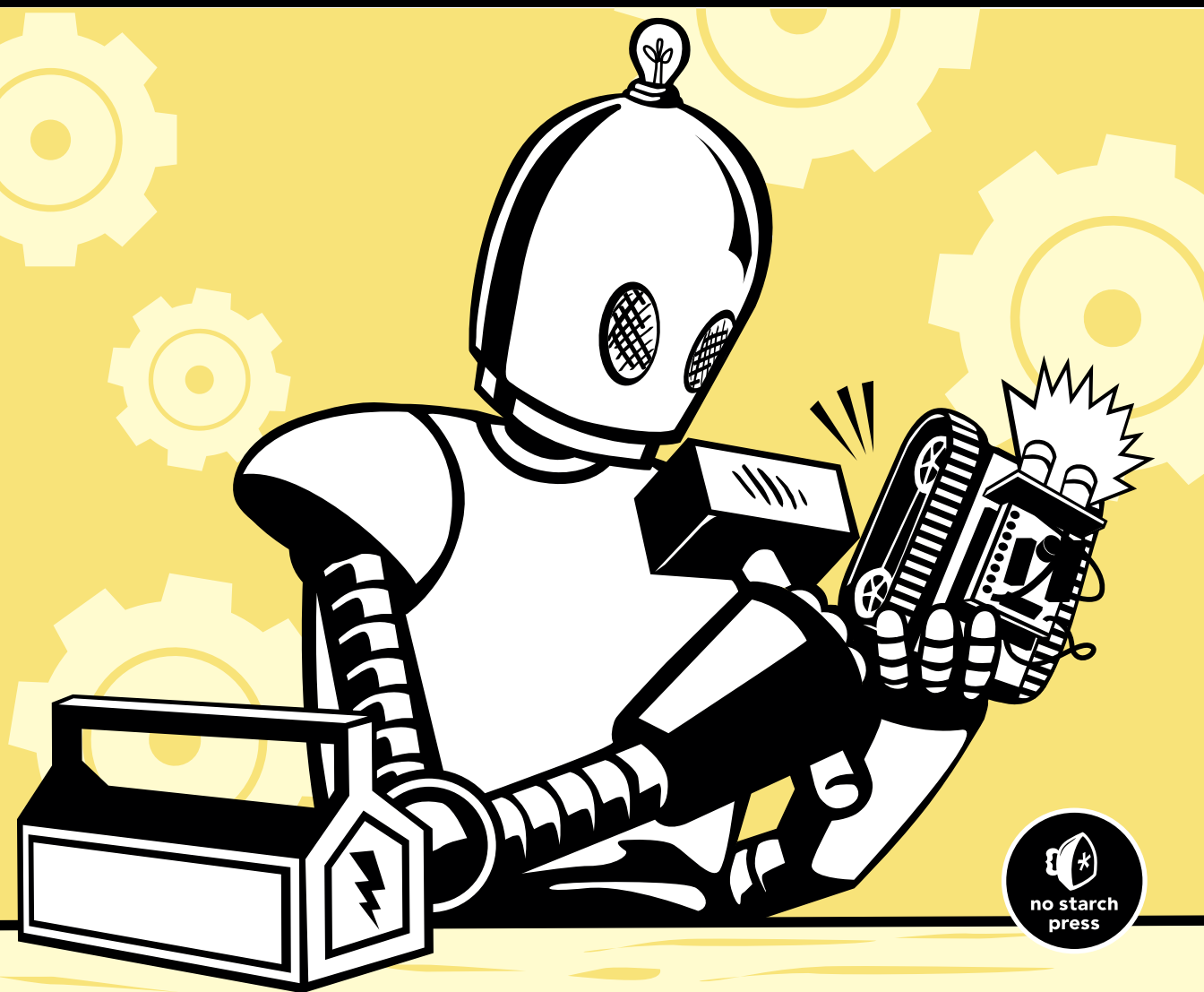
ARDUINO WORKSHOP

A *HANDS-ON* INTRODUCTION WITH *65 PROJECTS*

JOHN BOXALL



no starch press

ARDUINO WORKSHOP

ARDUINO WORKSHOP

A Hands-On Introduction with 65 Projects

by John Boxall

no starch press

San Francisco

Printed in USA

First printing

17 16 15 14 13 1 2 3 4 5 6 7 8 9

ISBN-10: 1-59327-448-3
ISBN-13: 978-1-59327-448-1

Publisher: William Pollock
Production Editor: Serena Yang
Cover Illustration: Charlie Wylie
Interior Design: Octopod Studios
Developmental Editor: William Pollock
Technical Reviewer: Marc Alexander
Copyeditor: Lisa Theobald
Compositor: Susan Glinert Stevens
Proofreader: Emelie Battaglia

Circuit diagrams made using Fritzing (*http://fritzing.org/*)

For information on distribution, translations, or bulk sales, please contact No Starch Press, Inc. directly:

No Starch Press, Inc.
38 Ringold Street, San Francisco, CA 94103
phone: 415.863.9900; fax: 415.863.9950; info@nostarch.com; www.nostarch.com

Library of Congress Cataloging-in-Publication Data
A catalog record of this book is available from the Library of Congress.

For the two people who have always believed in me:
my mother and my dearest Kathleen

BRIEF CONTENTS

CONTENTS IN DETAIL

ACKNOWLEDGMENTS

First of all, a huge thank you to the Arduino team: Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino, and David Mellis. Without your vision, thought, and hard work, none of this would have been possible.

Many thanks to my technical reviewer Marc Alexander for his contributions, expertise, suggestions, support, thoughts, and long conversations, and for having the tenacity to follow through with such a large project.

I also want to thank the following organizations for their images and encouragement: adafruit industries, Agilent Technologies, Gravitech, Freetronics, Oomlout, Seeed Studio, Sharp Corporation, and SparkFun. Furthermore, a big thanks to Freetronics for the use of their excellent hardware products. And thank you to all those who have contributed their time making Arduino libraries, which makes life much easier for everyone.

Kudos and thanks to the Fritzing team for their wonderful open source circuit schematic design tool, which I've used throughout this book.

And a thank you to the following people (in no particular order) from whom I've received encouragement, inspiration and support: Iraphne Childs, Limor Fried, Jonathan Oxer, Philip Lindsay, Nicole Kilah, Ken Shirriff, Nathan Kennedy, David Jones, and Nathan Seidle.

Finally, thank you to everyone at No Starch Press, including Sondra Silverhawk for suggesting the book; Serena Yang for her dedicated editing, endless patience, and suggestions; and Bill Pollock for his support and guidance and for convincing me that sometimes there is a better way to explain something.

1

GETTING STARTED

Have you ever looked at some gadget and wondered how it *really* worked? Maybe it was a remote control boat, the system that controls an elevator, a vending machine, or an electronic toy? Or have you wanted to create your own robot or electronic signals for a model railroad, or perhaps you'd like to capture and analyze weather data over time? Where and how do you start?

The Arduino board (shown in Figure 1-1) can help you find some of the answers to the mysteries of electronics in a hands-on way. The original creation of Massimo Banzi and David Cuartielles, the Arduino system offers an inexpensive way to build interactive projects, such as remote-controlled robots, GPS tracking systems, and electronic games.

The Arduino project has grown exponentially since its introduction in 2005. It's now a thriving industry, supported by a community of people united with the common bond of creating something new. You'll find both individuals and groups, ranging from interest groups and clubs to local hackerspaces and educational institutions, all interested in toying with the Arduino.

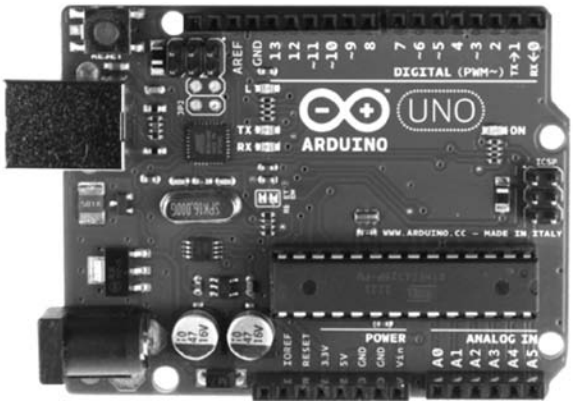


Figure 1-1: The Arduino board

To get a sense of the variety of Arduino projects in the wild, you can simply search the Internet. You'll find a list of groups offering introductory programs and courses with like-minded, creative people.

The Possibilities Are Endless

A quick scan through this book will show you that you can use the Arduino to do something as simple as blinking a small light, or even something more complicated, such as interacting with a cellular phone—and many different things in between.

