Introduction

I first discovered the Arduino in 2008 when I was looking for ways to connect temperature sensors to my PC so I could make a cloud detector. I wanted to try out a cloud detection concept I'd read about on a weather forum, and as it was experimental, I didn't want to spend a lot of money on it in case it failed. There were many solutions on the market, but the Arduino appealed to me the most. Not only did it seem to be an easy and cheap way to connect the sensors I required, but it could be used for other cool things. Thousands of projects in blogs, video sites, and forums showed the amazing things people were doing with their Arduinos. There seemed to be a huge sense of community with everyone trying to help one another.

It was obvious that I could have a lot of fun with an Arduino. However, I didn't want to be trawling through websites for information. I wanted to buy a book on the subject, something I could hold in my hand and read on the train to work. After looking around, I found one book. Unfortunately, it was very basic and out of date. Worse, it didn't give me anything practical to do with the Arduino, and I didn't warm to the teaching style, either. What I wanted was a hands-on book that taught me both programming and electronics as I built things instead of having to wade through pages of theory first. Such a book just didn't exist at the time.

Then I started Earthshine Electronics to sell kits based on the Arduino. To go with the kit, I produced a small tutorial booklet to get people started. This little booklet ended up being extremely popular, and I got hundreds of queries from people asking when I would be adding more projects or if I sold a printed version. In fact, I had already thought that it would be great to produce a comprehensive beginner's book, crammed with projects and written in an easy-to-follow style. That is how this book came about. This book has proven so successful at teaching people about the Arduino that it has since been updated to this second edition with improvements and updated sections relevant to the changes in the Arduino world since I began.

I have written this book with the presumption that you have never done either computer programming or electronics before. I also presume you're not interested in reading lots of theory before you actually get down to making something with your Arduino. Hence, right from the start of the book, you will be diving right into making a simple project. From there, you will work through a total of 50 projects until you become confident and proficient at Arduino development. I believe that the best way to learn anything is by learning as you go and getting your hands dirty.

The book works like this: the first project introduces basic concepts about programming the Arduino and also about electronics. The next project builds on that knowledge to introduce a little bit more. Each project after that builds on the previous projects. By the time you have finished all 50 projects, you will be confident and proficient at making your own projects. You'll be able to adapt your new skills and knowledge to connect just about anything to your Arduino and make great projects for fun or to make your life easier.

Each project starts off with a list of required parts. I have chosen common parts that are easy to source. I also provide a circuit diagram showing exactly how to connect the Arduino and parts together using jumper wires and a breadboard. To create the parts images and breadboard diagrams for the book, I used the excellent open-source program Fritzing. The program allows designers to document their prototypes and then go on to create PCB layouts for manufacture. It is an excellent program and a brilliant way of demonstrating a breadboard circuit to others. Pop on over to `http://fritzing.org` and check it out.

After you have made your circuit, I supply a code listing to type into the Arduino's program editor (the IDE) which can then be uploaded to your Arduino to make the project work. You will very quickly have a fully working project. It is only after you have made your project and seen it working that I explain how it works. The hardware will be explained to you in such a way that you know how the components work and how to connect them to the Arduino correctly.

The code will then be explained to you step by step so you understand exactly what each section of the code does. By dissecting the circuit and the code, you will understand how the whole project works and can then apply the skills and knowledge to later projects and then to your own projects in the future.

The style of teaching in this book is very easy to follow. Even if you have absolutely no experience of either programming or electronics, you will be able to follow along easily and understand the concepts as you go. More importantly, you will have fun. The Arduino is a great and fun open source product. With the help of this book, you'll discover just how easy it is to get involved in physical computing to make your own devices that interact with their environment.

—Mike McRoberts

Downloading the Code

The code for the examples shown in this book is available on the Apress web site, www.apress.com. A link can be found on the book's information page under the Source Code/Downloads tab. This tab is located underneath the Related Titles section of the page.

Contacting the Author

Should you have any questions or comments—or even spot a mistake you think I should know about—you can contact me at mike@earthshineelectronics.com, on Twitter as @TheArduinoGuy or on Google+ as Mike McRoberts. I am available, upon request (when I am free) to run Arduino Workshops or demonstrations at Hackerspaces and other organisations.

CHAPTER 1

Getting Started

Since the Arduino Project started in 2005, over 500,000 boards have been sold worldwide to date. The number of unofficial clone boards sold no doubt outweighs the number of official boards, and it's likely that over a million Arduino boards or its variants are out in the wild. Its popularity is ever increasing as more and more people realize the amazing potential of this incredible open source project and its ability to create cool projects quickly and easily with a relatively shallow learning curve.

The biggest advantage of the Arduino over other microcontroller development platforms is the ease of use in which non-"techie" people can pick up the basics and create their own projects in a relatively short amount of time. Artists in particular seem to find it the ideal way to create interactive works of art quickly and without specialist knowledge of electronics. There is a huge community of people using Arduinos and sharing their code and circuit diagrams for others to copy and modify. Most of this community is also very willing to help others and to provide guidance and the Arduino Forum is the place to go if you want answers quickly.

However, despite the huge amount of information available on the Internet for beginners, most of this information is spread across various sources, making it tricky for beginners to obtain the information they want. This is where this book fits in. Within the pages you are about to read are 50 projects that are all designed to take you step by step through the world of electronics and programming your Arduino in an easy to follow manner. I believe that the best way to learn anything is to jump in and just do it. That is why this book will not bore you with pages and pages of theory before you start to use your Arduino. I know what it is like when you first get an Arduino, or any new gadget: you want to plug it in, connect an LED, and get it flashing right away, not read through pages of manuals first. This author understands that excitement to get going and that is why we will dive right into connecting things to our Arduino, uploading code, and getting started right away. This is, I believe, the best way to learn a subject and especially a subject like physical computing, which is what the Arduino is all about.

How to Use This Book

The book starts with an introduction to the Arduino, how to set up the hardware, install the software, upload your first sketch, and ensure that your Arduino and the software are working correctly. We then explain the Arduino IDE (integrated development environment) and how to use it before we dive right into some projects, progressing from very basic stuff through to advanced topics. Each project will start with a description of how to set up the hardware and what code is needed to get it working. We will then separately describe the code and the hardware and explain in some detail how it works. Everything will be explained in clear and easy-to-follow steps. The book contains a lot of diagrams and photographs to make it as easy as possible to check that you are following along with the project correctly.

You will come across some terms and concepts in the book that you may not understand at first. Don't worry; these will become clear as you work your way through the projects.

What You Will Need

In order to follow along with the projects in this book, you will need various components. To carry out all of the projects will require purchasing a lot of parts first. This could be expensive, so I suggest that you start by purchasing the components for the projects in the first few chapters and obtain the parts listed at the start of the project pages. As you progress through the book, you can obtain the parts needed for subsequent projects.

There are a handful of other items you will need or may find useful. Of course, you will need to obtain an Arduino board or one of the many clone boards on the market such as the Freeduino, Seeeduino (yes, there really are three *eee*s), Boarduino, Sanguino, Roboduino or any of the other "duino" variants. These are all fully compatible with the Arduino IDE, Arduino Shields and everything else that you can use with an official Arduino Board. Remember that the Arduino is an Open Source project; therefore anyone is free to make a clone or other variant of the Arduino. However, if you wish to support the development team of the original Arduino board, get an official board from one of the recognized distributors. For the projects in this book, we will be using an Arduino Uno, although any of the available Arduino boards will work just as well.

You will need access to the Internet to download the Arduino IDE, the software used to write your Arduino code and upload it to the board, and also to download the Code Samples within this book (if you don't want to type them out yourself), as well as any code libraries that may be necessary to get your project working.

You will also need a well-lit table or other flat surface to lay out your components; this will need to be next to your desktop or laptop PC to enable you to upload the code to the Arduino. Remember that you are working with electricity (although it is low voltage DC), and therefore metal tables or surfaces will need to be covered in a non-conductive material such as a tablecloth or paper before laying out your materials. Also of some benefit, although not essential, may be a pair of wire cutters, a pair of long-nosed pliers, and a wire stripper. A notepad and pen will also come in handy for drawing out rough schematics and working out concepts and designs.

