CHAPTER 7

Wired!

Some of the questions I hear the most are related to data hubs and wires—and for good reason! A block's data hub can be confusing, especially when a block has numerous options on the hub. And wires can be just as difficult to figure out.

In this chapter, I want to take a short break from learning about NXT-G programming blocks and give you some background and tips on how to use data hubs and wires. I hope that any confusion you have will be cleared up by the end of this chapter.

Passing Around Information

To help you understand hubs and wires, let me start with a fake programming block called the `COLOR` block. This block is shown in Figure 7-1.

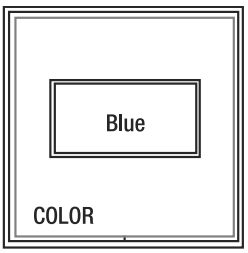

Figure 7-1. *The COLOR block*

This imaginary block is one of the simplest blocks you'll ever encounter. It can hold *one* color. This block holds Blue. It will always hold Blue and nothing else. There is no way to change the color. There's even worse news—the block has no way of sharing this color with a robot. It's a very boring and useless block.

What would make this block useful to us? Well, first, it would be nice to be able to change the color. My favorite color is green, so I'd at least like to change the block to a Green block. I might not be able to do anything else with the block at this point, but at least it will contain my favorite color!

One of the things the block lacks is a way to get inside the block and change Blue to Green. What's so great about creating the `COLOR` block is that I can change it whenever I like (because

it's a fake block). The first thing I'm going to do is attach a very small color keyboard to the block so I can change the color. This color keyboard is a strange type of keyboard, though; it will only let me type colors. If I try to type in "Jim" or "five," the keyboard will buzz to let me know that it's not going to cooperate. Take a look at the updated `COLOR` block in Figure 7-2.

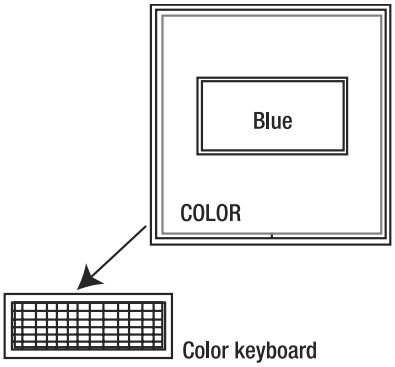

Figure 7-2. *I've added a fake color keyboard, so I can change the Blue block to a Green block.*

Perfect! Now I can type Green. Later, if I want to change to Yellow or Red, I can simply type the new color, and the block will change.

Now I've changed the color, and I have a Green block. Other than looking at it, there's really not much I can do with it. Just like I added a small keyboard to the block, I think I'll now connect a small, fake color screen to the block that will take whatever color is stored inside and display it. This screen is just like my weird keyboard; it will only display a color. (If I had a "direction screen" and I connected it to the block, it wouldn't know what to do with a color. But if I connected it to a `DIRECTION` block that holds North, East, West, or South, then it would definitely work!)

Figure 7-3 shows my new color screen connected to the Green block.

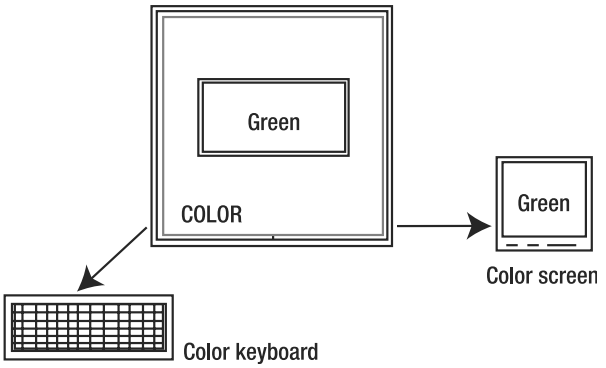

Figure 7-3. *The color screen lets me see what color is stored in the COLOR block.*

So, let's review how this works: the COLOR block can hold only *one* color, not a number or a day or a name.

Next, the COLOR block has a color keyboard attached. I can change the color the COLOR block holds but only by using this special keyboard, and this keyboard will let me type in *only* colors.

Finally, I've attached a color screen to the COLOR block. This special screen will only display colors and nothing else, not names or types of food.

If I detach the color keyboard, can I still display the color inside? Yes, but only if I keep the color screen attached.

If I detach the color screen, can I still change the color inside the block? Yes, again, but only if I keep the color keyboard attached.

