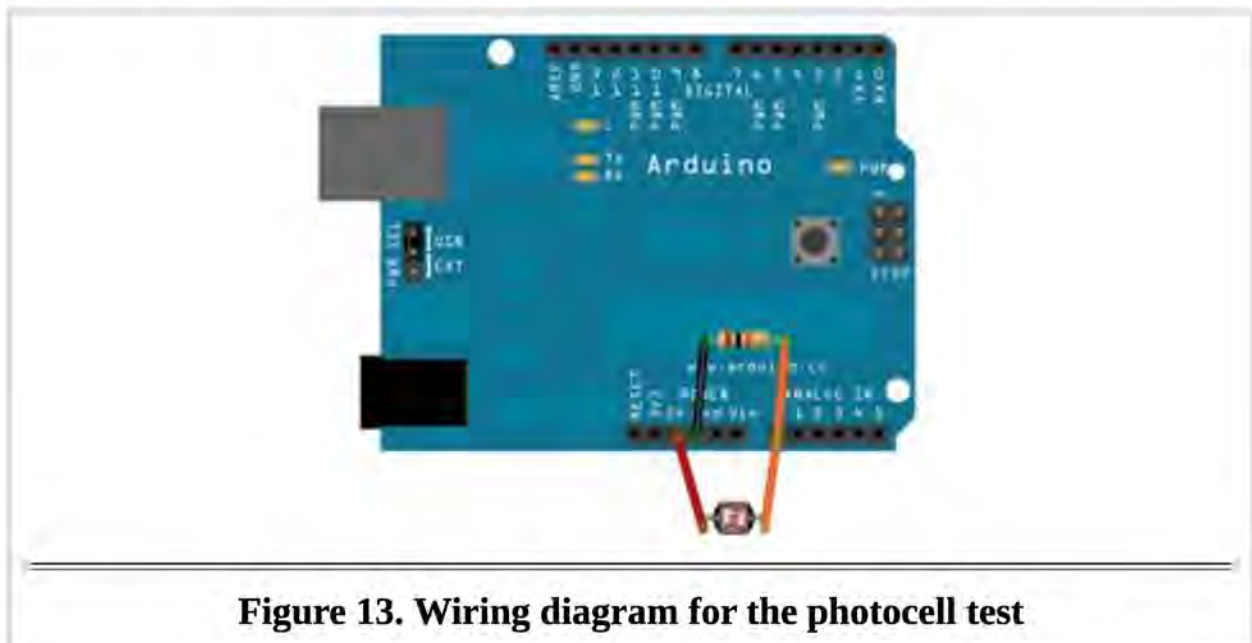
Now that we have a working bird perch sensor, we need a way for the feeder to alert us when it is running low on seed. How will we do this? A photocell can help.

5.4 The Seed Sensor

A photocell measures light intensity, with higher intensity correlating with higher current and lower intensity with lower current. For a more detailed explanation and tutorial on photocells, visit Ladyada's ever-helpful website.[\[47\]](#) By placing the photocells at a level beneath the seeds poured into the feeder, we can detect when the seed level dips below the sensor, exposing it to more light and thereby alerting us that the feeder needs to be refilled.

Before drilling holes into the feeder for placement of the photocell, we need to write some code and test it using the same approach that we did for our homemade foil resistor.

Connect one of the photocell leads to the Arduino 5v pin and connect the other photocell lead to the Arduino analog pin 0. Then bridge a 10k ohm resistor between the Arduino analog pin 0 and the Arduino ground pin, as shown in Figure 13, [Wiring diagram for the photocell test](#). Does this electrical pattern look familiar? Yep, it's the same wiring configuration used previously with Arduino sensors in this book. This is a frequent pattern for various types of sensors that connect with the Arduino.



With the photocell connected, connect the Arduino to the computer via the USB serial cable and launch the Arduino IDE. Using the same technique used for the foil switch, monitor analog pin 0 values in the Arduino IDE's serial monitor window and capture baseline values for when the sensor is bathed in ambient, standard lighting conditions. Then cover the sensor with your finger to block incoming light. Note the difference in value.