Finally, the most important thing you will need is enthusiasm and a willingness to learn. The Arduino is designed as a simple and cheap way to get involved in microcontroller electronics and nothing is too hard to learn if you are willing to give it a go. This book will help you on that journey, and introduce you to this exciting and creative hobby.

What Exactly Is an Arduino?

Wikipedia states "**Arduino** is a single-board microcontroller designed to make the process of using electronics in multidisciplinary projects more accessible. The hardware consists of a simple open-source hardware board designed around an 8-bit Atmel AVRmicrocontroller, though a new model has been designed around a 32-bit Atmel ARM. The software consists of a standard programming language compiler and a boot loader that executes on the microcontroller."

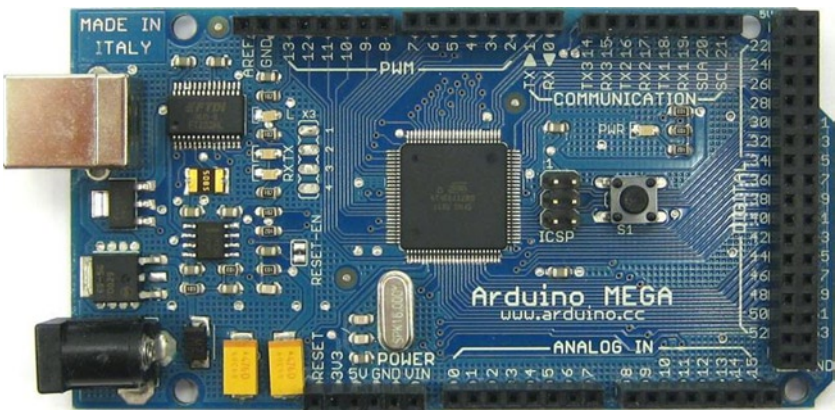

Figure 1-1. *An Arduino Mega (image by David Mellis)*

To put the above definition in layman's terms, an Arduino is a tiny computer that you can program to process inputs and outputs between the device and external components you connect to it. The Arduino is what is known as a physical or embedded computing platform. For example, a simple use of an Arduino would be to turn a light on for a set period of time, let's say for 30 seconds, after a button has been pressed. In this example, the Arduino would have a lamp connected to it as well as a button. The Arduino would sit patiently waiting for the button to be pressed. When you press the button, the Arduino would turn the lamp on and start counting. Once it had counted for 30 seconds, it would turn the lamp off, and then continue to wait for another button press. You could use this setup to control a lamp in a cupboard, for example.

You could extend this concept so that the device detects when the cupboard door has been opened or some other event has occurred, and automatically turns the lamp on, turning it off after a set period. You could go even further and connect a passive infrared (PIR) sensor to detect movement and to turn the lamp on when it has been triggered. These are some simple examples of how you could use an Arduino.

The Arduino can be used to develop stand-alone interactive objects or it can be connected to a computer, a network, or even the Internet to retrieve and send data to and from the Arduino, and then act on that data. For example, it could be used to send a set of data received from sensors to a website to be displayed in the form of a graph.

The Arduino can be connected to LEDs, dot-matrix displays (see Figure 1-2), buttons, switches, motors, temperature sensors, pressure sensors, distance sensors, GPS receivers, Ethernet or WiFi modules, or just about anything that outputs data or can be controlled. A look around the Internet will bring up a wealth of projects in which an Arduino has been used to read data from or control an amazing array of devices.

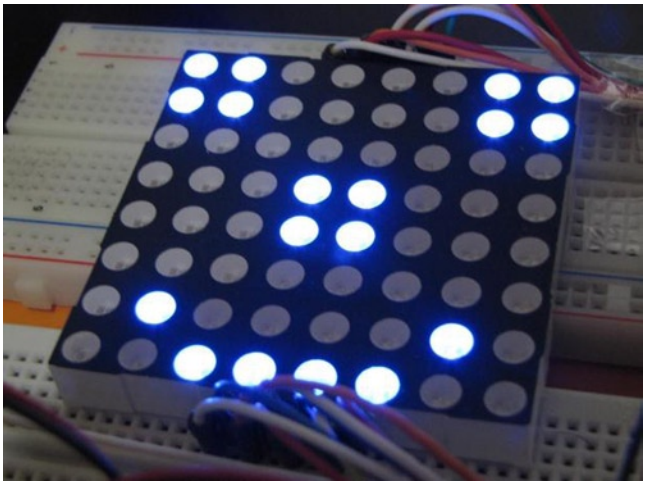

Figure 1-2. *A dot-matrix display controlled by an Arduino (image courtesy of Bruno Soares)*

The Arduino board is made up of an Atmel AVR microprocessor, a crystal or oscillator (a crude clock that sends time pulses at a specified frequency to enable it to operate at the correct speed) and a 5V voltage regulator. (Some Arduinos may use a switching regulator, and some, like the Due, are not 5 volt). Depending on what type of Arduino you have, it may also have a USB socket to enable it to be connected to a PC or Mac to upload or retrieve data. The board exposes the microcontroller's I/O (input/output) pins to enable you to connect those pins to other circuits or to sensors, etc.

To program the Arduino (make it do what you want it to), you also use the Arduino IDE, which is a piece of free software that enables you to program in the language that the Arduino understands. In the case of the Arduino, the language is based on C/C++ and can even be extended through C++ libraries. The IDE enables you to write a computer program, which is a set of step-by-step instructions that you then upload to the Arduino. Your Arduino will then carry out those instructions and interact with whatever you have connected to it. In the Arduino world, programs are known as "sketches".

The Arduino hardware and software are both Open Source, which means that the code, schematics, design, etc., are all open for anyone to take freely and do with what they like. Hence, there are many clone boards and other Arduino-based boards available to purchase or to make from a schematic. Indeed, there is nothing stopping you from purchasing the appropriate components and making your own Arduino on a breadboard or on your own homemade PCB (printed circuit board). The only caveat that the Arduino team makes to this is that you cannot use the word *Arduino*, as this is reserved for the official board. Hence, the clone boards all have names such as Freeduino, Roboduino, etc.

The Arduino can also be extended with the use of "shields," which are circuit boards containing other devices (for example, GPS receivers, LCD displays, Ethernet modules, etc.) that you can simply connect to the top of your Arduino to get extra functionality. Shields also extend the pins (the places on your Arduino where you can output or input data) to the top of their own circuit board, so you still have access to all of them. You don't have to use a shield if you don't want to, as you can make the exact same circuitry using a breadboard, some Stripboard or Veroboard (boards made up of strips of copper in a grid for home-soldered projects), or by making your own PCB. Most of the projects in this book are made using circuits on a breadboard.

As the designs are open source, a clone board, such as the Freeduino, can be 100 percent compatible with the Arduino and therefore any software, hardware, shields, etc. Some clones are compatible in most respects but may have intentional differences to support special features. Also, the Due (which is genuine Arduino) does have some issues such as its 3 volt operation, which may not work with all shields.

There are many different variants of the Arduino available. The most common one is Uno, released in 2010 (currently on Revision 3) and this is the board you will most likely see being used in the vast majority of Arduino projects across the Internet. You can also get the Due Leonardo, Duemilanove, Mega 2560, Mega ADK, Fio, Arduino Ethernet, Mini, Nano, Lilypad, and Bluetooth Arduinos. The latest additions to the product line are the Arduino Leonardo and the Arduino Due, which is the Arduino team's first incursion into using ARM processors instead of AVR architecture processors The Due has a 32-bit processor instead of the usual 8-bit processor in the other Arduino variants, runs at 84MHz, and has 512KB of flash memory.

Probably the most versatile Arduino, and hence the reason it is so popular, is the Uno (prior to the Uno, the Duemilanove was the most popular). This is because they use a standard 28 pin chip attached to an IC (integrated circuit) socket. The beauty of this system is that if you make something neat with an Arduino and then want to turn it into something permanent, instead of using a relatively expensive Arduino board, you can simply use the Arduino to develop your device and program the chip, then pop the chip out of the board and place it into your own circuit board in your custom device. You would then have made a custom-embedded device, which is really cool. Then for a couple of quid or bucks, you can replace the AVR chip in your Arduino with a new one. The chip must be pre-programmed with the Arduino Bootloader (software programmed onto the chip to enable it to be used with the Arduino IDE), but you can either purchase an AVR Programmer to burn the bootloader yourself or you can buy a chip ready programmed, and most of the Arduino parts suppliers will provide these.

