Getting Started

1.0 Introduction

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

NOTE

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

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

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

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

problems that can provide inspiration and assistance as you pursue your own projects.

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

NOTE

You may be used to referring to software source code as a "program" or just "code." In the Arduino community, source code that contains computer instructions for controlling Arduino functionality is referred to as a *sketch*. The word *sketch* will be used throughout this book to refer to Arduino program code.

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

NOTE

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

Arduino Software

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

Arduino Hardware

The Arduino board is where the code you write is executed. The board can only control and respond to electricity, so you'll attach specific components to it that enable it to interact with the real world. These

components can be sensors, which convert some aspect of the physical world to electricity so that the board can sense it, or actuators, which get electricity from the board and convert it into something that changes the world. Examples of sensors include switches, accelerometers, and ultrasonic distance sensors. Actuators are things like lights and LEDs, speakers, motors, and displays.

Arduino boards can be enhanced by add-ons called *shields*, which you connect to an Arduino by stacking them on top with their pins connected to all the headers of the Arduino. Different models of Arduino and certain Arduino-compatibles may have their own add-ons similar to, but not compatible with shields. This is because some models of boards use a different layout of connector pins and have a different shape from the most common Arduino, the Uno. For example, the Arduino MKR and the Adafruit Feather lines of boards are both physically much smaller than the Uno, but have different pin layouts. The add-on boards that are designed to plug directly into MKR boards are called carriers, and for the Feather line, they are called Featherwings.

There are a variety of official boards that you can use with Arduino software and a wide range of Arduino-compatible boards produced by companies and individual members of the community. In addition to all the boards on the market, you'll even find Arduino-compatible controllers inside everything from 3D printers to robots. Some of these Arduino-compatible boards and products are also compatible with other programming environments such as MicroPython or CircuitPython.

The most popular boards contain a USB connector that is used to provide power and connectivity for uploading your software onto the board. Figure 1-1 shows a basic board that most people start with, the Arduino Uno. It is powered by an 8-bit processor, the ATmega328P, which has 2 kilobytes of SRAM (static random-access memory, used to store program variables), 32 kilobytes of flash memory for storing your sketches, and runs at 16 MHz. A second chip handles USB connectivity.

Basic board: the Arduino Uno. Photograph courtesy todo.to.it.

The Arduino Leonardo board uses the same form factor as the Uno, but uses a different processor, the ATmega32U4, which runs your sketches

and also takes care of USB connectivity. It is slightly cheaper than the Uno, and also offers some interesting features, such as the ability to emulate various USB devices such as mice and keyboards. The Arduino-compatible Teensy and Teensy+ boards from PJRC (*http://www.pjrc.com/teensy/*) are also capable of emulating USB devices.

Another board with a similar pin layout and even faster processor is the Arduino Zero. Unlike the Arduino Uno and Leonardo, it cannot tolerate input pin voltages higher than 3.3 volts. The Arduino Zero has a 32-bit processor running at 48 MHz and has 32 kilobytes of RAM and 256 kilobytes of flash storage. Adafruit's Metro M0 Express and SparkFun's RedBoard Turbo comes in the same form factor as the Arduino Zero, and also offers compatibility with multiple environments, including the Arduino IDE and CircuitPython.

ARDUINO AND USB

The Arduino Uno has a second microcontroller onboard to handle all USB communication; the small surface-mount chip (the ATmega16U2, ATmega8U2 in older versions of the Uno) is located near the USB socket on the board. This can be programmed separately to enable the board to appear as different USB devices (see ??? for an example).

Older Arduino boards, and some of the Arduino-compatible boards, use a chip from the FTDI company that provides a hardware USB solution for connection to the serial port of your computer. Some of the cheaper clones that you will encounter on eBay or Amazon may use a chip that performs a similar function, such as the CH340. You will probably need to install a driver to use CH340-based boards.

There's another class of USB-enabled Arduino-compatible boards you may encounter, which have no dedicated chip to handle USB communication. Instead, these boards use a technique called *bit-banging*, in which software running on the board manipulates I/O pins to send and receive USB signals. These boards, which include the popular original Adafruit Trinket, generally do not work well with modern computers, though you may have luck with an older computer. (Adafruit has released the Adafruit Trinket M0, which does not have this problem, and as a bonus, is much faster than its predecessor.)

