1

MAKE: PROJECTS 

The Tools

This book is a cookbook of sorts, and this chapter covers the key ingredients. The concepts and tools you'll use in every chapter are introduced here. There's enough information on each tool to get you to the point where you can make it say **"Hello World!"** Chances are you've used some of the tools in this chapter before—or ones just like them. Skip past the things you know and jump into learning the tools that are new to you. You may want to explore some of the less-familiar tools on your own to get a sense of what they can do. The projects in the following chapters only scratch the surface of what's possible for most of these tools. References for further investigation are provided.

◂◂ Happy Feedback Machine by Tuan Anh T. Nguyen
The main pleasure of interacting with this piece comes from the feel of flipping the switches and turning the knobs. The lights and sounds produced as a result are secondary, and most people who play with it remember how it feels rather than its behavior.

It Starts with the Stuff You Touch

All of the objects that you'll encounter in this book—tangible or intangible—will have certain behaviors. Software objects will send and receive messages, store data, or both. Physical objects will move, light up, or make noise. The first question to ask about any object is: what does it do? The second is: how do I make it do what it's supposed to do? Or, more simply, what is its interface?

An object's interface is made up of three elements. First, there's the physical interface. This is the stuff you touch—such as knobs, switches, keys, and other sensors—that react to your actions. The connectors that join objects are also part of the physical interface. Every network of objects begins and ends with a physical interface. Even though some objects in a network (such as software objects) have no physical interface, people construct mental models of how a system works based on the physical interface. A computer is much more than the keyboard, mouse, and screen, but that's what we think of it as, because that's what we see and touch. You can build all kinds of wonderful functions into your system, but if those functions aren't apparent in the things people see, hear, and touch, they will never be used. Remember the lesson of the VCR clock that constantly blinks 12:00 because no one can be bothered to learn how to set it? If the physical interface isn't good, the rest of the system suffers.

Second, there's the software interface—the commands that you send to the object to make it respond. In some projects, you'll invent your own software interface; in others, you'll rely on existing interfaces to do the work for you. The best software interfaces have simple, consistent functions that result in predictable outputs. Unfortunately, not all software interfaces are as simple as you'd like them to be, so be prepared to experiment a little to get some software objects to do what you think they should do. When you're learning a new software interface, it helps to approach it mentally in the same way you approach a physical interface. Don't try to use all the functions at once; first, learn what each function does on its own. You don't learn to play the piano by starting with a Bach fugue—you start one note at a time. Likewise, you don't learn a software interface by writing a full application with it—you learn it one function at a time. There are many projects in this book; if you find any of their software functions confusing, write a simple program that demonstrates just that function, then return to the project.

Finally, there's the electrical interface—the pulses of electrical energy sent from one device to another to be interpreted as information. Unless you're designing new objects or the connections between them, you never have to deal with this interface. When you're designing new objects or the networks that connect them, however, you have to understand a few things about this interface, so that you know how to match up objects that might have slight differences in their electrical interfaces.
x

It's About Pulses

In order to communicate with each other, objects use communications protocols. A protocol is a series of mutually agreed-upon standards for communication between two or more objects.

Serial protocols like RS-232, USB, and IEEE 1394 (also known as FireWire and i.Link) connect computers to printers, hard drives, keyboards, mice, and other peripheral devices. Network protocols like Ethernet and TCP/IP connect multiple computers through network hubs, routers, and switches. A communications protocol usually defines the rate at which messages are exchanged, the arrangement of data in the messages, and the grammar of the exchange. If it's a protocol for physical objects, it will also specify the electrical characteristics, and sometimes even the physical shape of the connectors. Protocols don't specify what happens between objects, however. The commands to make an object do something rely on protocols in the same way that clear instructions rely on good grammar—you can't give useful instructions if you can't form a good sentence.

One thing that all communications protocols have in common—from the simplest chip-to-chip message to the most complex network architecture—is this: it's all about pulses of energy. Digital devices exchange information by sending timed pulses of energy across a shared connection. The USB connection from your mouse to your computer uses two wires for transmission and reception, sending timed pulses of electrical energy across those wires. Likewise, wired network connections are made up of timed pulses of electrical energy sent down the wires. For longer distances and higher bandwidth, the electrical wires may be replaced with fiber optic cables , which carry timed pulses of light. In cases where a physical connection is inconvenient or impossible, the transmission can be sent using pulses of radio energy between radio transceivers (a transceiver is two-way radio, capable of transmitting and receiving). The meaning of data pulses is independent of the medium that's carrying them. You can use the same sequence of pulses whether you're sending them across wires, fiber optic cables, or radios. If you keep in mind that all of the communication you're dealing with starts with a series of pulses—and that somewhere there's a guide explaining the sequence of those pulses—you can work with any communication system you come across.
X

Computers of All Shapes and Sizes

You'll encounter at least four different types of computers in this book, grouped according to their physical interfaces. The most familiar of these is the personal computer. Whether it's a desktop or a laptop, it's got a keyboard, screen, and mouse, and you probably use it just about every working day. These three elements—the keyboard, the screen, and the mouse—make up its physical interface.

The second type of computer you'll encounter in this book, the microcontroller, has no physical interface that humans can interact with directly. It's just an electronic chip with input and output pins that can send or receive electrical pulses. Using a microcontroller is a three-step process:

1. You connect sensors to the inputs to convert physical energy like motion, heat, and sound into electrical energy.
2. You attach motors, speakers, and other devices to the outputs to convert electrical energy into physical action.
3. Finally, you write a program to determine how the input changes affect the outputs.

In other words, the microcontroller's physical interface is whatever you make of it.

