The Programmable Brick Handbook

Epistemology and Learning Group
MIT Media Laboratory*

May 6, 1997

The Programmable Brick is a hand-held, battery-powered computer that can receive inputs from electronic sensors (including touch, light, and sound sensors) and operate LEGO motors. The "Brick" is designed for a variety of educational robotics uses, including mobile robot projects, data-taking applications, and "ubiquitous computing" applications (projects that embed computers in the world around us).

This document explains how to use the Programmable Brick. Included is an overview of the Brick, an introduction to common sensors, and a programming reference.

The Programmable Brick is the result of nearly a decade of work on hand-held children's computers done at the Epistemology and Learning Group at the MIT Media Laboratory, under the guidance of Seymour Papert. The Programmable Brick work builds on Papert's pioneering work with the Logo computer language, and subsequent work with the computer-controlled LEGO/Logo system.

The primary development team on the Programmable Brick project has included Fred Martin, Seymour Papert, Mitchel Resnick, Randy Sargent, and Brian Silverman. This document discusses a version of the Programmable Brick hardware that was designed mostly by Fred Martin, using a software system that was designed mostly by Brian Silverman. This manual was written by Fred Martin.

> *This manual is specific to the Programmable Brick Model 120 —see Figure 1 on page 11—running the September 26, 1995 release of Brick Logo software.*

*20 Ames Street Room E15–315, Cambridge, MA 02139. To reach the current members of the Programmable Brick design team, send e-mail to `pbrick-design@media.mit.edu`.

Overview

This manual is organized as several sections and a number of appendices. The main sections present the ideas and information you need to use the Programmable Brick:

- Section 1, *Introduction*, presents the concept of the Programmable Brick and some motivations behind its creation.
- Section 2, *The Programmable Brick*, discusses the Brick itself and explains its commonly used features.
- Section 3, *Motors*, explains the Brick Logo motor primitives.
- Section 4, *Sensors*, explains the uses of common electronic sensors and associated Brick Logo primitives.

The appendices provide additional helpful information:

- Appendix A, *Getting Started*, explains how to set up the Brick materials (both hardware and software).
- Appendix B, *MS-DOS Computer Information*, presents information specific to IBM-compatible personal computers.
- Appendix C, *Battery Maintenance*, explains how to take care of the Brick's internal rechargeable battery.
- Appendix D, *The Brick Interface/Charger Unit*, provides reference information about the functioning of this device.
- Appendix E, *Brick Logo Quick Reference*, is a handy synopsis of Brick Logo commands.
- Appendix F, *Error Messages*, explains common errors and their solutions.
- Appendix G, *Known Bugs*, lists known problems with the current Brick hardware and software.
- Appendix H, *Version History*, discusses update changes from previous versions of Brick hardware and software.

- Appendix I, *Bibliography*, presents a sampling of works that influenced the Programmable Brick project.
- Appendix J, *Suppliers*, lists suppliers for materials useful for Programmable Brick projects.

Contents

List of Figures

1 Introduction

This introduction presents a (very) brief history of the research leading to the development of the Programmable Brick, and a quick scenario illustrating its use.

1.1 History of the Brick

The Programmable Brick is the result of more than twenty-five years' of work in developing computer-rich, constructionist activities for children, which began with the creation of the Logo programming language in the late 1960's under the guidance of Seymour Papert.

The early work was done at MIT's Artificial Intelligence Laboratory. Hand-built mobile robots (called "floor turtles") were cabled to big mainframe computers, and children wrote Logo programs to control how these turtles moved about. The turtles carried marker pens, so childrens' Logo programs would cause the turtles to make drawings on paper taped down to the floor.

By the late 1970's, these turtles moved off of the floor and onto the computer screen. The "screen turtle," an iconic image of a turtle on the computer display, was a fast, cheap, and effective alternative to the electromechanical floor turtles. With the explosion of microcomputers in the 1980's, many children were introduced to computing by writing Logo programs for screen turtles.

In the mid 1980's, a collaboration began between Papert's research group, which had moved to the MIT Media Laboratory, and the LEGO Group of Denmark, makers of the ubiquitous children's toy. The result was a system that allowed childen's Logo programs to control the movement of their LEGO constructions, which could be equipped with little electric motors and sensors. In a sense, Logo was returning to its roots of being interconnected with physical things-in-the-world, but with an important new dimension. The "turtle" in a LEGO/Logo project became anything that a child could build with LEGO parts. The commercial LEGO/Logo system, now in its second generation, is in use in thousands of elementary and middle schools in the United States.[1]

The Programmable Brick extends the LEGO/Logo environment, allowing children to create robotic devices that are portable and/or mobile. Children are using the Programmable Brick to build robot vehicles, perform remote data-taking ex-

[1]The commercial version of the LEGO/Logo materials, marketed by LEGO Dacta, is known by its product name *LEGO Control Lab*.

periments, and create computationally active environments, in the spirit of the recent “ubiquitous computing” work of Xerox PARC.

1.2 Using the Brick

Working with the Programmable Brick is a lot like building with the commercial LEGO/Logo systems. During project development, the Brick may be hooked up to a desktop computer, and users can control their LEGO motors directly by typing commands on the keyboard. Logo programs may be written, downloaded, and run from the keyboard.

The difference is that because users’ programs are actually downloaded to the Brick, it may be detached from the desktop computer (the link is a simple telephone-wire-style serial connection), and any programs that have been downloaded can be executed without the benefit of the desktop system. The Brick has its own little display screen as well as a knob and two buttons, so that different programs can be started and stopped, all when the Brick is away from the desktop host machine.

Since the Brick is small, portable, and battery-powered, a new strain of LEGO/Logo projects are possible, including mobile robots that don’t have to carry an awkward tether, and remote data-collection projects can be set up for an extended period of time.

2 The Programmable Brick

This section introduces the Programmable Brick, explaining its various inputs, outputs, and connectors. Appendix A of this guide demonstrates how to hook the Brick up to a desktop computer and operate the Brick Logo software system.

Figure 1 is a photograph of the Programmable Brick, showing the connectors, buttons, and other interface objects on the Brick. In the discussion that follows, please refer back to this diagram to relate the features being presented.

2.1 Motor Outputs

The Programmable Brick can control four motors, driving them in either direction and at user-controllable levels of power. Motors are connected to the Brick using 2×2 square LEGO connectors, the LEGO 9 volt connector system. To use a motor, simply plug it into a Brick motor output.

The four motor connectors are located along the lower edge of the Brick. Motor A is on the left, and Motor D is on the right. Above each motor connector, a pair of LEDs (light emitting diodes) indicates the motor output's state. The green LED indicates the motor output is on, turning a motor in one direction, and the red LED indicates it is turning in the other direction. Depending on the orientation of the motor connector, a motor may run clockwise or counterclockwise when initially turned on.

The Programmable Brick was designed to be used with 9 volt LEGO motors. Other hobby motors, even small ones, may overload the Brick's circuitry. This is because the 9v LEGO motors were specially designed to draw relatively small amounts of electrical current, while the average toy motor is not designed this way.

The motor outputs may also be used to control 9v LEGO lamps and beepers, as well as flashlight bulbs. When connecting flashlight bulbs, make sure to use bulbs rated around 7.5 volts (bulbs rated for lower voltages may burn out, while bulbs rated for higher voltages will be too dim). Radio Shack part number 272–1133 is an ideal bulb.

2.2 Sensor Inputs

The Programmable Brick can support up to six sensors at once. There are three each of two different types of sensor connectors. Sensors A through C are for 9v

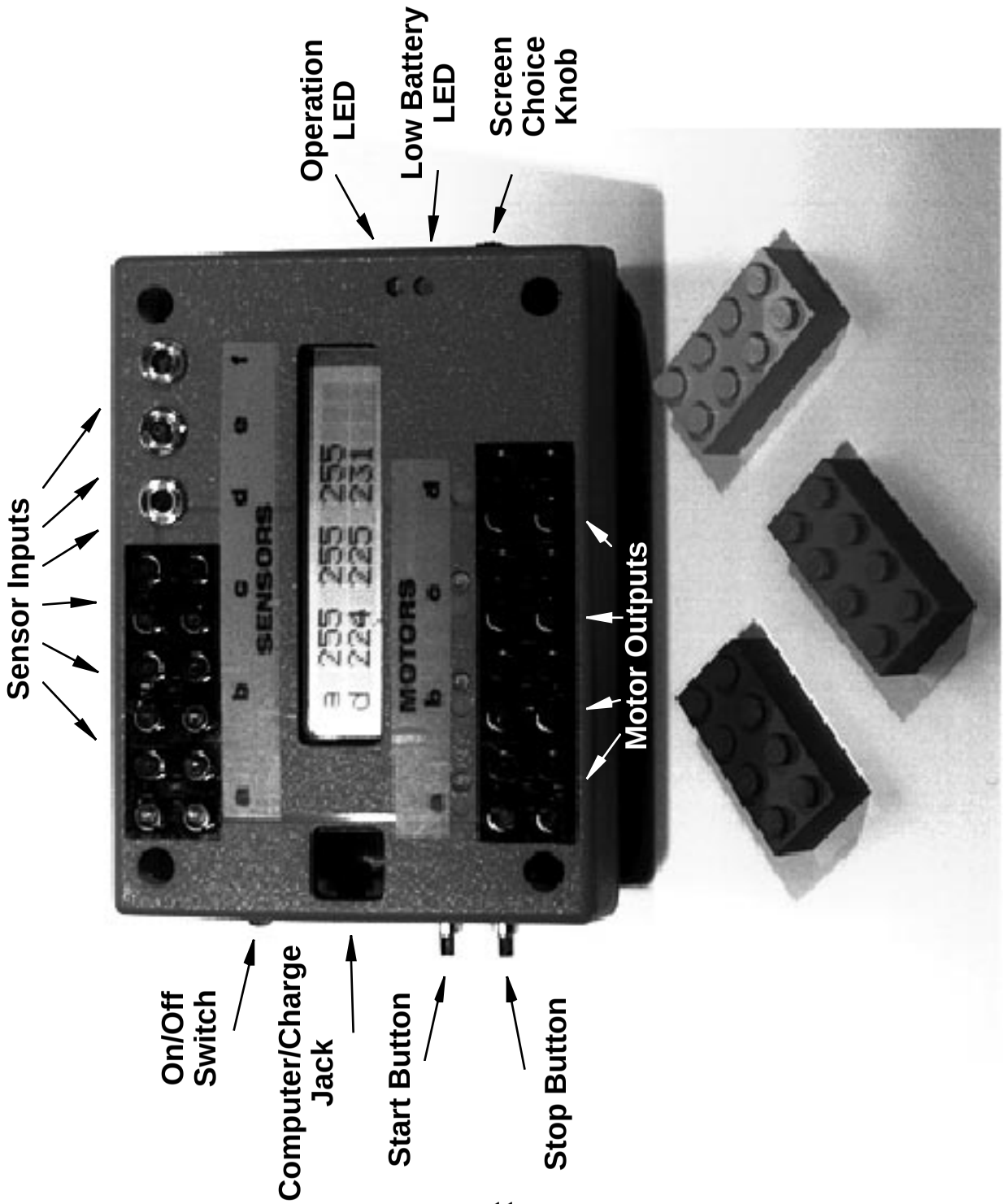

Figure 1: Photograph of the Programmable Brick

LEGO sensors (which include touch sensors, reflected light sensors, and rotation sensors), and sensors D through F are for custom-made sensors.