Finally, you may find Arduino-compatible boards that have no USB connection whatsoever. Instead, they offer only serial pins that you cannot directly connect to

a computer without a special adapter. See "Serial Hardware" for a list of some available adapters.

If you want a board for learning that will run all the sketches in this book, the Uno is a great choice. If you want more performance than the Uno then consider the Zero, or a similar board such as the Metro M0 Express or RedBoard Turbo.

CAUTION NEEDED WITH SOME 3.3 VOLT BOARDS

Many of the newer boards operate on 3.3 volts rather than 5 volts used by older boards such as the Uno. Some such boards can be permanently damaged if a pin receives 5 volts, even for a fraction of a second, so check the documentation for your board to see if it is tolerant of 5 volts before wiring things up when there is a risk of pin levels higher than 3.3 volts.

Arduino boards come in other form factors, which means that the pins on such boards have a different layout and aren't compatible with shields designed for the Uno. The MKR1010 is an Arduino board that uses a much smaller form factor. Its pins are designed for 3.3V I/O (it is **not** 5V-tolerant) and like the Zero, it uses an ARM chip. However, the MKR1010 also includes WiFi and and a circuit to run from and recharge a LIPO battery. Although the MKR family of boards is not compatible with shields designed for the Uno, Arduino offers a selection of add-on boards for the MKR form factor called *carriers*. Similarly, Adafruit has a huge collection of *Featherwing* add-on boards for their Feather line of development boards, many of which are Arduino-compatible.

You can get boards as small as a postage stamp, such as the Adafruit Trinket M0; larger boards that have more connection options and more powerful processors, such as the Arduino Mega and Arduino Due; and boards tailored for specific applications, such as the Arduino LilyPad for wearable applications, the Arduino Nano 33 IoT for wireless projects, and the Arduino Nano Every for embedded applications (standalone projects that are often battery-operated).

Other Arduino-compatible boards are also available, including the following:

- Arduino Nano, a tiny board with USB capability, from Gravitech (*http://store.gravitech.us/arna30wiatn.html*)
- Bare Bones Board, a low-cost board available with or without USB capability, from Modern Device (*http://www.moderndevice.com/products/bbb-kit*) and Educato, a shield compatible version (https://moderndevice.com/product/educato/)
- Adafruit Industries (*http://www.adafruit.com/*) has a vast collection of Arduino and Arduino-compatible boards and accessories (boards, modules, and components).
- Sparkfun (https://www.sparkfun.com/categories/242) has lots of Arduino and Arduino-compatible accessories for Arduino projects.
- Seeed Studio (*http://www.seeedstudio.com/*) sells Arduino and Arduino-compatible boards as well as many accessories. They also offer a flexible expansion system for Arduino and other embedded boards called Grove (*http://wiki.seeedstudio.com/Grove_System/*), which uses a modular connector system for sensors and actuators.
- Teensy and Teensy++, tiny but extremely versatile boards, from PJRC (*http://www.pjrc.com/teensy/*)

An exhaustive list of Arduino-compatible boards is available at *Wikipedia*.

See Also

An overview of Arduino boards: *https://www.arduino.cc/en/Main/Products*.

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

1.1 Installing the Integrated Development Environment (IDE)

Problem

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

Solution

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

For Windows you can download a Windows installer, which you just double click to run when you have downloaded it, but you will need to have administrative privileges to do this. If you are running Windows 10 or Windows 8.1, you can use the Microsoft Store to install Arduino without needing admin privileges. Alternatively, you can download the Windows zip file, and unzip the file to any convenient directory that you have write access to.

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

If you use the Windows installer or install Arduino from the Microsoft Store, you can launch Arduino from the Start Menu.