The newer Arduino Uno has the advantage over the previous Arduino, the Duemilanove, in that it has a programmable USB chip on board which enables you to flash the chip in such a way that when you plug the device into your PC it will show up as any USB device you like, such as a keyboard, mouse, or joystick. This enables you to use the Arduino as an interface for creating your own USB devices. This is, however, an advanced feature and not for the faint hearted.

If you do a search on the Internet for *Arduino*, you will be amazed at the huge amount of websites dedicated to the Arduino or in which someone has used an Arduino to create a cool project. The Arduino is an amazing device and will enable you to create anything, from interactive works of art (see Figure 1-3) to robots. With a little enthusiasm about learning how to program an Arduino and make it interact with other components as well as a bit of imagination, you can build anything you can think of.

Figure 1-3. *Anthros art installation by Richard V. Gilbank controlled using an Arduino*

This book will give you the necessary skills needed to make a start in this exciting and creative hobby. So now that you know what an Arduino is, let's get one hooked up to our computer and start using it.

Setting Up Your Arduino

This section will explain how to set up your Arduino and the IDE for the first time. The instructions for both Windows and Macs are given. If you use Linux, then refer to the Getting Started instructions on the Arduino website at http://playground.arduino.cc/learning/linux. I will also presume you are using an Arduino Uno (see Figure 1-4), Duemilanove, Nano, Diecimila or Mega 2560 (or their equivalent clone) and are installing on either Windows 7 or a recent version of OSX (Lion or Mountain Lion). If you have a different type of board, then refer to the corresponding page in the Getting Started guide of the Arduino website.

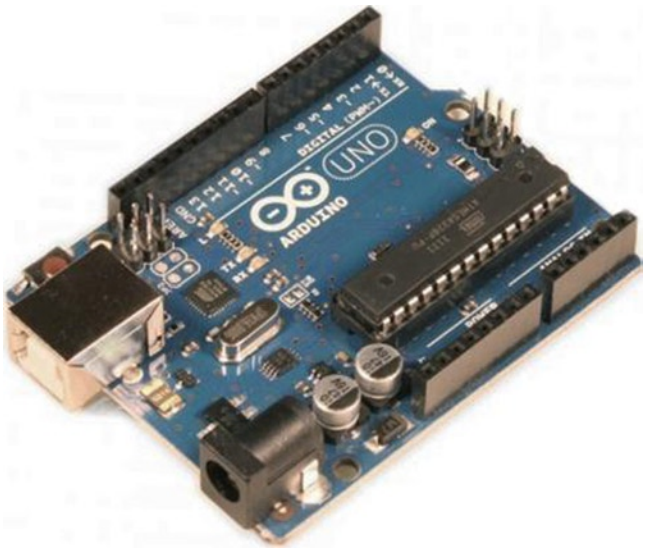

Figure 1-4. *An Arduino Uno (Image courtesy of Earthshine Electronics)*

You will also need a USB cable (A to B plug type) which is the same kind of cable used for most modern USB printers. If you have an Arduino Nano, you will need a USB A to Mini-B cable instead.

Next, you need to download the Arduino IDE. This is the software you will use to write your programs (or sketches) and upload them to your board. For the latest IDE go to the Arduino download page at `http://arduino.cc/en/Main/Software` and obtain appropriate the version for your operating system.

If you have a Mac, once the Zip file has downloaded, unzip it and then you will see the Arduino icon. Drag it across to the Applications folder and drop it in there to install the program. You simply double-click the icon to start it. For Windows, download the ZIP file and, once complete, unzip it. Then put the unzipped folder in a place that suits you, keeping the directory structure in place.

Now you need to connect your Arduino board before installing the drivers and software. Connect the USB cable to the Arduino and plug the other end into a USB socket on your computer. You will see the green Power LED (marked PWR) light up on your board to show you it has power. If you are on a Mac, then there are no drivers to install. If you are on Windows, then it will now attempt to install the drivers for the Arduino. This auto attempt will fail and you will get a message that the "Device driver software was not successfully installed" (Figure 1-5); do not worry about this.

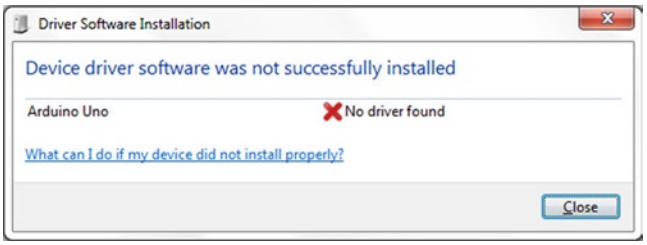

Figure 1-5. *The automatic attempt by Windows to install the drivers will fail. This is normal*

Click on the Start Menu and then click on Control Panel. Navigate to System and Security, click on System, and then open the Device Manager. On the list of hardware underneath Other Devices, you should see something similar to Figure 1-6 in which you have "Arduino Uno" with a yellow hazard icon over it.

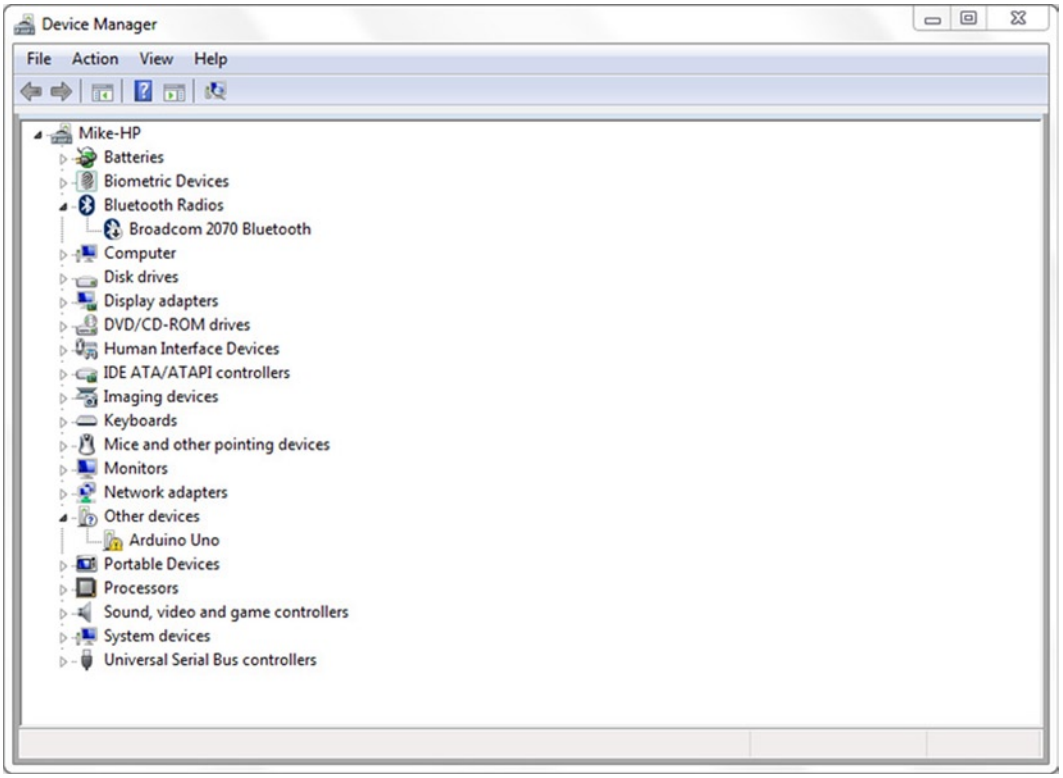

Figure 1-6. *The Windows Device Manager*

Right click on the Arduino Uno icon in the list and choose “Update Driver Software” (Figure 1-7).