For example, have a look at Philip Lindsay's device, shown in Figure 1-2. It can receive text messages from cellular phones and display them on a large sign for use in dance halls. This device uses an Arduino board and a cellular phone shield to receive text messages from other phones (similar to Project 65). The text message is sent to a pair of large, inexpensive dot-matrix displays for everyone to see.

Figure 1-2: SMS (short message service) text marquee

You can purchase large display boards that are easy to interface with an Arduino, so you don't have to make your own display from scratch. (For more information, visit *http://www.labradoc.com/i/follower/p/project-sms-text-scroller*.)

How about creating a unique marriage proposal? Tyler Cooper wanted an original way to propose to his girlfriend, so he built what he calls a "reverse geocache box"—a small box that contained an engagement ring, as shown in Figure 1-3. When the box was taken to a certain area (measured by the internal GPS), it unlocked to reveal a romantic message and the ring. You can easily reproduce this device using an Arduino board, a GPS receiver, and an LCD module (as used in Chapter 13), with a small servo motor that acts as a latch to keep the box closed until it's in the correct location. The code required to create this is quite simple—something you could create in a few hours. The most time-consuming part is choosing the appropriate box in which to enclose the system. (For more information, visit *http://learn.adafruit.com/reverse-geocache-engagement-box/*.)

Figure 1-3: Marriage proposal via Arduino

Here's another example. Kurt Schulz was interested in monitoring the battery charge level of his moped. However, after realizing how simple it is to work with Arduino, his project morphed into what he calls the "Scooterputer": a complete moped management system. The Scooterputer can measure the battery voltage, plus it can display the speed, distance traveled, tilt angle, temperature, time, date, GPS position, and more. It also contains a cellular phone shield that can be controlled remotely, allowing remote tracking of the moped and engine shutdown in case it's stolen. The entire system can be controlled with a small touchscreen, shown in

Figure 1-4. Each feature can be considered a simple building block, and anyone could create a similar system in a couple of weekends. (See *http://www.janspace.com/b2evolution/arduino.php/2010/06/26/scooterputer/*.)

Figure 1-4: The Scooterputer display (courtesy of Kurt Schulz)

Then there's John Sarik, who enjoys the popular Sudoku math puzzles; he also likes working with Nixie numeric display tubes. With those two drivers in mind, John created a huge 81-digit Sudoku game computer! The user can play a full 9-by-9 game, with the Arduino in control of the digits and checking for valid entries. Although this project might be considered a more advanced type, it is certainly achievable and the electronics are not complex. The device is quite large and looks great mounted on a wall, as shown in Figure 1-5. (See *http://trashbearlabs.wordpress.com/2010/07/09/nixie-sudoku/*.)

The team at Oomlout even used the Arduino to create a TwypeWriter. They fitted an Arduino board with an Ethernet shield interface connected to the Internet, which searches Twitter for particular keywords. When a keyword is found, the tweet is sent to an electric typewriter for printing. The Arduino board is connected to the typewriter's keyboard circuit, which allows it to emulate a real person typing, as shown in Figure 1-6. (See *http://oomlout.co.uk/blog/twitter-monitoring-typewritter-twypwriter/*.)

These are only a few random examples of what is possible using an Arduino. You can create your own projects without much difficulty—and after you've worked through this book, they are certainly not out of your reach.

Figure 1-5: Nixie tube Sudoku

Figure 1-6: The TwypeWriter

Strength in Numbers

The Arduino platform increases in popularity every day. If you're more of a social learner and enjoy class-oriented situations, search the Web for "Cult of Arduino" to see what people are making and to find Arduino-related groups. Members of Arduino groups introduce the world of Arduino from an artist's perspective. Many group members work to create a small Arduino-compatible board at the same time. These groups can be a lot of fun, introduce you to interesting people, and let you share your Arduino knowledge with others.

Parts and Accessories

As with any other electronic device, the Arduino is available from many retailers that offer a range of products and accessories. When you're shopping, be sure to purchase the original Arduino, not a knock-off, or you run the risk of receiving faulty or poorly performing goods; why risk your project with an inferior board that could end up costing you more in the long run? For a list of Arduino suppliers, visit *http://arduino.cc/en/Main/Buy/*.

Here's a list of current suppliers (in alphabetical order) that I recommend for your purchases of Arduino-related parts and accessories:

- Adafruit Industries (*http://www.adafruit.com/*)
- DigiKey (*http://www.digikey.com/*)
- Jameco Electronics (*http://www.jameco.com/*)
- Little Bird Electronics (*http://www.littlebirdelectronics.com/*)
- Newark (*http://www.newark.com/*)
- nicegear (*http://www.nicegear.co.nz/*)
- Oomlout (*http://www.oomlout.co.uk/*)
- RadioShack (*http://www.radioshack.com/*)
- RS Components (*http://www.rs-components.com/*)
- SparkFun Electronics (*http://www.sparkfun.com/*)

As you'll see in this book, I use several Arduino-compatible products from Freetronics (*http://www.freetronics.com/*). However, you will find that all the required parts are quite common and easily available from various resellers.

But don't go shopping yet. Take the time to read the first few chapters to get an idea of what you'll need so that you won't waste money buying unnecessary things immediately.

Required Software

You should be able to program your Arduino with just about any computer using a piece of software called an *integrated development environment (IDE)*. To run this software, your computer should have one of the following operating systems installed:

- Mac OS X or higher
- Windows XP 32- or 64-bit, or higher
- Linux 32- or 64-bit (Ubuntu or similar)

Now is a good time to download and install the IDE, so jump to the heading that matches your operating system and follow the instructions. Make sure you have or buy the matching USB cable for your Arduino from the supplier as well. Even if you don't have your Arduino board yet, you can still download and explore the IDE. Because the IDE version number can change quite rapidly, the number in this book may not match the current version, but the instructions should still work.

NOTE *Unfortunately, as this book went to press, there were issues with Windows 8 installations. If you have Windows 8, visit the Arduino Forum at* http://arduino.cc/forum/index.php/topic,94651.15.html *for guidance and discussion.*

Mac OS X

In this section, you'll find instructions for downloading and configuring the Arduino IDE in Mac OS X.

Installing the IDE

To install the IDE on your Mac, follow these instructions:

1. Using a web browser such as Safari, visit the software download page located at *http://arduino.cc/en/Main/Software/*, as shown in Figure 1-7.

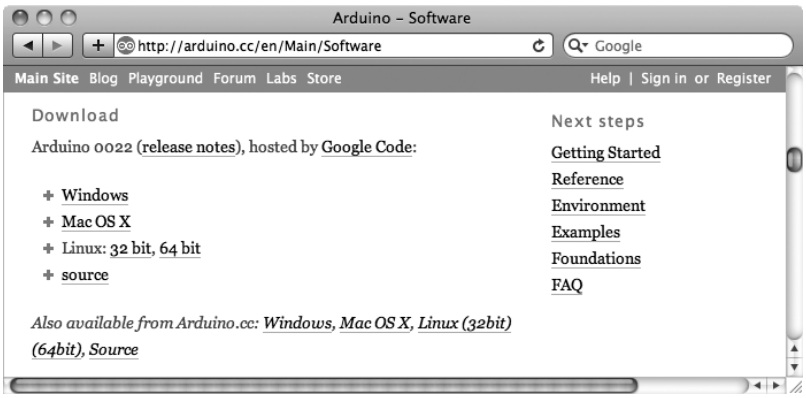


Figure 1-7: The IDE download page in Safari

2. Click the **Mac OS X** link. The file will start downloading, and it will appear in the Downloads window shown in Figure 1-8.

Figure 1-8: File download is complete.

3. Once it's finished downloading, double-click the file to start the installation process. You will then be presented with the window shown in Figure 1-9.

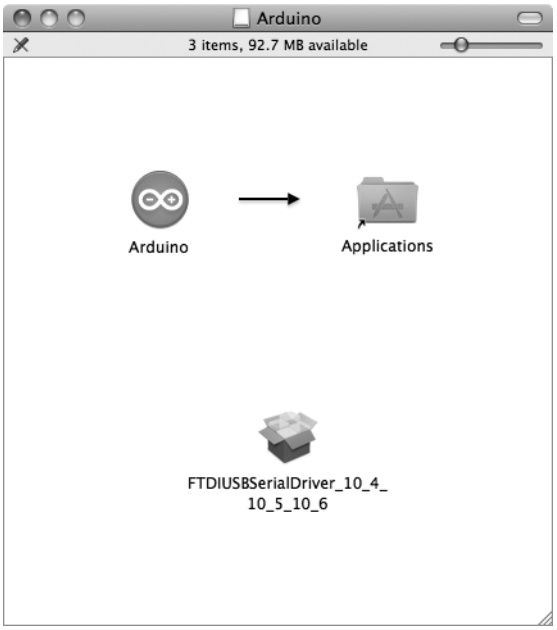


Figure 1-9: Your new Arduino IDE folder

NOTE *The third file icon shown in Figure 1-9 needs to be installed only if you have an Arduino board older than the current Uno.*

4. Drag the Arduino icon over the Applications folder and release the mouse button. A temporary status window will appear as the file is copied.
5. Now connect your Arduino to your Mac with the USB cable. After a moment, the dialog shown in Figure 1-10 will appear.

