

Recipes to Begin, Expand, and Enhance Your Projects

Arduino Cookbook



O'REILLY®

Michael Margolis

Arduino Cookbook

by Michael Margolis

Copyright © 2011 Michael Margolis and Nicholas Weldin. All rights reserved.
Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (<http://my.safaribooksonline.com>). For more information, contact our corporate/institutional sales department: (800) 998-9938 or corporate@oreilly.com.

Editors: Simon St. Laurent and Brian Jepson

Production Editor: Teresa Elsey

Copyeditor: Audrey Doyle

Proofreader: Teresa Elsey

Indexer: Lucie Haskins

Cover Designer: Karen Montgomery

Interior Designer: David Futato

Illustrator: Robert Romano

Printing History:

March 2011: First Edition.

Nutshell Handbook, the Nutshell Handbook logo, and the O'Reilly logo are registered trademarks of O'Reilly Media, Inc. *Arduino Cookbook*, the image of a toy rabbit, and related trade dress are trademarks of O'Reilly Media, Inc.

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 O'Reilly Media, Inc., was aware of a trademark claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the publisher and authors assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

ISBN: 978-0-596-80247-9

[LSI]

1299267108

Table of Contents

Preface	xiii
1. Getting Started	1
1.1 Installing the Integrated Development Environment (IDE)	4
1.2 Setting Up the Arduino Board	6
1.3 Using the Integrated Development Environment (IDE) to Prepare an Arduino Sketch	8
1.4 Uploading and Running the Blink Sketch	11
1.5 Creating and Saving a Sketch	13
1.6 Using Arduino	15
2. Making the Sketch Do Your Bidding	19
2.1 Structuring an Arduino Program	20
2.2 Using Simple Primitive Types (Variables)	21
2.3 Using Floating-Point Numbers	23
2.4 Working with Groups of Values	25
2.5 Using Arduino String Functionality	28
2.6 Using C Character Strings	30
2.7 Splitting Comma-Separated Text into Groups	32
2.8 Converting a Number to a String	34
2.9 Converting a String to a Number	36
2.10 Structuring Your Code into Functional Blocks	38
2.11 Returning More Than One Value from a Function	41
2.12 Taking Actions Based on Conditions	44
2.13 Repeating a Sequence of Statements	45
2.14 Repeating Statements with a Counter	47
2.15 Breaking Out of Loops	49
2.16 Taking a Variety of Actions Based on a Single Variable	50
2.17 Comparing Character and Numeric Values	52
2.18 Comparing Strings	54
2.19 Performing Logical Comparisons	55

2.20	Performing Bitwise Operations	56
2.21	Combining Operations and Assignment	58
3.	Using Mathematical Operators	61
3.1	Adding, Subtracting, Multiplying, and Dividing	61
3.2	Incrementing and Decrementing Values	62
3.3	Finding the Remainder After Dividing Two Values	63
3.4	Determining the Absolute Value	64
3.5	Constraining a Number to a Range of Values	65
3.6	Finding the Minimum or Maximum of Some Values	66
3.7	Raising a Number to a Power	67
3.8	Taking the Square Root	68
3.9	Rounding Floating-Point Numbers Up and Down	68
3.10	Using Trigonometric Functions	69
3.11	Generating Random Numbers	70
3.12	Setting and Reading Bits	72
3.13	Shifting Bits	75
3.14	Extracting High and Low Bytes in an int or long	77
3.15	Forming an int or long from High and Low Bytes	78
4.	Serial Communications	81
4.1	Sending Debug Information from Arduino to Your Computer	86
4.2	Sending Formatted Text and Numeric Data from Arduino	89
4.3	Receiving Serial Data in Arduino	92
4.4	Sending Multiple Text Fields from Arduino in a Single Message	95
4.5	Receiving Multiple Text Fields in a Single Message in Arduino	98
4.6	Sending Binary Data from Arduino	101
4.7	Receiving Binary Data from Arduino on a Computer	105
4.8	Sending Binary Values from Processing to Arduino	107
4.9	Sending the Value of Multiple Arduino Pins	109
4.10	How to Move the Mouse Cursor on a PC or Mac	112
4.11	Controlling Google Earth Using Arduino	115
4.12	Logging Arduino Data to a File on Your Computer	121
4.13	Sending Data to Two Serial Devices at the Same Time	124
4.14	Receiving Serial Data from Two Devices at the Same Time	128
4.15	Setting Up Processing on Your Computer to Send and Receive Serial Data	131
5.	Simple Digital and Analog Input	133
5.1	Using a Switch	136
5.2	Using a Switch Without External Resistors	139
5.3	Reliably Detecting the Closing of a Switch	141
5.4	Determining How Long a Switch Is Pressed	144

5.5	Reading a Keypad	149
5.6	Reading Analog Values	152
5.7	Changing the Range of Values	154
5.8	Reading More Than Six Analog Inputs	155
5.9	Displaying Voltages Up to 5V	158
5.10	Responding to Changes in Voltage	161
5.11	Measuring Voltages More Than 5V (Voltage Dividers)	162
6.	Getting Input from Sensors	165
6.1	Detecting Movement	167
6.2	Detecting Light	170
6.3	Detecting Motion (Integrating Passive Infrared Detectors)	171
6.4	Measuring Distance	173
6.5	Measuring Distance Accurately	176
6.6	Detecting Vibration	180
6.7	Detecting Sound	181
6.8	Measuring Temperature	185
6.9	Reading RFID Tags	187
6.10	Tracking the Movement of a Dial	190
6.11	Tracking the Movement of More Than One Rotary Encoder	193
6.12	Tracking the Movement of a Dial in a Busy Sketch	195
6.13	Using a Mouse	197
6.14	Getting Location from a GPS	201
6.15	Detecting Rotation Using a Gyroscope	206
6.16	Detecting Direction	208
6.17	Getting Input from a Game Control Pad (PlayStation)	211
6.18	Reading Acceleration	213
7.	Visual Output	217
7.1	Connecting and Using LEDs	220
7.2	Adjusting the Brightness of an LED	223
7.3	Driving High-Power LEDs	224
7.4	Adjusting the Color of an LED	226
7.5	Sequencing Multiple LEDs: Creating a Bar Graph	229
7.6	Sequencing Multiple LEDs: Making a Chase Sequence (Knight Rider)	232
7.7	Controlling an LED Matrix Using Multiplexing	234
7.8	Displaying Images on an LED Matrix	236
7.9	Controlling a Matrix of LEDs: Charlieplexing	239
7.10	Driving a 7-Segment LED Display	245
7.11	Driving Multidigit, 7-Segment LED Displays: Multiplexing	248
7.12	Driving Multidigit, 7-Segment LED Displays Using MAX7221 Shift Registers	250

