14

PROGRAMMING THE LEGO MINDSTORMS RCX

Advanced Methods

The LEGO Mindstorms Robotic Invention System has become a watershed for hobby robotics. The Mindstorms set allows both child and adult alike to experiment with robotics, without the usual requirements of constructing a frame and body or soldering electronic circuits. As such, Mindstorms provides a quick and simple introduction to robotics, and especially the programming behind it.

Yet the robots you can build with the Mindstorms go far beyond simple automated toys. There is a surprising amount of power under the yellow plastic of the Mindstorms robot module (the RCX). Much of this power is either not easily recognizable in the standard programming environment that comes on the Mindstorms CD-ROM or is not available, for whatever technical reasons.

Fortunately, you can tap the full potential of the Mindstorms robotics system by using alternative programming environments. This chapter discusses two popular alternative environments, including one that works with Microsoft Visual Basic.

Using Visual Basic to Program the RCX

Microsoft Visual Basic provides a convenient method for programming the LEGO Mindstorms RCX. You don't even need the full Visual Basic package (some $250 or more retail). You can also use any program that supports Visual Basic for Applications—such as Microsoft Word 97 or Corel WordPerfect 9 or later versions.

The text that follows will work equally well when using Visual Basic 5.0 or later or Visual Basic for Applications. However, some menu commands may be different between the two products as well as between different versions. For the sake of brevity, from here on we'll refer to Visual Basic as "VB" and Visual Basic for Applications as "VBA."

Note: by necessity, this chapter does not discuss programming with VB or VBA. It is assumed that you are already familiar with at least the basics of VB or VBA and that you know how to create and work with user forms and code modules. If VB and/or VBA are new to you, pick up a good introductory book on the subject at your library or neighborhood bookstore.

Before attempting to program in VB/VBA visit the main LEGO Mindstorms Web page (*www.mindstorms.com*) and look for the Software Developer's Kit page. Download the informational document on the "PBrick" programming syntax for the *spirit.ocx* ActiveX control. As of this writing, the document is available only in Adobe Acrobat format, so you will need a copy of the Adobe Acrobat reader to view the file. The Acrobat reader is available free at *www.adobe.com* and many other places; see the link on the LEGO Mindstorms page to download this software. You may also wish to download the sample VB program for later review.

When you use RCX with Visual Basic, you program the RCX by using a "middleware" component. This component is *spirit.ocx,* a Windows file that serves as an interface between the programming environment (VB or VBA) and the RCX itself. Spirit.ocx comes with the LEGO Mindstorms installation CD and is placed on your computer when you install the software. Keep this in mind: you will not be able to perform any programming until the Mindstorms software has been loaded. If you haven't done so already, use the standard Mindstorms programmer software to run the RCX through a couple of its basic paces. Once you know the RCX works with the standard programmer software you're ready to forge ahead with VB!

RUNNING THE TEST PROGRAM

To begin, start Visual Basic in the usual way. If you are using VBA, start the Visual Basic Editor (for example, in Word 97 and later you would choose *Tools, Macro, Visual Basic Editor*). Once you are in the Editor, follow these steps:

1. Create a new form by choosing *Insert, UserForm.*
2. Add the spirit.ocx component by choosing *Tools, Additional Controls.* Locate the "Spirit Control" and click on it to add a checkmark beside it. Choose OK to close the Additional Controls dialog box. (Note: this step need only be done once; thereafter the spirit.ocx control will be registered with VB/VBA for use in other projects.)
3. A new LEGO icon should appear in the Toolbox (choose *View, Toolbox,* if the Toolbox is not visible).
4. Click on the LEGO icon (this is the Spirit Control) and drag anywhere over the user form you created in step 1.
5. For ease of use, make the LEGO icon about a quarter-inch square and place it in the lower-right corner, as shown in Fig. 14.1.
6. Verify the proper settings of the Spirit Control by clicking on it and examining its properties in the Properties box. Specifically, check that the serial communications port is set properly (usually either *COM1* or *COM2*), that the LinkType is *Infrared* (assuming you're using the standard infrared tower that comes with the Mindstorms set), and that the PBrick type is RCX.