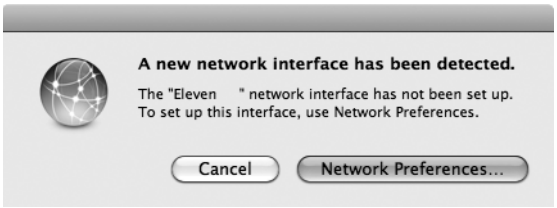


Figure 1-10: A new Arduino board is detected. Your dialog may read Uno *instead of* Eleven.

6. Click **Network Preferences...**, and then click **Apply** in the Network box. You can ignore the "not configured" status message.

Setting Up the IDE

Once you have downloaded the IDE, use the following instructions to open and configure the IDE:

1. Open the Applications folder in Finder (shown in Figure 1-11) and double-click the Arduino icon.

Figure 1-11: Your Applications folder

2. A window may appear warning you about opening a web app. If it does, click **Open** to continue. You will then be presented with the IDE, as shown in Figure 1-12.

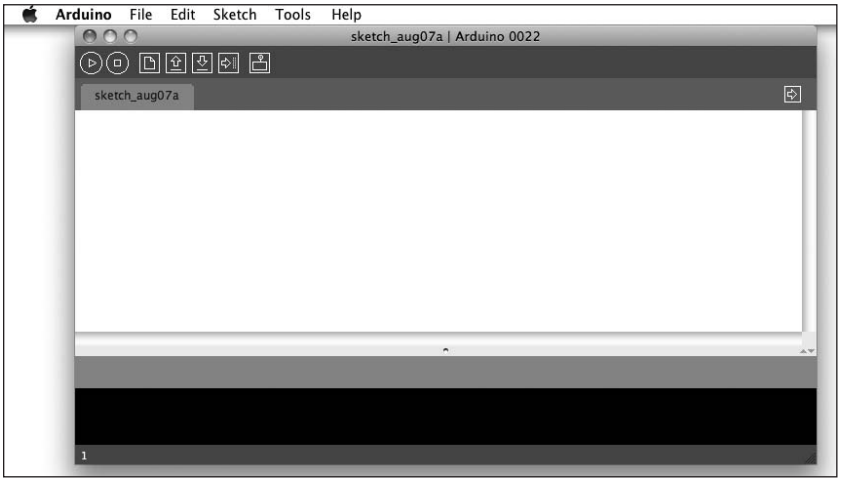


Figure 1-12: The IDE in Mac OS X

3. You're almost there—just two more things to do before your Arduino IDE is ready to use. First, you need to tell the IDE which type of socket the Arduino is connected to. Select **Tools ▸ Serial Port** and select the **/dev/tty.usbmodem1d11** option, as shown in Figure 1-13.

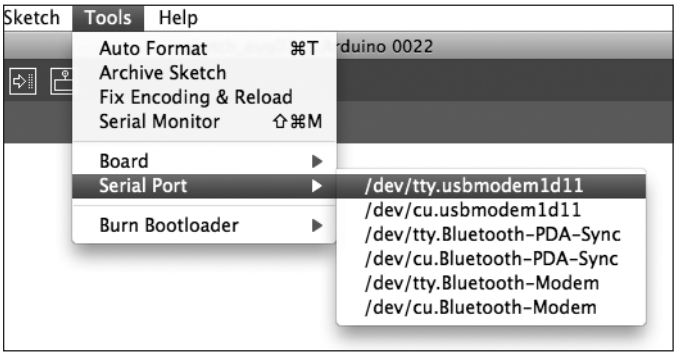


Figure 1-13: Selecting the USB port

4. The final step is to tell the IDE which Arduino board you have connected. This is crucial, since Arduino boards do differ. For example, if you have the most common board, the Uno, then select **Tools ▸ Board ▸ Arduino Uno**, as shown in Figure 1-14. The differences in Arduino boards are explained in more detail in Chapter 11.

Now your hardware and software are ready to work for you. Next, move on to "Safety" on page 18.

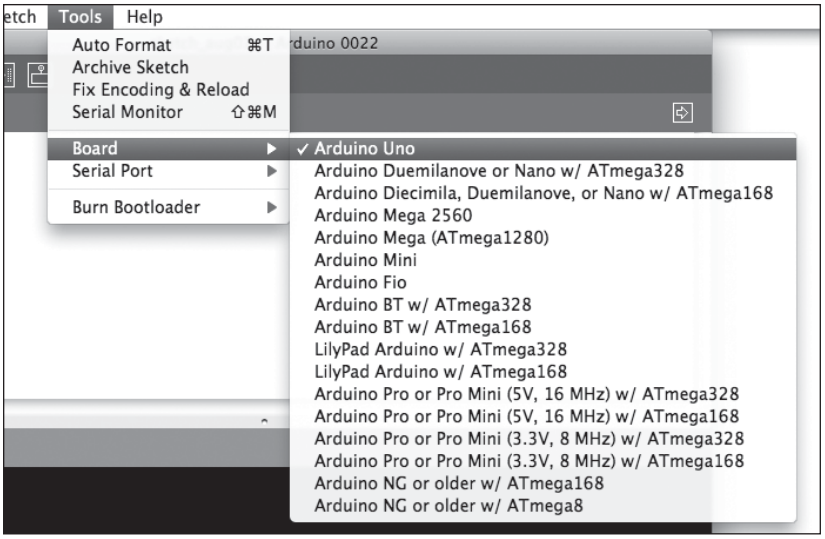


Figure 1-14: Selecting the correct Arduino board

Windows XP and Later

In this section, you'll find instructions for downloading the IDE, installing drivers, and configuring the IDE in Windows.

Installing the IDE

To install the Arduino IDE for Windows, follow these instructions:

1. Using a web browser such as Firefox, visit the software download page located at *http://arduino.cc/en/Main/Software/*, as shown in Figure 1-15.

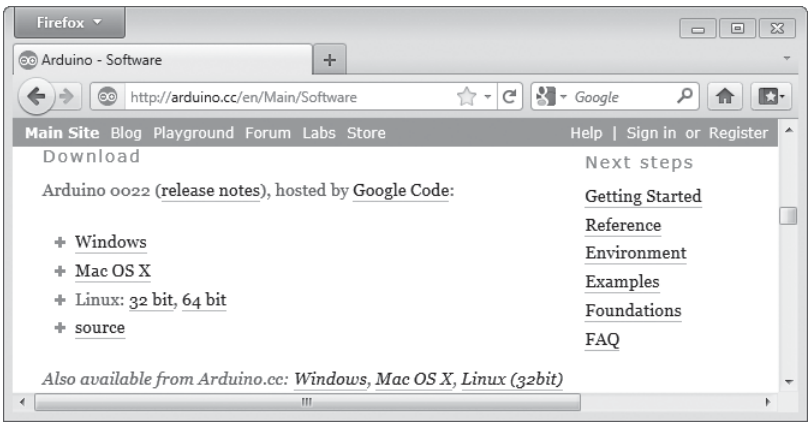


Figure 1-15: The IDE download page in Windows Firefox

2. Click the **Windows** link, and the dialog shown in Figure 1-16 will appear. Select **Open with Windows Explorer**, and then click **OK**. The file will start to download, as shown in Figure 1-17.

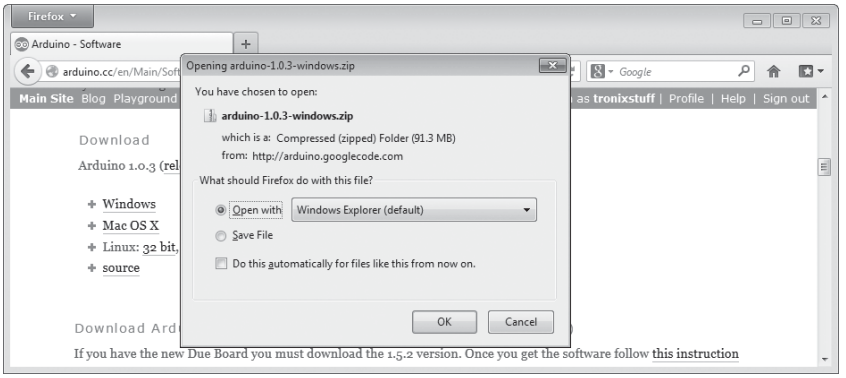


Figure 1-16: Downloading the file

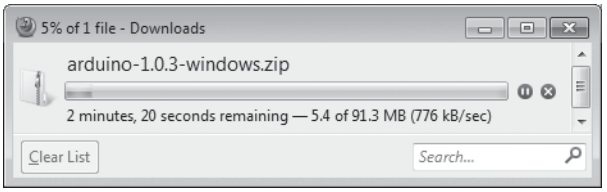


Figure 1-17: Firefox shows the progress of your download.