7.13	Controlling an Array of LEDs by Using MAX72xx Shift Registers	253
7.14	Increasing the Number of Analog Outputs Using PWM Extender Chips (TLC5940)	255
7.15	Using an Analog Panel Meter As a Display	259
8.	Physical Output	261
8.1	Controlling the Position of a Servo	264
8.2	Controlling One or Two Servos with a Potentiometer or Sensor	266
8.3	Controlling the Speed of Continuous Rotation Servos	267
8.4	Controlling Servos from the Serial Port	269
8.5	Driving a Brushless Motor (Using a Hobby Speed Controller)	271
8.6	Controlling Solenoids and Relays	272
8.7	Making an Object Vibrate	273
8.8	Driving a Brushed Motor Using a Transistor	276
8.9	Controlling the Direction of a Brushed Motor with an H-Bridge	277
8.10	Controlling the Direction and Speed of a Brushed Motor with an H-Bridge	280
8.11	Using Sensors to Control the Direction and Speed of Brushed Motors (L293 H-Bridge)	282
8.12	Driving a Bipolar Stepper Motor	287
8.13	Driving a Bipolar Stepper Motor (Using the EasyDriver Board)	290
8.14	Driving a Unipolar Stepper Motor (ULN2003A)	293
9.	Audio Output	297
9.1	Playing Tones	299
9.2	Playing a Simple Melody	301
9.3	Generating More Than One Simultaneous Tone	303
9.4	Generating Audio Tones and Fading an LED	305
9.5	Playing a WAV File	308
9.6	Controlling MIDI	311
9.7	Making an Audio Synthesizer	314
10.	Remotely Controlling External Devices	317
10.1	Responding to an Infrared Remote Control	318
10.2	Decoding Infrared Remote Control Signals	321
10.3	Imitating Remote Control Signals	324
10.4	Controlling a Digital Camera	327
10.5	Controlling AC Devices by Hacking a Remote Controlled Switch	330
11.	Using Displays	333
11.1	Connecting and Using a Text LCD Display	334

11.2	Formatting Text	337
11.3	Turning the Cursor and Display On or Off	340
11.4	Scrolling Text	342
11.5	Displaying Special Symbols	345
11.6	Creating Custom Characters	347
11.7	Displaying Symbols Larger Than a Single Character	349
11.8	Displaying Pixels Smaller Than a Single Character	352
11.9	Connecting and Using a Graphical LCD Display	355
11.10	Creating Bitmaps for Use with a Graphical Display	359
11.11	Displaying Text on a TV	361
12.	Using Time and Dates	367
12.1	Creating Delays	367
12.2	Using millis to Determine Duration	368
12.3	More Precisely Measuring the Duration of a Pulse	372
12.4	Using Arduino As a Clock	373
12.5	Creating an Alarm to Periodically Call a Function	380
12.6	Using a Real-Time Clock	384
13.	Communicating Using I2C and SPI	389
13.1	Controlling an RGB LED Using the BlinkM Module	392
13.2	Using the Wii Nunchuck Accelerometer	397
13.3	Interfacing to an External Real-Time Clock	401
13.4	Adding External EEPROM Memory	404
13.5	Reading Temperature with a Digital Thermometer	408
13.6	Driving Four 7-Segment LEDs Using Only Two Wires	412
13.7	Integrating an I2C Port Expander	416
13.8	Driving Multidigit, 7-Segment Displays Using SPI	418
13.9	Communicating Between Two or More Arduino Boards	421
14.	Wireless Communication	425
14.1	Sending Messages Using Low-Cost Wireless Modules	425
14.2	Connecting Arduino to a ZigBee or 802.15.4 Network	431
14.3	Sending a Message to a Particular XBee	438
14.4	Sending Sensor Data Between XBees	440
14.5	Activating an Actuator Connected to an XBee	446
15.	Ethernet and Networking	451
15.1	Setting Up the Ethernet Shield	453
15.2	Obtaining Your IP Address Automatically	455
15.3	Resolving Hostnames to IP Addresses (DNS)	458
15.4	Requesting Data from a Web Server	462
15.5	Requesting Data from a Web Server Using XML	466

15.6	Setting Up an Arduino to Be a Web Server	469
15.7	Handling Incoming Web Requests	471
15.8	Handling Incoming Requests for Specific Pages	474
15.9	Using HTML to Format Web Server Responses	479
15.10	Serving Web Pages Using Forms (POST)	483
15.11	Serving Web Pages Containing Large Amounts of Data	486
15.12	Sending Twitter Messages	493
15.13	Sending and Receiving Simple Messages (UDP)	496
15.14	Getting the Time from an Internet Time Server	502
15.15	Monitoring Pachube Feeds	507
15.16	Sending Information to Pachube	510
16.	Using, Modifying, and Creating Libraries	515
16.1	Using the Built-in Libraries	515
16.2	Installing Third-Party Libraries	517
16.3	Modifying a Library	518
16.4	Creating Your Own Library	522
16.5	Creating a Library That Uses Other Libraries	527
17.	Advanced Coding and Memory Handling	531
17.1	Understanding the Arduino Build Process	532
17.2	Determining the Amount of Free and Used RAM	535
17.3	Storing and Retrieving Numeric Values in Program Memory	537
17.4	Storing and Retrieving Strings in Program Memory	540
17.5	Using #define and const Instead of Integers	542
17.6	Using Conditional Compilations	543
18.	Using the Controller Chip Hardware	547
18.1	Storing Data in Permanent EEPROM Memory	551
18.2	Using Hardware Interrupts	554
18.3	Setting Timer Duration	557
18.4	Setting Timer Pulse Width and Duration	559
18.5	Creating a Pulse Generator	562
18.6	Changing a Timer's PWM Frequency	565
18.7	Counting Pulses	567
18.8	Measuring Pulses More Accurately	569
18.9	Measuring Analog Values Quickly	571
18.10	Reducing Battery Drain	572
18.11	Setting Digital Pins Quickly	574
A.	Electronic Components	579
B.	Using Schematic Diagrams and Data Sheets	585

C. Building and Connecting the Circuit	591
D. Tips on Troubleshooting Software Problems	595
E. Tips on Troubleshooting Hardware Problems	599
F. Digital and Analog Pins	603
G. ASCII and Extended Character Sets	607
Index	611

Getting Started

1.0 Introduction

The Arduino environment has been designed to be easy to use for beginners who have no software or electronics experience. With Arduino, you can build objects that can respond to and/or control light, sound, touch, and movement. Arduino has been used to create an amazing variety of things, including musical instruments, robots, light sculptures, games, interactive furniture, and even interactive clothing.