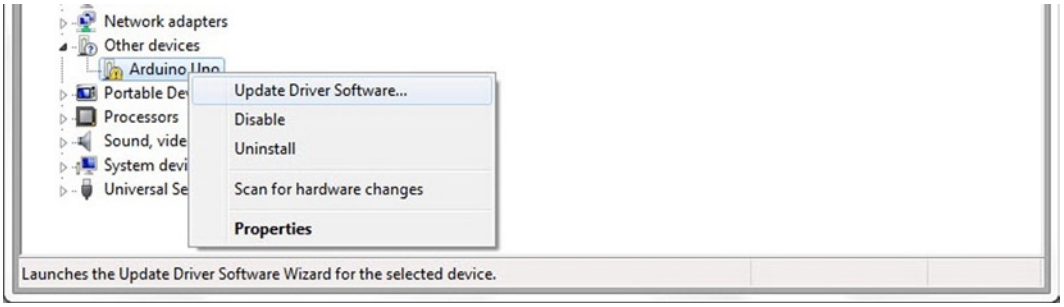

Figure 1-7. *Right click and choose "Update Driver Software"*

Now choose “Browse my computer for driver software”.

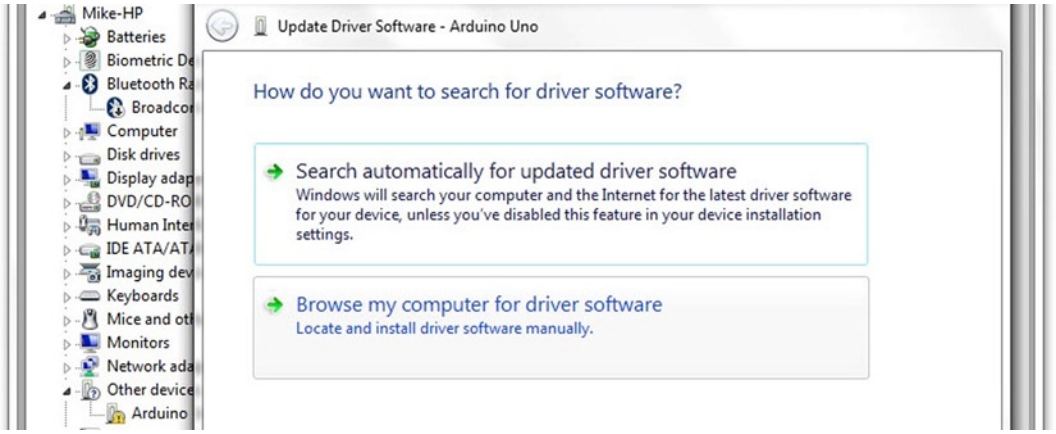

Figure 1-8. *Click on “Browse my computer for driver software”*

Next, browse to the driver folder of the Arduino installation, and then click the Next button. Windows will now finish the driver installation. If you get a message that says “Windows can’t verify the publisher of this driver software” then click the “Install this driver software anyway.” If you have a Mac, then there are no drivers to install.

Now that the drivers are installed, you are ready to open up the Arduino IDE. For Windows, double-click the **arduino.exe** file inside the unzipped Arduino folder. For a Mac, click the Arduino icon in the Applications folder. The IDE will now open up and present you with a blank sketch as in Figure 1-9.

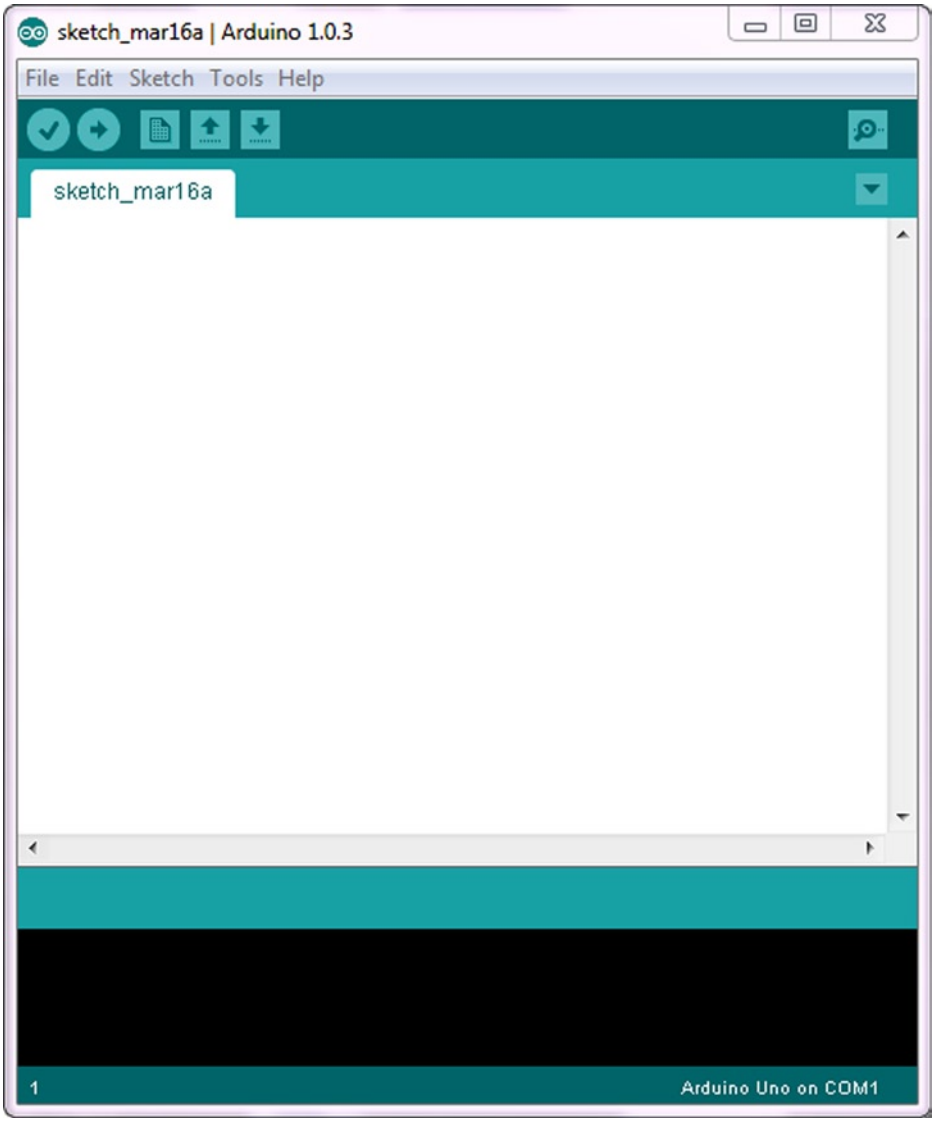

Figure 1-9. *The Arduino IDE*

Next, open up an example sketch to test out the IDE and the Arduino. Click File, then Examples, then 01.Basics, and finally, Blink (see Figure 1-10).