Arduino splash screen (Arduino 1.8.9 on Windows 10)

IDE main window (Arduino 1.8.9 on a Mac)

NOTE

The first time you run Arduino on Windows, you may see a warning that says “Windows Defender Firewall has blocked some features of this app”, specifying `javaw.exe` as the source of the warning. The Arduino IDE is a Java-based application, which is why the warning comes from the Java program instead of `Arduino.exe`. It is not clear which, if any, features of Arduino depend on your permitting access through the firewall, so

you can either accept the defaults or press Cancel. If you want to change your mind later, you'll need to open the Windows Defender Firewall control panel, choose the option to "Allow apps to communicate through Windows Defender Firewall," locate "Java(TM) Platform SE Binary" in the list, and use the Details button to confirm that it's the `javaw.exe` program from your Arduino installation before you change the settings in the control panel.

The Arduino download for the Mac is a zip file. If you are using the Safari web browser, the zip file will be automatically unzipped, and the Arduino application will pop up. Otherwise you'll need to extract the zip yourself. Move the application to somewhere convenient—the `Applications` folder is a sensible place. Double-click the application. The splash screen will appear, followed by the main program window.

Linux versions are increasingly available from your distributions package manager, but these versions are often not the most current release, so it is best to download the version from *http://arduino.cc/en/Main/Software*. It is available for 32 or 64 bit, and an ARM version which can be used on the Raspberry PI and other Linux ARM boards.

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

On Windows, if you used the installer then when you plug in the board it will automatically associate the installed driver with the board, this may take a little time. If this process fails, or you installed Arduino using the zip file then visit procedure at https://www.arduino.cc/en/Guide/HomePage, click the link for your board from the list on that page, and follow the instructions there.

If you are using an earlier board (any board that uses FTDI drivers) and you are online, you can let Windows search for drivers and they will install automatically. If you don't have Internet access, or are using Windows XP you should specify the location of the drivers. Use the file selector to navigate to the `drivers\FTDI USB Drivers` directory, located in the directory where you unzipped the Arduino files. When this driver has installed, the Found New Hardware Wizard will appear again, saying a new serial port has been found. Follow the same process again.

NOTE

You may need to go through the sequence of steps to install the drivers twice to ensure that the software is able to communicate with the board.

On the Mac, current Arduino boards, such as the Uno, can be used without additional drivers. When you first plug the board in, a notification may pop up saying a new network port has been found, you can dismiss this. If you are using earlier boards that need FTDI drivers, you can get these from http://www.ftdichip.com/Drivers/D2XX.htm.

On Linux, most distributions use a standard driver that is already installed, and usually have FTDI support as well, but follow the Linux link given in this chapter's introduction for specific information for your distribution.

If you are using an Arduino-compatible board that is not made by Arduino, then a similar installation process may be necessary for the board to be recognised by the IDE. Increasingly this is handled automatically by the operating system, but you may have to check the specific board's documentation for details, especially if it is an Arduino compatible board rather than an official board.

Discussion

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

See Also

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

In addition to the IDE, there is also an online editing environment called *Arduino Create*. In order to use this you will need to create an account and download a plugin that enables the web site to communicate with the board to upload code. It has cloud storage where your sketches are saved and provides facilities for sharing code. At the time this book

was written, *Arduino Create* was a fairly new, still evolving service. If you would like the ability to create Arduino sketches without having to install a development environment on your computer then have a look at *Arduino Create* here: https://create.arduino.cc/.

If you are using a Chromebook, Arduino Create's Chrome App requires a monthly subscription of US$1 per month. It has a time-limited trial so you can try it out. There is another alternatives to compiling and uploading Arduino code from a Chromebook: Codebender is a web-based IDE like Arduino Create, but is also supports a number of third-party Arduino-compatible boards. They have pricing plans available for classooms and schools as well. See https://edu.codebender.cc/.

1.2 Setting Up the Arduino Board

Problem

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

Solution

Plug the board in to a USB port on your computer and check that the LED power indicator on the board illuminates. Most Arduino boards have an LED power indicator that stays on whenever the board is powered.

The onboard LED (labelled in Figure 1-4) should flash on and off when the board is powered up (most boards come from the factory preloaded with software to flash the LED as a simple check that the board is working).

Basic Arduino board, the Uno rev3

NOTE

The current boards in the Arduino Uno form factor have some pins that weren't present on older boards, and you may encounter some older Arduino shields that don't have these pins. Fortunately, this usually does

not affect the use of older shields: most will continue to work with the new boards, just as they did with earlier boards (but your mileage may vary).