Following is a brief description of the sensors; these sensors are described in detail in Section 4, *Sensors*.

2.2.1 LEGO Sensor Inputs

Connectors for sensors A, B, and C, located along the top edge of the Programmable Brick, can be used to connect any of the 9v LEGO sensors used with the LEGO Control Lab product.

Touch sensor. The touch sensor is a switch that can detect contact (when the switch nub is pressed in).

Reflected light sensor. The reflectance sensor has a light emitter and a light detecter. It measures how much light from its own light source is reflected back into its light detector.

Angle sensor. The angle sensor keeps track of the rotary movements of a LEGO axle that is mounted through the sensor.

Temperature probe. The temperature sensor reports a value that corresponds to the temperature at the tip of the sensor.

Each of the LEGO sensors has a square 2×2 LEGO connector just like the motor cables. To attach the sensor, simply plug the connector onto the metal studs on the Programmable Brick. Orientation is not important.

> *Note to users of LEGO Control Lab: In the LEGO Control Lab product, a distinction is made between "active" sensors (which have a blue plug) and "passive" sensors (which have a yellow plug). On the Control Lab interface, there are separate pads for connecting active and passive sensors.*
>
> *The Programmable Brick does not make such a distinction. Either type of sensor can plug into any of the three LEGO-compatible sensor inputs.*

2.2.2 Mini-Plug Sensors

In addition to the three LEGO-compatible sensor inputs, there are three sensor inputs designed for custom-made sensors. These sensors use the "stereo mini-plug" connector (the type of connector commonly found on Sony Walkman-style headphones).

Several custom-made sensors are available to work with the Programmable Brick. These are described in Section 4.

Sound sensor. The sound sensor can be used to detect loud noises.

Bend sensor. The bend sensor is like a touch sensor, except that it can detect a range of contact forces, not simply on and off.

Ambient light sensor. The ambient light sensor is similar to the reflectance sensor in that it detects light, but it does not have its own light source, and is better for detecting room (i.e., ambient) lighting.

2.3 User Input and Output

There are two buttons, a switch, a knob, and a display on the Brick. Here is a description of their function.

2.3.1 Power Switch

The Brick's power switch is located near the upper left corner of the Brick in photograph of Figure 1. To turn the Brick on, flip the switch to the position labelled "On."

The Brick should be turned off when not in use. The Brick's memory is "non-volatile," meaning that the Brick remembers its program and any data it has recorded when it is turned off. Even if the battery runs down so low that the Brick can't be turned on, the memory will not lose data. When the Brick is charged up again, the data and programs will be present in the Brick's memory.

The Brick has an internal rechargeable battery that will not need replacement over years of normal use. Battery life, charging, and other details related to battery management are discussed in Appendix C, *Battery Maintenance*.

2.3.2 Brick Display

The Brick has a thirty-two character LCD (liquid crystal) display, organized as two rows of sixteen characters each. User programs may print messages to this display; also, the Brick maintains a "menu" of user programs loaded onto the Brick that may be selected at any time.

The Brick can also show continous sensor readings on the display. In the Brick photograph (Figure 1), it is possible to see the sensor readout, showing the six sensor values.

2.3.3 Buttons and Knob

The Brick has two buttons and one knob for interacting with the Brick's display and selecting any programs that may be loaded.

The knob, located near the lower right corner on the side edge of the brick, is used to scroll through a list of choices, which are shown on the Brick display. At any given time, only one choice is displayed on the Brick's screen; this choice is changed by rotating the knob.

To execute the currently-displayed choice, press the button labelled "START." An asterisk will be displayed in the lower right corner of the screen, indicating that the program is running.

To stop a single program, scroll to it on the menu, and press the START button.

To stop all programs that may be running, and turn all of the motors off, press the button labelled "STOP."

Brick Logo primitives for using the buttons and knob are explained in Appendix E, which discusses the Brick's software system.

2.3.4 Status LEDs

The Brick has a number of status LEDs (light emitting diodes).

Each of the motor outputs has a pair of LEDs that indicate the status of the associated motor. The green LED indicates the motor is on one way, and the red LED indicates the motor is on the other way.

There are two additional status LEDs. The green LED labelled "READY" indicates that the Brick is turned on and is operating normally. If the "READY" LED is not on, or is blinking on and off, there is a problem with the Brick. Please refer to Section A.4 for information on how to restart a Brick that has "crashed."

The red LED labelled "LOW BATT", for "low battery," indicates that the Brick's battery is low and should be recharged. Often, however, when the Brick's battery runs down, it doesn't have enough power to even light the low battery indicator. So if a Brick is turned on, and neither the "READY" nor the "LOW BATT" LEDs are lit, it usually means that the battery is completely discharged. Turn the Brick off, and plug it in to begin recharging it.

2.4 Infrared Control

The Brick has an infrared sensor (located on its right edge) that is used to decode signals from household TV and VCR remotes. The remotes may be used to send numeric instructions to the Brick, causing it to execute one of several pre-loaded programs, for example.

The Brick interprets the infrared codes transmitted by Sony brand remotes. This allows it to work with any original-equipment Sony remotes, or universal remotes set to control Sony television and VCR products.

The Brick's infrared sensor has an omni-directional performance characteristic, meaning that it is able to see the infrared remote's signal over a wide range of angles. If the Brick is being used in a room with white ceilings, it is often possible to bounce the infrared signal off of the ceiling and into the Brick's sensor.

2.5 Computer/Charge Connector

The Computer/Charge Connector is located between the power switch and the start and stop buttons. It looks just like a modular phone jack, and, indeed it is a modular phone jack. The Brick, however, should **never** be plugged into an active telephone circuit! Permanent and serious damage to the Brick is likely to result.[2]

The Brick is connected to the Interface and Charger Unit via this connector. When the Brick is plugged in, it both (a) communicates with its host desktop computer for downloading new programs or uploading data that the Brick has recorded, and (b) recharges the Brick's battery.

[2]Telephone-style wiring and plugs are commonly used for computer networking applications for their convenience, low cost, and performance. Despite the fact that computer modems plug into the telephone line, many of these networking products, including the Programmable Brick, are not meant to be connected to telephone circuits.

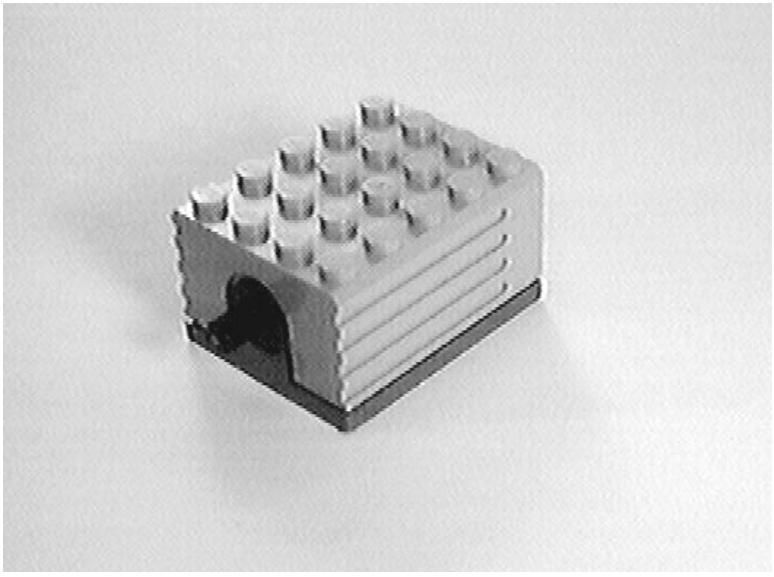

Figure 2: The Nine Volt LEGO Motor

3 Motors

The Programmable Brick was designed to be used with nine volt LEGO motors. Figure 2 shows the motor alone, and Figure 3 shows the motor attached to its special LEGO connector cable. One end of the cable plugs underneath the motor, attaching to the metal studs in the middle, and the other end connects to one of the Brick's motor outputs.

The Brick can control up to four motors, which are referred to as motor A through motor D. Motors A, B, and C are capable of bi-directional motor control—they each can drive a motor forward or backward under software control, while Motor D can only turn a motor on and off.

The rest of this section explains Brick Logo primitives for operating the motors.

3.1 Turning Motors On and Off

To control a motor, first specify the motor or motors to be controlled, and then give the command. For example, the sequence

```
a, on
```

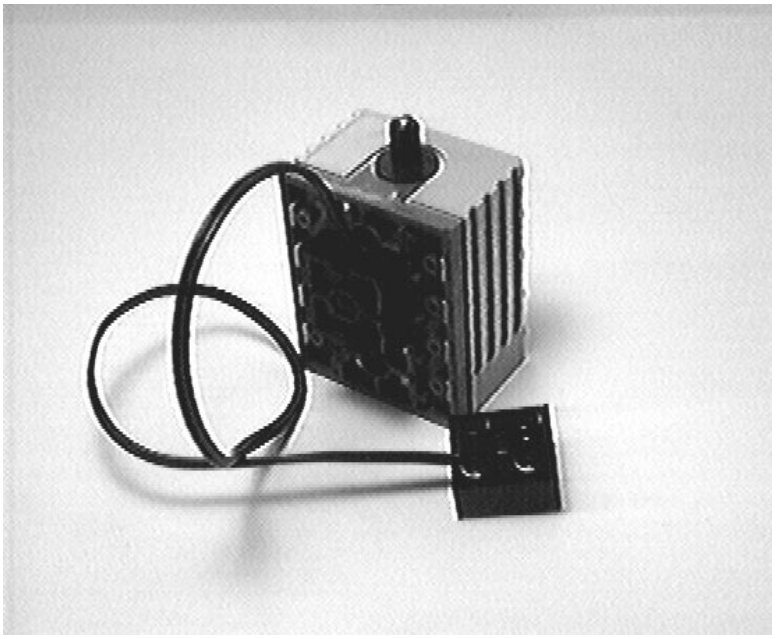

Figure 3: LEGO Motor and Motor Cable

selects motor A and then turns it on. Notice the comma after the letter “a”; this syntax is meant to suggest the common English imperative form, as in “Francis, come here.”

The command “off” is used to turn motors off:

```
a, off
```

In addition, the command “toggle” may be used to invert the on/off state of the motors: motors that are on are turned off, and motors that are off are turned on. For example:

```
a, on b, off ab, toggle
```

This sequence turns motor A on and motor B off, and then toggles them both; motor A is turned off while motor B is turned on.

3.2 Selecting Multiple Motors

There are a couple of ways to turn on multiple motors. For example, to turn on motors A and B, either

```
a, on b, on
```

or

```
ab, on
```

will work. Many, but not all, multiple-motor selections are allowed. The combinations allowed are:

```
ab, bc, ac, abc, abcd,
```

3.3 Timed Motor Commands

The command “onfor” is used to turn a motor on for a particular period of time. `onfor` takes as input the amount of time, which is specified in tenths of seconds. Thus,

```
a, onfor 10
```

turns motor A on for one second.