7. Change the name of the Spirit Control you've added to *PBrickCtrl.* (This step is optional; however, it conveniently conforms to the examples provided in the PBrick documentation provided by LEGO.)
8. Click on any blank area of the form, and change the name of the form (in the Properties box) to *RCXFrm.* While you can choose any name for the form, the sample programs that follow later in this chapter use the name RCXFrm to reference the PBrickCtrl control.

Adding the Spirit Control (spirit.ocx) component to the form allows you to write VB code so as to interface with the RCX. You are now ready to begin programming:

1. Create a new code module by choosing *Insert, Module.*
2. Type the BasicTest code shown below. Be on the lookout for typographical errors.

```
Sub BasicTest()
RCXFrm.PBrickCtrl.InitComm
RCXFrm.PBrickCtrl.PlaySystemSound (2)
End Sub
```

3. Verify that your RCX is on and that it is positioned no more than about a foot from the infrared tower.
4. In VB/VBA, run the BasicText program (choose *Run, Run Sub,* or press F5). (Note: you do not need to depress the Run button on the RCX in order to execute the BasicTest code.)

If all is working properly, the RCX should emit a short tone. If you get an error or the tone doesn't sound, recheck that the RCX is operating properly. Verify that the IR tower is functioning by verifying that the dim green light is on when the BasicTest program is being downloaded. This light will extinguish a few seconds after downloading is complete.

PROGRAMMING MOTOR ACTIONS

Sounding tones is hardly the life's work of the LEGO RCX unit, so let's try some more advanced programming techniques, including running two motors. For the following test, we'll assume that your RCX robot has two motors, attached to outputs A and C. Type the following

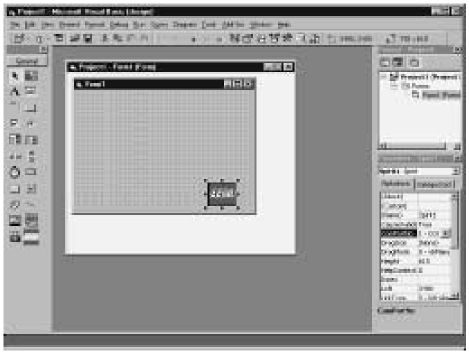

FIGURE 14.1 **The test form in Visual Basic, with the spirit.ocx control added.**

code, either in the same module in which you created BasicTest earlier or in a new module. Again, be watchful for typographical errors. We'll discuss what the code does in a bit.

Listing 14.1 Testmotors

```
Option Explicit

Public Const SWEEP_DOWN_SOUND = 2
Public Const SWEEP_UP_SOUND = 3
Public Const SWEEP_FAST_SOUND = 5

Sub TestMotors()
With RCXFrm.PBrickCtrl
 .InitComm
 .SelectPrgm 0
 .BeginOfTask 0
  .Wait 2, 30
  .SetPower "02", 2, 7
  .SetFwd "02"
  .On "02"
  .Wait 2, 50
  .SetRwd "02"
  .Wait 2, 50
  .Off "02"
  .PlaySystemSound SWEEP_FAST_SOUND
 .EndOfTask
End With
MsgBox "Download complete"
End Sub
```

Running the TestMotors program When you are done typing, run the TestMotors program in VB/VBA. A message box appears when downloading is complete. For this test, you must select program 1 on the RCX, using the *Pgm* button. Press *Run* when you're ready to run the program. The RCX should spin its motors forward and reverse for a short burst each way. When done, the RCX will emit its "up-sweep" tone to tell you it's finished.

Examining the TestMotors program The TestMotors program is actually straightforward. You may want to increase your understanding of what the program does by reviewing the PBrick documentation (described earlier in this chapter) from LEGO. Here is the first line of the program:

```
With RCXFrm.PBrickCtrl
```

The *With* statement is a standard VB/VBA command. It allows you to reference an object—in this case, RCXFrm.PBrickCtrl—using a shorthand syntax. Refer to the VB/VBA documentation for additional information on using *With*. All of the statements that follow, except for MsgBox, are commands built into the spirit.ocx component:

```
.InitComm
```

.InitComm (note the period prefix) sets up communications between the IR tower and the RCX. You must always include this statement before sending other commands to the RCX.

```
.SelectPrgm 0
.BeginOfTask 0
```

The *.SelectPrgm 0* statement selects program 1 in the RCX (e.g., press the *Pgm* button until program 1 appears in the RCX's LCD display). Much of the programming with spirit.ocx involves zero-based values, so *SelectPrgm 0* is program 1, *SelectPrgm 1* is program 2, and so forth. Recall that you can store up to five programs in the RCX at any one time.