3. Once the download is complete, double-click the file, and the window shown in Figure 1-18 will appear.

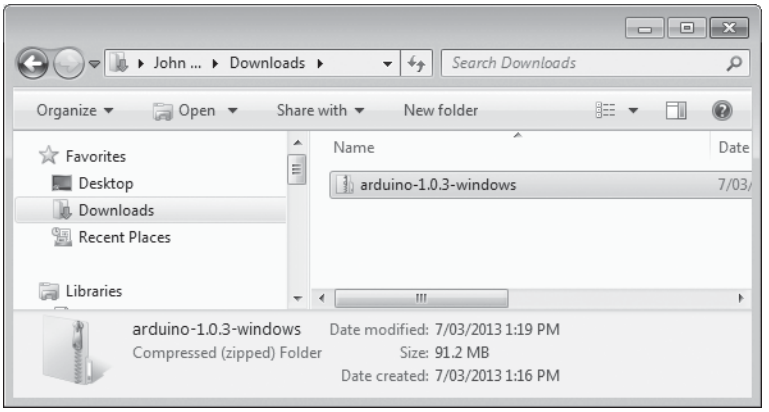


Figure 1-18: The IDE package

4. Copy the folder named *arduino-0022* (or something similar) to the location where you store your applications. Once the copying is finished, locate the folder and open it to reveal the Arduino application icon, as shown in Figure 1-19. You may wish to copy the icon and place a shortcut on the desktop for easier access in the future.

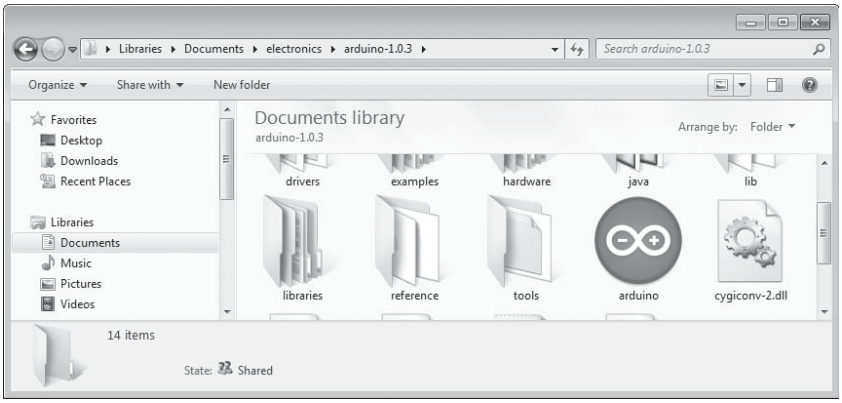


Figure 1-19: Your IDE folder with the Arduino application icon selected

Installing Drivers

The next task is to install the drivers for your Arduino board's USB interface.

1. Connect your Arduino to your PC with the USB cable. After a few moments an error message will be displayed, which will say something like "Device driver software not successfully installed." Just close that dialog or balloon.
2. Navigate to the Windows Control Panel. Open the Device Manager and scroll down until you see the Arduino, as shown in Figure 1-20.

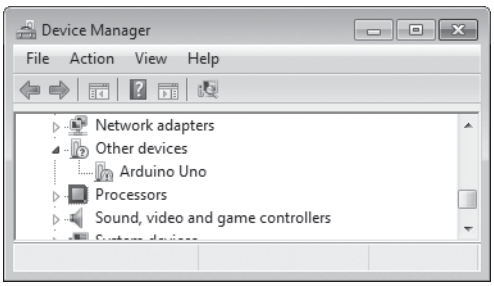


Figure 1-20: The Device Manager

3. Right-click **Arduino Uno** under Other Devices and select **Update Driver Software**. Then, select the **Browse my computer for driver software** option that appears in the next dialog. Another Browse For Folder dialog will appear; click **Browse**, and navigate to the *drivers* folder in the newly installed Arduino software folder (shown in Figure 1-21). Click **OK**.

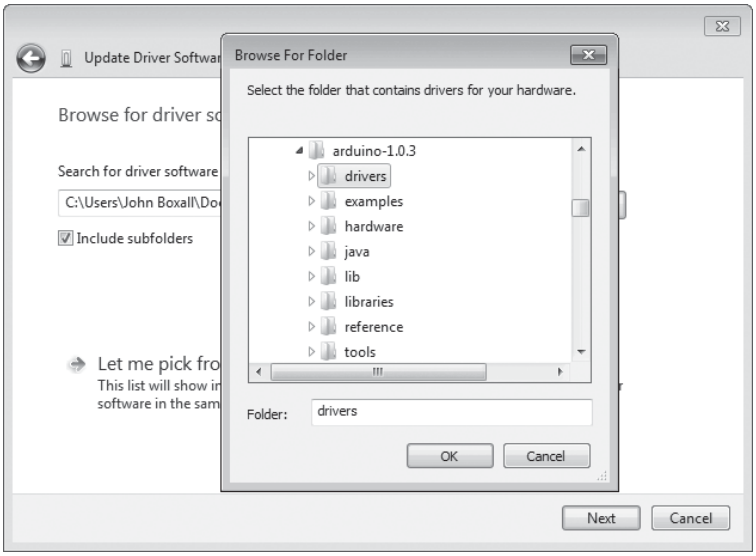


Figure 1-21: Locating the drivers *folder*

4. Click **Next** in the dialog that follows. Windows may present a message stating that it "cannot verify the publisher of the driver software." Click **Install this software anyway**. After a short wait, Windows will tell you that the driver is installed and the COM port number the Arduino is connected to, as shown in Figure 1-22.

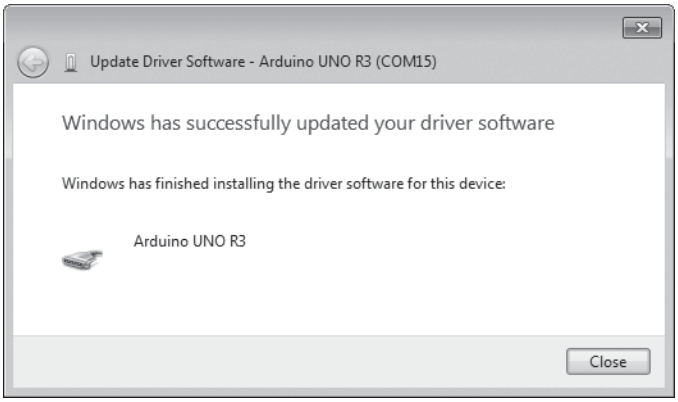


Figure 1-22: The drivers have been updated successfully.

Setting Up the IDE

Okay, we're almost there—just two more things to do to finish setting up the IDE.

1. Open the Arduino IDE. You need to tell the IDE which type of socket the Arduino is connected to by selecting **Tools ▸ Serial Port** and selecting the COM port number that appeared in the Update Driver Software window.
2. The final step is to tell the IDE which Arduino board we have connected. This is crucial, as the Arduino boards do differ. For example, if you have the Uno, select **Tools ▸ Board ▸ Arduino Uno**. The differences in Arduino boards are explained in more detail in Chapter 11.

Now that your Arduino IDE is set up, you can move on to "Safety" on page 18.

Ubuntu Linux 9.04 and Later

If you are running Ubuntu Linux, here are instructions for downloading and setting up the Arduino IDE.

Installing the IDE

Use the following instructions to install the IDE:

1. Using a web browser such as Firefox, visit the software download page located at *http://arduino.cc/en/Main/Software/*, as shown in Figure 1-23.