If you're not a beginner, please feel free to skip ahead to recipes that interest you.

Arduino is used in many educational programs around the world, particularly by designers and artists who want to easily create prototypes but do not need a deep understanding of the technical details behind their creations. Because it is designed to be used by nontechnical people, the software includes plenty of example code to demonstrate how to use the Arduino board's various facilities.

Though it is easy to use, Arduino's underlying hardware works at the same level of sophistication that engineers employ to build embedded devices. People already working with microcontrollers are also attracted to Arduino because of its agile development capabilities and its facility for quick implementation of ideas.

Arduino is best known for its hardware, but you also need software to program that hardware. Both the hardware and the software are called "Arduino." The combination enables you to create projects that sense and control the physical world. The software is free, open source, and cross-platform. The boards are inexpensive to buy, or you can build your own (the hardware designs are also open source). In addition, there is an active and supportive Arduino community that is accessible worldwide through the Arduino forums and the wiki (known as the Arduino Playground). The forums and the

wiki offer project development examples and solutions to problems that can provide inspiration and assistance as you pursue your own projects.

The recipes in this chapter will get you started by explaining how to set up the development environment and how to compile and run an example sketch.



Source code containing computer instructions for controlling Arduino functionality is usually referred to as a *sketch* in the Arduino community. The word *sketch* will be used throughout this book to refer to Arduino program code.

The Blink sketch, which comes with Arduino, is used as an example for recipes in this chapter, though the last recipe in the chapter goes further by adding sound and collecting input through some additional hardware, not just blinking the light built into the board. [Chapter 2](#) covers how to structure a sketch for Arduino and provides an introduction to programming.



If you already know your way around Arduino basics, feel free to jump forward to later chapters. If you're a first-time Arduino user, patience in these early recipes will pay off with smoother results later.

Arduino Software

Software programs, called *sketches*, are created on a computer using the Arduino integrated development environment (IDE). The IDE enables you to write and edit code and convert this code into instructions that Arduino hardware understands. The IDE also transfers those instructions to the Arduino board (a process called *uploading*).

Arduino Hardware

The Arduino board is where the code you write is executed. The board can only control and respond to electricity, so specific components are attached to it to enable it to interact with the real world. These components can be sensors, which convert some aspect of the physical world to electricity so that the board can sense it, or actuators, which get electricity from the board and convert it into something that changes the world. Examples of sensors include switches, accelerometers, and ultrasound distance sensors. Actuators are things like lights and LEDs, speakers, motors, and displays.

There are a variety of official boards that you can use with Arduino software and a wide range of Arduino-compatible boards produced by members of the community.

The most popular boards contain a USB connector that is used to provide power and connectivity for uploading your software onto the board. [Figure 1-1](#) shows a basic board, the Arduino Uno.

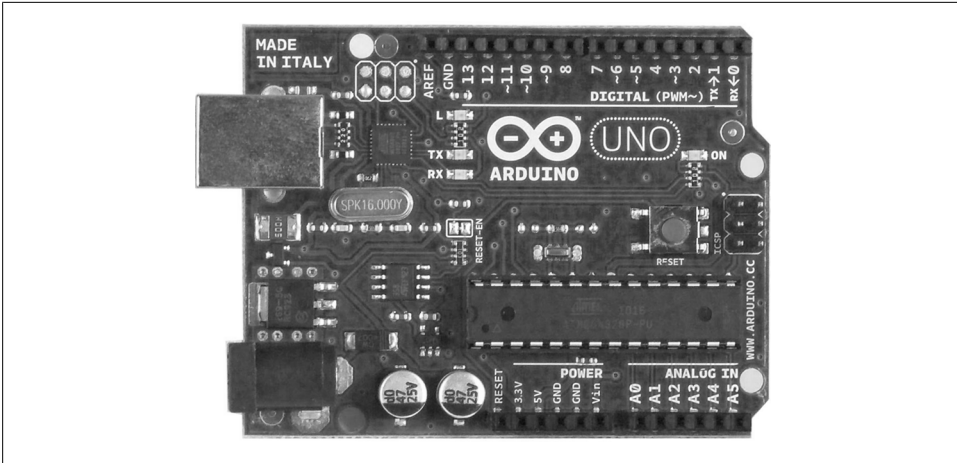


Figure 1-1. Basic board: the Arduino Uno

You can get boards as small as a postage stamp, such as the Arduino Mini and Pro Mini; larger boards that have more connection options and more powerful processors, such as the Arduino Mega; and boards tailored for specific applications, such as the LilyPad for wearable applications, the Fio for wireless projects, and the Arduino Pro for embedded applications (standalone projects that are often battery-operated). Many other Arduino-compatible boards are also available, including the following:

- Arduino Nano, a tiny board with USB capability, from Gravitech (<http://store.gravitech.us/arna30wiatn.html>)
- Bare Bones Board, a low-cost board available with or without USB capability, from Modern Device (<http://www.moderndevice.com/products/bbb-kit>)
- Boarduino, a low-cost breadboard-compatible board, from Adafruit Industries (<http://www.adafruit.com/>)
- Seeedduino, a flexible variation of the standard USB board, from Seeed Studio Bazaar (<http://www.seeedstudio.com/>)
- Teensy and Teensy++, tiny but extremely versatile boards, from PJRC (<http://www.pjrc.com/teensy/>)

A comprehensive list of Arduino-compatible boards is available at <http://www.freeduino.org/>.

See Also

An overview of Arduino boards: <http://www.arduino.cc/en/Main/Hardware>.

Online guides for getting started with Arduino are available at <http://arduino.cc/en/Guide/Windows> for Windows, <http://arduino.cc/en/Guide/MacOSX> for Mac OS X, and <http://www.arduino.cc/playground/Learning/Linux> for Linux.

1.1 Installing the Integrated Development Environment (IDE)

Problem

You want to install the Arduino development environment on your computer.

Solution

The Arduino software for Windows, Mac, and Linux can be downloaded from <http://arduino.cc/en/Main/Software>.

The Windows download is a ZIP file. Unzip the file to any convenient directory—*Program Files/Arduino* is a sensible place.



A free utility for unzipping files, called 7-Zip, can be downloaded from <http://www.7-zip.org/>.

Unzipping the file will create a folder named *Arduino-00<nn>* (where *<nn>* is the version number of the Arduino release you downloaded). The directory contains the executable file (named *Arduino.exe*), along with various other files and folders. Double-click the *Arduino.exe* file and the splash screen should appear (see [Figure 1-2](#)), followed by the main program window (see [Figure 1-3](#)). Be patient, as it can take some time for the software to load.