As a point of reference, the 0 after the *.SelectPrgm* statement is known as a *parameter.* Many of the statements used to program the RCX with the spirit.ocx component require that you use of one or more parameters.

The *.BeginOfTask 0* statement tells the RCX that the code that follows is its main task. This functionality will occur when you press the *Run* button on the RCX. Each program can have up to 10 tasks. The RCX is designed to run each task simultaneously, which allows your programs to be multithreaded. For example, you might have your RCX play a tune while driving a zigzag course. Each of these actions is contained in its own task in one program.

```
.Wait 2, 30
```

The *.Wait* statement tells the RCX to wait a brief period of time. Wait uses two parameters: the *2* tells the RCX that the parameter that follows is a literal "constant". The *30* indicates 30/100s of a second, or about a third of a second. Other *Wait* statements in the remainder of the program perform a similar function.

```
.SetPower "02", 2, 7
.SetFwd "02"
.On "02"
```

These three statements set up the drive motors and turn them on. *.SetPower* sets the power to motors 0 and 2 (labeled A and C on the RCX) to full. The *2* indicates a literal constant, and the *7* indicates the value (1 is slow, 7 is fast, and there are several speeds in between). Similarly, *.SetFwd* sets the direction of motors 0 and 2, and *.On* turns them on.

```
.SetRwd "02"
```

Similar to *.SetFwd, .SetRwd* sets the direction of motors 0 and 2 in reverse.

```
.Off "02"
.PlaySystemSound SWEEP_FAST_SOUND
```

The *.Off* statement turns motors 0 and 2 off. The *.PlaySystemSound* statement, previously used in the BasicTest program earlier in this chapter, sounds a tone. Note the use of the *SWEEP_FAST_SOUND* constant variable, which is defined at the top of the program (a constant is a variable whose value does not change throughout the execution of the program). You can—and should—get into the habit of using constants instead of literal numeric values. It's a lot easier to see what *PlaySystemSound(SWEEP_FAST_SOUND)* means than *PlaySystemSound(5),* though both do exactly the same thing.

Consult the documentation for VB/VBA if you're new to the concept of using constants.

```
.EndOfTask
```

The *.EndOfTask* statement tells the RCX that the task begun earlier is now complete.

CLOSING THE COMMUNICATIONS PORT

In the program examples given in the previous section, the communications port, such as COM1 or COM2, is opened so the spirit.ocx component can send signals out of the Mindstorm's IR tower. This is done with the *.InitComm* statement. In each of the program examples we just examined, the communications port is left open. This is to simplify download timing, but in general it's not an advisable practice because it leaves the communications port opened, and therefore locked against use by other programs on your computer.

Use the *.CloseComm* statement to close the communications port. You can integrate this statement with your programs—for example, as the last command sent to the RCX. However, you must be careful not to close the communications port before downloading is complete or else your program will not function properly. One way to get around this is to use a waiting loop in VB; another is to use the *DownloadDone* event, which is a signal sent by the RCX when it has received all of its programming. The PBrick documentation from LEGO explains how to use the *DownloadDone* event.

Yet another approach is to specifically run a small program that closes the communications port. Here's all the code you really need for the job:

```
Sub CloseComm()
RCXFrm.PBrickCtrl.CloseComm
End Sub
```

GOING FURTHER

There are many more things you can do with the spirit.ocx component and VB/VBA, including reading sensor values (this is done with the Poll statement), adjusting the power output of the IR tower, even turning the RCX off remotely. Sadly, we don't have the room to delve into these subjects in more detail.

Fortunately, you can turn to the PBrick documentation provided by LEGO on the Mindstorms Web site (*www.mindstorms.com*) for additional information. Be sure to also check out the additional examples and resources on RCX programming at the support site for this book, *www.robotoid.com*.