The new connections provide a pin (IOREF) for shields to detect the analog reference voltage (so that analog input values can be correlated with the supply voltage), SCL and SDA pins to enable a consistent pin location for I2C devices. The location of the I2C pins had varied on some earlier boards such as the Mega due to different chip configurations, and in some cases, certain shields required workarounds such as the addition of jumper wires to connect the shield's I2C pins to the ones on the Mega. Shields designed for the new layout should work on any board with the new pin locations. An additional pin (next to the IOREF pin) is not being used at the moment, but enables new functionality to be implemented in the future without needing to change the pin layout again.

Discussion

If the power LED does not illuminate when the board is connected to your computer, the board is probably not receiving power (try a different USB socket or cable).

The flashing LED is being controlled by code running on the board (new boards are pre-loaded with the Blink example sketch). If the onboard LED is flashing, the sketch is running correctly, which means the chip on the board is working. If the power LED is on but the onboard LED (usually labeled L) is not flashing, it could be that the factory code is not on the chip; follow the instructions in Recipe 1.3 to load the Blink sketch onto the board to verify that the board is working. If you are not using a standard board, it may not have an onboard LED, so check the documentation for details of your board.

The Leonardo and Zero-class boards (Arduino Zero, Adafruit Metro M0, SparkFun RedBoard Turbo) have the same footprint as the Uno (its headers are in the same position, enabling shields to be attached). They are significantly different in other respects. The Leonardo has an 8-bit chip like the Uno, but because it doesn't have a separate chip for handling USB communications, the Leonardo only accepts program uploads immediately after the board has been reset. You'll see the

Leonardo's onboard LED pulse gently while it's waiting for an upload. The Leonardo is 5V tolerant. The Zero has a 32 bit ARM chip, with more memory for storing your program and running it. There is also a pin that provides a DAC (Digital to Analog Convertor), which means you can get a varying analog voltage from it. This can be used to generate audio signals at much higher quality than an Uno. The Zero is not 5V tolerant, nor are the similar boards from Adafruit (Metro M0 Express) or SparkFun (RedBoard Turbo).

The MKR1010 uses the same chip as the Zero (and like the Zero, is not 5V tolerant), but in a smaller form factor. It also includes Wifi, so is able to connect to the internet through a WiFi network. The MKR form factor does not support shields that are designed for the Uno pin layout.

All the 32 bit boards have more pins that support interrupts than most of the 8-bit boards, which are useful for applications that must quickly detect signal changes (see ???). The one 8-bit exception to this is the Arduino Uno WiFi Rev 2, which supports interrupts on any of its digital pins.

See Also

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

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

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

Problem

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

Solution

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

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

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

After you've started the IDE, go to the File→Examples menu and select 01. Basics→Blink, as shown in Figure 1-6. The code for blinking the built-in LED will be displayed in the Sketch Editor window (refer to Figure 1-5).

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

You should see a message that reads "Compiling sketch..." and a progress bar in the message area below the text-editing window. After a second or two, a message that reads "Done Compiling" will appear. The black console area will contain the following additional message:

```
Sketch uses 930 bytes (2%) of program storage space. Maximum
is 32256 bytes.

Global variables use 9 bytes (0%) of dynamic memory, leaving
2039 bytes for

local variables. Maximum is 2048 bytes.
```

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

Arduino IDE on macOS

Discussion

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

IDE menu (selecting the Blink example sketch) on Windows 10

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

```
Sketch too big; see

http://www.arduino.cc/en/Guide/Troubleshooting#size for tips
on reducing it.
```

If this happens, you need to make your sketch smaller to be able to put it on the board, or get a board with higher flash memory capacity. If your global variables are using too much memory, you'll see a different error instead:

```
Not enough memory; see
http://www.arduino.cc/en/Guide/Troubleshooting#size

for tips on reducing your footprint.
```

In that case, you'll need to go through your code and reduce the amount of memory that you are allocating to global variables, or get a board with a higher SRAM (dynamic memory) capacity.

If there are errors in the code, the compiler will print one or more error messages in the console window. These messages can help identify the error—see ??? on software errors for troubleshooting tips.