Just as we did for the capacitive test, we will write the same type of procedures and conditional statements to test for luminosity thresholds. Indeed, you could copy and paste code from the foil test and simply change variable names and connected pin assignments to create the working program.

[TweetingBirdFeeder/SeedPhotocellTest.pde](#)

```
#define SEED          500
#define ONBOARD_LED   13
#define PHOTOCELL_SENSOR 0
int seed_value = 0;
byte seed_state = 0;
void setup()
{
    // for serial window debugging
    Serial.begin(9600);

    // set pin for onboard led
    pinMode(ONBOARD_LED, OUTPUT);
}

void SendSeedAlert(int seed_value, int seed_state)
{
    digitalWrite(ONBOARD_LED, seed_state ? HIGH : LOW);
    if (seed_state)
        Serial.print("Refill seed, seed_value=");
    else
        Serial.print("Seed refilled, seed_value=");
    Serial.println(seed_value);
}

void loop() {
    // wait a second each loop iteration
    delay(1000);
```

```

// poll photocell value for seeds
seed_value = analogRead(PHOTOCELL_SENSOR);

switch (seed_state)
{
case 0: // bird feeder seed filled
    if (seed_value >= SEED)
    {
        seed_state = 1;
        SendSeedAlert(seed_value, seed_state);
    }
    break;

case 1: // bird feeder seed empty
    if (seed_value < SEED)
    {
        seed_state = 0;
        SendSeedAlert(seed_value, seed_state);
    }
    break;
}
}

```

Measuring and assigning the defined `SEED` threshold value for the photocell is much easier and more reliable than the capacitive foil test we did earlier. While you can use your finger to cover up the photocell and measure the value change, it is more authentic to test with real seed. If you don't want to drill holes in your bird feeder to set the photocell just yet, use a paper cup and place the photocell toward the bottom of the cup.

Similar to the calibration procedure we used in the Water Level Notifier project, add these lines after the `seed_value = analogRead(PHOTOCELL_SENSOR);` request in the sketch's main program loop:

```

Serial.print("seed_value=");
Serial.println(seed_value);

```

Record the `seed_value` starting value, then fill the cup with seed and measure the new value. Use these values to set the starting and threshold values for the photocell.

If you do not see any change, check your wiring and try again. In my tests with the photocell, the baseline value fluctuated between 450 and 550. It reported below 100 when my finger covered the sensor. Use whatever upper and lower limit values you recorded with your tests, keeping in mind that they will need to be recalibrated once the sensor is mounted inside the feeder.

Now that monitoring is working for both the perch and light sensors, we need a way to communicate when those sensor thresholds have been exceeded. It's not very practical to run an Ethernet cable from an indoor network hub to an outdoor tree limb. Not to mention that trying to fit a bulky Arduino+Ethernet shield assembly into the confined space of a bird feeder would be a challenge. We will use the convenience of low-power wireless communication to transmit these sensor notifications to an indoor computer. Then we will use that computer's faster processing and larger storage capacity to analyze and act upon the data received.

5.5 Going Wireless

Although more ubiquitous 802.11b/g Wi-Fi shields exist for the Arduino (such as Sparkfun's WiFly Shield), the most prevalent means of Arduino wireless communication is via XBee radios. The initial outlay for a set of XBee devices can be a bit pricey. This is because in addition to the XBee radios, you also need an FTDI USB cable to connect one of the radios to a computer to act as a wireless serial port.

The other XBee is typically connected to an Arduino. And to make it easier to make these connections, additional kits are available to mount the XBee radios to an assembly that better exposes the connection pins while also displaying data transfers via onboard LEDs. Such visual indicators can be quite helpful when debugging or troubleshooting a paired XBee connection. Nevertheless, the fact that XBees offer a low-power and long-range (up to 150 feet) solution coupled with easy connectivity and data transfer protocol make them an ideal wireless technology for our project needs.

In order to more easily interface with the XBee radios, Adafruit has designed an adapter kit that “doesn’t suck,” but it does require you to solder a few small components onto the adapter board. Assemble the adapter following the instructions posted on Ladyada’s website.[\[48\]](#)

Once assembled, configuring and pairing the XBee radios isn’t too difficult, though one of the more helpful utilities used to configure them only runs on Windows.