Using Not Quite C (NQC) to Program the RCX

At last count, there were over a *dozen* programming alternatives for the LEGO Mindstorms RCX. One of the first, and still one of the most popular, is NQC—the letters stand for "Not Quite C." NQC is a stand-alone, text-based programming environment for the RCX. It is capable of performing the same basic programming functions as the Visual Basic approach, described earlier, but the programming language is radically different.

NQC is a freely distributed program available at *http://www.enteract.com/~dbaum/nqc/*. Versions of NQC are available for Windows-based systems, as well as the Apple Macintosh and Linux. Fetch the version you want, and place the NQC files in their own directory on your computer's hard disk drive. Assuming the IR tower is properly connected to your computer and the Mindstorms software has been previously installed, you're ready to go!

Note that the following steps assume you're using a Windows-based PC. Consult the documentation that comes with NQC if you are using a different computer.

CREATING A NQC PROGRAM

As its name suggests, Not Quite C uses a C-language syntax for programming. For those unfamiliar with C, the syntax can look daunting. However, with just a little bit of study, you'll find NQC is not difficult to use. If anything, it's often easier to interact with the RCX using NQC than with Visual Basic.

You may use any text editor to prepare an NQC file. In Windows, for example, you can use the Notepad program. You should store your NQC program files in the same directory as the *nqc.exe* program itself. Listing 14.2 shows a simple NQC program that does an amazing amount of computational work. Run this program on an RCX with two motors attached to the A and C outputs and with a light sensor connected to Input 1 and pointing forward. When you do, the RCX will seek out any bright light in the room. Aim a flashlight at the light sensor, for example, and the robot will come toward the light.

Figure 14.2 shows the prototypical RCX "rover" robot, set up for the sample programs in this section.

Listing 14.2 photophile.nqc.

```
#define LIGHT    SENSOR_1
#define MOTOR   OUT_A+OUT_C

task main()
{
      SetPower(MOTOR, 7);
      SetDirection (MOTOR, OUT_REV);
      SetSensorType(LIGHT, SENSOR_TYPE_LIGHT);
      while(true) {
            if(LIGHT > 60)
                   On(MOTOR);
            else
                   Off(MOTOR);
      }
}
```

If you key in this program in order to try it out, name it *photophile.nqc* (*photophile* means "lover of light"). Be on guard for typographical errors, and do not omit any of the brace characters! As with most C-based languages, capitalization is important. For example, *while* is correct, but not *While*.

EXAMINING THE NCQ PROGRAM

Let's take a closer look at this program. The first two lines:

```
#define LIGHT        SENSOR_1
#define MOTOR        OUT_A+OUT_C
```

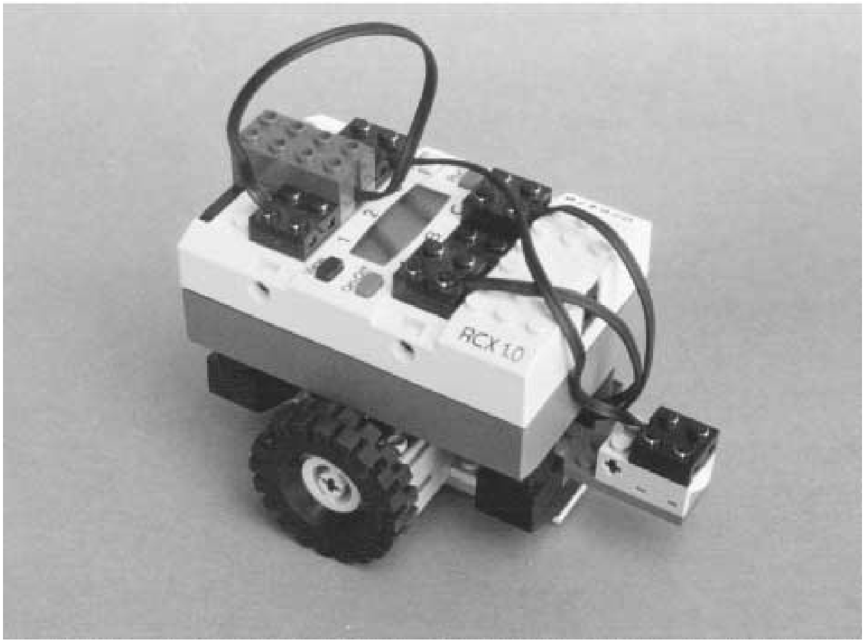

FIGURE 14.2 **The basic "rover" robot to be used with the programs in this chapter.**