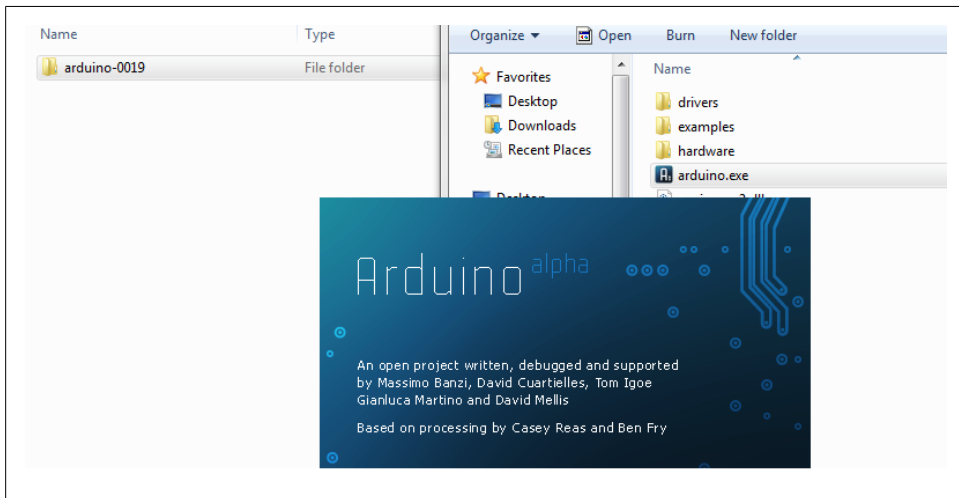


Figure 1-2. Arduino splash screen (version 0019 in Windows 7)

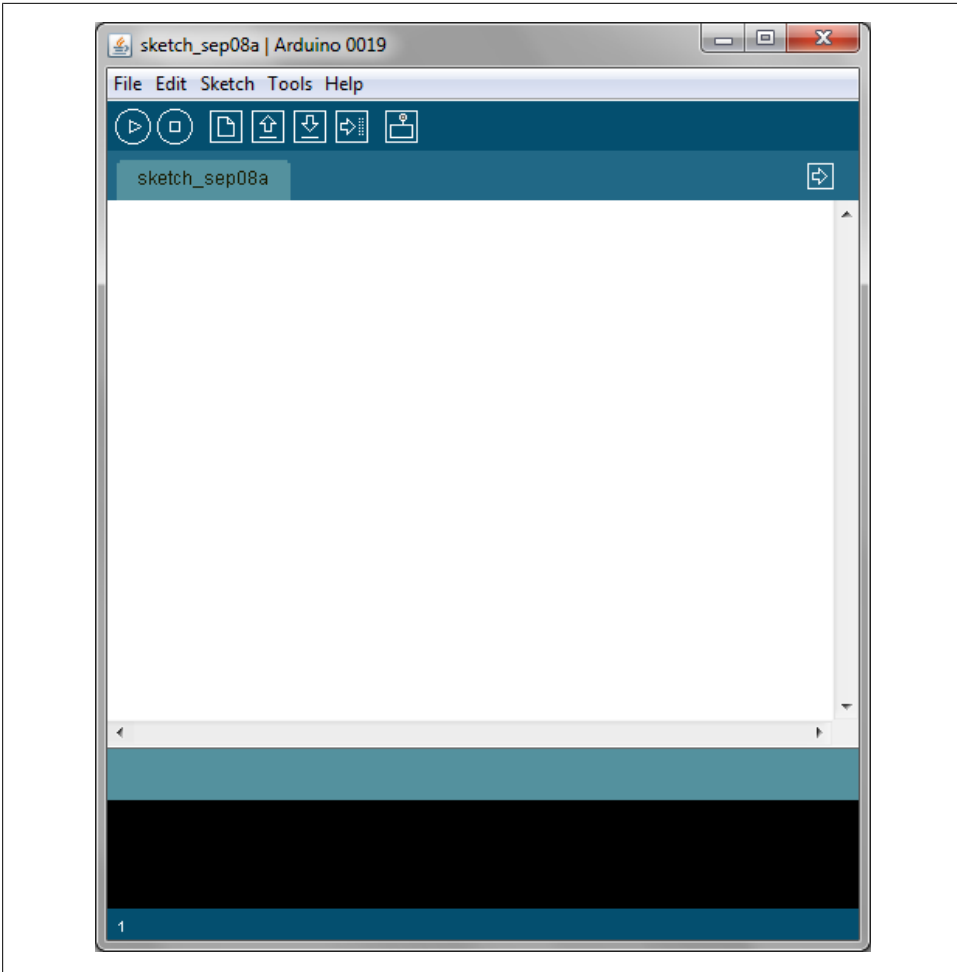


Figure 1-3. Arduino IDE main window (version 0019 in Windows 7)

The Arduino download for the Mac is a disk image (*.dmg*); double-click the file when the download is complete. The image will mount (it will appear like a memory stick on the desktop). Inside the disk image is the Arduino application. Copy this to somewhere convenient—the *Applications* folder is a sensible place. Double-click the application once you have copied it over (it is not a good idea to run it from the disk image). The splash screen will appear, followed by the main program window.

Linux installation varies depending on the Linux distribution you are using. See the Arduino wiki for information (<http://www.arduino.cc/playground/Learning/Linux>).

To enable the Arduino development environment to communicate with the board, you need to install drivers.

On Windows, use the USB cable to connect your PC and the Arduino board and wait for the Found New Hardware Wizard to appear. If you are using Windows Vista or Windows 7 and are online, you can let the wizard search for drivers and they will install automatically. On Windows XP, you should specify the location of the drivers. Use the file selector to navigate to the *drivers* directory, located in the directory where you unzipped the Arduino files. When the driver has installed, the Found New Hardware Wizard will appear again, saying a new serial port has been found. Follow the same process as before.



It is important that you go through the sequence of steps to install the drivers two times, or the software will not be able to communicate with the board.

On the Mac, the latest Arduino boards, such as the Uno, can be used without additional drivers, but if you are using earlier boards, you will need to install driver software. There is a package named *FTDIUSBSerialDriver*, with a range of numbers after it, inside the disk image. Double-click this and the installer will take you through the process. You will need to know an administrator password to complete the process.

On Linux, most distributions have the driver already installed, but follow the Linux link given in this chapter's [introduction](#) for specific information for your distribution.

Discussion

If the software fails to start, check the troubleshooting section of the Arduino website, <http://arduino.cc/en/Guide/Troubleshooting>, for help solving installation problems.

See Also

Online guides for getting started with Arduino are available at <http://arduino.cc/en/Guide/Windows> for Windows, <http://arduino.cc/en/Guide/MacOSX> for Mac OS X, and <http://www.arduino.cc/playground/Learning/Linux> for Linux.