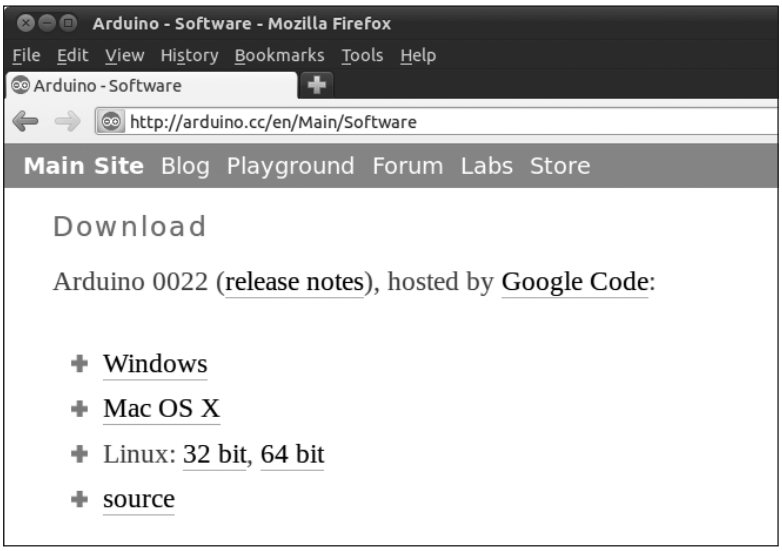


Figure 1-23: The IDE download page in Ubuntu Firefox

2. Click the Linux **32-bit** or **64-bit** link, depending on your system. When the dialog in Figure 1-24 appears, select **Open with Archive Manager** and click **OK**.

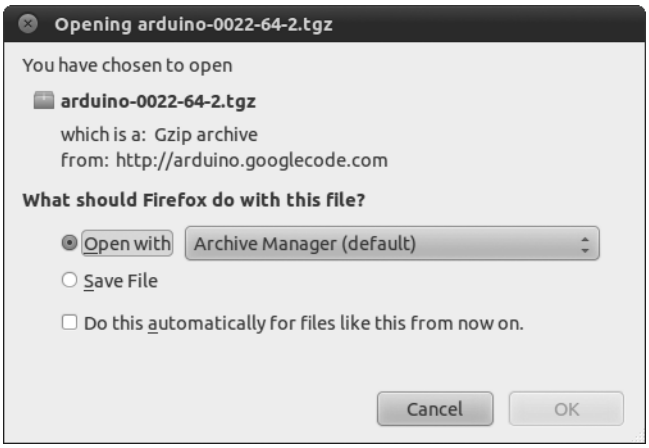


Figure 1-24: Downloading the file

3. After the file has downloaded, it will be displayed in the Archive Manager, as shown in Figure 1-25. Copy the *arduino-0022* folder (or something similar) to your usual application or Home folder.

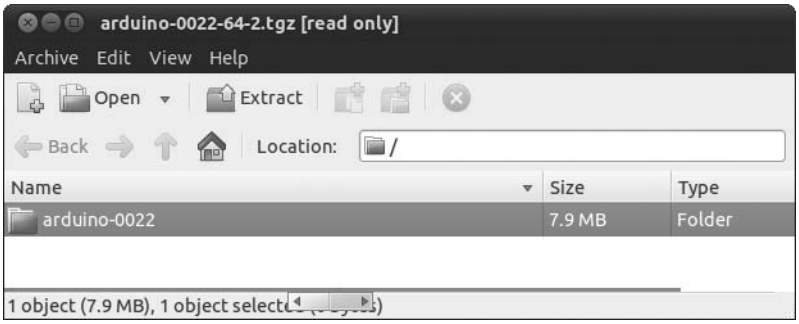


Figure 1-25: The IDE package

Setting Up the IDE

Next, you'll configure the IDE.

1. Connect your Arduino to your PC with the USB cable. At this point you want to run the Arduino IDE, so locate the *arduino-0022* folder you copied earlier and double-click the *arduino* file that's selected in Figure 1-26.

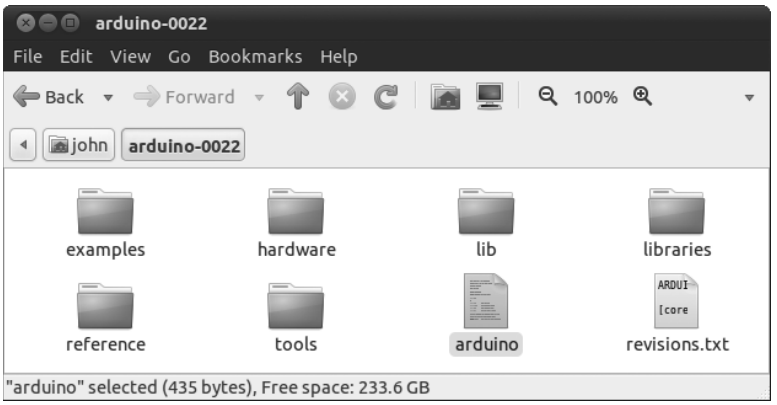


Figure 1-26: Your Arduino IDE folder with the arduino *file selected*

2. If the dialog shown in Figure 1-27 appears, click **Run**, and you will be presented with the IDE, as shown in Figure 1-28.

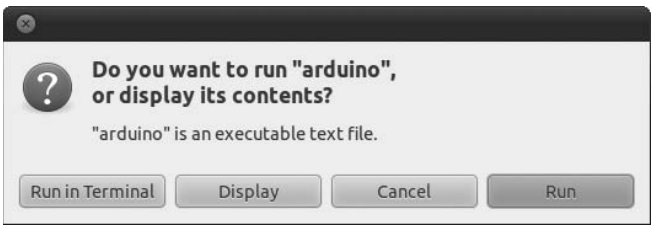


Figure 1-27: Granting permission to run the IDE

Figure 1-28: The IDE in Ubuntu

3. Now that the IDE is running, we need to tell it which type of socket the Arduino is connected to. Select **Tools ▸ Serial Port** and select the **/dev/ttyACM*x*** port, where ***x*** is a single digit (there should be only one port with a name like this).
4. Next, tell the IDE which Arduino you have connected. This is crucial, as Arduino boards do differ. For example, if you have the Uno, select **Tools ▸ Board ▸ Arduino Uno**. The differences in Arduino boards are explained in more detail in Chapter 11.

   Now your hardware and software are ready to work for you.

Safety

As with any hobby or craft, it's up to you to take care of yourself and those around you. As you'll see in this book, I discuss working with basic hand tools, battery-powered electrical devices, sharp knives, and cutters—and sometimes soldering irons. At no point in your projects should you work with the mains current. Leave that to a licensed electrician who is trained for such work. Remember that contacting the mains current will kill you.

Looking Ahead

You're about to embark on a fun and interesting journey, and you'll be creating things you may never have thought possible. You'll find 65 Arduino projects in this book, ranging from the very simple to the relatively complex. All are designed to help you learn and make something useful. So let's go!

2

EXPLORING THE ARDUINO BOARD AND THE IDE

In this chapter you'll explore the Arduino board as well as the IDE software that you'll use to create and upload Arduino *sketches* (Arduino's name for its programs) to the Arduino board itself. You'll learn the basic framework of a sketch and some basic functions that you can implement in a sketch, and you'll create and upload your first sketch.

The Arduino Board

What exactly is Arduino? According to the Arduino website (*http://www.arduino.cc/*), it is

> an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software. It's intended for artists, designers, hobbyists, and anyone interested in creating interactive objects or environments.

In simple terms, the Arduino is a tiny computer system that can be programmed with your instructions to interact with various forms of input and output. The current Arduino board model, the Uno, is quite small in size compared to the average human hand, as you can see in Figure 2-1.

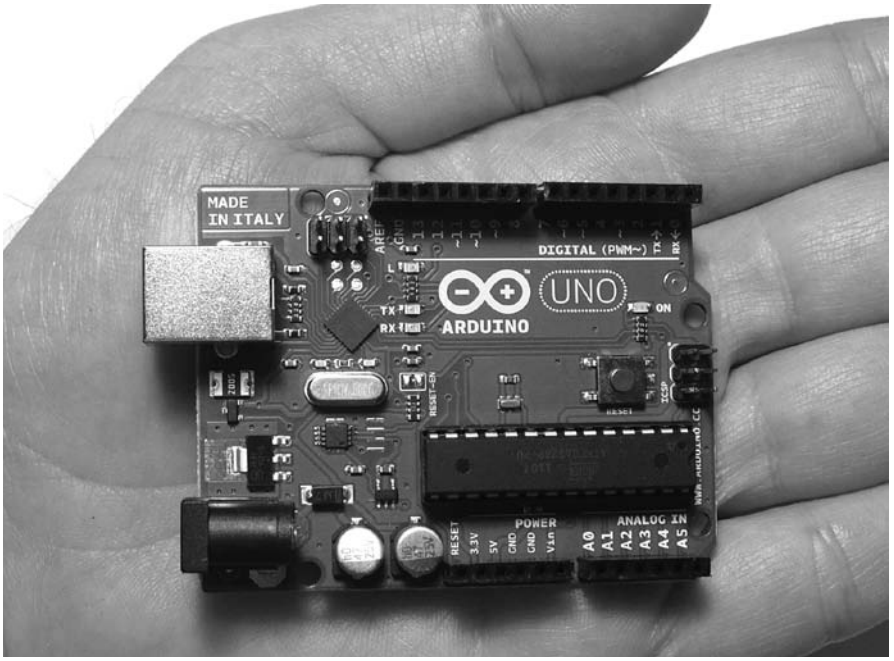


Figure 2-1: An Arduino Uno is quite small.