To wait for a period of time and leave a motor off, use the “wait” command. For example, the sequence

```
a, onfor 10 wait 10 onfor 10
```

turns motor A on for one second, waits a second, and then again turns it on for a second.

3.4 Changing Motor Direction

The command “rd,” for *r*everse *d*irection, makes a motor spin in the opposite direction. For example, the sequence

```
a, onfor 10 rd onfor 10
```

turns motor A on for one second, reverses its direction, and then turns it on for another second.

In addition to the `rd` command, the commands “thisway” and “thatway” may be used to set motor direction. The `thisway` command sets the direction to the

one in which the green motor LEDs are illuminated, and the `thatway` commands sets the opposite direction, in which the red LEDs are illuminated.

The difference between using `rd` and `thisway` or `thatway` is that the first reverses the current direction, while the latter two set the direction state to a known value.

Note that when a motor is turned on, the direction that it actually spins depends on the orientation of its two wire connectors—the one plugged into the Brick and the one plugged into the motor—in addition to the motor direction selected by the Brick. If either of the connectors is reversed, the motor's spin will also reverse.

3.5 Setting Motor Power Level

Motors can be driven at nine degrees of power from off to fully on. The command to do this is "setpower," which takes an input determining the power level. The power levels range from 0, which is off, to 8, which is fully on. Motors begin in the full-power state, and, as with all motor commands, `setpower` only affects the motor(s) currently selected.

For example,

```
ab, setpower 6
```

sets Motors A and B to power level 6.

The power control works by rapidly switching motors on and off, with a duty cycle proportional to the power level. For example, at power level 5, motors are turned on for five phases and off for three. These phases are typically interleaved, so power 5 might look like *on-on-off-on-on-off-on-off*. (This technique is known as "pulse width modulation.")

Actual power levels are not strictly proportional, however. Power 7, the step just beneath the full power 8, provides less power than the ratio of $\frac{7}{8}$ would suggest. This is due to an electrical effect in which the motors are actively braked during the off phase of the duty cycle, rather than simply being left to coast.

When motor output are used to control incandescent (flashlight) lamps, power levels are fairly proportional.

4 Sensors

There are two fundamental kinds of sensors: *switch sensors*, which provide on/off-type of readings (for example, the LEGO touch switch), and *continuous sensors*, which provide continuous levels of reading (for example, a light sensor).

The LEGO angle sensor is a special case of a continuous sensor, in which the sensor's electrical signals are converted into a count of number of rotations. There are special Brick Logo commands for using this sensor.

There are two connector styles for attaching sensors to the Brick:

LEGO Connector LEGO sensors plug onto connectors just like the LEGO motor connectors.

Mini Plug Connector Custom-made sensors plug into round stereo minijack connectors.

There are three of each type of sensor connector. The sensors are named with letters A through F from left to right along the top edge of the Brick. (On some Bricks, the sensors are incorrectly labelled with the numerals 6 through 1 from left to right.)

The Brick has two other special sensor functions. An internal timer keeps track of elapsed time with a precision of one thousandth of a second ($\frac{1}{1000}$ sec). Also, the Brick can determine its remaining battery level, reported as a percentage of full charge.

The remainder of this section introduces standard sensors to be used with the Programmable Brick and the Brick Logo primitives for using them.

4.1 The LEGO Touch Sensor

The LEGO touch sensor is shown in Figure 4. To determine the state of the sensor, the "switch" primitive is used. The primitive reports a true or false value depending on whether the switch is pressed.

There are three variants of the switch command, depending on which sensor port is being tested: `switcha`, `switchb`, and `switchc`, for a touch sensor plugged into port A, B, or C, respectively.

The switch primitive reports true if the switch is pressed and false if it is not. It is typically used with an `if`, `waituntil`, or `when` command. For example, the statement

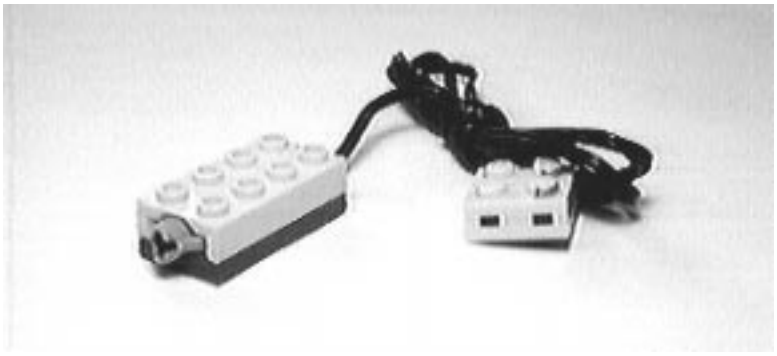

Figure 4: The LEGO Touch Sensor

```
if switcha [a, onfor 20]
```

causes motor A to turn on for two seconds if the touch switch plugged into sensor A is pressed. (Note that this statement must be in a loop in order for the switch to be repeatedly tested.) In a similar fashion, the statement

```
waituntil [switcha]
```

causes the computer to wait until the sensor A switch is pressed.

The switch command can be used in conjunction with the "not" primitive. Thus,

```
waituntil [not switcha]
```

waits until the sensor A switch is *not* pressed—that is, until it is released. A standard method for debouncing a switch press is to wait for it to be pressed, and then wait for it to be released:

```
waituntil [switcha]
waituntil [not switcha]
```

4.2 Continuous Sensors

Continuous sensors provide readings that indicate a range of values. For example, a light sensor reports a number that indicates the amount of light being detected, or a temperature sensor reports a number that indicates the amount of warmth being detected.

The "sensor" primitive is used with any of the continuous-level sensors, such as the LEGO reflectance sensor, the LEGO temperature sensor, and the custom-made bend and light sensors. The primitive reports a value from 0 to 255 depending on the property being detected, though the actual range obtained is characteristic of the particular sensor.

The sensor primitive has six forms, corresponding to the six sensor ports: `sensora`, `sensorb`, `sensorc`, `sensord`, `sensore`, and `sensorf`.

When using continuous sensors, it is common to test the value of the sensor to determine if it is above or below a certain threshold. For example, suppose the reflectance sensor reports a value near 190 when pointed at a dark surface, and a value near 170 when pointed at a light surface. Then a reasonable test for the dark surface would be, "Does the sensor report a value greater than 180?" Translated into a Brick Logo statement, this would look like:

```
ifelse sensorc > 180 [print [Dark surface!]]  [print
[Light surface!]]
```

This statement would print "`Dark surface!`" on the Brick's display if the port C sensor reported a value greater than 180; otherwise, the message "`Light surface!`" would be displayed. In a real program, some action would probably be taken based on the change in sensor value (instead of or addition to the print statement).

4.2.1 The LEGO Reflectance Sensor

The LEGO reflectance sensor is primarily used to measure the reflectivity of a surface. It performs this measurement with two particular electrical components. One emits a beam of red light, and the other detects how much light is received. When aimed at a surface, the light from the emitter is reflected into the detector. Bright surfaces reflect a lot of light while dark surfaces reflect less light (assuming a constant distance from the reflection surface to the sensor, and constant room lighting).

The sensor works best at distances between one-eighth and one-half of an inch from the reflecting surface. It is also helpful to shield the sensor from ambient room lighting. Also, since the emitter shines red light, the device actually measures reflectivity to red light, which may or may not correlate to one's visual impression of the general reflectivity of a surface.

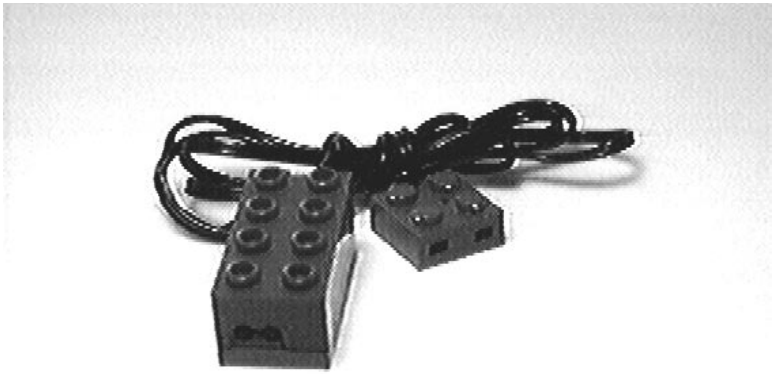

Figure 5: The LEGO Reflectance Sensor

The reflectance sensor can also be used to detect ambient light (i.e., room lighting), simply by aiming it toward open space. Its detector, however, has a lens that gives it a fairly narrow beam of detection. Depending on one's intended application, this may be a desirable or undesirable property.

The LEGO reflectance sensor plugs into sensor ports A, B, or C, and hence is used with the sensor primitives `sensora`, `sensorb`, and `sensorc`. Typical reflectance readings range from 170 to 190, though lower values may be detected if ambient light is shining directly into the sensor.

4.2.2 The LEGO Temperature Sensor

The LEGO temperature sensor measures heat. The metal tip of the sensor is the heat-sensitive component. The tip may be submerged in water.

The values reported by the sensor have an inverse relationship to standard temperature measurements: higher values indicate lower temperatures. Also, the sensor's readings are not linear with respect to standard temperature scales. A useful experiment would be to measure the correlation between the values reported by the sensor and a standard temperature scale.

The LEGO temperature sensor plugs into sensor ports A, B, or C, and is used with the sensor primitives `sensora`, `sensorb`, and `sensorc`.

4.2.3 The Bend Sensor

The bend sensor is custom-made sensor that measures the amount of bending in a flexible plastic strip. It was designed for measuring the amount of flex in a person's finger for the Nintendo Power Glove, but has many robotic applications.

The sensor reports increasing values for increasing amounts of flex. It is sensitive only when bent in one of the two possible ways of flexing from the flat resting position.

Bend sensors plug into sensor ports D, E, or F and are used with the sensor primitives `sensord`, `sensore`, and `sensorf`.

4.2.4 The Photocell Light Sensor

The photocell light sensor is a custom-made sensor used for measuring ambient light. Unlike the LEGO reflectance sensor, it does not have a lens on its detection element, so it is sensitive to general, undirected room lighting.

Depending on how a given sensor was wired, it may yield increasing or decreasing values with increasing amounts of light, so it is best to experiment with each particular sensor device to determine its characteristic.

Photocell light sensors plug into sensor ports D, E, or F and are used with the sensor primitives `sensord`, `sensore`, and `sensorf`.

4.2.5 The Sound Sensor

The sound sensor is a custom-made sensor that consists of an integrated microphone/amplifier assembly. It reports sound level as a waveform centered around a value of about 147. Sounds will create peaks above this value and troughs below it; stronger sounds will create higher peaks and lower troughs.