1.2 Setting Up the Arduino Board

Problem

You want to power up a new board and verify that it is working.

Solution

Plug the board into a USB port on your computer and check that the green LED power indicator on the board illuminates. Standard Arduino boards (Uno, Duemilanove, and Mega) have a green LED power indicator located near the reset switch.

An orange LED near the center of the board (labeled “Pin 13 LED” in Figure 1-4) should flash on and off when the board is powered up (boards come from the factory preloaded with software to flash the LED as a simple check that the board is working).

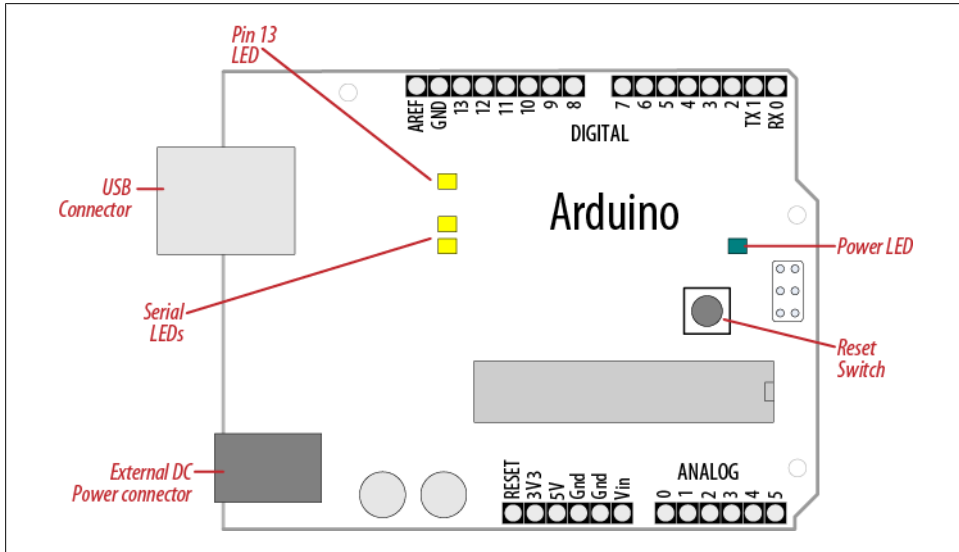


Figure 1-4. Basic Arduino board (Uno and Duemilanove)

Discussion

If the power LED does not illuminate when the board is connected to your computer, the board is probably not receiving power.

The flashing LED (connected to digital output pin 13) is being controlled by code running on the board (new boards are preloaded with the Blink example sketch). If the pin 13 LED is flashing, the sketch is running correctly, which means the chip on the board is working. If the green power LED is on but the pin 13 LED is not flashing, it could be that the factory code is not on the chip; follow the instructions in [Recipe 1.3](#) to load the Blink sketch onto the board to verify that the board is working. If you are not using a standard board, it may not have a built-in LED on pin 13, so check the documentation for details of your board.

See Also

Online guides for getting started with Arduino are available at <http://arduino.cc/en/Guide/Windows> for Windows, <http://arduino.cc/en/Guide/MacOSX> for Mac OS X, and <http://www.arduino.cc/playground/Learning/Linux> for Linux.

A troubleshooting guide can be found at <http://arduino.cc/en/Guide/Troubleshooting>.

1.3 Using the Integrated Development Environment (IDE) to Prepare an Arduino Sketch

Problem

You want to get a sketch and prepare it for uploading to the board.

Solution

Use the Arduino IDE to create, open, and modify sketches that define what the board will do. You can use buttons along the top of the IDE to perform these actions (shown in [Figure 1-5](#)), or you can use the menus or keyboard shortcuts (shown in [Figure 1-6](#)).

The Sketch Editor area is where you view and edit code for a sketch. It supports common text editing keys such as Ctrl-F (⌘+F on a Mac) for find, Ctrl-Z (⌘+Z on a Mac) for undo, Ctrl-C (⌘+C on a Mac) to copy highlighted text, and Ctrl-V (⌘+V on a Mac) to paste highlighted text.

[Figure 1-6](#) shows how to load the Blink sketch (the sketch that comes preloaded on a new Arduino board).

After you've started the IDE, go to the File→Examples menu and select 1.Basics→Blink, as shown in [Figure 1-6](#). The code for blinking the built-in LED will be displayed in the Sketch Editor window (refer to [Figure 1-5](#)).

Before the code can be sent to the board, it needs to be converted into instructions that can be read and executed by the Arduino controller chip; this is called *compiling*. To do this, click the compile button (the top-left button with a triangle inside), or select Sketch→Verify/Compile.

You should see a message that reads “Compiling...” in the message area below the text editing window. After a second or two, a message that reads “Done Compiling” will appear. The black console area will contain the following additional message:

```
Binary sketch size: 1008 bytes (of a 32256 byte maximum)
```

The exact message may differ depending on the Arduino version; it is telling you the size of the sketch and the maximum size that your board can accept.