The third type of computer in this book, the network server, is basically the same as a desktop computer—it may even have a keyboard, screen, and mouse. Even though it can do all the things you expect of a personal computer, its primary function is to send and receive data over a network. Most people don't think of servers as physical things because they only interact with them over a network, using their local computers as physical interfaces to the server. A server's most important interface for most users' purposes is its software interface.

The fourth group of computers is a mixed bag: mobile phones, music synthesizers, and motor controllers, to name a few. Some of them will have fully developed physical interfaces, some will have minimal physical interfaces but detailed software interfaces, and most will have a little of both. Even though you don't normally think of these devices as computers, they are. When you think of them as programmable objects with interfaces that you can manipulate, it's easier to figure out how they can all communicate, regardless of their end function.
X

Good Habits

Networking objects is a bit like love. The fundamental problem in both is that when you're sending a message, you never really know whether the receiver understands what you're saying, and there are a thousand ways for your message to get lost or garbled in transmission.

You may know how you feel but your partner doesn't. All he or she has to go on are the words you say and the actions you take. Likewise, you may know exactly what message your local computer is sending, how it's sending it, and what all the bits mean, but the remote computer has no idea what they mean unless you program it to understand them. All it has to go on are the bits it receives. If you want reliable, clear communications (in love or networking), there are a few simple things you have to do:

- Listen more than you speak.
- Never assume that what you said is what they heard.
- Agree on how you're going to say things in advance.
- Ask politely for clarification when messages aren't clear.

Listen More Than You Speak

The best way to make a good first impression, and to maintain a good relationship, is to be a good listener. Listening is more difficult than speaking. You can speak anytime you want, but since you never know when the other person is going to say something, you have to listen all the time. In networking terms, this means you should write your programs such that they're listening for new messages most of the time, and sending messages only when necessary. It's often easier to send out messages all the time rather than figure out when it's appropriate, but it can lead to all kinds of problems. It usually doesn't take a lot of work to limit your sending, and the benefits far outweigh the costs.

Never Assume

What you say is not always what the other person hears. Sometimes it's a matter of misinterpretation, and other times, you may not have been heard clearly. If you assume that the message got through and continue on obliviously, you're in for a world of hurt. Likewise, you may be inclined to first work out all the logic of your system—and all the steps of your messages before you start to connect things—then build it, and finally test it all at once. Avoid that temptation.

It's good to plan the whole system out in advance, but build it and test it in baby steps. Most of the errors that occur when building these projects happen in the communication between objects. Always send a quick "Hello World!" message from one object to the others, and make sure that the message got there intact before you proceed to the more complex details. Keep that "Hello World!" example on hand for testing when communication fails.

Getting the message wrong isn't the only misstep you can make. Most of the projects in this book involve building the physical, software, and electrical elements of the interface. One of the most common mistakes people make when developing hybrid projects like these is to assume that the problems are all in one place. Quite often, I've sweated over a bug in the software transmission of a message, only to find out later that the receiving device wasn't even connected, or wasn't ready to receive messages. Don't

assume that communication errors are in the element of the system with which you're most familiar. They're most often in the element with which you're least familiar, and therefore, are avoiding. When you can't get a message through, think about every link in the chain from sender to receiver, and check every one. Then check the links you overlooked.

Agree on How You Say Things

In good relationships, you develop a shared language based on shared experience. You learn the best ways to say things so that your partner will be most receptive, and you develop shorthand for expressing things that you repeat all the time. Good data communications also rely on shared ways of saying things, or protocols. Sometimes you make up a protocol for all the objects in your system, and other times you have to rely on existing protocols. If you're working with a previously established protocol, make sure you understand all the parts before you start trying to interpret it. If you have the luxury of making up your own protocol, make sure you've considered the needs of both the sender and receiver when you define it. For example, you might decide to use a protocol that's easy to program on your web server, but that turns out to be impossible to handle on your microcontroller. A little thought to the strengths and weaknesses on both sides of the transmission, and a bit of compromise before you start to build, will make things flow much more smoothly.

Ask Politely for Clarification

Messages get garbled in countless ways. Perhaps you hear something that may not make much sense, but you act on it, only to find out that your partner said something entirely different from what you thought. It's always best to ask nicely for clarification to avoid making a stupid mistake. Likewise, in network communications, it's wise to check that any messages you receive make sense. When they don't, ask for a repeat transmission. It's also wise to check, rather than assume, that a message was sent. Saying nothing can be worse than saying something wrong. Minor problems can become major when no one speaks up to acknowledge that there's an issue. The same thing can occur in network communications. One device may wait forever for a message from the other side, not knowing, for example, that the remote device is unplugged. When you don't receive a response, send another message. Don't resend it too often, and give the other party time to reply. Acknowledging messages may seem like a luxury, but it can save a whole lot of time and energy when you're building a complex system.

X

Tools

As you'll be working with the physical, software, and electrical interfaces of objects, you'll need physical tools, software, and (computer) hardware.

Physical Tools

If you've worked with electronics or microcontrollers before, chances are you have your own hand tools already. Figure 1-1 shows the ones used most frequently in this book. They're common tools that can be obtained from many vendors. A few are listed in Table 1-1.

In addition to hand tools, there are some common electronic components that you'll use all the time. They're listed as well, with part numbers from the retailers featured most frequently in this book. Not all retailers will carry all parts, so there are many gaps in the table.

NOTE: You'll find a number of component suppliers in this book. I buy from different vendors depending on who's got the best and the least expensive version of each part. Sometimes it's easier to buy from a vendor that you know carries what you need, rather than search through the massive catalog of a vendor who might carry it for less. Feel free to substitute your favorite vendors. A list of vendors can be found in the Appendix.