Using the screen Utility

A helpful serial monitoring utility for *nix-based platforms like Mac OS X and Linux is the `screen` application. To use `screen` in OS X, determine the serial port that the FTDI USB cable is using by launching the Arduino IDE. Select Serial Port from the Arduino IDE Tools menu to identify the serial port assigned to it.

In my case, the device name of the FTDI USB-to-XBee adapter connection was `devtty.usbserial-A6003SHc`, though yours may be different, depending on other devices connected to your computer. Open up the `Terminal` application and type

`screen devtty.YOURDEVICE 9600`. This will open the serial port and allow you to enter and receive characters at a communication rate of 9600 baud. To gracefully exit the `screen` utility, press Control-A followed by Control-\.

Following the instructions for the XBee point-to-point sample program available on Ladyada's web page,[\[49\]](#) attach the power, ground (Gnd), receive (RX), and transmit (TX) pins of one of the adapter-mounted XBees to the Arduino's 5V, Gnd, digital 2, and digital 3 pins. Plug the Arduino into your computer, upload the test program, open up the Arduino serial window, and make sure it is set to a rate of 9600 baud. Leave the Arduino plugged in and attach the other paired XBee to your computer via the FTDI USB cable. Open up a serial terminal session: Hyperterminal in Windows, the `screen` on a Mac, or various serial communication programs such as Minicom for Linux.[\[50\]](#)

Once you have established a serial connection, type in a few characters in your serial application's input screen. If your XBee radios are properly configured, you will see those typed characters appear in the Arduino serial window. If you connected the XBees to the Arduino and the FTDI cable using XBee adapters,

you should also see the adapter's green transmit and red receive LEDs blinking as the characters are wirelessly transmitted from one XBee radio to the other.

If you do not see the characters being displayed on the receiving window, review the XBee radio wiring to the appropriate Arduino ports. You can also swap the radios to verify both are recognized using the FTDI USB connection. Type an AT in your serial application's terminal window and verify that it returns an OK acknowledgment.

If the XBees still don't seem to be communicating with one another, contact the retailer you purchased them from for further assistance.

With the XBee radios successfully paired, we will reconnect both the photocell and foil resistor to the XBee-outfitted Arduino and combine the code that will receive the sensor's trigger condition events. For the complete wiring diagram,

refer to Figure 14, [*Tweeting bird feeder with sensors and XBee radio attached to Arduino.*](#)

Consider using a breadboard, or solder the resistors along with the wires being connected to the Arduino pins. If you opt for a breadboard for testing purposes, keep in mind that it probably won't fit in the feeder, so be prepared to solder the wiring permanently in place after the tests prove successful. And depending on the orientation of the Arduino Uno or Nano as it is positioned into the bird feeder, you may need to use straight header pins on the XBee adapter board instead of the default right-angled pins that accompany the kit. The goal is to make sure everything fits inside the feeder in a secure and serviceable way. And

recall that unlike the female headers on an Arduino Uno, the Arduino Nano uses male pins instead. As such, you will need to use female jumper wires to better accommodate the connections to the Nano's male pins.

Finishing the Sketch

We are going to poll both the foil and photocell sensors for activity, first to the Arduino IDE serial window, then—with one small replacement in our code—to the XBees. Essentially, we are simply going to combine the threshold code for the foil and photocell that we tested earlier while sending threshold alerts to the serial port of the XBee radio. Adding this logic to what we already wrote to test the perch and seed status, we can complete the sketch for the project.