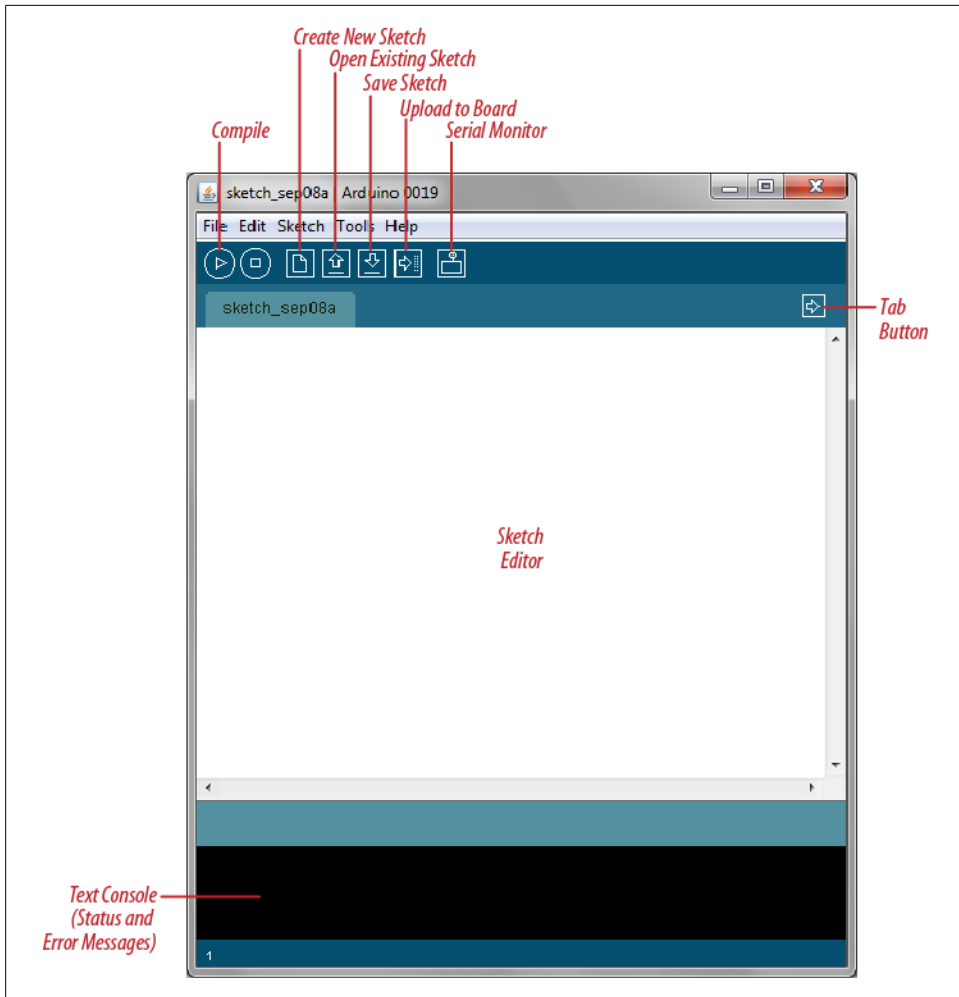


Figure 1-5. Arduino IDE

Discussion

Source code for Arduino is called a *sketch*. The process that takes a sketch and converts it into a form that will work on the board is called *compilation*. The IDE uses a number of command-line tools behind the scenes to compile a sketch. For more information on this, see [Recipe 17.1](#).

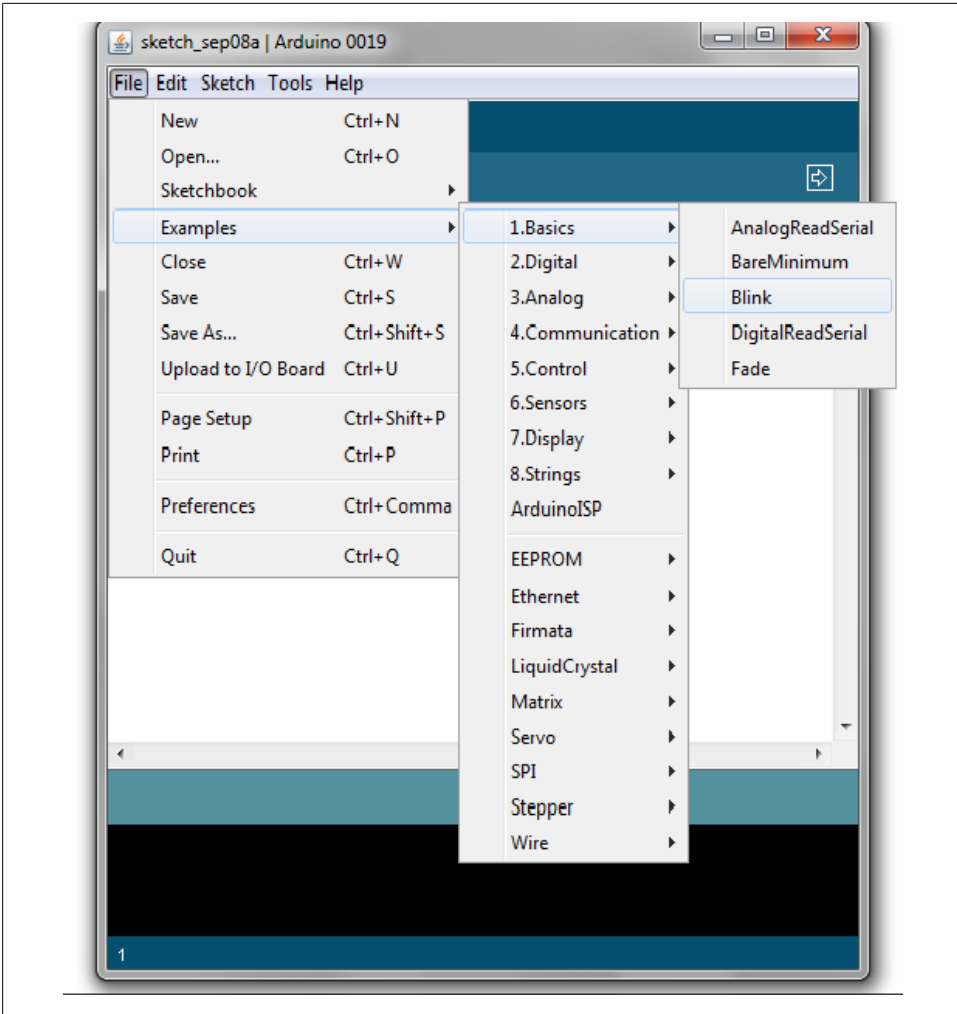


Figure 1-6. IDE menu (selecting the Blink example sketch)

The final message telling you the size of the sketch indicates how much program space is needed to store the controller instructions on the board. If the size of the compiled sketch is greater than the available memory on the board, the following error message is displayed:

Sketch too big; see <http://www.arduino.cc/en/Guide/Troubleshooting#size> for tips on reducing it.

If this happens, you need to make your sketch smaller to be able to put it on the board, or get a board with higher capacity.

If there are errors in the code, the compiler will print one or more error messages in the console window. These messages can help identify the error—see [Appendix D](#) on software errors for troubleshooting tips.



To prevent accidental overwriting of the examples, the Arduino IDE does not allow you to save changes to the provided example sketches. You must rename them using the Save As menu option. You can save sketches you write yourself with the Save button (see [Recipe 1.5](#)).

As you develop and modify a sketch, you should also consider using the File→Save As menu option and using a different name or version number regularly so that as you implement each bit, you can go back to an older version if you need to.



Code uploaded onto the board cannot be downloaded back onto your computer. Make sure you save your sketch code on your computer. You cannot save changes back to the example files; you need to use Save As and give the changed file another name.

See Also

[Recipe 1.5](#) shows an example sketch. [Appendix D](#) has tips on troubleshooting software problems.

1.4 Uploading and Running the Blink Sketch

Problem

You want to transfer your compiled sketch to the Arduino board and see it working.

Solution

Connect your Arduino board to your computer using the USB cable. Load the Blink sketch into the IDE as described in [Recipe 1.3](#).

Next, select Tools→Board from the drop-down menu and select the name of the board you have connected (if it is the standard Uno board, it is probably the first entry in the board list).