Let me give you another way to describe this COLOR block:

- The COLOR block will accept a color as input from the keyboard.
- The COLOR block will also provide a color as output to the screen.

There are some programming words for you in that description: input and output. When thinking about blocks, always remember that any information that is provided to a block is *input*. Any information that the block can give out (share) can be considered *output*.

Now, let's look at a block with a few more options. Take a look at the fake CUP block in Figure 7-4.

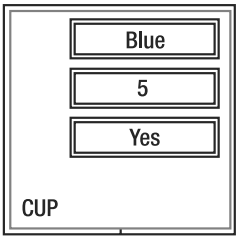

Figure 7-4. *The CUP block is a little more complex.*

The CUP block can hold three pieces of information: the cup's Color, its Height in inches, and a Yes answer if the cup is empty or a No answer if the cup is not empty. Now, here's where it gets fun.

Take a look at Figure 7-5. The COLOR block has an easier way for me to provide input to the block and to receive output from the block. It's called a *hub*.

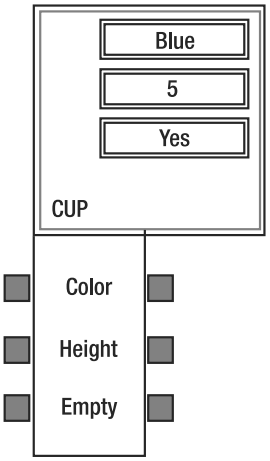

Figure 7-5. *The CUP block has a hub for connecting things.*

You can see in the figure that there are three input plugs on the left side of the hub and three output plugs on the right side. These are where I will plug in keyboards, screens, and other items.

Just like the COLOR block, though, the CUP block is very picky about what types of devices are connected to it. For the Color input plug, I can only connect something that supplies a color. We already know that a color keyboard will work. I could connect a color keyboard and change the color of the cup from Blue to Green. But there's a better way!

Remember that COLOR block we played around with earlier? Well, it has a data hub, too; it was just hidden inside the block. If I click on the lower left edge of the COLOR block, the COLOR block's hub will pop down; this is shown in Figure 7-6.

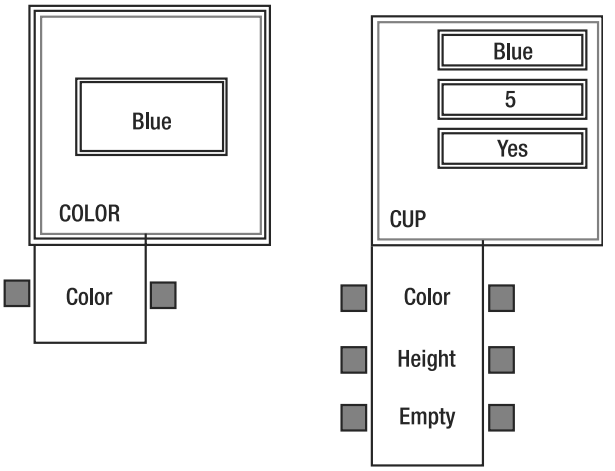

Figure 7-6. *The COLOR block also has a hub.*

The input plug on the left side of the COLOR block is where I can plug in a color keyboard to change the color inside the block. The output plug on the right can be connected to a color screen, but in truth, it can be connected to any *input* plug that can accept a color. Notice the CUP block has an input plug that will accept a color! So instead of connecting a color keyboard to the CUP block, I can use a simple wire to connect the output plug on the COLOR block to the input plug on the CUP block (see Figure 7-7).

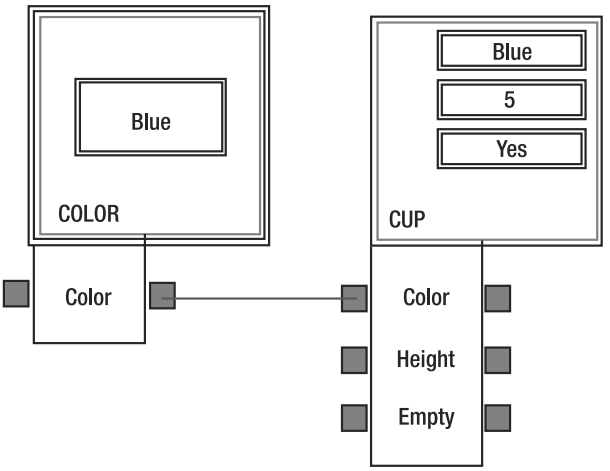

Figure 7-7. *I'll connect a wire from the COLOR block to the CUP block.*

I can also connect a height keyboard that can only be used to type in a cup's height in inches. If I try to type in anything besides a number, the keyboard won't work. I'll also connect a logic keyboard to the Empty input plug. A logic keyboard is a very special keyboard—it can only be used to provide Yes or No answers. Not Maybe or Sometimes; only Yes or No.