TweeingBirdFeeder/TweeingBirdFeeder.pde

```
#include <CapSense.h>;
#include <NewSoftSerial.h>

#define ON_PERCH      1500
#define SEED          500
#define CAP_SENSE     30
#define ONBOARD_LED   13
#define PHOTOCELL_SENSOR 0
// Set the XBee serial transmit/receive digital pins
NewSoftSerial XBeeSerial = NewSoftSerial(2, 3);
CapSense foil_sensor     = CapSense(10,7); // capacitive sensor
                                   // resistor bridging digital pins 10 and 7,
                                   // wire attached to pin 7 side of resistor

int perch_value  = 0;
byte perch_state = 0;
int seed_value   = 0;
byte seed_state  = 0;

void setup()
{
    // for serial window debugging
    Serial.begin(9600);

    // for XBee transmission
    XBeeSerial.begin(9600);

    // set pin for onboard led
    pinMode(ONBOARD_LED, OUTPUT);
```

```

}

void SendPerchAlert(int perch_value, int perch_state)
{
    digitalWrite(ONBOARD_LED, perch_state ? HIGH : LOW);
    if (perch_state)
    {
        XBeeSerial.println("arrived");
        Serial.print("Perch arrival event, perch_value=");
    }
    else
    {
        XBeeSerial.println("departed");
        Serial.print("Perch departure event, perch_value=");
    }
    Serial.println(perch_value);
}

void SendSeedAlert(int seed_value, int seed_state)
{
    digitalWrite(ONBOARD_LED, seed_state ? HIGH : LOW);
    if (seed_state)
    {
        XBeeSerial.println("refill");
        Serial.print("Refill seed, seed_value=");
    }
    else
    {
        XBeeSerial.println("seedOK");
        Serial.print("Seed refilled, seed_value=");
    }
    Serial.println(seed_value);
}

void loop() {
    // wait a second each loop iteration
    delay(1000);

    // poll foil perch value
    perch_value = foil_sensor.capSense(CAP_SENSE);

    // poll photocell value for seeds
    seed_value = analogRead(PHOTOCELL_SENSOR);

    switch (perch_state)

```

```

{
  case 0: // no bird currently on the perch
    if (perch_value >= ON_PERCH)
    {
      perch_state = 1;
      SendPerchAlert(perch_value, perch_state);
    }
    break;

  case 1: // bird currently on the perch
    if (perch_value < ON_PERCH)
    {
      perch_state = 0;
      SendPerchAlert(perch_value, perch_state);
    }
    break;
}

switch (seed_state)
{
  case 0: // bird feeder seed filled
    if (seed_value >= SEED)
    {
      seed_state = 1;
      SendSeedAlert(seed_value, seed_state);
    }
    break;

  case 1: // bird feeder seed empty
    if (seed_value < SEED)
    {
      seed_state = 0;
      SendSeedAlert(seed_value, seed_state);
    }
    break;
}
}

```

Note the references to the capacitive sensing and new software serial libraries at the beginning of the sketch. We initialize the variables that will use references from those libraries along with our threshold variables. Then we set up the connections to the serial window and XBee radio along with the Arduino onboard LED on pin 13. Once initialized, the program simply runs a loop and

waits for the perch or seed thresholds to be exceeded or reset. If a change condition is detected, the running sketch will transmit these trigger messages to the Arduino IDE's serial window as well as to the XBee radio.

With both the foil and photocells correctly reporting their values, redirect serial output from the Arduino IDE serial window to the XBee attached to the Arduino. Open up the serial application window to the FTDI-connected XBee, and if all goes well, the data you saw being displayed in the Arduino IDE serial window should now be showing up in the FTDI-connected serial application window on your computer. Isn't wireless communication cool?

At this point, the hardware for our project is connected and tested and may look similar to the bird feeder components I assembled in Figure 15, [*A pet bird can help out with the testing and debugging of threshold values that trigger perch landing and departure events.*](#)



Figure 15. A pet bird can help out with the testing and debugging of threshold values that trigger perch landing and departure events.

But before we start packing the hardware into the bird feeder, we need to create one more crucial component. Using the Python language, we can write a short program that will listen for bird landings, poll for seed status, and post notable changes to Twitter. Let's go write some code.

5.6 Tweeting with Python

A number of different languages capable of accomplishing the tasks of monitoring and interpreting incoming serial console messages and transmitting outbound messages to the serial port exist. There are also a number of Twitter libraries for various programming languages.

Python was chosen for this and several other scripts in the book due to the language's

easy-to-follow syntax, its default inclusion in Linux and Mac OS X operating systems, and its “batteries included” approach to bundling a number of relevant libraries (such as SQLite) in its base distribution. To learn more about programming in Python, check out *Learning Python* [LA03].