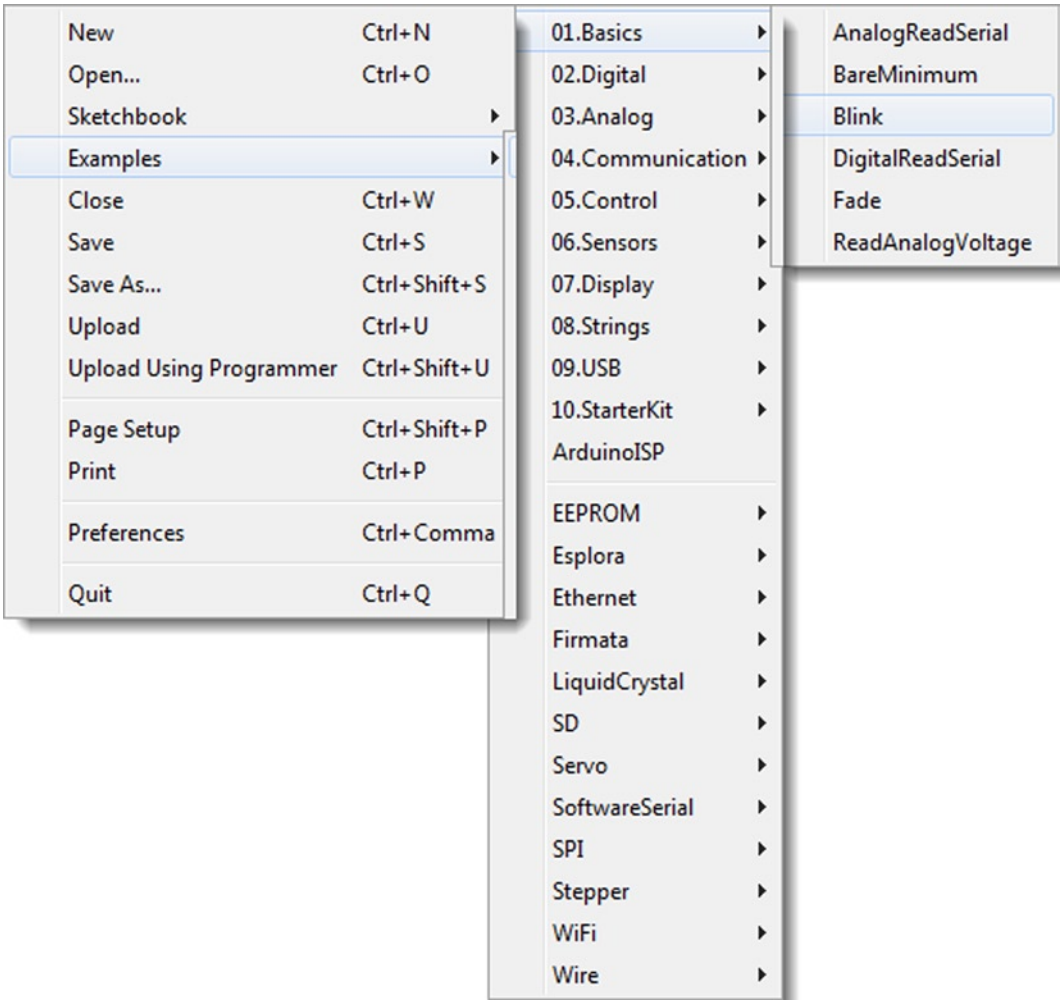

Figure 1-10. *The Arduino File Menu. Choose the Blink sketch*

This will load the Blink example sketch into the IDE and will look something like Figure 1-11.

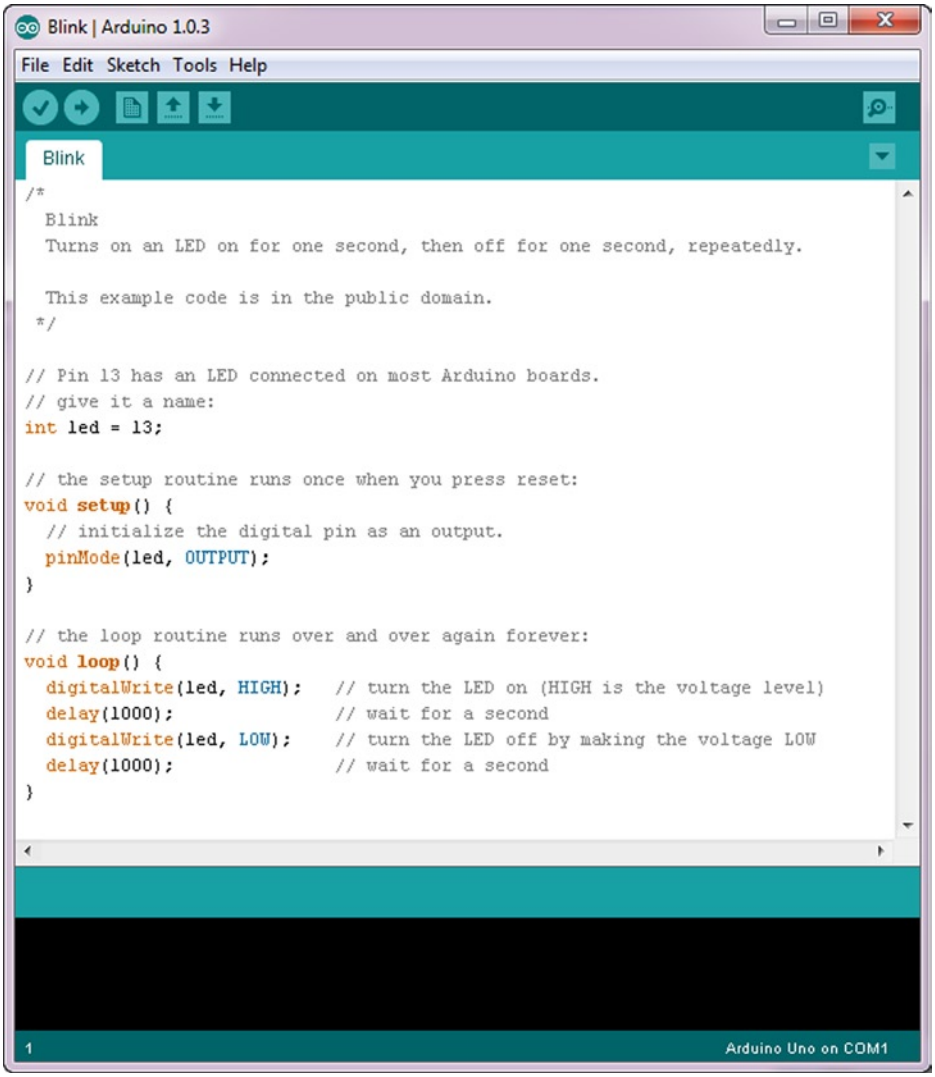

Figure 1-11. *The IDE with the Blink sketch loaded*

Next, you will need to select your board from the list (see Figure 1-12) in **Tools ➤ Board**. For an **Arduino Uno**, select this from the top of the list. If you have an older Arduino Duemilanove or clone with an Atmega328 chip, you will need to select **Arduino Duemilanove or Nano w/ Atmega328.** If you have an even older board with an Atmega168 chip, select **Arduino Diecimila, Duemilanove, or Nano w/ ATmega168**, or you may even have a **Leonardo**, **Mega** or a **DUE**. Choose whichever board matches yours.

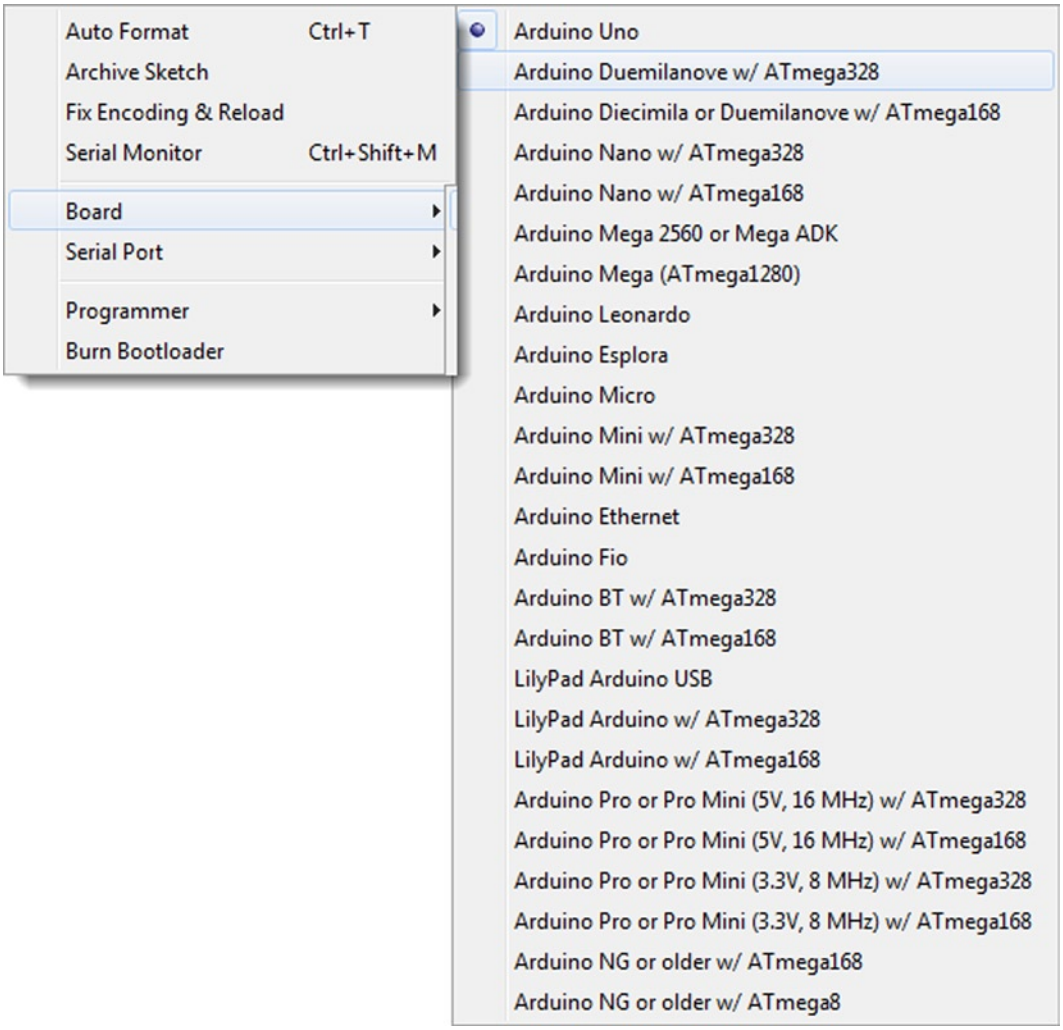

Figure 1-12. *Select your board type*

Select the serial device of the Arduino board from **Tools ➤ Serial Port** (see Figure 1-13). If you are not sure what your port is, disconnect the Arduino and check the ports available, then reconnect the Arduino and see which port has now appeared (you may need to close and reopen the menu to get it to show).