These two statements define constants (unchanging variables) used elsewhere in the program. Constants are defined by using the *#define* keyword followed by the name of the constant and finally by the value of that constant. Note that the value of the constants *LIGHT* and *MOTOR* are themselves constants! In this case, the constants *SENSOR_1, OUT_A,* and *OUT_B* are built-in constants, with values already defined by NQC. We use our own constants to make working with the RCX even easier. See Table 14.1 for a list of the most commonly used built-in constants.

You will note that the *MOTOR* constant refers to two outputs, both A and C (*OUT_A OUT_C*). This allows us to operate both motors together, which will make it a little easier to command the robot to go forward or backward. The next two lines of the program are:

```
task main()
{
```

Each NQC program can have up to 10 tasks. Each task can be run simultaneously. The task that is run when you press the Run button on the RCX is called *main.* You can create your own names for other tasks you add in your program, but main always refers to the, well, "main" task that the RCX automatically runs.

Note the open brace (the "{" character) that follows the task *main()* statement. In NQC, as in C, multiple program statements are defined in *blocks* or *compound statements.* For

TABLE 14.1 NQC STANDARD CONSTANTS

CONSTANT NAME	FUNCTION	EQUIVALENT VALUE
SetSensor		
SENSOR_1	Input 1	0
SENSOR_2	Input 2	1
SENSOR_3	Input 3	2
SetSensorMode		
SENSOR_MODE_RAW	Raw value from sensor (0 to 1023)	hex 0x00
SENSOR_MODE_BOOL	Return Boolean (0 or 1) value	hex 0x20
SENSOR_MODE_EDGE	Count number of rising/ falling edges	hex 0x40
SENSOR_MODE_PULSE	Count number of pulses	hex 0x60
SENSOR_MODE_PERCENT	Show value as percentage	hex 0x80
SENSOR_MODE_CELSIUS	Temperature sensor Celsius reading	hex 0xa0
SENSOR_MODE_FAHRENHEIT	Temperature sensor Fahrenheit reading	hex 0xc0
SENSOR_MODE_ROTATION	Rotation encoder	hex 0xe0
SENSOR_TYPE_TOUCH	Pushbutton switch	1
SENSOR_TYPE_TEMPERATURE	Temperature sensor	2
SENSOR_TYPE_LIGHT	Powered light detector	3
SENSOR_TYPE_ROTATION	Rotation encoder	4
Outputs		
OUT_A	Select motor A	1 << 0
OUT_B	Select motor B	1 << 1
OUT_C	Select motor C	1 << 2
Output modes		
OUT_FLOAT	Let motors coast	0
OUT_OFF	Stop motors	hex 0x40
OUT_ON	Run motors	hex 0x80
Output directions		
OUT_REV	Motors in reverse	0
OUT_TOGGLE	Motors change direction	0x40
OUT_FWD	Motors forward	0x80

TABLE 14.1 NQC STANDARD CONSTANTS (*Continued*)

CONSTANT NAME	FUNCTION	EQUIVALENT VALUE
Output power levels		
OUT_LOW	Motors at low speed	0
OUT_HALF	Motors at medium speed	3
OUT_FULL	Motors at full speed	7
Sounds for PlaySound		
SOUND_CLICK	Short beep	0
SOUND_DOUBLE_BEEP	Two beeps	1
SOUND_DOWN	Tone scale down	2
SOUND_UP	Tone scale up	3
SOUND_LOW_BEEP	“Error” beep	4

each { character there must always be a } character, indicating the end of the block. You will use blocks in *if, while,* and other statements. The one thing you need to remember about blocks and the brace characters that define them, is this: always make sure you have a close brace for every open brace. The next two lines of the program are as follows:

```
SetPower(MOTOR, 7);
SetDirection (MOTOR, OUT_REV);
```