A simple way to use the sensor is as a detector for loud sounds by looking for a high peak (or a low trough). For example, the statement

```
when [sensord > 160] [note 80 5]
```

sets up a process to repeatedly check the value of sensor D and make a short "beep" when it rises above 160. Lowering the threshold point of 160 will make the sensor respond to quieter sounds.

It is also possible to sample a short snippet of sound data and record the waveform. For example, the procedure

```
to sample
  erase
  repeat 5000 [record sensord]
end
```

quickly samples 5000 points of sensor data, which then can be downloaded to a host computer for processing. The Brick records data in this manner at a rate of about 11,000 samples per second. (The Brick has a limited data buffer; see Appendix E.6 for information about using the Brick's data-taking primitives.)

The sound sensor plugs into sensor ports D, E, and F, and hence is used with the sensor primitives `sensord`, `sensore`, and `sensorf`.

4.3 The LEGO Angle Sensor

The LEGO Angle Sensor is used to measure the rotation of an axle that is inserted through the sensor. Each revolution of the axle yields a measurement of sixteen counts. Turned in one direction, the sensor counts up; turned in the other direction, it counts down. This allows faithful readings even when the shaft changes direction.

The "counter" command is used to report the number of rotations; the "resetc" (reset counter) command is used to set the corresponding counter to zero. There are three forms of each of the two commands, corresponding to the three LEGO sensor ports: `countera`, `counterb`, and `counterc` to report values for each of three sensors, and `resetca`, `resetcb`, and `resetcc` to reset these counters to zero.

It is possible to determine angular velocity by taking differences in the positional count at regular intervals. For example, the `global` command can be used to set up global variables to keep track of the velocity of counter A, the last count reading, and a temporary variable to be used in the velocity calculation:

```
global [velocitya lastcounta tempa]
```

Then the `every` primitive can be used to take a velocity reading at a regular interval; for example, every second:

```
every 10 [settempa countera
          setvelocitya tempa - lastcounta
          setlastcounta tempa]
```

Every second, `tempa`, a temporary variable, is set to be the current count. Then, the current velocity reading is determined: the difference between the current count (`tempa`) and the previous count (`lastcounta`). Finally, the `lastcounta` variable is set to the current count for the next iteration.

In this example, the velocity readings are calculated every second, but the best length of this interval should be determined based on how rapidly the count advances in the desired application.

The Brick takes sensor readings 300 times per second. The axle should not turn faster than about 18 times per second in order that the Brick does not lose track of the counter transitions (300 counts/second divided by 16 counts/revolution is 18.75 rotations/second).

4.4 Timing Sensor

The Brick has a timing function that keeps track of elapsed time. The command "timer" reports the elapsed time in milliseconds (thousandths of a second). The command "resett" resets the timer value to zero.

These commands may be demonstrated with the following code sample. Try running this sequence from a Brick screen slot:

```
resett wait 50 print timer
```

What happens? First, the `resett` command resets the timer to zero. Then the Brick waits for five seconds (the `wait 50` command). Next the Brick prints the current elapsed time on the screen. What value is shown?[3]

The Brick's internal crystal, which operates the Brick microprocessor and thereby controls the timing function, is accurate to a few parts per million. Therefore the Brick's timer can be considered a fairly accurate source of time data.

4.5 Battery Level Sensor

The "battery" primitive reports the Brick's battery level as a percentage of full charge. The following statement demonstrates usage:

```
type [Level is] type battery print [%]
```

[3] The Brick should display 5000—the number of milliseconds of time that elapses in the `wait 50` statement.

The level reported by the `battery` primitive is not valid when the Brick is being charged by the Interface/Charger Unit. When charging, it would typically report a 100% charge, unless the actual battery level is very low. Therefore, to perform this test, it's a good idea to put the above statement into a Brick screen item slot (see Appendix A.3) and run it when the Brick is disconnected from its charger.

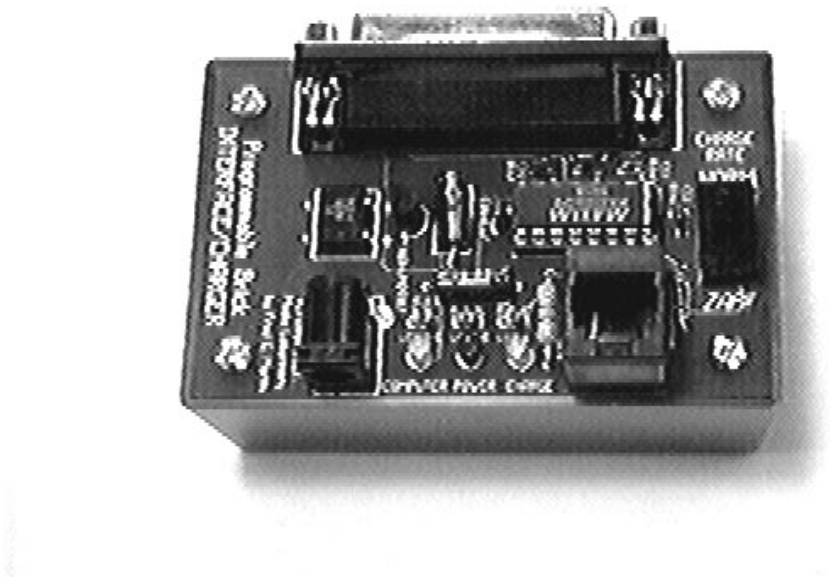

Figure 6: The Brick Interface/Charger Unit

A Getting Started

This section provides step-by-step instructions for setting up the Brick's hardware and software. The order of activities will be: first setting up the hardware, then installing the software, and then running the software.

> *Note: The diagrams and instructions that follow assume that you will be using a Macintosh computer with the Brick. If you are using an MS-DOS IBM compatible computer, please refer to the additional instructions in Appendix B,* MS-DOS Computer Information*.*

A.1 Setting up the Brick Hardware

The first step is to locate the five components that are needed to set up the Brick system:

1. *The Brick itself.*

2. *The Interface/Charger Unit, or I/C Unit.* The I/C Unit is depicted in Figure 6. It is smaller than the Programmable Brick, quite light in weight, and contained in a dark blue or black plastic case.

3. *The DC power adapter.* The power adapter is used to provide power to the I/C Unit (both for its own electronics and to charge the Brick).

4. *The Macintosh modem cable.* Shown in Figure 8, the modem cable is used to connect the I/C unit to the Macintosh.

5. *The telephone cable.* The Brick is connected to the I/C Unit using a standard telephone extension cable.

Figure 7 illustrates the method for setting up the Programmable Brick system hardware. Here are step-by-step instructions for setting up the system as indicated in the diagram.

1. Locate the Macintosh modem cable (depicted in Figure 8), and plug the small round end into the Macintosh's modem port. The modem port has a little picture of a telephone handset above it, like this:

2. Locate the Interface/Charger Unit. Plug the other end of the modem cable—a large, D-shaped connector—into the corresponding jack on the I/C Unit.

3. Locate the DC wall adapter and unroll its cable. Connect the round plug to the I/C unit; then plug the AC prongs into a household outlet. At this point, the red LED labelled "POWER" on the I/C Unit should light.

   If the red LED does not light, check that the adapter is indeed plugged into the household wall outlet. If it still doesn't light, try another outlet. If it still doesn't light, there is a problem with either the DC adapter or the Interface/Charger Unit itself; you will need to contact us for technical support.

4. Locate the telephone cable. Plug one end of the cable to the jack on the I/C Unit, and the other end into the jack on the Programmable Brick.