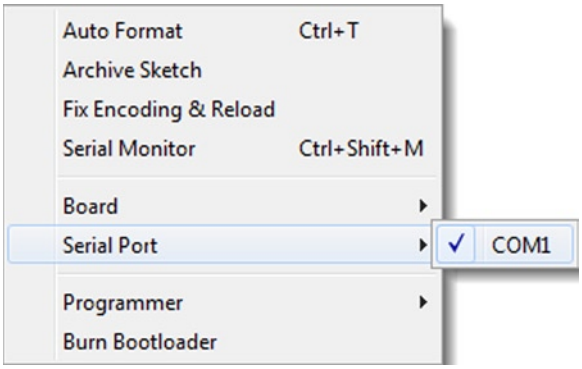

Figure 1-13. *Select the port*

Upload Your First Sketch

Now that you have installed the drivers and the IDE and have the correct board and ports selected, you can upload an example Blink sketch to the Arduino to test everything is working properly before moving on to the first project. Once you have loaded the Blink sketch into the Arduino IDE, you can upload it to the Arduino by simply clicking the Upload button (the second button from the left that is a right-facing arrow) and look at your Arduino (if you have an Arduino Mini, NG, or other board, you may need to press the reset button on the board prior to pressing the Upload button). The IDE will say "Compiling sketch . . .", which will then change to "Uploading . . . ." Next, the RX and TX lights should start to flash to show that data is being transmitted from your computer to the board. Once the sketch has successfully uploaded, the words "Done uploading" will appear in the IDE status bar and the RX and TX lights will stop flashing.

After a few seconds, you should see the Pin 13 LED (the tiny LED above the TX and RX LEDs) start to flash on and off at one second intervals. If it does, then you have just successfully connected your Arduino, installed the drivers and software, and uploaded an example sketch. The Blink sketch is a very simple sketch that blinks the LED 13, which is a tiny orange LED soldered to the board and also connected to digital pin 13 from the microcontroller (see Figure 1-14).

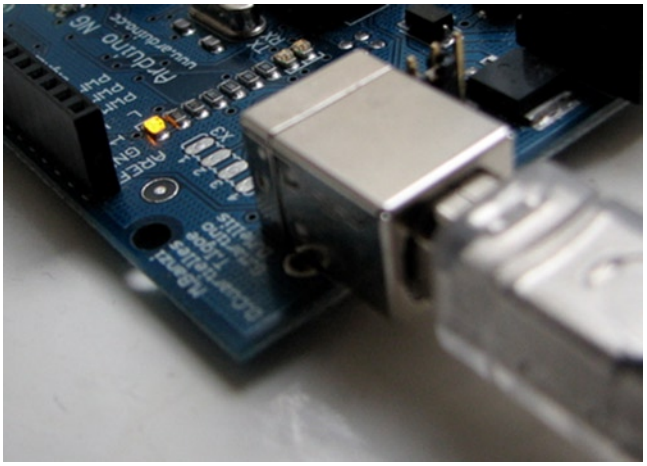

Figure 1-14. *LED 13 blinking*

Before we move onto Project 1, let's take a look at the Arduino IDE and I'll explain what each of the parts of the program do.

The Arduino IDE

The Arduino IDE (Integrated Development Environment) is what you will use to write the code for your Arduino, verify it, and upload it to your board. The current IDE version 1.x was released in November 2011. Previously, the Beta version numbers ran from 0001 to 0023 and version 1.0 was the first release candidate of the software. In version 1.0, the file extensions for the sketches changed from .pde to .ino to avoid conflicts with the Processing software (Processing is a project that the original IDE was based on). There were also some major changes to the Arduino language. If you want to port older Arduino code to the new IDE, you should read up on the Arduino website in the reference section about how the commands work if you get any errors with the older code.

When you open up the Arduino IDE, it will look very similar to the Windows version in the image below (Figure 1-15). If you are using OSX or Linux, there may be some slight differences, but the IDE is pretty much the same no matter what OS you use.

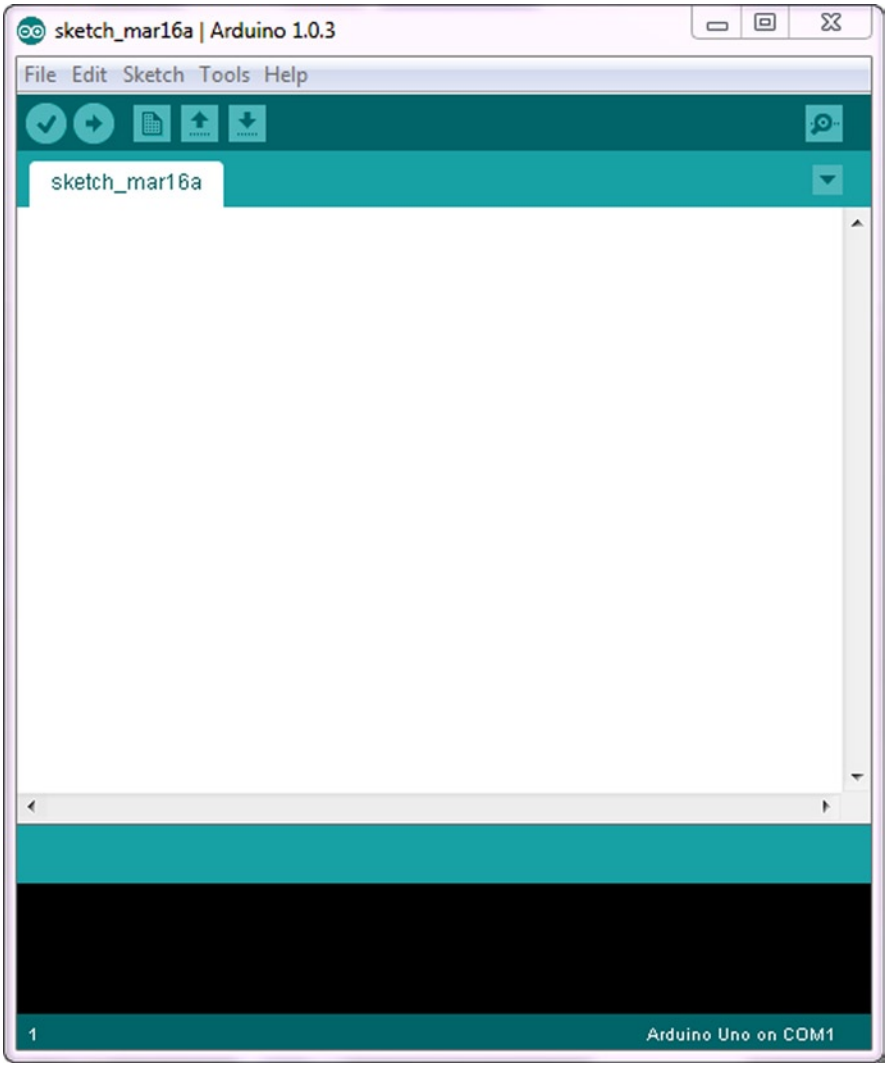

Figure 1-15. *What the IDE looks like when the application opens*

The IDE is split into four parts: the File Menu across the top of the program (or at the top of your screen in OSX), the Toolbar below this, the code or Sketch Window in the center, and the message window in the bottom. The Toolbar consists of six buttons, and underneath the Toolbar is a tab, or set of tabs, with the filename of the sketch within the tab. There is also one further button on the far right hand side which brings up the Serial Monitor window.