What I would like to do with the CUP block is to connect it to a screen that will display one of two things (but not both):

- Fill the [Color] cup with [Height] inches of water.
- The [Color] cup is not empty.

To do this, I can use a screen to print the color and height that are provided by the CUP block (Figure 7-8 shows my setup so far).

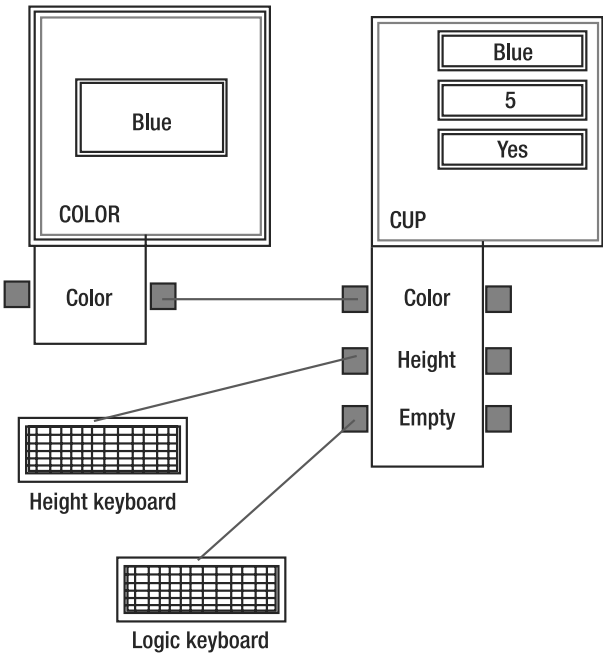

Figure 7-8. *Everything is hooked up and ready to use.*

But before I print the color and height, I need another special block that can examine the contents of the cup and determine if it is empty or not empty. Now, all I need to do is reveal my new `EXAMINE` block, shown in Figure 7-9.

The `EXAMINE` block can perform a nice trick. It takes a Yes or No answer (logic) and, depending on the answer, performs action 1 or action 2. Action 1 will occur if the answer is Yes; Action 2 will occur if the answer is No (for more information on logic and the `LOGIC` block, feel free to jump ahead to Chapter 8).

I can use this block to examine the contents of the `CUP` block. It will first look at the data plug labeled Empty. If the data the Empty data plug provides is Yes, the `EXAMINE` block will use the "EXAMINE = Yes" screen. If the data is No, the block will use the "EXAMINE = No" screen.

Figure 7-9 shows that when this program is run, the screen will display "Fill the Blue cup with 5 inches of water." It does this because the `EXAMINE` block receives the Yes data from the `CUP` block. It then performs the actions required for a Yes answer.

If I go back and change the Color to Yellow and the Height of the cup to 3 (using a color keyboard and height keyboard), this information will be passed from the `CUP` block to the `EXAMINE` block. If I change the logic answer from Yes to No using the logic keyboard, the `EXAMINE` block will receive the No data from the `CUP` block and perform the action required for a No answer: the screen displays "The Yellow cup is not empty."