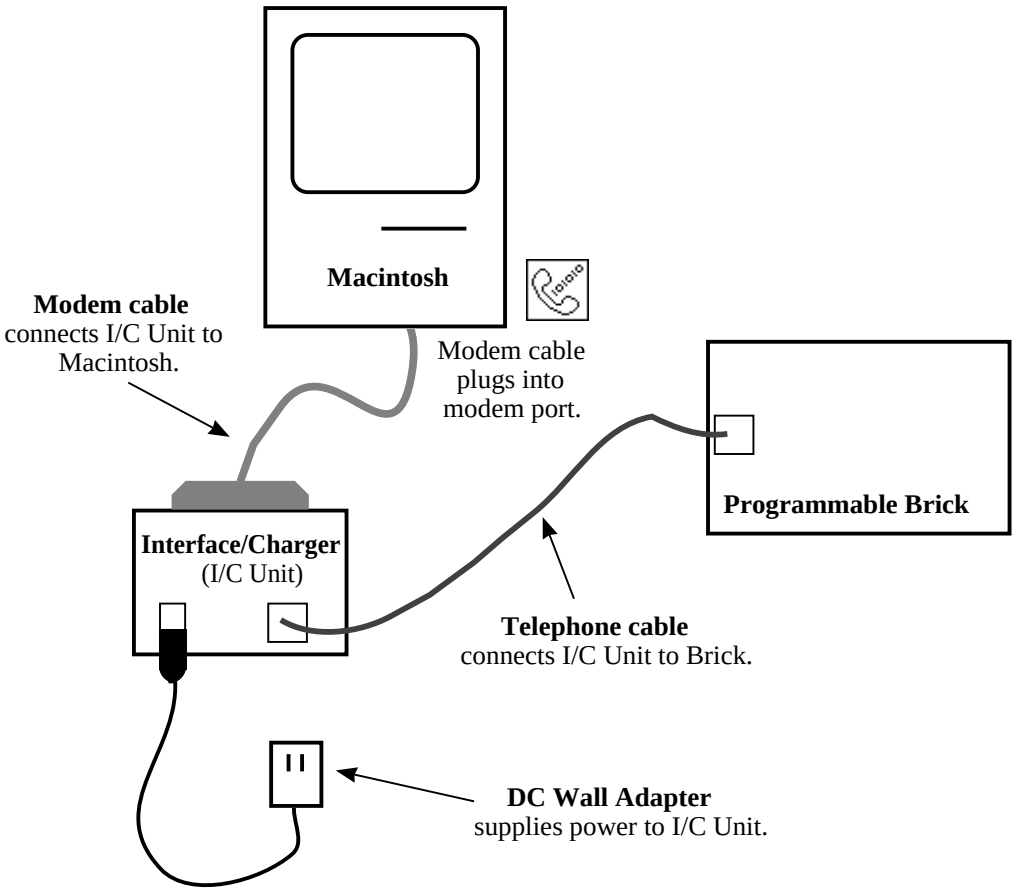

Figure 7: Programmable Brick, Interface/Charger, and Host Macintosh

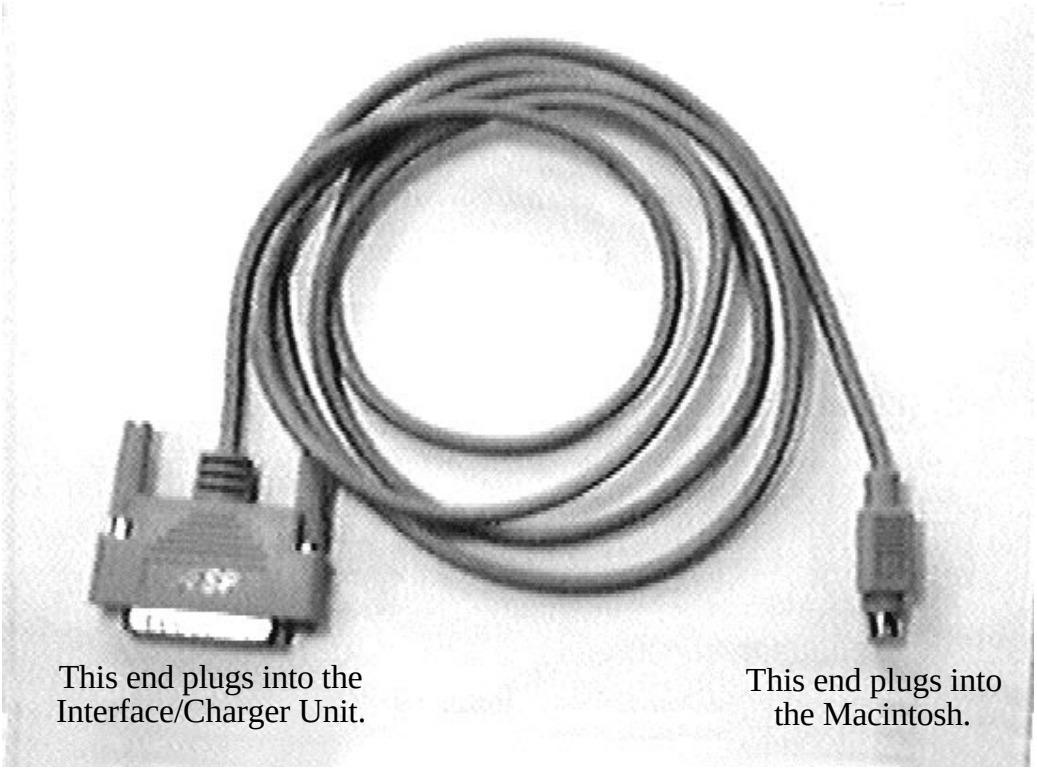

Figure 8: Macintosh Modem Cable

5. On the Interface/Charger Unit, set the switch labelled "CHARGE RATE" to the "NORM" position (normal charge). At this point, the yellow LED labelled "CHARGE" on the I/C Unit should light.

   If the yellow LED does not light, first make sure you have selected Normal Charge and the Brick is plugged into the I/C Unit. Still no light? Unplug the Brick and check if the red power light on the I/C Unit is on. If not, go back to Step 3, above.

   If the red power light is on (with the Brick unconnected), go ahead and plug the Brick back in. Check to see if the yellow light is on very dimly. This would mean that the Brick battery is fully charged.

   If you still see no light, proceed on the assumption that the Brick battery is so fully charged and the Charge LED is simply too dim to see.

A.2 Installing the Brick Software

The Brick software system makes use of *Logo MicroWorlds*, a modern commercial implementation of the Logo language that is sold by Logo Computer Systems, Inc. (LCSI).[4] The Brick software requires version 1.02 or later of MicroWorlds; if you have an earlier version, contact LCSI for upgrade information.

To install the Brick software, insert the Brick software distribution diskette into the Macintosh computer. On the disk will be a folder named "BRICK LOGO". Copy this folder onto your hard drive.

There are four files inside this folder:

BRICK LOGO 120 This file is the main Microworlds project for using the Programmable Brick.

LOADBRICK (RED) This file is a Microworlds project that is used for reloading the Brick's operating program.

RED.CODE This file contains data to be downloaded to the Brick by the LOADBRICK program.

-TOOLS- This file contains patches to Microworlds to allow it to properly run the BRICK LOGO program.

[4]Contact information for LCSI can be found in Appendix J, *Suppliers*.

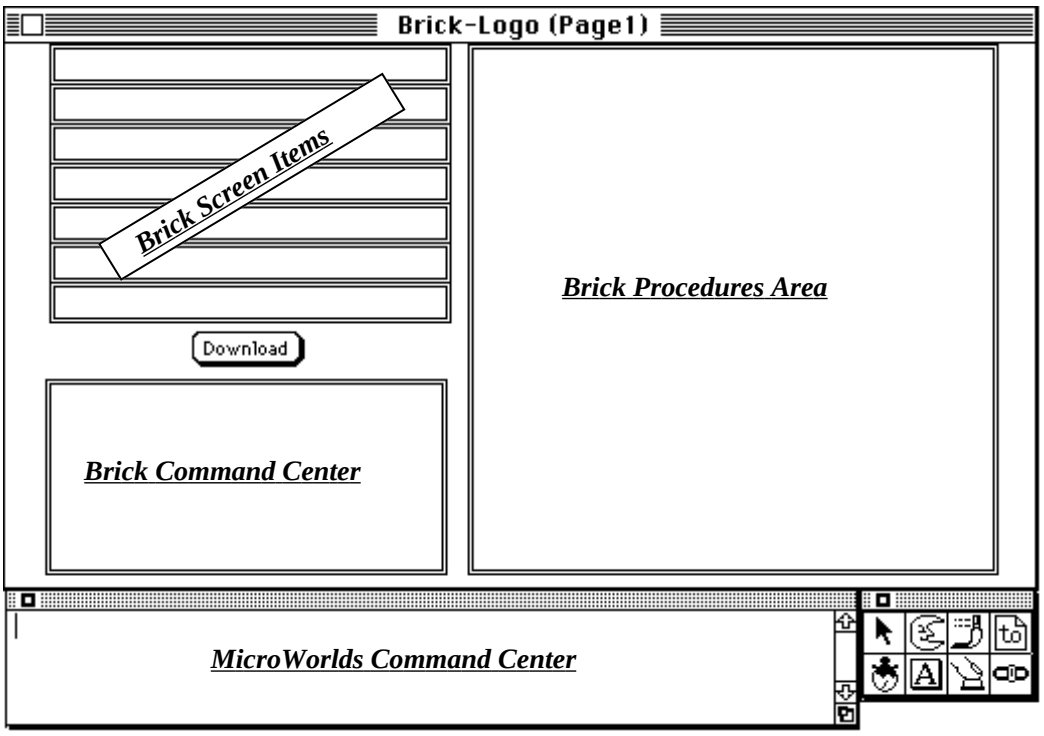

Figure 9: Brick Logo Screen with Annotations

A.3 Running the Brick Software

To start up the Brick Logo software, double-click on the Microworlds project named "BRICK LOGO." Click past the Microworlds splash screen, and the display shown in Figure 9 will be displayed. Here is the function of the various elements of the screen display:

Brick Command Center. In the lower left corner of the Brick Logo page is a region labelled *Brick Command Center*. This is for typing commands directly to the Brick. When you type a command in this box, it is sent to the Brick immediately and run.

Brick Procedures Area. The large rectangle on the right-hand side of the page, labelled *Brick Procedures Area*, is for programs to be downloaded to the Brick. Your Brick programs are typed into the box, and downloaded to the Brick when the *Download* button is clicked.

Brick Screen Items. Above the Brick Command Center are seven single-line text boxes. Commands typed into these windows are displayed on the Brick's screen (selectable using the Brick's knob) after programs are downloaded to the Brick. Pressing the Brick's "START" button causes the item currently being displayed to be run.

Download Button. Above the Brick Command Center is a button labelled *Download*. When this button is clicked, all procedures in the Procedures Area, as well as the Screen Items, are downloaded to the Brick. Any previous procedures are erased from the Brick's memory.

The Download button will highlight while the information is being downloaded to the Brick; also, the message "`Downloading...`" will be displayed in the Microworlds Command Center. When the download process is complete, the message "`done.`" will be displayed there.

MicroWorlds Command Center. The window at the bottom of the screen is the Microworlds Command Center. Commands typed into this window are executed by MicroWorlds rather than the Brick itself; for example, the commands to load and save Brick programs to the host computer hard drive are typed here. Also, status messages during download are displayed here.

At this point, check to see that the green "COMPUTER" LED on the Interface/Charger Unit is lit. *If the green LED is* not *lit, MicroWorlds will not be able to communicate with the Brick. Check that the I/C Unit is correctly plugged into the modem port.*

Turn the Brick on. Rotate its knob, and you should see various messages displayed on its screen, like `on`, `off`, `rd`, and a few others. Turn the knob all the way clockwise, and you should see two rows of numbers—the sensor readout.

If the Brick screen display does not come up, try turning the Brick off and then on again. If it does not come up after a few tries, the Brick will need to have its operating program reloaded. Skip ahead and follow the instructions in the next section, *Reloading the Brick Operating Program*, and then come back here after the Brick is working normally.

Plug the Brick into the I/C Unit, and the Brick is ready to accept commands from MicroWorlds. Click the mouse in the Brick Command Center, and type the command to turn on motor A:

```
a, on
```

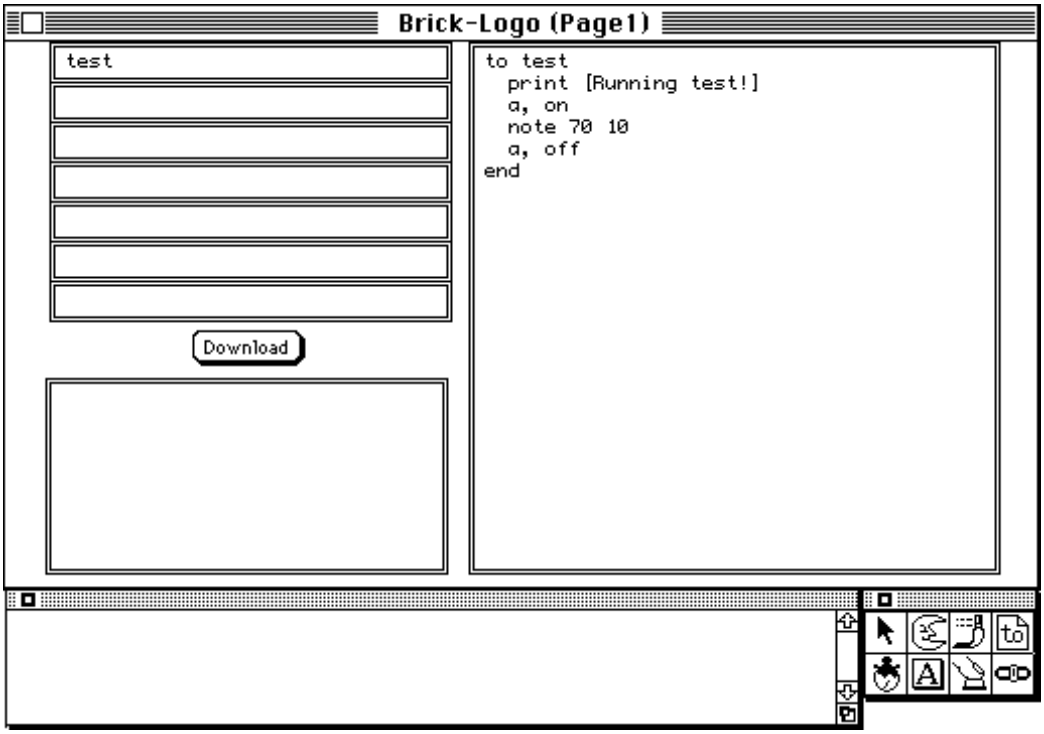

Figure 10: Brick Logo Test Program

After pressing return, MicroWorlds will send the command to the Brick, and the green LED above motor port A should turn on. If there is a motor plugged into port A, it will turn on as well. Did it work? If not, check to make sure that you typed the command exactly as indicated (lower-case "a", comma, space, lower-case "on") and that you typed it into the Brick's command center, not the MicroWorlds command center (see Figure 9).