For this project, the basic functionality we need the `tweetingbirdfeeder.py` Python script to accomplish is this:

1. Record the date and time when a bird lands and departs from the perch in the `birdfeeding` table in the `tweetingbirdfeeder` database.
2. Record the date and time when seed levels are depleted and replenished in the `seedstatus` table, which is also part of the `tweetingbirdfeeder` database.
3. Listen for inbound and transmit outbound serial messages via the FTDI cable-connected XBee radio and respond to events by storing the date and time of their occurrence and condition.
4. Connect to Twitter via OAuth authentication and post changes in bird feeding and seed level status.

The only additional Python libraries that need to be installed for this project are `pyserial` and `python-twitter`.

Beyond relaying tweets to a designated Twitter account, it would be helpful to

visualize trends in the data we will be tweeting, such as the frequency and date/time of bird landings and the average time between seed refills. We can then see how these trends map out over an hour, a day, a month, and a year. To do this, we will need to capture the data in a structured format.

Configure the Database

Since Python 2.5 and higher supports the SQLite database out of the box, and because our data needs don't require an overengineered standalone database server, SQLite is the ideal choice for the job. While we could have dumped these values to a plain-text comma separated value (CSV) file, organizing the data into a structured SQLite file affords us two benefits: First, we will be better prepared for future data analysis queries. Second, we will have greater flexibility to capture and manage other types of data events later on simply by adding new columns to the table.

In order to create a database in `sqlite3` file format, we can use the `sqlite3` command-line tool. This tool is already installed on Mac OS X. On Linux, it will most likely need to be retrieved from the distribution's repository. In the case of Debian-based distributions like Ubuntu, issuing `sudo apt-get install sqlite3 libsqlite3-dev` should install the application. Windows users will need to download the `sqlite3.exe` utility from the SQLite website.[\[51\]](#)

Once installed, type `sqlite3` within a terminal window. This will display something like the following:

```
SQLite version 3.7.6
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite>
```

Your installation of SQLite may report a different version number.

Next, we will enter the SQL statement to create our new database. To do so, exit the `sqlite` command shell by typing `.q` followed by a carriage return at the `sqlite>` prompt. Then relaunch the `sqlite3` tool, followed by the name of the database to

be opened.

For this project, we will call the database `tweetingbirdfeeder`, with the filename `tweetingbirdfeeder.sqlite`. Because this database file does not yet exist, SQLite will automatically create the file for us. The database file will be created from the same directory that you launched the `sqlite3` tool from. For example, if you launched `sqlite3` from your home directory, the database file will be created there.

SQLite Manager

While the SQLite command-line tools provide all you need to create and manage SQLite databases, it's sometimes easier to work with a graphic user interface. This is especially applicable if you need to scroll through rows of data in a single window. Several open source GUI-based SQLite database explorer-style apps exist. If you are a Mozilla Firefox web browser user, I recommend using the cross-platform SQLite Manager Firefox plug-in.[\[52\]](#)

Installing the plug-in is easy. From the Firefox's Tools→Add-ons menu selection, search for SQLite Manager and click the Install button. Open SQLite Manager from Firefox's Tools→SQLite Manager menu option. Creating a new database is as simple as clicking on the New Database icon in the SQLite Manager window toolbar. Saving and opening SQLite database files is just as easy.

We will create a new table in the `tweetingbirdfeeder.sqlite` database that we will call `birdfeeding` with the following structure:

Column Name	Data Type	Primary Key?	Autoinc?	Allow Null?	Unique?
id	INTEGER	YES	YES	NO	YES
time	DATETIME	NO	NO	NO	NO
event	TEXT	NO	NO	NO	NO

We can create this table by submitting the following SQL statement to the `sqlite`

command-line tool:

```
[~]$ sqlite3 tweetingbirdfeeder.sqlite
SQLite version 3.7.6
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> CREATE TABLE "birdfeeding" ("id" INTEGER PRIMARY KEY NOT NULL UNIQUE,
"time" DATETIME NOT NULL,"event" TEXT NOT NULL);
```

With the `birdfeeding` table established, we need another table, one that has a similar structure in the same database and is called `seedstatus`:

Column Name	Data Type	Primary Key?	Autoinc?	Allow Null?	Unique?
id	INTEGER	YES	YES	NO	YES
time	DATETIME	NO	NO	NO	NO
event	TEXT	NO	NO	NO	NO