NOTE

To prevent you from accidentally overwriting the example code, the Arduino IDE does not allow you to save changes to the built-in example sketches. You must rename them using the Save As menu option. You can save sketches you write yourself with the Save button (see Recipe 1.5).

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

NOTE

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

See Also

Recipe 1.5 shows an example sketch. ??? has tips on troubleshooting software problems.

1.4 Uploading and Running the Blink Sketch

Problem

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

Solution

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

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

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

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

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

```
/dev/cu.usbmodem-xxxxxxx(Arduino/Genuino Uno)
```

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

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

Each board will have different values for *xxxxxxx*. Select either entry.

On Linux, if your board is an Uno it will probably be listed as:

```
/dev/ttyACMx(Arduino/Genuino Uno)
```

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

```
/dev/ttyUSB-x
```

x is usually 0, but you will see 1, 2, etc. if you have multiple boards connected at once. Select the entry that corresponds to your Arduino.

NOTE

If you have so many entries in the Port menu that you can't figure out which one goes to your Arduino, try this: look at the menu with the Arduino unplugged from your computer, then plug the Arduino in and look for the menu option that wasn't there before. Another approach is

to select the ports, one by one, and try uploading, until you see the lights on the board flicker to indicate that the code is uploading.

Click the upload button (in Figure 1-5, it's the second button from the left), or choose Sketch→Upload (Ctrl-U, ⌘+U on a Mac).

The IDE will compile the code, as in Recipe 1.3. After the software is compiled, it is uploaded to the board. If this is a fresh out-of-the-box Arduino that's pre-loaded with the Blink sketch, you will see the onboard LED (labeled as Onboard LED in Figure 1-4) stop blinking. When the upload begins, two LEDs (labeled as Serial LEDs in Figure 1-4) near the onboard LED should flicker for a couple of seconds as the code uploads. The onboard LED should then start flashing as the code runs. The location of the onboard LED differs across some Arduino models, such as the Leonardo, MKR boards, and third-party Arduino clones.

Discussion

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

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

NOTE

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

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

serial port are displayed in the status bar at the bottom of the Arduino window

See Also

The Arduino page: *http://www.arduino.cc/en/Guide/Troubleshooting*.

1.5 Creating and Saving a Sketch

Problem

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

Solution

To open an editor window ready for a new sketch, launch the IDE (see Recipe 1.3), go to the File menu, and select New. Delete the boilerplate code that is in the Sketch Editor window, and paste the following code in its place (it's similar to the Blink sketch, but the blinks last twice as long):

```
void setup()
{
  pinMode(LED_BUILTIN, OUTPUT);
}


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

```
    delay(2000);                    // wait for two seconds

  }
```

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

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

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

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

Discussion

When you save a file in the IDE, a standard dialog box for the operating system will open. It suggests that you save the sketch to a folder called `Arduino` in your `My Documents` folder (or your `Documents` folder on a Mac). You can replace the default sketch name with a meaningful name that reflects the purpose of your sketch. Click Save to save the file.

NOTE

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

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

Arduino sketches are saved as plain text files with the extension `.ino`. Older versions of the IDE used the `.pde` extension, also used by

Processing. They are automatically saved in a folder with the same name as the sketch.

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

NOTE

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

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

NOTE

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

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

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

NOTE

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

If you try and save a sketch file that is not in a folder with the same name as the sketch, the IDE will inform you that this can't be opened as is and suggest you click OK to create the folder for the sketch with the same name.

NOTE

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

Sketches made with older versions of Arduino software have a different file extension (`.pde`). The IDE will open them, when you save the sketch it will create a file with the new extension (`.ino`). Code written for early versions of the IDE may not be able to compile in version 1.0. Most of the changes to get old code running are easy to do. See ??? for more details.

1.6 Using Arduino

Problem

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

Solution

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

The sketch is based on the LED blinking code from the previous recipe, but instead of using a fixed delay, the rate is determined by a light-sensitive sensor called a light dependent resistor (LDR for short, see Recipe 6.3). Wire the LDR as shown in Figure 1-7.

Arduino with light dependent resistor

NOTE

If you are not familiar with building a circuit from a schematic, see ??? for step-by-step illustrations on how to make this circuit on a breadboard.

NOTE

LDRs contain a compound (cadmium sulfide) that is a hazardous substance. You can use a phototransistor if you live in a jurisdiction where it is difficult to obtain an LDR, or if you simply prefer to not use an LDR. A phototransistor has a long lead and a short lead, much like an LED. You can wire it exactly as shown in the figure, but you must connect the long lead to 5V and the short lead to the resistor and pin 0. Be sure to buy a phototransistor such as Adafruit part number 2831 (https://www.adafruit.com/product/2831), that can sense visible light so you can test it with a common light source.

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