Try the command to reverse the direction of the motor:

```
rd
```

The red LED for motor A will turn on, indicating the opposite direction as the green one. If there is a motor plugged in, it will change direction, though it might not be evident except for the "skip" it will make if it was plugged in and running when the `rd` command was executed.

Now let's try a simple Brick program. In the Brick Procedures Area, type in the following Brick Logo procedure:

```
to test
  print [Running test!]
  a, on
  note 70 10
  a, off
end
```

In one of the Brick screen item slots, type the name of this procedure, "test." The MicroWorlds screen should look like screen snapshot shown in Figure 10. In the figure, the first screen item slot is used to indicate the `test` procedure, but any of the seven slots will work fine.

Click on the Download button, and the procedure and screen items will be downloaded to the Brick. The green COMPUTER LED on the Interface/Charger Unit will flash briefly while the program is being downloaded (since this is a small program, there won't be much flashing). When the download process is finished, the Download button will un-highlight.

The program should now be loaded on the Brick. Rotate the Brick's knob to look through its screen items until the word `test` is displayed (the blank screen items are indicated by "`---`"). Press the "START" button, and the `test` procedure will be run. The Brick's display will read `Running test!` and the Brick will turn on motor A for one second while beeping.

This completes the introduction to basic Brick operation. From here, the next subsection, *Reloading the Brick's Operating Program*, explains what to do if the Brick fails due to a software error. Section 3 explains the Brick's motor control primitives, and Section 4 explains how to use standard Brick sensors. Appendix E is the full Brick Logo language reference.

A.4 Reloading the Brick Operating Program

When the Brick "crashes"—fails unexpectedly due to a software error—it is necessary to reload the Brick's operating program. This is done with a MicroWorlds project named "LOADBRICK".

To use the "LOADBRICK" project, double-click on the "LOADBRICK" file. This will open the MicroWorlds application, if it is not already running. If there is another MicroWorlds project open, it will be necessary to close that project before the "LOADBRICK" project will open.

When the "LOADBRICK" project loads into MicroWorlds, a short list of instructions will be shown on the screen, along with a screen button containing the word "loadbrick." By following these instructions, repeated here, the Brick's operating program will be reloaded:

1. *Make sure the Brick is connected to the host computer.* In order to reload the Brick operating program, it must be connected to the host computer. Appendix A.1 explains how to do this.

2. *Turn the Brick off.* There is a special sequence to boot the Brick into the mode where its operating program can be reloaded, and the first step is to turn the Brick off.

3. *Hold down the "START" button, and turn the Brick on.* When the Brick is powered up in this fashion, the red "LOW BATT" LED should flash for a fraction of a second, and then the green "READY" LED should turn on. At this point, the "START" button may be released.

   If the green "READY" LED does not come on, or if all of the Brick's LEDs are flashing wildly, then the Brick did not boot up properly. Try turning the Brick off and on again, holding down the "START" button until the green "READY" LED turns on.

4. *Click the mouse on the "loadbrick" button.* The screen button will highlight, and the computer will commence loading the Brick's operating program.

   On the Interface/Charger Unit, the green "COMPUTER" LED will flash while the download is in progress.

   When the download is finished, the screen button will un-highlight, and the green "COMPUTER" LED will stop flashing.

5. *Turn the Brick off, and then turn it on again.* This time, *do not* hold down the "START" button—turn the Brick on normally. The Brick's "READY" LED should come on, and its standard listing of screen items should be displayed.

B MS-DOS Computer Information

The Programmable Brick software for MS-DOS computers is currently under development. We are working on a Microsoft Windows version, and don't expect to develop a DOS-only version. Please contact us for more details.

C Battery Maintenance

This section explains how to operate the battery charging system on the Interface/Charger Unit. For users who don't want to be bothered with too much detail, it's adequate to remember just three things:

1. Leave the I/C Unit on "normal charge" at all times.
2. Turn the Brick off and keep it plugged in to the I/C Unit when not in use.
3. Make sure that the I/C Unit itself has power from the wall adapter.

   This is tricky because power from the Brick will cause the I/C Unit's "POWER" LED to light up *even if the I/C Unit is unplugged*. In this circumstance, the I/C Unit is actually draining power from the Brick's battery, not charging it! Therefore, to check that the I/C Unit is powered, it's necessary to unplug the Brick and check that the I/C Unit's "POWER" LED stays lit.

Following is additional information about the battery and charging system.

C.1 Charging Modes

The Interface/Charger Unit allows two charging modes, selected by the slide switch labelled "CHARGE RATE":

Normal Charge. In the normal charge mode, the Brick will be completely charged in about twelve to fourteen hours. When the yellow LED on the I/C Unit is lit, the system is in normal charge mode.

Zap Charge. In the zap charge mode, the Brick will be fully charged in about three to four hours. After this time, the Brick's battery pack will start to get warm, and the Brick should be removed from charge or placed into normal charge.

The Brick should not be left on zap charge for periods of more than 24 hours; permanent damage to the Brick's battery will result.

A few additional notes about battery charging and maintenance:

- When leaving the Brick to charge overnight, the Brick's own power should be turned off, and the I/C Unit should be set to Normal Charge. (If the Brick is left on, its battery will charge too slowly.)

- To make sure the Brick's battery is not over-charged, zap charge should only be used when the Brick is being attended.

  During normal use, the Brick will be alternately plugged into the host computer and then removed for testing. While the Brick is plugged in, the battery will charge a little. If the Brick is to be used for an extended period in this fashion, putting the charger into Zap Mode will help keep the Brick fully charged. Just make sure to put in back into Normal Mode when leaving the Brick at the end of the day.

- In a clutch, the I/C Unit may be used without its DC power adapter. In this case, it will draw power from the Brick. The Brick should not, however, be left connected to an unpowered I/C unit overnight, because the Brick's battery will become depleted—even if the Brick itself is turned off.

C.2 Battery Life

On a full charge, the Programmable Brick should last for a maximum of eighteen hours of continuous operation—enough for an overnight data-taking experiment.

When operating motors, however, the battery life is significantly shorter—about one to four hours, depending on how often and how many motors are in operation. If one motor is operating continously, expect a battery life of about two hours.

To set up projects where the Brick needs to operate for longer periods of time unattended, leave the Brick plugged in to the Interface/Charger Unit while the Brick is running. Set the I/C Unit to the normal charge mode.

C.3 Battery Level Readout

The "battery" command reports the battery level as a percentage of full charge. This command will normally report a full charge while the Brick is being charged from the Interface/Charger Units, so in order to get a proper reading, one must either (a) run a battery level command using the Brick's buttons and knob while

it is disconnected from the I/C Unit, or (b) remove power from the I/C Unit and issue the battery command from the computer screen Brick Command Center.

To try out the `battery` command, put the statement "`print battery`" in one of the Brick screen item slots, and run it while the Brick is disconnected from the computer.

D The Brick Interface/Charger Unit

The Brick Interface/Charger Unit, or I/C Unit, serves a dual function, as indicated by its name:

- It interfaces the Brick to a desktop computer.
- It recharges the Brick's internal battery.

This section presents reference information on the function of the connectors, LEDs, and switch on the Interface/Charger Unit. The I/C Unit is depicted in Figure 6 on page 28.

D.1 Connectors

The I/C Unit has three connectors:

Computer Connector. The large, D-shaped connector is used to connect the I/C Unit to the host computer.

Brick Connector. The telephone-style connector is used to connect the I/C Unit to the Brick.

DC Power Jack. The remaining connector supplies power to the I/C Unit, for operating its own circuitry and for recharging the Brick's internal battery.

D.2 Status LEDs

The I/C Unit has three status LEDs:

Computer. The green LED labelled "COMPUTER" lights when the host computer is sending a communications signal to the Brick.

> *Note to Macintosh users: The Macintosh typically does not generate the communications signal until* after *the Brick software is started up. Therefore, when the I/C Unit is first connected to the Mac, this LED won't light up.*

Power. The red LED labelled "POWER" lights when the I/C Unit is receiving power.

> *Note: The I/C Unit may receive power from* either *the DC wall adapter or the Programmable Brick. If the I/C Unit is connected to the Brick and the DC power is disconnected, the I/C Unit will drain the Brick's battery to receive its own power.*

Charge. The yellow LED labelled "CHARGE" lights when the I/C Unit is charging the Brick's battery. The I/C Unit must be plugged into DC power in order to charge the Brick (see above); otherwise, it will drain power from the Brick.

The yellow LED turns off when the I/C Unit is charging in Zap Mode (see below). Also, when the Brick is completely charged, the charge LED may not light, even when in Normal mode.

D.3 Charge Rate Switch

When left to charge, the Brick itself should be turned off. The I/C Unit has a slide switch labelled "CHARGE RATE" that selects one of two charging modes.

Normal Charge. Labelled "NORM", the normal charge mode fully charges the Brick in twelve to fourteen hours. During normal charge mode the yellow "CHARGE" LED should light.

Zap Charge. Labelled "ZAP!!", the zap charge mode fully charges the Brick in about three hours. During zap charge mode the yellow "CHARGE" LED will *not* light.

Please see Appendix C, *Battery Maintenance*, for more information about battery charging and caveats about using the Zap Charge mode.

D.4 Adapter Specifications

The Interface/Charge Unit was designed to work with a range of power adapters. The specifications for a compatible adapter are:

VOLTAGE	12–15 v
CURRENT	300–500 mA
POLARITY	AC or DC
PLUG STYLE	coaxial power plug
PLUG SIZE	2.1 mm ID, 5.5 mm OD

E Brick Logo Quick Reference

Brick Logo is the language used to write programs that run on the Brick. Brick Logo is similar to the versions of Logo that are part of the commercial LEGO Dacta products (both *LEGO tc logo* and *LEGO Control Lab*). Previous experience with either of these two products, as well as any other Logo experience, will translate easily to writing programs for the Programmable Brick.

E.1 Motors

Motors A, B, and C are bi-directional (the motors' can be reversed under software control). Motor D is uni-directional—the Brick can only turn the motor on and off, and the direction is determined by the way the cable is connected.

`a,` Selects motor A for subsequent commands.

`b,` Selects motor B.

`c,` Selects motor C.

`d,` Selects motor D.

`ab,` Selects motors A and B together.

`bc,` Selects motors B and C.

`ac,` Selects motors A and C.

`abc,` Selects motors A, B, and C.

`abcd,` Selects all motors.

`on` Turns selected motor(s) on.

`off` Turns selected motor(s) off.

`toggle` Inverts on/off state of selected motor(s); i.e., motors that are off go on, and motors that are on go off.