Just like the `birdfeeding` table, submitting the following SQL statement to the sqlite command-line tool will generate the desired structure for the `seedstatus` table:

```
[~]$ sqlite3 tweetingbirdfeeder.sqlite
SQLite version 3.7.6
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> CREATE TABLE "seedstatus" ("id" INTEGER PRIMARY KEY NOT NULL,
"time" DATETIME NOT NULL ,"event" TEXT NOT NULL );
```

Now that the database has been defined, we can work on the code to import the database and Serial and Twitter libraries, then listen for serial events being generated and timestamp and store these events to the appropriate database table.

We'll conclude the event capture by posting a tweet of the situation. But before you can programmatically tweet to Twitter, you need to create a Twitter account

and sign up for a Twitter API key and related OAuth credentials. Let's go get ourselves an API key.

Twitter API Credentials

Before sending tweets to Twitter, you need a Twitter account to send them to. And before you send tweets to Twitter programmatically via a language or library that supports OAuth,[53] you need to create an application identifier for the intended Twitter account. While you could use an existing Twitter account, I prefer creating new accounts whenever a new project demands it. That way, followers of my existing account are not accosted by tweets of my latest experiments. It also offers a way to share your application's tweets selectively. With these considerations in mind, create a new account and application ID specifically for the bird feeder project.

Using your new Twitter account credentials, visit <http://dev.twitter.com> and select the "Register an app" option. On the New Twitter Application page, enter a unique name for your application, a description at least ten characters long, and a valid website for the app. Enter a temporary one if you don't have a permanent website from which you will offer your application for download. Then select Client under Application Type and select Read & Write under Default Access type. You can upload a custom icon if you like, but it's not required. Then enter the CAPTCHA validation and click the Register Application button at the bottom of the screen. Read and accept the Twitter API Terms of Service to proceed.

Once your request has been approved, a unique API Key, OAuth Consumer key, and Consumer secret will be generated. Click the My Access Token menu item on the right side of the page to access your application's all important Access Token (oauth_token) and super-secret Access Token Secret (oauth_token_secret). Copy these unique codes and store them in a safe, secure file. You will need these values to programmatically interact with your new Twitter account. Remember to keep these values a secret! You don't want any unscrupulous individuals getting hold of your secret token and using it to spam your friends and raise the ire of the Twitter community.

With a Twitter account and a valid application API key in hand, you can use these credentials in our Python-based Tweeting Bird Feeder application.

The Python-Twitter Library

Even though we have API access to Twitter, we still need to talk to Twitter from Python. We can do so with help a little help from the Python-Twitter library.[\[54\]](#) To install both the Pyserial and Python-Twitter libraries, download the latest version and execute the standard `sudo python setup.py install` command. If you're installing this library on Mac OS X 10.6 (Snow Leopard) or higher, the `easy_install` Python setup tool is preinstalled. However, due to quirks in the 64-bit libraries, you will need to precede the command with an i386 architecture flag to install the Python-Twitter library without errors. The complete command for this is `sudo env ARCHFLAGS="-arch i386" easy_install python-twitter`.