The *SetPower* statement sets the power to the motors. The *7* means full power; use *1* for low power or other values in between. *SetDirection* sets the direction of the outputs, in this case reverse. This will make the robot move toward the light.

```
SetSensorType(LIGHT, SENSOR_TYPE_LIGHT);
```

A single light sensor, connected to input 1, is used for the robot. The *SetSensorType* statement sets the input—specified here as *LIGHT*—to accept a powered light.

```
    while(true) {
       if(LIGHT > 60)
              On(MOTOR);
       else
              Off(MOTOR);
    }
```

The main body of the program is a *while* loop, which thanks to the *true* expression repeats the program until you depress the *Run* button on the RCX a second time or turn the RCX off. The important part of the program is the *if* statement, which reads (in “human” terms):

"if the output of the light sensor is greater than 60, turn the motors on;

otherwise

turn the motors off"

Light sensors on the RCX return a value of 0 to 100, with 0 being absolute darkness and 100 being fairly bright light. The value of 60 was selected as a kind of threshold. If you operate the RCX in dim room light, pointing a flashlight at the sensor will cause the motors to run. Turning the flashlight off will cause the motors to stop.

DOWNLOADING THE NQC PROGRAM

Now that the program has been written (and saved), you can download it to the RCX by using the main *nqc.exe* program. This program does two things: it compiles your text programs into a form that is suitable for the RCX, and it transfers the code to the RCX via the Mindstorms' IR tower.

NQC is a command-line program. To use it, open a new MS-DOS window by choosing *Start, Programs, MS-DOS Prompt.* (Note: if you don't have this option, choose *Start,* select *Run,* type *command.com,* and then press OK.) If necessary, switch to the NQC program directory using the *CD* (change directory) command, such as:

```
cd \nqc
```

This assumes that *nqc.exe* and your programs are in a directory named *NQC*. Then compile and download your program with the following command:

```
nqc -d program.nqc
```

where *program.nqc* is the actual program name you want to use. If you saved the sample program as *photophile.nqc,* for example, type the following:

```
nqc -d photophile.nqc
```

and press the Enter key. NQC will then compile the program and download it to the RCX. If there are syntax errors in your program, NQC will alert you of them and display the approximate line where the error occurs (usually the actual error is a line or two above). Assuming the program compiles correctly, NQC displays "Downloading program…" and then finally "Complete" when downloading is finished. Run the downloaded program by pressing the *Run* button on the RCX.

Note: Unless you tell it otherwise, NQC assumes that the IR tower is connected to COM1 of your computer. Use the set command in DOS to set a different communications port, such as:

```
set RCX_PORT=COM2
```

This tells NQC to use COM2 instead. Valid values are COM1, COM2, COM3, and COM4. If you type one of these in the DOS window you have opened for NQC, the value will remain only until you close the window. If you want to make the setting permanent, edit the *autoexec.bat* file (it's in the root of the C drive) and add the *set* command there. It will take effect the next time you start your computer.

ALTERING THE BEHAVIOR OF THE ROBOT

It's easy to alter the behavior of your NQC-controlled robot creations. One small change you can make is to have the motors turn the other way when the light shines on the sensor. This has the effect of creating a "photophobic" robot—a robot that appears to run away from the light. The complete code example is shown in Listing 14.3. If you retype this program, name it *photophobe.nqc*.

Listing 14.3 photophobe.nqc.

```
#define LIGHT        SENSOR_1
#define MOTOR        OUT_A+OUT_C

task main()
{
       SetPower(MOTOR, 7);
       SetDirection (MOTOR,OUT_FWD);
       SetSensorType(BUTTON, SENSOR_TYPE_LIGHT);
       while(true) {
               if(LIGHT > 60)
                      On(MOTOR);
               else
                      Off(MOTOR);
       }
}
```

CREATING A MULTITASKING CONTROL PROGRAM

One of the most important capabilities of the RCX is that it is a multitasking device. You can run up to 10 tasks "simultaneously" in one program. The microcontroller in the RCX divvies up a little bit of its processing time to each task, so in human terms things appear to happen simultaneously. Of course, in microcontroller terms it's handling one instruction at a time, but at very fast speeds.