Now select Tools→Serial Port. You will get a drop-down list of available serial ports on your computer. Each machine will have a different combination of serial ports, depending on what other devices you have used with your computer.

On Windows, they will be listed as numbered COM entries. If there is only one entry, select it. If there are multiple entries, your board will probably be the last entry.

On the Mac, your board will be listed twice if it is an Uno board:

```
/dev/tty.usbmodem-XXXXXXX  
/dev/cu.usbmodem-XXXXXXX
```

If you have an older board, it will be listed as follows:

```
/dev/tty.usbserial-XXXXXXX  
/dev/cu.usbserial-XXXXXXX
```

Each board will have different values for `XXXXXXX`. Select either entry.

Click on the upload button (in [Figure 1-5](#), it's the fifth button from the left), or choose `File→Upload to I/O board`.

The software will compile the code, as in [Recipe 1.3](#). After the software is compiled, it is uploaded to the board. If you look at your board, you will see the LED stop flashing, and two lights (labeled as Serial LEDs in [Figure 1-4](#)) just below the previously flashing LED should flicker for a couple of seconds as the code uploads. The original light should then start flashing again as the code runs.

Discussion

For the IDE to send the compiled code to the board, the board needs to be plugged into the computer, and you need to tell the IDE which board and serial port you are using.

When an upload starts, whatever sketch is running on the board is stopped (if you were running the Blink sketch, the LED will stop flashing). The new sketch is uploaded to the board, replacing the previous sketch. The new sketch will start running when the upload has successfully completed.



Older Arduino boards and some compatibles do not automatically interrupt the running sketch to initiate upload. In this case, you need to press the Reset button on the board just after the software reports that it is done compiling (when you see the message about the size of the sketch). It may take a few attempts to get the timing right between the end of the compilation and pressing the Reset button.

The IDE will display an error message if the upload is not successful. Problems are usually due to the wrong board or serial port being selected or the board not being plugged in.

If you have trouble identifying the correct port on Windows, try unplugging the board and then selecting `Tools→Serial Port` to see which COM port is no longer on the display list. Another approach is to select the ports, one by one, until you see the lights on the board flicker to indicate that the code is uploading.

See Also

The Arduino troubleshooting page: <http://www.arduino.cc/en/Guide/Troubleshooting>

1.5 Creating and Saving a Sketch

Problem

You want to create a sketch and save it to your computer.

Solution

To open an editor window ready for a new sketch, launch the IDE (see [Recipe 1.3](#)), go to the File menu, and select New. Paste the following code into the Sketch Editor window (it's similar to the Blink sketch, but the blinks last twice as long):

```
const int ledPin = 13;    // LED connected to digital pin 13

void setup()
{
  pinMode(ledPin, OUTPUT);
}

void loop()
{
  digitalWrite(ledPin, HIGH); // set the LED on
  delay(2000);                // wait for two seconds
  digitalWrite(ledPin, LOW);  // set the LED off
  delay(2000);                // wait for two seconds
}
```

Compile the code by clicking the compile button (the top-left button with a triangle inside), or select Sketch→Verify/Compile (see [Recipe 1.3](#)).

Upload the code by clicking on the upload button, or choose File→Upload to I/O board (see [Recipe 1.4](#)). After uploading, the LED should blink, with each flash lasting two seconds.

You can save this sketch to your computer by clicking the Save button, or select File→Save.

You can save the sketch using a new name by selecting the Save As menu option. A dialog box will open where you can enter the filename.

Discussion

When you save a file in the IDE, a standard dialog box for the operating system will open. It suggests that you save the sketch to a folder called *Arduino* in your *My Documents* folder (or your *Documents* folder on a Mac). You can replace the default sketch

name with a meaningful name that reflects the purpose of your sketch. Click Save to save the file.



The default name is the word *sketch* followed by the current date. Sequential letters starting from *a* are used to distinguish sketches created on the same day. Replacing the default name with something meaningful helps you to identify the purpose of a sketch when you come back to it later.

If you use characters that the IDE does not allow (e.g., the space character), the IDE will automatically replace these with valid characters.

Arduino sketches are saved as plain text files with the extension *.pde*. They are automatically saved in a folder with the same name as the sketch.

You can save your sketches to any folder on your computer, but if you use the default folder (the *Arduino* folder in your *Documents* folder) your sketches will automatically appear in the Sketchbook menu of the Arduino software and be easier to locate.



If you have edited one of the examples from the Arduino download, you will not be able to save the changed file using the same filename. This preserves the standard examples intact. If you want to save a modified example, you will need to select another location for the sketch.

After you have made changes, you will see a dialog box asking if you want to save the sketch when a sketch is closed.



The § symbol following the name of the sketch in the top bar of the IDE window indicates that the sketch code has changes that have not yet been saved on the computer. This symbol is removed when you save the sketch.

The Arduino software does not provide any kind of version control, so if you want to be able to revert to older versions of a sketch, you can use Save As regularly and give each revision of the sketch a slightly different name.

Frequent compiling as you modify or add code is a good way to check for errors as you write your code. It will be easier to find and fix any errors because they will usually be associated with what you have just written.



Once a sketch has been uploaded onto the board there is no way to download it back to your computer. Make sure you save any changes to your sketches that you want to keep.

If you open sketches you get from other people that are not in a folder with the same name as the sketch, the IDE will tell you and you can click OK to put them in a folder with the same name.



Sketches must be located in a folder with the same name as the sketch. The IDE will create the folder automatically when you save a new sketch.

1.6 Using Arduino

Problem

You want to get started with a project that is easy to build and fun to use.

Solution

This recipe provides a taste of some of the techniques that are covered in detail in later chapters.

The sketch is based on the LED blinking code from the previous recipe, but instead of using a fixed delay, the rate is determined by a light-sensitive sensor called a light dependent resistor or LDR (see [Recipe 6.2](#)). Wire the LDR as shown in [Figure 1-7](#).