Although it might not look like much to the new observer, the Arduino system allows you to create devices that can interact with the world around you. By using an almost unlimited range of input and output devices, sensors, indicators, displays, motors, and more, you can program the exact interactions required to create a functional device. For example, artists have created installations with patterns of blinking lights that respond to the movements of passers-by, high school students have built autonomous robots that can detect an open flame and extinguish it, and geographers have designed systems that monitor temperature and humidity and transmit this data back to their offices via text message. In fact, you'll find an almost infinite number of examples with a quick search on the Internet.

Now let's move on and explore our Arduino Uno *hardware* (in other words, the "physical part") in more detail and see what we have. Don't worry too much about understanding what you see here, because all these things will be discussed in greater detail in later chapters.

Let's take a quick tour of the Uno. Starting at the left side of the board, you'll see two connectors, as shown in Figure 2-2.

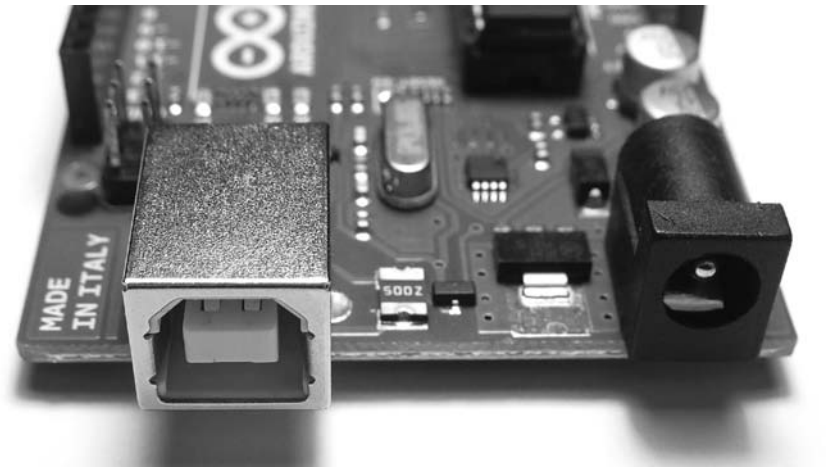


Figure 2-2: The USB and power connectors

On the far left is the Universal Serial Bus (USB) connector. This connects the board to your computer for three reasons: to supply power to the board, to upload your instructions to the Arduino, and to send data to and receive it from a computer. On the right is the power connector. Through this connector, you can power the Arduino with a standard mains power adapter.

At the lower middle is the heart of the board: the microcontroller, as shown in Figure 2-3.

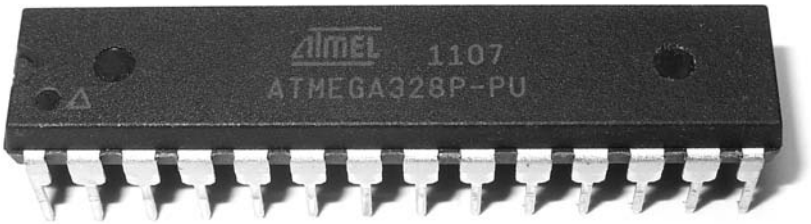


Figure 2-3: The microcontroller

The *microcontroller* is the "brains" of the Arduino. It is a tiny computer that contains a processor to execute instructions, includes various types of memory to hold data and instructions from our sketches, and provides various avenues of sending and receiving data. Just below the microcontroller are two rows of small sockets, as shown in Figure 2-4.

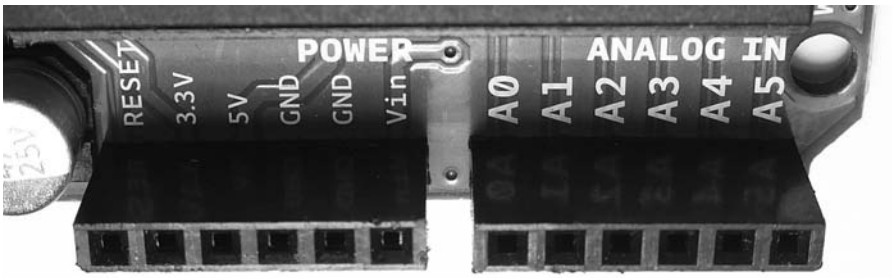


Figure 2-4: The power and analog sockets

The first row offers power connections and the ability to use an external RESET button. The second row offers six analog inputs that are used to measure electrical signals that vary in voltage. Furthermore, pins A4 and A5 can also be used for sending data to and receiving it from other devices. Along the top of the board are two more rows of sockets, as shown in Figure 2-5.

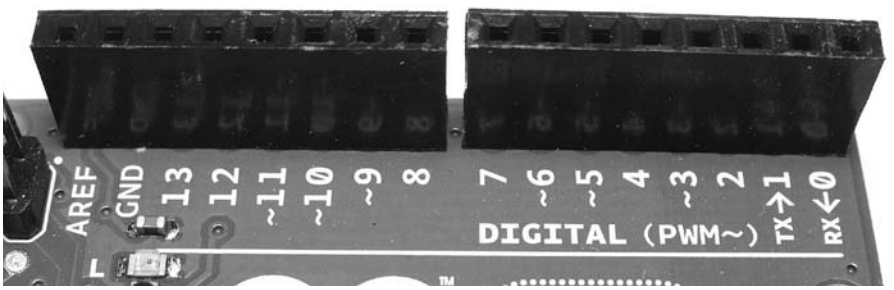


Figure 2-5: The digital input/output pins

Sockets (or pins) numbered 0 to 13 are digital input/output (I/O) pins. They can either detect whether or not an electrical signal is present or generate a signal on command. Pins 0 and 1 are also known as the *serial port*, which is used to send and receive data to other devices, such as a computer via the USB connector circuitry. The pins labeled with a tilde (~) can also generate a varying electrical signal, which can be useful for such things as creating lighting effects or controlling electric motors.

Next are some very useful devices called *light-emitting diodes (LEDs)*; these very tiny devices light up when a current passes through them. The Arduino board has four LEDs: one on the far right labeled ON, which indicates when the board has power, and three in another group, as shown in Figure 2-6.

The LEDs labeled *TX* and *RX* light up when data is being transmitted or received between the Arduino and attached devices via the serial port and USB. The *L* LED is for your own use (it is connected to the digital I/O pin number 13). The little black square part to the left of the LEDs is a tiny microcontroller that controls the USB interface that allows your Arduino to send data to and receive it from a computer, but you don't generally have to concern yourself with it.

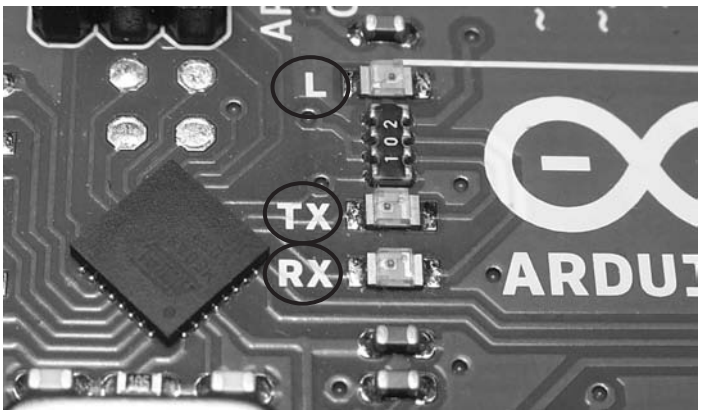


Figure 2-6: The onboard LEDs

And, finally, the RESET button is shown in Figure 2-7.

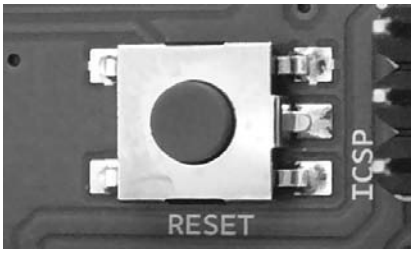


Figure 2-7: The RESET button

As with a normal computer, sometimes things can go wrong with the Arduino, and when all else fails, you might need to reset the system and restart your Arduino. This simple RESET button on the board (Figure 2-7) is used to restart the system to resolve these problems.