At last, we have all the accounts configured and dependencies installed, and we can complete the project by writing the Python script that will listen for messages on the receiving XBee radio serial port, timestamp the messages, save them in a database, and post the message to Twitter. Let's write the Python script that will codify this process.

```
'TweetingBirdFeeder/tweetingbirdfeeder.py'
```

```
# import DateTime, Serial port, SQLite3 and Twitter python libraries
from datetime import datetime
import serial
import sqlite3
import twitter

# import the os module to clear the terminal window at start of the program
# windows uses "cls" while Linux and OS X use the "clear" command
import os
if sys.platform == "win32":
    os.system("cls")
else:
    os.system("clear")

# Connect to the serial port, replacing YOUR_SERIAL_DEVICE with the
# name of the serial port of the FTDI cable-attached XBee adapter
XBeePort = serial.Serial('devtty.YOUR_SERIAL_DEVICE', \
```

```

        baudrate = 9600, timeout = 1)

# Connect to SQLite database file
sqlconnection = sqlite3.connect("tweetingbirdfeeder.sqlite3")

# create database cursor
sqlcursor = sqlconnection.cursor()

# Initialize Twitter API object
api = twitter.Api('Your_OAuth_Consumer_Key', 'Your_OAuth_Consumer_Secret', \
    'Your_OAuth_Access-Token', 'Your_OAuth_Access-Token_Secret')

def transmit(msg):
    # Get current date and time and format it accordingly
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    # Determine message and assign response parameters
    if msg == "arrived":
        tweet = "A bird has landed on the perch!"
        table = "birdfeeding"
    if msg == "departed":
        tweet = "A bird has left the perch!"
        table = "birdfeeding"
    if msg == "refill":
        tweet = "The feeder is empty."
        table = "seedstatus"
    if msg == "seedOK":
        tweet = "The feeder has been refilled with seed."
        table = "seedstatus"

    print "%s - %s" % (timestamp.strftime("%Y-%m-%d %H:%M:%S"), tweet)

    # Store the event in the SQLite database file
    try:
        sqlstatement = "INSERT INTO %s (id, time, event) \
VALUES(NULL, \"%s\", \"%s\")" % (table, timestamp, msg)
        sqlcursor.execute(sqlstatement)
        sqlconnection.commit()
    except:
        print "Could not store event to the database."
        pass

    # Post message to Twitter
    try:
        status = api.PostUpdate(msg)

```

```

except:
    print "Could not post Tweet to Twitter"
    pass

# Main program loop
try:
    while 1:
        # listen for inbound characters from the feeder-mounted XBee radio
        message = XBeePort.readline()

        # Depending on the type of message is received,
        # log and tweet it accordingly
        if "arrived" in message:
            transmit("arrived")

        if "departed" in message:
            transmit("departed")

        if "refill" in message:
            transmit("refill")

        if "seedOK" in message:
            transmit("seedOK")

except KeyboardInterrupt:
    # Exit the program when the Control-C keyboard interrupt been detected
    print("\nQuitting the Tweeting Bird Feeder Listener Program.\n")
    sqlcursor.close()
    pass

```

Once the necessary `datetime`, `serial`, `sqlite`, and `twitter` libraries are loaded, we clear the terminal window (sending a `cls` for Windows and a `clear` for any other operating system), connect to the receiving XBee radio serial port (the XBee that is attached to the computer via the FTDI cable). Then we connect to the `tweetingbirdfeeder.sqlite3` database file we created earlier and run an infinite `while` loop until the `Control-C` keyboard combination is triggered so we can gracefully exit the program. If the attached XBee radio receives a message it recognizes, we call the `def transmit(msg)` function that parses the `msg` variable, adds descriptive text to the event, saves the message to the database, and posts it to Twitter.

With the Arduino running and the XBee radios paired and powered, test the

threshold detections by touching the perch sensor and photocell enough times to trigger several event transmissions. Assuming no errors were reported in the terminal window of the running script, open the `tweetingbirdfeeder.sqlite3` file in the SQLite Manager's Browse and Search window and verify that entries for both sensors were timestamped when the related events were triggered. If everything checks out, log into the Twitter account that you used to post the events and verify that the events appear on the timeline.

We're almost done. Just a few more hardware assembly steps remain.

5.7 Putting It All Together

In order to make the project fully functional, we need to package up the Arduino+XBee hardware assembly inside a weatherized, protected layer within the bird feeder, mount the photocell near the base of the feeder, fill the feeder with seed, attach the Arduino+XBee hardware to a power source, and place the feeder outdoors but within range of the paired XBee radio attached to the computer.

Unless you live in an area with little rainfall, you will need to protect the electrical assembly from water damage. I have found double bagging the components in a small plastic freezer bag does a sufficient job of weatherproofing the Arduino+XBee. However, unless you plan on powering the electronics with a 9V battery that can be contained within the bundle (it might be good for short data collection sessions but won't last very long before its charge is exhausted), you will need to account for an external cord to attach to the Arduino so that continuous power can be delivered.

Cutting a small opening in the bag to allow the cable to enter works, but doing so exposes the insides to potential moisture condensation. To minimize this risk, tightly wrap the freezer bag and cable exit point with a continuous sheet of plastic wrap, climbing high enough up the power cord to ensure a good seal that won't slip or loosen with weather changes.