`rd` Reverses direction of selected motor(s).

`thisway` Sets selected motor(s) for one of the two possible directions (indicated by the green motor LED being illuminated). When motors are first turned on, they are in the "thisway" state.

`thatway` Sets selected motor(s) for the other of the two directions (indicated by the red motor LED being illuminated).

`onfor` *time* Turns selected motor(s) on for *time* tenths of seconds.

`setpower` *level* Sets the power level of the selected motor(s). Power levels range from 8 (full power) to 0 (off). The initial state of motors, when turned on, is full power.

E.2 Sensors

`switcha`

`switchb`

`switchc` Reports value of switch sensor (pressed is “true,” not pressed is “false.”)

`sensora`

`sensorb`

`sensorc`

`sensord`

`sensore`

`sensorf` Reports value of sensor as a number from 0 to 255.

`countera`

`counterb`

`counterc` Reports counts on angle sensor.

`resetca`

`resetcb`

`resetcc` Resets count to zero.

`timer` Reports amount of elapsed time in milliseconds. Reset by `resett` or pressing STOP button.

`resett` Resets elapsed time count to zero.

`battery` Reports battery level as a percentage of full charge (0 to 100).

E.3 Control Structures

`wait` *time* Waits (does nothing) for *time* tenths of seconds.

`waituntil [` *condition* `]` Waits until *condition* becomes true. Example: `waituntil [sensora > 180]`

`if` *condition* `[` *action* `]` Performs *action* if *condition* is true. Typically used in a loop to repeatedly test the condition. Example: `if switcha [ad, rd]`

`ifelse` *condition* `[` *action* `] [` *else-action* `]` Performs *action* if *condition* is true; otherwise, performs *else-action*. Example: `ifelse sensora > 180 [a, on d, off][a, off d, on]`

`repeat` *times* `[` *action* `]` Repeatedly performs *action* for *times* number of times. Example: `repeat 10 [ad, onfor 10 rd]`

`loop [` *action* `]` Indefinitely loops performing *action*. To exit, use `stop` command, which causes currently running procedure to terminate.

E.4 Input/Output

E.4.1 LCD Display

`print "`*word* Prints a single word to the LCD screen. Example: `print "hello`

`print [` *word1 word2 word3* . . . `]` Prints phrase to the computer screen. Example: `print [hello there matey]`

`print` *number* Prints a number to the LCD screen. Example: `print sensora`

`type` Used like `print`, but allows multiple statements to print onto the same display line. Example: `type [Sensor is] print sensora`

`top` Selects top line of display for subsequent printing.

`bottom` Selects bottom line of display.

E.4.2 Input

The following describes the action of the start and stop buttons.

START **button.** Pressing the START button causes the screen item currently displayed on the Brick's LCD screen to be run (if it was idle). An asterisk is displayed in the lower right corner of the screen while the item is running. If the screen item was already active when the START button is pressed, then the item's process is stopped.

STOP **button.** Pressing the STOP button causes all processes running on the Brick to be stopped. All motor outputs are turned off. Additionally, the internal motor state is reset to the power-on defaults: all motors at `setpower 8`, direction `thisway`, and talkto state `a,`.

E.4.3 Sound

`note` *midi-step duration* Plays a tone on the Brick's beeper. Pitch is determined by *midi-step* number, which represents successive semi-tones as value increases. Audible values range from about 40 (low tones) to 120 (high tones). *duration* is specified in tenths of seconds.

E.4.4 Infrared Communication

The Brick infrared commands from a Sony-brand infrared remote (or a universal remote programmed to transmit Sony codes). Keys 1 through 7 cause the first through seventh screen item, respectively, to be run.

When the Brick is running a program, the Power key will cause the program to stop (this is equivalent to pressing the Stop button). In addition, Brick Logo programs can use the following primitives to send and receive infrared codes. Note that if a Brick transmits the code corresponding to the "1" key to another Brick, the Brick receiving the transmission will run the screen item corresponding to the key. If this program is already running, receiving the code will stop execution; otherwise, it will initiate it.

`ir` Reports a number corresponding to a key on an infrared remote or signal transmitted from another Brick.

`irf` Reports number received by infrared sensor plugged into sensor port F.

`irsend` *value* Sends *value* from 0 to 255 to another Brick, using infrared transmitter accessory plugged into motor port D. Note translation table below.[5]

Transmitted Character	*Received Action*
128 or 18	runs/stops menu item 1
129 or 20	runs/stops menu item 2
130 or 19	runs/stops menu item 3
131 or 17	runs/stops menu item 4
132	runs/stops menu item 5
133	runs/stops menu item 6
134	runs/stops menu item 7
149 or 223	stops all processes & motors

E.4.5 Serial Line

The Brick can send characters over the serial line while it is executing Brick Logo programs. The serial line setting is 9600 baud, eight bit data, no parity.

`send` *char* Transmits lower byte of *char* over serial line.

E.4.6 Speech Output

The Brick can connect to a specially-modified version of RC Systems' voice board for natural-speech output.[6] The "say" primitive is used to transmit information to the voice board over the Brick's serial line connection:

`say "`*word* Outputs a single word to the voice board. Example: `say "hello`

`say [` *word1 word2 word3* . . . `]` Outputs phrase to the voice board. Example:
`say [hello there matey]`

[5]This table is used to translate the channel/volume up/down keys, from a Casio infrared watch, into the codes for buttons 1 through 4. The 149 code is the Power key, and the 223 code is the Stop key on Sony CD player remotes.

[6]Contact the authors for information about how to wire the voice board to the Brick.

`say` *number* Outputs a number to the voice board. For example, `say sensora` would result in the current value of sensor A being transmitted. The voice board converts the numeric representation (e.g., "193") to its spoken form (e.g., "one hundred ninety three").

After any power-on, it is necessary to send the voice board an odd-numbered-byte over the serial line, followed by a short delay, to establish communications baud rate. The carriage return character, 13, is a good choice. Also, it is necessary to send the carriage return to get the board to speak words that have been already transmitted:

```
to init-speech-board
  send 13 wait 1
end

to test-speech-board
  say [hello there.]  send 13
end
```

E.5 Multi-Tasking

The Brick can support up to eight concurrent process tasks. Each of the following primitives launches a new task.

E.5.1 Launching Processes

`launch [` *action* `]` Launches *action* as a separate process.

`forever [` *action* `]` Launches a process to repeatedly execute *action*. Equivalent to `launch [loop [` *action* `]]`.

`when [` *condition* `][` *action* `]` Launches a process to repeatedly test *condition* and execute *action* when it becomes true.

The condition clause for the `when` statement fires on edge-triggered logic; that is, *action* is run each time that *condition* changes from false to true. In the case in which the *condition* is true the first time the `when` statement is executed, the *action* is not run.

`every` *time* `[` *action* `]` Launches a process to execute *action* every *time* tenths-of-seconds.

E.5.2 Stopping Processes

Pressing the STOP button or sending the infrared stop code stops all running tasks, turns off motors, and resets the internal motor state (see E.4.2).

`stoprules` Stops all processes except the one executing the "stoprules" command.

E.6 Data Recording and Playback

There is a single global array for storing data which holds 5887 2-byte integer values. There is no error-checking to prevent against overrunning the data buffer.

`erase` Resets the data recording pointer to zero.

`record` *value* Records *value* in the data buffer, and advances the recording pointer.

`record#` Reports value of record pointer, indicating where the next data point to be recorded will go.

`resetr` Resets the recall pointer to zero.

`recall` Reports value of current data point, and advances the recall pointer.

`recall#` Reports value of recall pointer.

E.7 Procedures, Variables, and Comments

E.7.1 Procedure Definition

Procedures are defined using the keyword "to"; i.e.:

```
to test
  procedure body
end
```

E.7.2 Procedure Inputs

Inputs, or arguments, to procedures are declared using the standard Logo colon syntax; e.g.:

```
to test :input1 :input2
  top type [Input 1 is] print :input1
  bottom type [Input 2 is] print :input2
  wait 10
end
```

Procedure inputs are local variables.

E.7.3 Local Variables

Local variables are declared using the `let` keyword, accessed using Logo's colon syntax, and set using the `make` keyword:

```
to local-example
  let [alocal 5 anotherlocal 17]
  print :alocal ; prints "5"
  make "anotherlocal 3
  print :anotherlocal ; prints "3"
end
```

The "let" declaration should be made at the beginning of a procedure.

E.7.4 Global Variables

Global variables are declared using the `global` keyword, which takes a list of the names of globals to be created; i.e.:

`global [`*name1 name2 name3 . . .*`]`

This declaration should come at the beginning of the procedure buffer. After being declared, each global is set using a mechanism in which the global name is preceded by the word "set"; their values are accessed by using the global name as a reporter; e.g.:

```
global [myglobal]

to test
  setmyglobal 3
  print myglobal
  wait 10
end
```

Global variables maintain their value when the Brick is power-cycled.

E.7.5 Procedure Return Values

By default, procedures do not produce return values. Procedures may return a numeric value using the `output` primitive; e.g.:

```
to double :n
  output :n * 2
end
```

Procedures may terminate at any point using the `stop` primitive, which exits the procedure without producing a return value.

Care should be taken to ensure that a procedure either *always* or *never* exits with a return value.

E.7.6 Code Comments

There are two forms for comments in the procedure buffer:

- Any text between the `end` statement of one procedure and the `to` declaration of the next procedure is ignored.
- Any text after a semicolon (“;”) on any given line is ignored.

E.8 Numeric Operations

Brick Logo is based on signed 16-bit integer arithmetic (all numeric values are in the inclusive range from -32768 to $+32767$).

All of the following arithmetic and boolean operators must be preceded and followed by a space. For example, the following expression is *not* legitimate:

```
print 3+4
```

E.8.1 Arithmetic Operators

The following arithmetic operators are supported, using infix notation:

`+` — addition.

`-` — subtraction.

`*` — multiplication.

`/` — division.

`\` — remainder.

The minus sign may also be used as a prefix negation operator.

E.8.2 Boolean and Bitwise Operators

The Boolean operators always produce values of zero or one. In evaluating conditionals, zero is false; any value other than zero is true.

`and` — performs bitwise "and" function. Prefix.

`or` — performs bitwise "or" function. Prefix.

`not` — performs Boolean logical negation. Prefix.

`>` — performs Boolean test for greater-than. Infix.

`<` — performs Boolean test for less-than. Infix.

`=` — performs Boolean test for equality. Infix.

Since the Boolean operators produce values of one and zero, and non-zero results are considered true, the `and` and `or` operators, which are bitwise, can function as Booleans when combining the result of other conditionals. The following example illustrates correct usage:

```
if and (:value > 100) (:value < 150) [doit]
```

E.8.3 Precedence

Order of evaluation is from left to right; standard rules of precedence are *not* observed. Parentheses may be used to override the standard order of evaluation.

E.8.4 Random Numbers

Brick Logo includes a primitive for generating pseudo-random numbers. The "random" primitive takes as input the upper limit of the number to be generated, and reports a value between 0 and that number minus 1 (inclusive).