```
const int sensorPin = A0;          // connect sensor to
analog input 0



void setup()

{

  pinMode(LED_BUILTIN, OUTPUT);  // enable output on the led
pin

}



void loop()

{

  int rate = analogRead(sensorPin); // read the analog input
```

```
  digitalWrite(LED_BUILTIN, HIGH);  // set the LED on

  delay(rate);                      // wait duration
dependent on light level

  digitalWrite(LED_BUILTIN, LOW);   // set the LED off

  delay(rate);

}
```

The code in this recipe and throughout this book use the `const int` expression to provide meaningful names (`sensorPin`) for constants instead of numbers (`0`). See ??? for more on the use of constants.

Discussion

The value of the 4.7K resistor is not critical. Anything from 1K to 10K can be used. The light level on the sensor will change the voltage level on analog pin 0. The `analogRead` command (see Chapter 6) provides a value that ranges from around 200 when the sensor is dark to 800 or so when it is very bright (the sensitivity will vary depending on the type of LDR and resistor you use, and whether you use a phototransistor in place of the LDR). The analog reading determines the duration of the LED on and off times, so the blink delay increases with light intensity.

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

```
const int sensorPin = A0;    // connect sensor to analog
input 0



// Low and high values for the sensor readings. You may need
to adjust these.

const int low  = 200;

const int high = 800;
```

```
// the next two lines set the min and max delay between
blinks

const int minDuration = 100;  // minimum wait between blinks

const int maxDuration = 1000;  // maximum wait between
blinks



void setup()

{

  pinMode(LED_BUILTIN, OUTPUT);  // enable output on the led
pin

}



void loop()

{

  int rate = analogRead(sensorPin);    // read the analog
input



  // the next line scales the blink rate between the min and
max values

  rate = map(rate, low,high, minDuration,maxDuration); //
convert blink rate

  rate = constrain(rate, minDuration,maxDuration);     //
constrain the value
```

```
  digitalWrite(LED_BUILTIN, HIGH); // set the LED on

  delay(rate);                     // wait duration
dependent on light level

  digitalWrite(LED_BUILTIN, LOW);  // set the LED off

  delay(rate);

}
```

NOTE

If you're not seeing any change in values as you adjust the light, you will need to play with the values for `low` and `high`. If you are using a phototransistor and aren't getting changes in the blink rate, try a value of 10 for `low`.

Recipe 5.7 provides more details on using the `map` function to scale values. Recipe 3.5 has details on using the `constrain` function to ensure values do not exceed a given range.

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

```
const int sensorPin = A0;    // connect sensor to analog
input 0



// Low and high values for the sensor readings. You may need
to adjust these.

const int low  = 200;

const int high = 800;
```

```
// the next two lines set the min and max delay between
blinks

const int minDuration = 100;  // minimum wait between blinks

const int maxDuration = 1000;  // maximum wait between
blinks



void setup()

{

  pinMode(LED_BUILTIN, OUTPUT);  // enable output on the led
pin

  Serial.begin(9600);            // initialize Serial

}



void loop()

{

  int rate = analogRead(sensorPin);    // read the analog
input

  // the next line scales the blink rate between the min and
max values

  rate = map(rate, low,high, minDuration,maxDuration); //
convert blink rate

  rate = constrain(rate, minDuration,maxDuration);     //
constrain the value
```

```
  Serial.println(rate);            // print rate to serial
monitor

  digitalWrite(LED_BUILTIN, HIGH); // set the LED on

  delay(rate);                     // wait duration
dependent on light level

  digitalWrite(LED_BUILTIN, LOW);  // set the LED off

  delay(rate);