Along the top is the file menu with drop down menus headed under **Arduino**, **File**, **Edit**, **Sketch**, **Tools**, and **Help**. The buttons in the Toolbar (see Figure 1-16) provide convenient access to the most commonly used functions within this file menu.

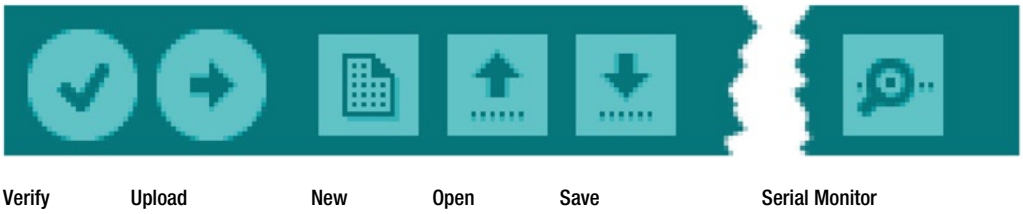

Figure 1-16. *The Toolbar*

The Toolbar buttons are listed in Figure 1-16. The functions of each of the buttons are as follows:-

Table 1-1. *The Toolbar Button Functions*

Verify	Checks the code for errors
Upload	Uploads the current sketch to the Arduino
New	Creates a new blank sketch
Open	Shows a list of sketches in your Sketchbook to open
Save	Saves the current Sketch to your Sketchbook
Serial Monitor	Displays serial data being sent from the Arduino

The **Verify** button is used to check that your code is correct and error-free before you upload it to your Arduino board.

The **Upload** button will upload the code within the current sketch window to your Arduino. You need to make sure that you have the correct board and port selected (in the Tools menu) before uploading. It is essential that you save your sketch before you upload it to your board in case a strange error causes your system to hang or the IDE to crash. It is also advisable to verify the code before you upload to ensure there are no errors that need to be debugged first.

The **New** button will create a completely new and blank sketch ready for you to enter your code into. The IDE will ask you to enter a name and a location for your sketch (try to use the default location if possible) and will then give you a blank sketch ready to be coded. The tab at the top of the sketch will now contain the name you have given to your new sketch.

The **Open** button will present you with a list of sketches stored within your sketchbook as well as a list of example sketches that you can try out with various peripherals once connected. The example sketches are invaluable for beginners to use as a foundation for your own sketch. Open the appropriate sketch for the device you are connecting and then modify the code to your own needs.

The **Save** button will save the code within the Sketch window to your sketch file. Once complete you will get a "Done Saving" message at the bottom of your code window.

The **Serial Monitor** is a very useful tool, especially for debugging your code. The monitor displays serial data being sent out from your Arduino (USB or Serial board). You can also send serial data back to the Arduino using the Serial Monitor. If you click the Serial Monitor button you will be presented with an image like the one in Figure 1-17.

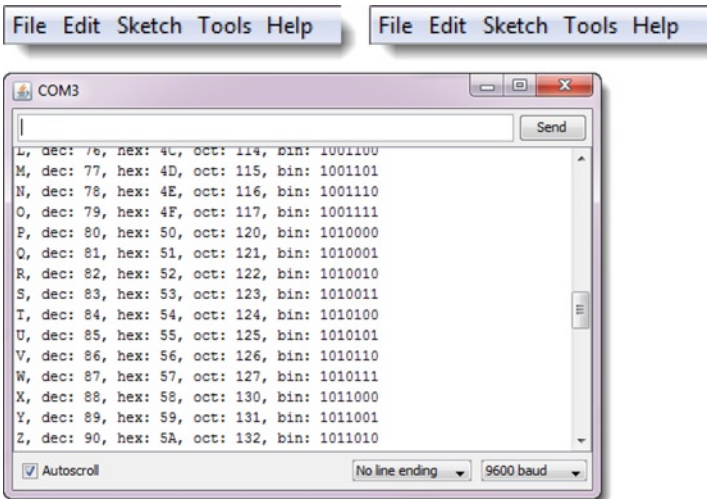

Figure 1-17. *The Serial Monitor in use*

On the bottom right hand side you can select the Baud Rate that the serial data is to be sent to/from the Arduino. The Baud Rate is the rate, per second, that state changes or bits (data) are sent to/from the board. The default setting is 9600 baud, which means that if you were to send a text novel over the serial communications line (in this case your USB cable), then 1200 letters, or symbols, of the novel, would be sent per second (9600 bits/8 bits per character = 1200 bytes or characters – bits and bytes will be explained later on).

At the top is a blank text box for you to enter text to send back to the Arduino and a **Send** button to send the text within that field. Note that the Serial Monitor will not receive any serial data unless you have set up the code inside your sketch to send serial data from the Arduino. Similarly, the Arduino will not receive any data sent unless you have coded it to do so.

There is a tick box on the bottom left where you can choose if you want the data in the serial monitor window to autoscroll or not.

The box to the left of the baud rate menu will affect the data sent from the serial monitor back to the Arduino. The default setting is "no line ending," meaning when you enter data into the text box on the serial monitor and press "send," the data will be sent as is. If you click the drop down menu, there are three other options for Newline, Carriage return. and Both NL+ Cr. By selecting one of these, the serial monitor will append an ascii code for a Newline, Carriage Return, or both on the end of any data entered into the serial monitor window when you click send. Bear this in mind when processing data sent from the serial monitor back to the Arduino.

Finally, the main area is where your serial data will be displayed. In the image above, the Arduino is running the `ASCIITable` sketch that can be found in the Communications examples. This program outputs ASCII characters, from the Arduino via serial (the USB cable) to the PC where the Serial Monitor then displays them.

Once you are proficient at communicating via serial to and from the Arduino, you can use other programs such as Processing, Flash, MaxMSP, etc. to communicate between the Arduino and your PC.

We will make use of the Serial Monitor later on in our projects when we read data from sensors and get the Arduino to send that data to the Serial Monitor, in human readable form for us to see.

The Message Window at the bottom of the IDE is where you will see error messages that the IDE will display to you when trying to connect to your board, upload code, or verify code.

Below the Message Window at the bottom left you will see a number. This is the current line that the cursor, within the code window, is at. If you have code in your window and you move down the lines of code (using the ↓ key on your keyboard) you will see the number increase as you move down the lines of code. This is useful for finding bugs highlighted by error messages.

Across the top of the IDE window (or across the top of your screen if you are using a Mac), you will see the various menus that you can click on to access more menu items (see Figure 1-18).

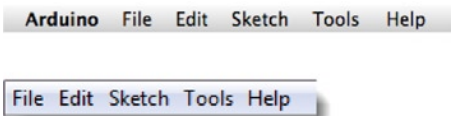

Figure 1-18. *The IDE menus (Top: OSX, Bottom: Windows)*

The first menu (on OSX) is the **Arduino menu** (see Figure 1-19). Within this is the **About Arduino** option, which when pressed will show you the current version number, a list of the people involved in making this amazing device, and some further information. On Windows PCs, the About Arduino item is on the Help menu.