The following program is a rudimentary but fully functional example of an RCX program with multiple concurrent tasks. The program is based on the *photophobe.ncq* example in the previous section. We have added separate tasks, one to play a little song (the first notes of something that sounds like "Mary Had a Little Lamb") and another to reverse the motors and spin if a touch sensor is activated.

When the program is run, the robot exhibits three events (sometimes called "behaviors" in modern robot artificial intelligence efforts):

Event 1. A song is played every few seconds. This event is free running, and no other event in the program affects it.

Event 2. When a strong enough light strikes the light sensor, the robot backs away from the light source (of course, "backs away" depends on how the motors and light sensor are mounted on the RCX, but you get the idea). The motors will continue to run as long as enough light strikes the sensor.

Event 3. When the touch sensor—mounted on the side of the RCX opposite the light sensor—is activated, Event 2 is suspended ("subsumed"). The robot reverses direction for a brief moment, then spins on its axis. Finally, it stops moving, and it is more than likely no longer facing in the same direction. At this time, Event 2 is reactivated so that the robot will "run away" from any light that shines into the light sensor.

See Listing 14.4 (let's call it *multitask.ncq*), which contains short comments that are indicated by the double slash ("//") characters. These comments serve to describe the main functionality of the program.

Listing 14.4 multitask.ncq.

```
// Constants definitions
#define LIGHT        SENSOR_1
#define SWITCH       SENSOR_2
#define MOTOR        OUT_A+OUT_C

// Main task; run when Run button is pressed on RCX
// starts all tasks
task main()
{
     start play_song;
     start run_from_light;
     start timed_backup;
}

// Task for running away from the light (same as photophobe.ncq,
// except that motors run a little slower)
task run_from_light()
{
     while (true) {
            SetPower(MOTOR, 3);
            SetDirection (MOTOR, OUT_FWD);
            SetSensorType(LIGHT, SENSOR_TYPE_LIGHT);
            if(LIGHT > 60)
                   On(MOTOR);
            else
                   Off(MOTOR);
     }
}

// Task for backing up and spinning in response to switch touch
task timed_backup()
{
     while (true) {
            SetPower(MOTOR, 3);
            SetSensor(SWITCH, SENSOR_TOUCH);
            if (SWITCH == 1) {
                  stop run_from_light;      // disallow run_from_light task
                   SetDirection (MOTOR, OUT_REV);
                   On(MOTOR);
                   Wait (50);
                   SetDirection (OUT_A, OUT_FWD);
                   Wait (150);
                   SetDirection (MOTOR, OUT_FWD);
                   Off(MOTOR);
                  start run_from_light;      // allow run_from_light task
```

```
        }
            }
}

// Task for playing a little tune
task play_song()
{
      while (true) {
               PlayTone(392,25);
               PlayTone(349,25);
               PlayTone(330,25);
               PlayTone(349,25);
               PlayTone(392,25);
               PlayTone(0,2);
               PlayTone(392,25);
               PlayTone(0,2);
               PlayTone(392,25);
               PlayTone(0,2);
               Wait (500);
      }
}
```

Feel free to experiment with the code for *multitask.ncq*. The only real caveat is that if you want a task to continue it should have its own loop. The *While* statement is one method for doing this, but NQC provides other looping statements you may wish to try. Also, remember that the RCX supports up to 10 tasks.

GOING FURTHER

Of course, there's far more to Not Quite C than we have discussed here. The NQC download includes complete documentation on its capabilities. For example, NQC supports a wide variety of programming statements, loops, variable assignments, conditional expressions, and more. With NQC you can develop highly sophisticated programs for the RCX robot, and with a surprisingly small amount of code. Look for additional NQC samples and resources at the support site for this book, *www.robotoid.com*.

From Here

To learn more about…	*Read*
Introduction to programming concepts	Chapter 7, "Programming Concepts—The Fundamentals"
Using LEGO parts to create custom robots	Chapter 12, "Build Custom LEGO-based Robots"
Using the LEGO Mindstorms Robotics Invention System	Chapter 13, "Creating Functionoids with the LEGO Mindstorms Robotics Invention System"
Computer control of robots	Part 5, Chapters. 28–34