`random` *limit* Reports a pseudo-random number between 0 and *limit* − 1 (inclusive).

E.9 File Management

To save and load Brick Logo programs, please use the following (rather than saving multiple copies of the Brick Logo project):

`saveall "`*filename* Saves procedures and screen items into file named *filename*.

`loadall "`*filename* Loads procedures and screen items from file named *filename*.

These commands must be typed into the MicroWorlds command center, located at the bottom of the computer screen, not the Brick command center.

It is also possible to save an entire Brick Logo project (procedure definitions and screen items) to a Brick. Use the following commands:

`savetobrick` Saves procedures and screen items to a Brick.

`loadfrombrick` Loads procedures and screen items from a Brick.

F Error Messages

This appendix contains a listing of common errors and their likely causes and solutions.

Sensor problem This message would be seen on the Brick's LCD display. It means that a motor or other short-circuit has been plugged into one of the LEGO sensor ports. When this happens, turn the Brick off and check the wiring of the devices plugged into the Brick's LEGO sensors.

Stack bug This message would be seen on the Brick's LCD display. This message indicates an internal system error that *may* be due to a mistake in a user program, or may be the result of a hardware problem or bug in the Brick's own software.

When this message occurs, the best solution is to reload the Brick's runtime program using LOADBRICK, and then try to download your program to the Brick again. If the problem recurs, try to remove the last bit of programming you just added before the problem occurred, and try again.

I don't know how to modem-port in startup This message in the Microworlds command center generally means that Microworlds did not not load the -TOOLS- file properly.

In order for Microworlds to load this file, make sure that either or both of the following methods is used:

- Put the -TOOLS- file in the same folder as the BRICK LOGO project file, and start Microworlds by double-clicking on the BRICK LOGO project (or an alias of it).
- Put the -TOOLS- file in the same folder as the Microworlds application, and start Microworlds by double-clicking on the application itself.

something *undefined . . . so no download* This message occurs after you have clicked "download" if Brick Logo cannot figure out the procedure definition of *something*. Check the procedure buffer and the menu items to make sure there are no typing mistakes; if the error persists there is probably a problem with invisible characters in the buffers. See solution in Appendix G.

Out of space This message will sometime appear after switching between the BRICK LOGO and LOADBRICK projects. The solution is simply to quit Microworlds and restart.

If the error persists, or occurs when attempting to download a Brick Logo program, check for the following:

- Make sure Microworlds has a big enough memory partition. It may be necessary to allocate three megabytes. Increasing it beyond this won't help, however.
- When downloading large Brick Logo programs (one hundred lines or more), it is necessary to patch Microworlds to give it more nodespace. Please use a copy of ResEdit to perform the patch:

  1. Open the `CODE` resources.
  2. Open resource number 3.
  3. Change the first byte from 28 to 40.
  4. Close and save changes.

  Please find a local Macintosh expert to perform the patch if you have never used ResEdit.

G Known Bugs

This section lists known bugs with the current brick hardware (Model 120) and software (Brick Logo 120, Microworlds version, date September 26, 1995).

G.1 Hardware

- Motor ports A through C will burn out if driving a short circuit for a moderate period of time (i.e., ten seconds or more). Please be extremely careful not to create a short circuit with the LEGO cables.

- There is a hardware design error related to motor port D. If motor D drives a short circuit, or a load of less than 2 ohms, the drive transistor will burn out.

  This shouldn't be a problem when using stock 9v LEGO motors because their stall resistance is greater than 2 ohms, so even a stalled motor won't cause the drive transistor to fail. Please be careful, though, when using other motors with motor D, and be careful not to create a short with the connector cables.

G.2 Software

- It is possible to generate invisible characters in the Brick Logo procedures buffer and screen items slots. These invisible characters cause Brick Logo to generate "`xxx undefined ...so no download`" error messages.

  Find the invisible characters by advancing one letter at a time using the arrow keys. If you find that you have pressed the forward arrow, but the cursor doesn't advance, you have just spaced over an invisible character. Press backspace to delete it.

H Version History

H.1 Hardware

The current Brick model, referred to as *Model 120*, contains numerous small improvements from the earlier version, *Model 100*. The Model 100 Brick was developed during early summer 1994 and used from August through December 1994. The Model 120 Brick was developed during the fall of 1994, was introduced in January 1995, and is currently in active use.

Changes from Model 100 to Model 120 include:

Plastic Cover. The Model 100 Brick did not have a plastic cover—the electronic "guts" of the Brick were exposed for users to admire. We decided, though, that an enclosed Brick would be more friendly to users, "pocketable," and generally more robust.

Better Connectors. The Model 100 used four metal screw-heads to attach to the LEGO connectors. Unfortunately the screws were not held tightly enough in the printed circuit board, and performed rather poorly. The Model 120 uses real LEGO connectors.

Better Motor Outputs. The 100 had two "full power" motor outputs and two "half power" motor outputs (all bi-directional). The 120 has three full power, bi-directional outputs and one full power uni-directional output. While the 120 has one fewer bi-directional motor output, we consider the inclusion of all full-powered outputs to be an improvement.

Short Circuit Protection. The Model 100 Brick would die if two of them were plugged together. The Model 120 includes a fuse to protect against this circumstance.

Improved Infrared Capability. Port D of the Model 120 Brick includes a hardware oscillator to generate the infrared carrier frequencies, as well as higher current drive capability.

Better Switches and Knob. The buttons and knob on the Model 120 Brick are higher-quality and more accessible.

H.2 Software

September 26, 1995 version

New features:

- `random` primitive added.

April 6, 1995 version

Bug fixes:

- `irsend` primitive now works correctly. Previously it transmitted the character out motor port C; it now uses port D.

March 23, 1995 version

New features:

- `irf` primitive to return infrared values detected on an infrared sensor plugged into sensor F.

Changes:

- `ir` primitive returns a value even if it is trapped by menu handler (e.g., the "1" key which triggers menu item 1). Also, low-level IR codes are no longer translated in any fashion, so the "1" key returns 128 rather than 1 (for example).

Bug fixes:

- Local variables were badly broken.
- The list object would break if it was longer than 256 bytes. This bug would be exercised, for example, in an unusually long procedure.
- The `phase` and `stopphase` primitives were removed. The primitives were broken, having been left in place after being superceded by `stoprules` in an earlier version.

I Bibliography

This bibliography presents a sampling of works that contain ideas that influenced the Programmable Brick, and may serve to inspire projects that use it.

- *Vehicles*, Valentino Braitenberg
- *The 6.270 Robot Builder's Guide*, Martin
- *Circuits to Control: Learning Engineering by Designing LEGO Robots*, Martin
- *Mindstorms*, Papert
- *Twenty Things to Do with a Computer*, Papert and Solomon
- *Xylophones, Hamsters, and Fireworks*, Resnick
- *Behavior Construction Kits*, Resnick

J Suppliers

This appendix lists contact information for recommended suppliers of materials useful for Programmable Brick projects. Included are: retail electronics companies; surplus electronics companies; LEGO Dacta; and Logo Computer Systems, Inc.

All Electronics All Electronics is a top-notch surplus electronics company. They have a wide ranging assortment of materials useful for robotics projects, including switches, motors, sensors, connectors, and wire. Their catalog is well-organized and includes a good supply of basic components.

All Electronics has a minimum order of $15, and charges a flat $4 fee for UPS Ground shipping (unless the order is exceptionally heavy). They generally pack and ship and order within two days of receiving it. Faster delivery options are available, but are expensive (e.g., $12 to ship a small parcel via UPS Blue two-day service).

Digi-Key Corporation
701 Brooks Ave. South • P.O. Box 677
Thief River Falls, MN 56701-0677
orders and info: (800) DIGI-KEY
fax: (218) 681-3380

Digi-Key is the undisputed leader in the mail-order retail electronics industry. Their extensive catalog, superb service, and willingness to deal with small orders (as well as large ones) are unmatched.

Get the Digi-Key catalog and use it for basic supplies, switches, optical devices, chips, cases, and connectors. Prices are competitive with any retail dealer.

Terms: $5 handling charge for orders under $25; MC/VISA, check, money order or COD; customer pays shipping on credit card and PO orders. Increasing volume discounts for orders over $100.

The Electronics Goldmine
POB 5408 • Scottsdale, AZ 85261
orders and info: (602) 451-7454
fax: (602) 451-9495

Unusual catalog comprised mostly of packaged electronic kits, with a small amount of very good surplus stock mixed in. A good source of parts for building sensors.

Terms: $10 minimum order; MC/VISA accepted; minimum $3.50 for UPS shipping.

LEGO Dacta
555 Taylor Road • Enfield, CT 06082
orders and info: (800) 527–8339 or (800) 243–4870

LEGO Dacta is the branch of the LEGO company that is responsible for educational sales (as distinct from retail toy sales). Dacta packages LEGO Technic sets for classroom use, including lesson plans and other support materials.

Call Dacta and get their "Gear Up for Learning" catalog, which has many LEGO Technics kits.

Logo Computer Systems, Inc.
(800) 321-LOGO

Logo Computer Systems, Inc. (LCSI) is the world's largest supplier of Logo software for educational use. Their latest version of Logo, called *Logo Microworlds*, includes features such as multi-tasking Logo program execution, dynamic creation of up to dozens of turtles, color paint tools, and user interface objects like buttons, sliders, and text windows. Microworlds is available for both the Macintosh and MS-DOS computer platforms.

Marlin P. Jones and Associates
P.O. Box 12685 • Lake Park, FL 33403-0685
orders and info: (407) 848-8236
fax: (407) 844-8764

Another good company for robotic surplus: motors, transformers, lots of relays, switches, optical stuff, power supplies, rechargeable batteries.

Terms: $1 handling charge for orders under $10; MC/VISA accepted; shipping charges are UPS rates.

MCM Electronics
650 Congress Park Drive • Centerville, Ohio 45459-4072

orders and info: (800) 543-4330
fax: (513) 434-6959

MCM is a low-priced leader for basic to high quality tools, electronics building supplies, and other general-purpose materials (e.g., solder, tape, glue, etc). They specialize in hard-to-find parts for VCR servicing, but their catalog is a must-have for anyone setting up a small electronics lab on a budget because of their excellent assortment, reliable service, and heavily discounted prices.

Terms: MCM is a retail company. $25 minimum for credit card orders; MC/VISA accepted; shipping charges are UPS rates plus $2.10 handling charge.

Radio Shack

National electronics chain; check yellow pages.

Radio Shack is the ubiquitous chain electronics parts store. Few localities would have a shop that could compete with Radio Shack's selection of in-store stock.

For Programmable Brick-related projects, the 'Shack can be mined to find: a variety of sensor parts (including switches, photocells, and mercury switches), hand tools, connectors, and wire.

Radio Shack is not known for lowball prices, but their selection and convenience more than make up for sometimes-a-little-high pricing.

RC Systems, Inc.

1609 England Avenue • Everett, WA 98203
phone: (206) 355–3800
fax: (206) 355–1098

RC Systems sells a custom text-to-speech board for voice output applications. It can be easily modified to work with the Programmable Brick. The board sells for about $150 in single quantities.