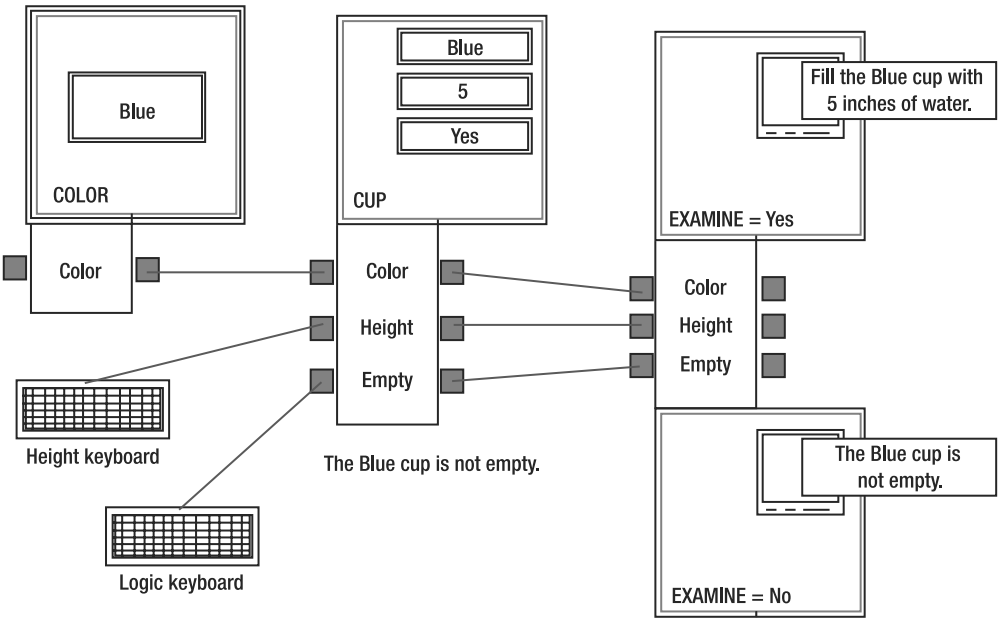

Figure 7-9. *The new EXAMINE block*

There are many more fake blocks that I could create, but I hope you're starting to understand how blocks can receive input data and provide output data. Both types of data (input and output) can be provided by you (by typing information in or selecting options in a configuration panel), or the data can be provided by other blocks using wires.

One thing you also need to know is that some blocks might not have any input plugs; some blocks might not have any output plugs, and a few, rare blocks have no data plugs at all.

You'll also be happy to hear this bit of information: I've created a bunch of fake blocks that can only accept color or height (and I could create a *bunch* of fake types of input), but you're fortunate, because when it comes to NXT-G, you only need to know about three types of data:

- *Text*: Letters, words, sentences, and even numbers can be considered text.
- *Number*: Numbers can be positive or negative, and sometimes they are limited to integers (only numbers like –3, 0, 4, 8, or 10 and no numbers with decimal points like 4.3 or 8.5).
- *Logic*: This can be Yes or No (another way to say it is True or False).

When you are programming, the only data that can be passed to and from a block are text, numbers, and logic types (Yes/No or True/False)—only these three! And just as your CUP block wouldn't let you use a logic keyboard to type in the color, a block's input and output data plugs will be very picky about the types of data they accept. You'll find an NXT-G block with plugs that accept Text and Logic, but no Number plug. Another NXT-G block will have no input plugs but maybe two output plugs that only provide Number data.

The good news is that if you ever drag a wire from one plug to an incompatible plug (if you try to drag a wire from a Text plug to a Logic plug, for example), the wire will be *broken*. By that,

I mean that the wire will become a dashed gray line indicating that you made a mistake. If you correctly connected a wire, the wire will have a color:

- The wire is yellow for the Number data type.
- The wire is orange for the Text data type.
- The wire is green for the Logic data type.
- If the wire is gray (and dashed), the wire is broken and will not work.

It takes practice to drag and connect wires from plug to plug. Sometimes, the wires will do strange things and go off in strange directions. You'll just have to play around with them until you figure out how to control them properly.

Well, now it's time to get back to real NXT-G blocks. You'll see in the figures I provide that many times I'll have a block's data hub opened. If you hover the mouse pointer over a data plug, it will show you the name of the data plug (something like Empty or Height in my examples).

For some of the plugs, it's fairly easy to figure out what type of data type they use (the Number plug requires the, duh, Number data type). Others aren't so easy to figure out. You can either check the help documentation, which provides a detailed description of a block's data hub plugs along with the types of data they accept *or* you can just experiment and drag some wires to it; the color of the wire will tell you if you're correct, or a gray wire will tell you to try again.

My last bit of good news is that you cannot ruin a program with incorrect wires! If you connect a wire that's incompatible, just click the input end of the wire (a wire always has an input end and an output end), and the wire will disappear. No worries!

This is a *lot* of information to absorb, and you've only scratched the surface of what wires can do for you. But there's *power* in wires! Wires can save you time by allowing you to use existing data over and over again; wires can be split, meaning you can split one wire and provide two different blocks with the same data! Wires can also go in the other direction, so you can send an output wire from the end of your program all the way back to an input plug at the start of your program! Keep your eyes open throughout this book to learn some new ways to use wires. Experiment on your own, and you'll discover even more uses for wires.

OK, up next in Chapter 8 is a short discussion on a method robots use for making decisions using Yes and No answers.