}
```

You can use the sensor to control the pitch of a sound by connecting a small speaker to the pin, as shown in Figure 1-8.

Connections for a speaker with the LDR circuit

You will need to increase the on/off rate on the pin to a frequency in the audio spectrum. This is achieved, as shown in the following code, by decreasing the min and max durations:

```
const int outputPin = 9;    // Speaker connected to digital
pin 9

const int sensorPin = A0;    // connect sensor to analog
input 0



const int low  = 200;

const int high = 800;



const int minDuration = 1;   // 1ms on, 1ms off (500 Hz)

const int maxDuration = 10;  // 10ms on, 10ms off (50 hz)
```

```
void setup()

{

  pinMode(outputPin, OUTPUT);  // enable output on the led
pin

}



void loop()

{

  int sensorReading = analogRead(sensorPin);    // read the
analog input

  int rate = map(sensorReading, low,high,
minDuration,maxDuration);

  rate = constrain(rate, minDuration,maxDuration);    //
constrain the value



  digitalWrite(outputPin, HIGH);   // set the LED on

  delay(rate);                     // wait duration
dependent on light level

  digitalWrite(outputPin, LOW);    // set the LED off

  delay(rate);

}
```

See Also

See Recipe 3.5 for details on using the `constrain` function.

See Recipe 5.7 for a discussion on the `map` function.

If you are interested in creating sounds, see ??? for a full discussion on audio output with Arduino.

1.7 Using Arduino with boards not included in the standard distribution

Problem

You want to use a board such as the Arduino MKR 1010, but it does not appear in the boards menu.

ADDING OTHER BOARDS TO THE BOARDS MENU

The procedure described here is similar for other boards you may want to add to the boards menu. Check the documentation for you board to find the location of the definition files for your board.

Solution

In order to use the MKR 1010 with the Arduino software you need to add its details to the Arduino software you have already downloaded.

To do this got to Tools→Board→Boards Manager.

Selecting Boards Manager (Linux version of Arduino IDE shown)

As this window opens the list of board definitions available online will be checked to ensure you have the latest versions available, wait till this has finished.

The window that opens shows you the Board definitions that are already installed, and ones that are available to download.

The Boards Manager

To find the MKR 1010 you can scroll down the list, or type its name in the filter box. For the MKR 1010, you'll need to select the Arduino SAMD boards. Once you have selected it, click on install and it will be downloaded and added to the Arduino IDE. This may take some time.

Once it has finished you can add other boards, or click on Close to finish using the Boards Manager. If you look at the board menu now you should have the option of selecting the MKR 1010.

The MKR 1010 is now installed and can be programmed using the Arduino IDE

Discussion

The files that you download when you do this describe how to map the programming concepts in Arduino that connect to specific bits of hardware in chip, to where that hardware is located in a specific chip, or family of chips.

Once you have added the description for a particular chip then you will often be able to work with a family of boards that use that chip. For example, adding support for the MKR 1010 board also provides support for the Arduino Zero as both boards use the same microcontroller chip.

To facilitate support for the growing number of Arduino and Arduino-compatible boards, the Arduino IDE added a Boards Manager in release 1.6. The Boards Manager was developed to enable people to easily add and remove board details from their installation. It also enables you to update the board support files if newer versions are available, or chose the version you use if you need to use a particular one. The Arduino IDE no longer includes the description files for all the Arduino boards, so even if you download the latest IDE you may not have the descriptions for the board you have.

The Boards Manager also enables third parties to add the details of their boards to the system. If their board descriptions are available online in the correct format then you can add the location as one of the places for Boards Manager to use to populate the list it produces. This means those

files will also get checked whenever the Boards Manager updates its details, so you get notified of updates and can use the same mechanism to update them once they are installed. To do this go to Arduino→Preferences and click on the icon to the right of the Additional Boards Manager URLs

Preferences after clicking on the icon to the right of the Additional Boards Manager URLs entry

If the people who made the board provide a URL to add to Arduino then paste it into the 'additional URLs' dialog box (on a separate line if there are any other entries). If there isn't an explicit URL then Click on the text below the box to go to the web page that maintains a list of unofficial Arduino board description URLs, and see if you can find a link there.

If you want to use a Teensy board (https://www.pjrc.com/teensy/) then you need to download a separate installer program from the Teensy website. You can get instructions and the download for that here. It is important that you use a Teensy installer that has support for the IDE version that you are using. There is usually a compatible version produced within a week or two of a new Arduino release.

See Also

- Quick start guides for various Arduino boards https://www.arduino.cc/en/Guide/HomePage

1.8 Using a 32-Bit Arduino (or Compatible)

Problem

You want 32 bit performance in the Uno form factor.

Solution

The Arduino Zero has the familiar pin layout of the Uno but has a much more memory and a faster processor. If you have trouble obtaining a Zero, Adafruit's Metro M0 Express and SparkFun's RedBoard Turbo are compatible alternatives.

The Arduino/Genuino Zero board. Photograph courtesy arduino.cc

Despite the similar physical layout of the pins, there are a number of differences. What distinguishes these boards from the Uno is that they use a 32-bit ARM chip, the Microchip SAMD21. This sketch, similar to the previous recipe, highlights some significant differences between the ARM-based boards and the Uno.