One of the great advantages of the Arduino system is its ease of expandability—that is, it's easy to add more hardware functions. The two rows of sockets along each side of the Arduino allow the connection of a *shield*, another circuit board with pins that allow it to plug into the Arduino. For example, the shield shown in Figure 2-8 contains an Ethernet interface that allows the Arduino to communicate over networks and the Internet, with plenty of space for custom circuitry.

Notice how the Ethernet shield also has rows of sockets. These enable you to insert one or more shields on top. For example, Figure 2-9 shows that another shield with a large numeric display, temperature sensor, extra data storage space, and a large LED has been inserted.

Note that you do need to remember which shield uses which individual inputs and outputs to ensure that "clashes" do not occur. You can also purchase completely blank shields that allow you to add your own circuitry. This will be explained further in Chapter 8.

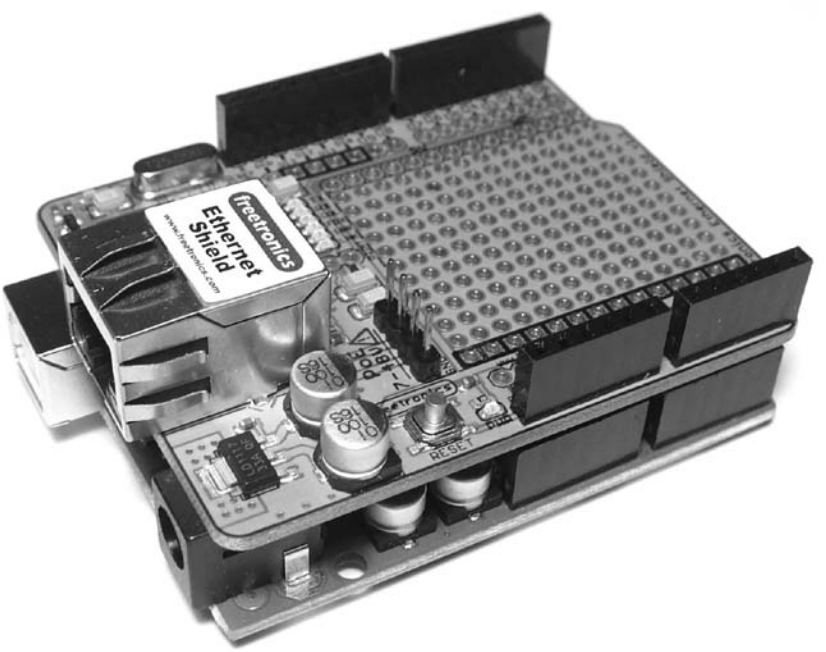


Figure 2-8: Arduino Ethernet interface shield

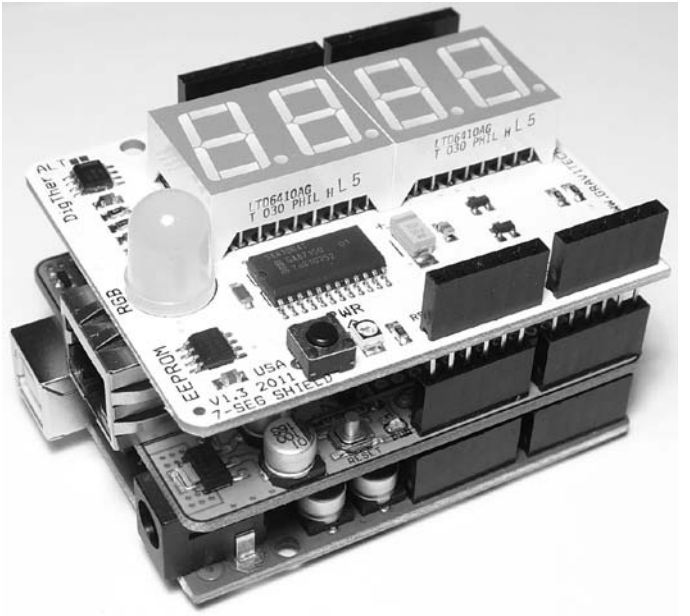


Figure 2-9: Numeric display and temperature shield

The companion to the Arduino hardware is the *software*, a collection of instructions that tell the hardware what to do and how to do it. Two types of software can be used: The first is the integrated development environment (IDE), which is discussed in this chapter, and the second is the Arduino sketch you create yourself.

The IDE software is installed on your personal computer and is used to compose and send sketches to the Arduino board.

Taking a Look Around the IDE

As shown in Figure 2-10, the Arduino IDE resembles a simple word processor. The IDE is divided into three main areas: the command area, the text area, and the message window area.

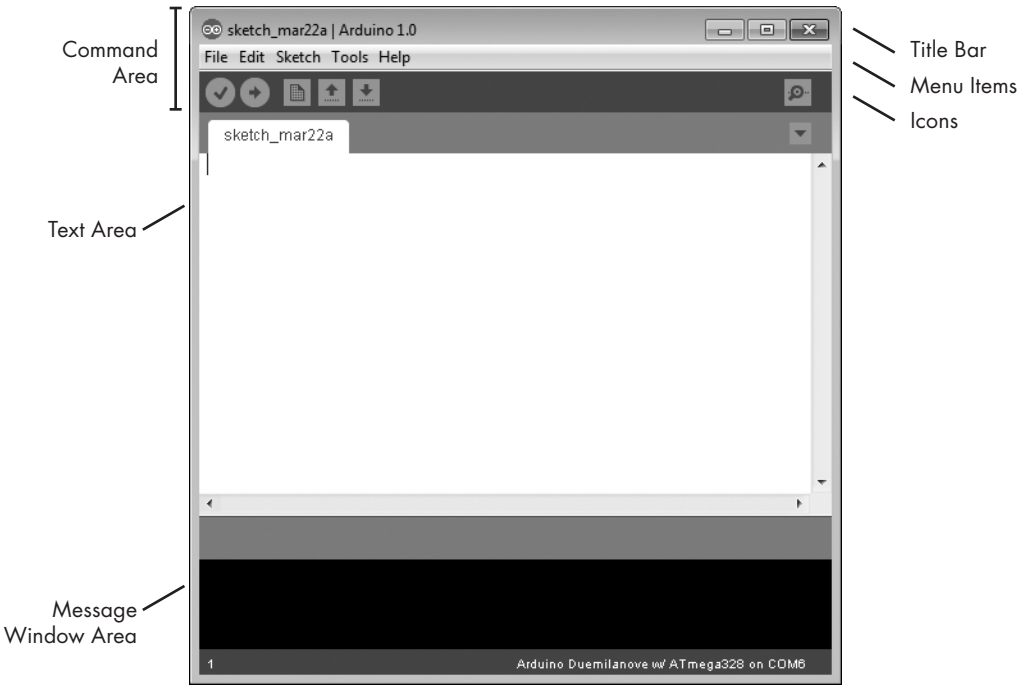


Figure 2-10: The Arduino IDE

The Command Area

The command area is shown at the top of Figure 2-10 and includes the title bar, menu items, and icons. The title bar displays the sketch's filename (*sketch_mar22a*), as well as the version of the IDE (*Arduino 1.0*). Below this is a series of menu items (File, Edit, Sketch, Tools, and Help) and icons, as described next.

Menu Items

As with any word processor or text editor, you can click one of the menu items to display its various options.

File Contains options to save, load, and print sketches; a thorough set of example sketches to open; as well as the **Preferences** submenu

Edit Contains the usual copy, paste, and search functions common to any word processor

Sketch Contains the function to verify your sketch before uploading to a board, and some sketch folder and import options

Tools Contains a variety of functions as well as the commands to select the Arduino board type and USB port

Help Contains links to various topics of interest and the version of the IDE

The Icons

Below the menu toolbar are six icons. Mouse over each icon to display its name. The icons, from left to right, are as follows:

Verify Click this to check that the Arduino sketch is valid and doesn't contain any programming mistakes.

Upload Click this to verify and then upload your sketch to the Arduino board.

New Click this to open a new blank sketch in a new window.

Open Click this to open a saved sketch.

Save Click this to save the open sketch. If the sketch doesn't have a name, you will be prompted to create one.

Serial Monitor Click this to open a new window for use in sending and receiving data between your Arduino and the IDE.

The Text Area

The text area is shown in the middle of Figure 2-10; this is where you'll create your sketches. The name of the current sketch is displayed in the tab at the upper left of the text area. (The default name is the current date.) You'll enter the contents of your sketch here as you would in any text editor.

The Message Window Area

The message window area is shown at the bottom of Figure 2-10. Messages from the IDE appear in the black area. The messages you see will vary and will include messages about verifying sketches, status updates, and so on.