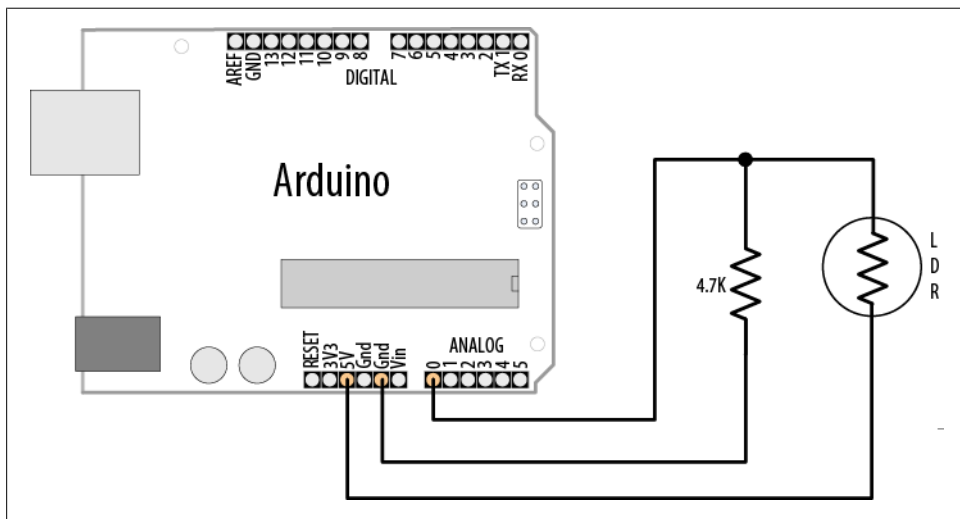


Figure 1-7. Arduino with light dependent resistor

The following sketch reads the light level of an LDR connected to analog pin 0. The light level striking the LDR will change the blink rate of the internal LED connected to pin 13:

```

const int ledPin = 13;    // LED connected to digital pin 13
const int sensorPin = 0; // connect sensor to analog input 0

void setup()
{
  pinMode(ledPin, OUTPUT); // enable output on the led pin
}

void loop()
{
  int rate = analogRead(sensorPin); // read the analog input
  Serial.println(rate);
  rate = map(rate, 200,800,minDuration, maxDuration); // convert to blink rate
  digitalWrite(ledPin, HIGH); // set the LED on
  delay(rate); // wait duration dependent on light level
  digitalWrite(ledPin, LOW); // set the LED off
  delay(rate);
}

```

Discussion

The value of the 4.7K resistor is not critical. Anything from 1K to 10K can be used. The light level on the LDR will change the voltage level on analog pin 0. The `analogRead` command (see [Chapter 6](#)) provides a value that ranges from around 200 when the LDR is dark to 800 or so when it is very bright. This value determines the duration of the LED on and off times, so the blink rate increases with light intensity.

You can scale the blink rate by using the Arduino `map` function as follows:

```

const int ledPin = 13;    // LED connected to digital pin 13
const int sensorPin = 0; // connect sensor to analog input 0

// the next two lines set the min and max delay between blinks
const int minDuration = 100; // minimum wait between blinks
const int maxDuration = 1000; // maximum wait between blinks

void setup()
{
  pinMode(ledPin, OUTPUT); // enable output on the led pin
}

void loop()
{
  int rate = analogRead(sensorPin); // read the analog input
  // the next line scales the blink rate between the min and max values
  rate = map(rate, 200,800,minDuration, maxDuration); // convert to blink rate
  digitalWrite(ledPin, HIGH); // set the LED on
  delay(rate); // wait duration dependent on light level
  digitalWrite(ledPin, LOW); // set the LED off
  delay(rate);
}

```

[Recipe 5.7](#) provides more details on using the `map` function to scale values.

If you want to view the value of the rate variable on your computer, you can print this to the Arduino Serial Monitor as shown in the revised loop code that follows. The sketch will display the blink rate in the Serial Monitor. You open the Serial Monitor window in the Arduino IDE (see [Chapter 4](#) for more on using the Serial Monitor):

```
const int ledPin = 13;    // LED connected to digital pin 13
const int sensorPin = 0;  // connect sensor to analog input 0

// the next two lines set the min and max delay between blinks
const int minDuration = 100; // minimum wait between blinks
const int maxDuration = 1000; // maximum wait between blinks

void setup()
{
  pinMode(ledPin, OUTPUT); // enable output on the led pin
  Serial.begin(9600);      // initialize Serial
}

void loop()
{
  int rate = analogRead(sensorPin); // read the analog input
  // the next line scales the blink rate between the min and max values
  rate = map(rate, 200, 800, minDuration, maxDuration); // convert to blink rate
  Serial.println(rate); // print rate to serial monitor
  digitalWrite(ledPin, HIGH); // set the LED on
  delay(rate); // wait duration dependent on light level
  digitalWrite(ledPin, LOW); // set the LED off
  delay(rate);
}
```

You can use the LDR to control the pitch of a sound by connecting a small speaker to the pin, as shown in [Figure 1-8](#).

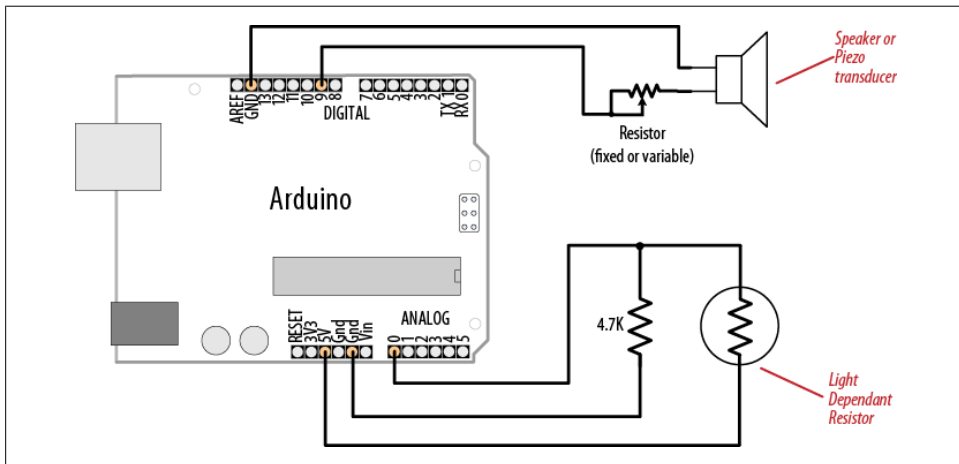


Figure 1-8. Connections for a speaker with the LDR circuit

You will need to increase the on/off rate on the pin to a frequency in the audio spectrum. This is achieved, as shown in the following code, by dividing the rate by 100 in the line after the `map` function:

```
const int ledPin = 13;    // LED connected to digital pin 13
const int sensorPin = 0; // connect sensor to analog input 0

const int minDuration = 100; // minimum wait between blinks
const int maxDuration = 1000; // maximum wait between blinks

void setup()
{
  pinMode(ledPin, OUTPUT); // enable output on the led pin
}

void loop()
{
  int sensorReading = analogRead(sensorPin); // read the analog input
  int rate = map(sensorReading, 200,800,minDuration, maxDuration);
  rate = rate / 100; // add this line for audio frequency
  digitalWrite(ledPin, HIGH); // set the LED on
  delay(rate); // wait duration dependent on light level
  digitalWrite(ledPin, LOW); // set the LED off
  delay(rate);
}
```

See Also

See [Chapter 9](#) for more on creating sound with Arduino.