```
const int outputPin = A0;     // headphones connected to
analog 0

const int sensorPin = A1;     // connect sensor to analog
input 1



const int low  = 200;

const int high = 800;



const int sampleCount = 16;  // number of samples used to
render one cycle



const int minDur = 1000000/(sampleCount*500); // period in
uS for 500 Hz

const int maxDur = 1000000/(sampleCount*50);  // period for
50 hz
```

```
// table of values for 16 samples of one sine wave cycle

static int sinewave[sampleCount] = {

  0x7FF,0xADD,0xDA7,0xF4D,0xFFF,0xF77,0xDA7,0xB40,

  0x7FF,0x521,0x257,0xB1, 0x0,  0x87, 0x257,0x4BE

};



void setup()

{

  pinMode(outputPin, OUTPUT);   // enable output on the led
pin

  analogWriteResolution(12);    // set the Arduino DAC
resolution

}



void loop()

{

  int sensorReading = analogRead(sensorPin);    // read the
analog input

  int duration = map(sensorReading, low,high,
minDur,maxDur);

  duration = constrain(duration, minDur,maxDur);    //
constrain the value

  for(int sample=0; sample < sampleCount; sample++) {

    analogWrite(outputPin, sinewave[sample]);
```

```
      delayMicroseconds(duration);

    }

  }
```

Before you can load sketches on the Zero, Adafruit Metro M0 or M4, or SparkFun RedBoard, open the Arduino Boards Manager and install the appropriate package (see Recipe 1.7): Arduino SAMD Boards, Adafruit SAMD Boards, or SparkFun SAMD Boards. If you are using an Adafruit or SparkFun board, you'll need to add their board manager URL to the Arduino IDE first. See *https://learn.adafruit.com/add-boards-arduino-v164/setup* (Adafruit) or *https://learn.sparkfun.com/tutorials/installing-arduino-ide/board-add-ons-with-arduino-board-manager* (SparkFun) for details. After you've installed support for your SAMD board, use the Tools menu to configure the Arduino IDE to use hat board and set the correct serial port for connecting to it. Next, upload the code using the Arduino IDE.

THESE SAMD-BASED BOARDS ARE NOT 5 VOLT TOLERANT

You must not connect more than 3.3 volts to their I/O pins or you can damage the board!

Discussion

Connections for audio output with the LDR circuit for the Zero board

Although the wiring may appear similar at first glance to Figure 1-8, the sensor input and audio output use different pins. These boards have a digital to analog converter (DAC) that can create more realistic audio than the binary output of a standard digital pins. However, the DAC is only available on analog pin 0 so the sensor input is here connected to analog pin 1.

Another difference that may not be obvious from the figure is that these boards can only drive up to 7mA on a pin compared to 40mA on the Uno. And because the pin voltage ranges from 0 to 3.3 volts compared to the 0 to 5 volt range of the Uno, the maximum power delivered to a pin is almost 10 time less than the Uno. For that reason, the output pins should be connected to headphones or an amplifier input as it will not drive a speaker directly.

The sketch uses a lookup table of 16 samples per sine wave cycle, however these boards are fast enough to handle much higher resolutions and you can increase the number of samples to improve the purity of the signal.