At the bottom right of the message area, you should see the name of your Arduino board type as well as its connected USB port—*Arduino Duemilanove w/ATmega328 on COM6* in this case.

Creating Your First Sketch in the IDE

An Arduino sketch is a set of instructions that you create to accomplish a particular task; in other words, a sketch is a *program*. In this section you'll create and upload a simple sketch that will cause the Arduino's LED (shown in Figure 2-11) to blink repeatedly, by turning it on and then off for 1 second intervals.

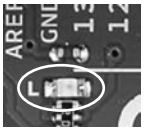


Figure 2-11: The LED on the Arduino board, next to the capital L

NOTE *Don't worry too much about the specific commands in the sketch we're creating here. The goal is to show you how easy it is to get the Arduino to do something so that you'll keep reading when you get to the harder stuff.*

To begin, connect your Arduino to the computer with the USB cable. Then open the IDE, choose **Tools ▸ Serial Port**, and make sure the USB port is selected. This ensures that the Arduino board is properly connected.

Comments

First, enter a comment as a reminder of what your sketch will be used for. A *comment* is a note of any length in a sketch, written for the user's benefit. Comments in sketches are useful for adding notes to yourself or others, for entering instructions, or for noting miscellaneous details. When programming your Arduino (creating sketches), it's a good idea to add comments regarding your intentions; these comments can prove useful later when you're revisiting a sketch.

To add a comment on a single line, enter two forward slashes and then the comment, like this:

```
// Blink LED sketch by Mary Smith, created 09/09/12
```

The two forward slashes tell the IDE to ignore the text that follows when verifying a sketch. (As mentioned earlier, when you verify a sketch, you're asking the IDE to check that everything is written properly with no errors.)

To enter a comment that spans two or more lines, enter the characters **/*** on a line before the comment, and then end the comment with the characters ***/** on the following line, like this:

```
/*
Arduino Blink LED Sketch
by Mary Smith, created 09/09/12
*/
```

As with the two forward slashes that precede a single line comment, the /* and */ tell the IDE to ignore the text that they bracket.

Enter a comment describing your Arduino sketch using one of these methods, and then save your sketch by choosing **File ▸ Save As**. Enter a short name for your sketch (such as *blinky*), and then click **OK**.

The default filename extension for Arduino sketches is *.ino*, and the IDE should add this automatically. The name for your sketch should be, in this case, *blinky.ino*, and you should be able to see it in your Sketchbook.

The Setup Function

The next stage in creating any sketch is to add the `void setup()` function. This function contains a set of instructions for the Arduino to execute once only, each time it is reset or turned on. To create the `setup` function, add the following lines to your sketch, after the comments:

```
void setup()
{

}
```

Controlling the Hardware

Our program will blink the user LED on the Arduino. The user LED is connected to the Arduino's digital pin 13. A digital pin can either detect an electrical signal or generate one on command. In this project, we'll generate an electrical signal that will light the LED. This may seem a little complicated, but you'll learn more about digital pins in future chapters. For now, just continue with creating the sketch.

Enter the following into your sketch between the braces ({ and }):

```
  pinMode(13, OUTPUT); // set digital pin 13 to output
```

The number `13` in the listing represents the digital pin you're addressing. You're setting this pin to `OUTPUT`, which means it will generate (output) an electrical signal. If you wanted it to detect an incoming electrical signal, then you would use `INPUT` instead. Notice that the function `pinMode()` ends with a semicolon (`;`). Every function in your Arduino sketches will end with a semicolon.

Save your sketch again to make sure that you don't lose any of your work.

The Loop Function

Remember that our goal is to make the LED blink repeatedly. To do this, we'll create a `loop` function to tell the Arduino to execute an instruction over and over until the power is shut off or someone presses the RESET button.

Enter the code shown in boldface after the `void setup()` section in the following listing to create an empty `loop` function. Be sure to end this new section with another brace (}), and then save your sketch again.

```
/*
Arduino Blink LED Sketch
by Mary Smith, created 09/09/12
*/

void setup()
{
  pinMode(13, OUTPUT); // set digital pin 13 to output
}
void loop()
{
  // place your main loop code here:
}
```

WARNING *The Arduino IDE does not automatically save sketches, so save your work frequently!*

Next, enter the actual functions into `void loop()` for the Arduino to execute.

Enter the following between the `loop` function's braces, and then click **Verify** to make sure that you've entered everything correctly:

```
  digitalWrite(13, HIGH); // turn on digital pin 13
  delay(1000); // pause for one second
  digitalWrite(13, LOW); // turn off digital pin 13
  delay(1000); // pause for one second
```

Let's take this all apart. The `digitalWrite()` function controls the voltage that is output from a digital pin: in this case, pin 13 to the LED. By setting the second parameter of this function to `HIGH`, a "high" digital voltage is output; then current will flow from the pin and the LED will turn on. (If you were to set this parameter to `LOW`, then the current flowing through the LED would stop.)

With the LED turned on, the light pauses for 1 second with `delay(1000)`. The `delay()` function causes the sketch to do nothing for a period of time—in this case, 1,000 milliseconds, or 1 second.

Next, we turn off the voltage to the LED with `digitalWrite(13, LOW);`. Finally, we pause again for 1 second while the LED is off, with `delay(1000);`.

The completed sketch should look like this:

```
/*
 Arduino Blink LED Sketch
 by Mary Smith, created 09/09/12
*/

void setup()
{
  pinMode(13, OUTPUT); // set digital pin 13 to output
}
```

```
void loop()
{
  digitalWrite(13, HIGH); // turn on digital pin 13
  delay(1000); // pause for one second
  digitalWrite(13, LOW); // turn off digital pin 13
  delay(1000); // pause for one second
}
```

Before you do anything further, save your sketch!

Verifying Your Sketch

When you verify your sketch, you ensure that it has been written correctly in a way that the Arduino can understand. To verify your complete sketch, click **Verify** in the IDE and wait a moment. Once the sketch has been verified, a note should appear in the message window, as shown in Figure 2-12.

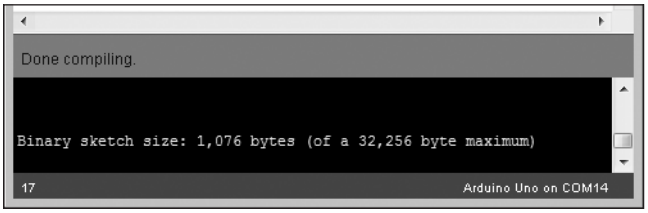


Figure 2-12: The sketch has been verified.

This "Done compiling" message tells you that the sketch is okay to upload to your Arduino. It also shows how much memory it will use (1,076 bytes in this case) of the total available on the Arduino (32,256 bytes).

But what if your sketch isn't okay? Say, for example, you forgot to add a semicolon at the end of the second `delay(1000)` function. If something is broken in your sketch, then when you click **Verify**, the message window should display a verification error message similar to the one shown in Figure 2-13.

Figure 2-13: The message window with a verification error

The message tells you that the error occurs in the `void loop` function, lists the line number of the sketch where the IDE thinks the error is located (`blinky:16`, or line 16 of your *blinky* sketch), and displays the error itself (the missing semicolon, `error: expected ';' before '}' token`). Furthermore, the IDE should also highlight in yellow the location of the error or a spot just after it. This helps you easily locate and rectify the mistake.

Uploading and Running Your Sketch

Once you're satisfied that your sketch has been entered correctly, save it, ensure that your Arduino board is connected, and click **Upload** in the IDE. The IDE may verify your sketch again and then upload it to your Arduino board. During this process, the TX/RX LEDs on your board (shown in Figure 2-6) should blink, indicating that information is traveling between the Arduino and your computer.

Now for the moment of truth: Your Arduino should start running the sketch. If you've done everything correctly, then the LED should blink on and off once every second!

Congratulations. You now know the basics of how to enter, verify, and upload an Arduino sketch.

Modifying Your Sketch

After running your sketch, you may want to change how it operates, by, for example, adjusting the on or off delay time for the LED. Because the IDE is a lot like a word processor, you can open your saved sketch, adjust the values, and then save your sketch again and upload it to the Arduino. For example, to increase the rate of blinking, change both `delay` functions to make the LEDs blink for one-quarter of a second by adjusting the delay to `250` like this:

```
delay(250); // pause for one-quarter of one second
```

Then upload the sketch again. The LED should now blink faster, for one-quarter of a second each time.

Looking Ahead

Armed with your newfound knowledge of how to enter, edit, save, and upload Arduino sketches, you're ready for the next chapter, where you'll learn how to use more functions, implement good project design, construct basic electronic circuits, and do much more.