Using a weatherized power cord (such as those sold for outdoor decorative lighting purposes) may be less expensive and easier to test in the short term. However, environmentally conscious individuals may prefer instead to spend a bit more money up front for a longer, more sustainable energy alternative in the form of a photovoltaic power supply.

When searching for an adequate, portable solar power solution, make sure it can deliver 5 volts, is built for rugged durability, and has a built-in rechargeable battery when backup power is needed. Products like the Solio Bolt provide a relatively inexpensive solution for short-term measurements.[\[55\]](#) If you prefer

photovoltaic solutions that offer greater internal battery charging capacities and voltage, be prepared to spend a bit more for the added capabilities. Companies like Sunforce Products offer a variety of solar backup power maintainers, trickle chargers, and controllers designed to take on greater loads.[\[56\]](#)

You should mount the solar panel far enough away from the feeder to gain maximum sun exposure. If possible, mount the panel at a ninety-degree angle to the sun for optimal energy capture. Depending on your location and average level of daylight intensity, you may need to seek alternatives such as consumer-grade wind turbine chargers or even pedal-powered dynamos.

We have accomplished quite a number of new objectives this project, from using photocell and homemade sensors and learning how to pair and wirelessly communicate between XBee-attached hardware to writing a script that records structured data, responds to events, and submits posts to Twitter via Twitter's API. We have also taken into account standalone Arduino+XBee radio energy requirements and ways to adequately shield our electronics from environmental changes.

We will reuse these valuable lessons with some of the other projects in the ensuing chapters.

5.8 Next Steps

The variety of home automation projects using capacitive and photocell sensor notifications is expansive. Here are just a few ideas to consider pursuing:

- Place an XBee+photocell-equipped Arduino, powered by a rechargeable battery pack, in your refrigerator or freezer to detect how often and how long the doors are left open. Use this data to calculate energy expended each month as a result of these encounters. If such expenditures are excessive, broadcast emails and/or tweets to cohabitants reminding them of their growing carbon footprint.
- If groping for a light switch is a frustratingly frequent chore, tweak the capacitive sensor configuration to be used to turn on basement or garage lights by touching foil tacked to a wall or to a table-mounted surface.
- Use the variable analog readings of a photocell to measure day-night cycles and sunny-overcast recordings and map these to seasonal gardening data. Did planting certain species of flowers, fruits, or vegetables before a certain time help or hinder their growth? How many days were the plants bathed in full sunlight compared to overcast or inclement skies?

In addition to these suggestions, there is still plenty of data analysis that can be done with the collected bird feeder data. Use a Python graphing library like CairoPlot to help visualize the average duration of and time in between bird visits.[\[57\]](#) How quickly was seed consumed? How much of an effect did weather have on feeding times and durations? Does changing the type of seed alter the duration and frequency of visits?

Consider sharing your tweets with other bird enthusiasts online to expand a social network of others collecting and sharing their bird feeder data.

Collectively compare patterns across various regions and geographies to infer population trends, environmental impacts, migration cycles, and other factors to better understand the habits of our feathered friends.

Footnotes

- [43] <http://www.makershed.com/ProductDetails.asp?ProductCode=MKGR1>
- [44] <http://www.adafruit.com>
- [45] <http://www.solio.com/chargers/>
- [46] <http://www.arduino.cc/playground/Main/CapSense>
- [47] <http://www.ladyada.net/learn/sensors/cds.html>.
- [48] <http://www.ladyada.net/make/xbee/>
- [49] <http://ladyada.net/make/xbee/point2point.html>
- [50] <http://alioth.debian.org/projects/minicom/>
- [51] <http://www.sqlite.org/download.html>
- [52] <https://addons.mozilla.org/en-US/firefox/addon/sqlite-manager/>
- [53] <http://oauth.net/>
- [54] <http://code.google.com/p/python-twitter/>
- [55] <http://www.solio.com/chargers/>
- [56] http://www.sunforceproducts.com/results.php?CAT_ID=1
- [57] <http://cairoplot.sourceforge.net/>