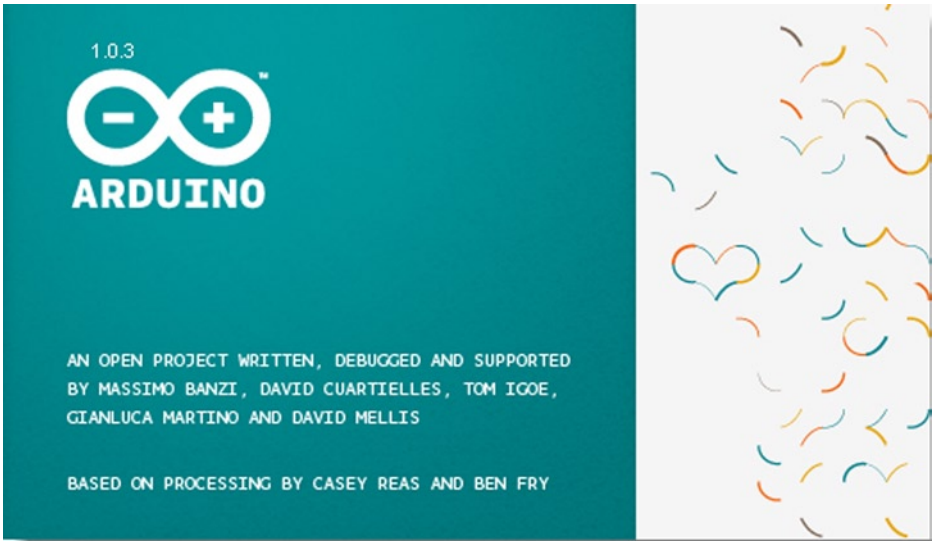

Figure 1-19. *The About Arduino menu*

The next menu is the **File** menu. (see Figure 1-20). Here, you get access to options to create a new sketch, take a look at sketches stored in your Sketchbook, example files, options to save your Sketch (or Save As, if you want to give it a different name). You also have the option to upload your sketch to the Arduino, upload using a programmer (we will not be using this feature) as well as the print options for printing out your code.

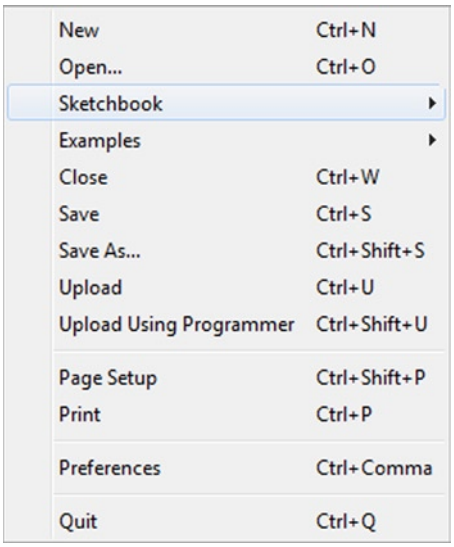

Figure 1-20. *The File Menu*

Near the bottom is the **Preferences** option. This will bring up the Preferences window where you can change various IDE options, such as where your default Sketchbook is stored, etc. Finally, there is the **Quit** option, which will quit the program.

Next is the **Edit** menu (see Figure 1-21) Here, you get options to enable you to cut, copy and paste sections of code. Select all of your code as well as find certain words or phrases within the code. Comment your code (adding comments to explain how it works), as well as increasing or decreasing indents. Also included are the useful Undo and Redo options, which come in handy when you make a mistake.

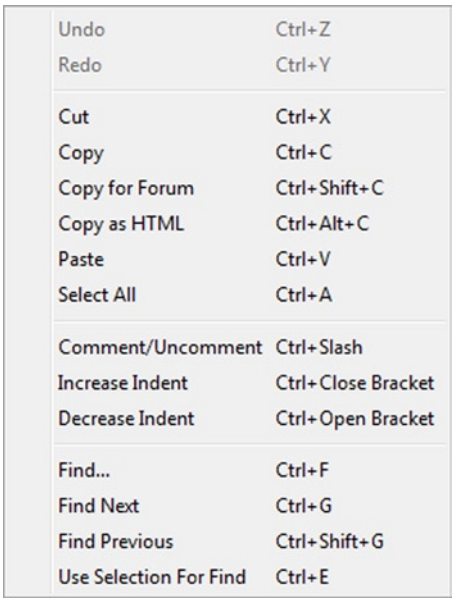

Figure 1-21. *The Edit Menu*

Our next menu is the **Sketch** menu (see Figure 1-22) which gives us access to the Verify/Compile functions and some other useful functions you will use later on. These include the Import Library option, which when clicked will bring up a list of the available libraries, stored within your libraries folder.

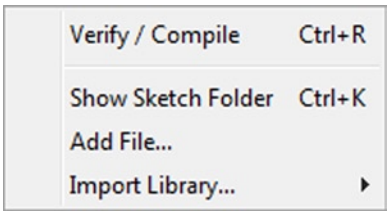

Figure 1-22. *The Sketch Menu*

A library is a collection of code that you can include in your sketch to enhance the functionality of your project. It is a way of preventing you from reinventing the wheel by reusing code already made by someone else for various pieces of common hardware you may encounter whilst using the Arduino.

For example, one of the libraries you will find is Stepper, which is a set of functions you can use within your code to control a stepper motor. Somebody else has kindly already created all of the necessary functions necessary to control a stepper motor, and by including the Stepper library into our sketch, we can use those functions to control the motor as we wish. By storing commonly used code in a library, you can reuse that code over and over in different projects and also hide the complicated parts of the code from the user. We will go into greater detail concerning the use of libraries later on.

Finally, within the Sketch menu is the Show Sketch Folder option, which will open up the folder were your sketch is stored. Also, there is the Add File option, which will enable you to add another source file to your sketch. This functionality allows you to split larger sketches into smaller files and then add them to the main Sketch.

The next menu in the IDE is the **Tools** menu (see Figure 1-23). Within this are the options to select the board and serial port we are using, as we did when setting up the Arduino for the first time. Also we have the Auto Format function that formats your code to make it look nicer.

Figure 1-23. *Tools Menu*

The Copy for Forum option will copy the code within the Sketch window, but in a format that when pasted into the Arduino forum (or most other Forums for that matter) will show up the same as it is in the IDE, along with syntax coloring, etc.

The Archive Sketch option will enable you to compress your sketch into a ZIP file and will ask you where you want to store it. The Fix Encoding & Reload option is to convert code created in older versions of the IDE into the newer format.

The programmer button will enable you to choose a programmer, in case you are using an external device to upload code to your Arduino or wish to burn code to a chip in your own project. We will simply be using the USB cable we purchased with our Arduino.

Finally, the Burn Bootloader option can be used to burn the Arduino Bootloader (piece of code on the chip to make it compatible with the Arduino IDE) to the chip. This option can only be used if you have an AVR programmer and have replaced the chip in your Arduino or have bought blank chips to use in your own embedded project. Unless you plan on burning lots of chips, it is usually cheaper and easier to just buy an ATmega chip (see Figure 1-24) with the Arduino Bootloader already pre-programmed. Many online stores stock pre-programmed chips and these can be purchased pretty cheaply. The chip used in the Arduino Uno is an Atmel ATmega328.

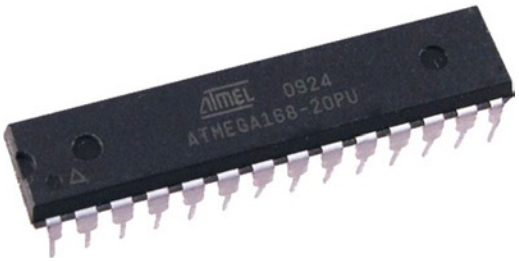

Figure 1-24. *An Atmel ATmega chip. The heart of your Arduino. (image courtesy of Earthshine Electronics)*

The final menu is the **Help** menu were you can find help menus for finding out more information about the IDE or links to the reference pages of the Arduino website and other useful pages.

The Arduino IDE is pretty basic and you will learn how to use it quickly and easily as we work through the projects. As you become more proficient at using an Arduino and programming in C (the programming language we use to code on the Arduino) you may find the Arduino IDE is too basic and wish to use something with better functionality. Indeed, many expert Arduino programmers do not use the IDE at all and instead use professional IDE programs (some of which are free) such as Eclipse, ArduIDE, GNU/Emacs, AVR-GCC, AVR Studio, and even Apple's XCode.

So, now that you have your Arduino software installed, the board connected and working, and you have a basic understanding of how to use the IDE, let's jump right in with Project 1 — LED Flasher.

Summary

In this chapter you have learnt what an Arduino is, a little bit about the different Arduino variants, what you can do with it, and what the basic components are that make up the Arduino board. Then you learnt how to install and set up the software and drivers for the Arduino, how to select the correct serial port, and upload a test sketch to your Arduino to make sure everything is working correctly.

Next we moved onto the IDE: how to use it and what the purpose of each of the buttons and menus is, including the serial monitor window. These are the basic concepts required to understand how to set up the software to work with the Arduino hardware. In the next chapter, we will put those concepts into practice by using the IDE to write our code and upload it to our Arduino board.