D. APPENDICES

Copyright 1984 All rights reserved.

APPENDIX 1

RCL Robot Task "PARTY" and its Tiny BASIC Translation

```
PARTY (a Task)

     1 Does PREPARE THE ROBOT
     2  and PREPARE THE VOICE
     3  and LOAD THE PHRASE called "SAY THE INTRODUCTION"
            with phonemes: "H.H.EH.L.L.OO.O1.PA1.AH.E.E.
            PA1.AE.AE.M.PA1.D.THV.U.PA1.AH1.AH1.R.PA0.B.
            Y.Y.PA1.F.AH.Y.V.PA1.EH1.EH2.K.PA0.S.PA1.I.N.
            T.EH.L.L.UH3.D.J.EH3.N.T.PA1.R.O1.B.AH1.T.ST
            OP."
     4  and CALL this robot task line "BEGINNING OF PARTY"
     5  and BEGIN A LOOP
     6  and EXIT IF ANY BUMPER TOUCHED
     7  and REPEAT THIS LOOP
     8  and TURN ON THE FLASHING LIGHTS
     9  and MOVE FORWARD
    10  and BEGIN A LOOP
    11  and EXIT IF ANY BUMPER TOUCHED
    12  and EXIT IF SONAR distance value is less than 95
    13  and REPEAT THIS LOOP
    14  and STOP ALL MOTION
    15  and SPEAK the phrase called: "SAY THE INTRODUCTION"
    16  and SPIN CLOCKWISE this many degrees:  180
    17  and MOVE FORWARD
    18  and BEGIN A LOOP
    19  and EXIT IF ANY BUMPER TOUCHED
    20  and REPEAT THIS LOOP
    21  and SPIN CLOCKWISE this many degrees:  180
    22  and TURN OFF THE FLASHING LIGHTS
    23  and JUMP to the line called "BEGINNING OF PARTY"
    24  and END
```

Stmt	Tiny BASIC Text	Meaning of Text
10	@#7803=#98	INITIALIZE I/O
20	@#780B=#A7:S=TOP:A=0	PREPARE THE VOICE
30	REM RB5X say the introduction	REMARK
40	P=#1B:GOSUB 3500	VOICE SUBROUTINE
50	P=#02:GOSUB 3500	VOICE SUBROUTINE
60	P=#00:GOSUB 3500	VOICE SUBROUTINE
70	P=#1B:GOSUB 3500	VOICE SUBROUTINE
80	P=#35:GOSUB 3500	VOICE SUBROUTINE
90	P=#37:GOSUB 3500	VOICE SUBROUTINE
100	P=#15:GOSUB 3500	VOICE SUBROUTINE
110	P=#00:GOSUB 3500	VOICE SUBROUTINE
120	P=#09:GOSUB 3500	VOICE SUBROUTINE
130	P=#29:GOSUB 3500	VOICE SUBROUTINE
140	P=#3E:GOSUB 3500	VOICE SUBROUTINE
150	P=#3E:GOSUB 3500	VOICE SUBROUTINE
160	P=#2F:GOSUB 3500	VOICE SUBROUTINE
170	P=#00:GOSUB 3500	VOICE SUBROUTINE
180	P=#0C:GOSUB 3500	VOICE SUBROUTINE
190	P=#3B:GOSUB 3500	VOICE SUBROUTINE
200	P=#32:GOSUB 3500	VOICE SUBROUTINE
210	P=#15:GOSUB 3500	VOICE SUBROUTINE
220	P=#31:GOSUB 3500	VOICE SUBROUTINE
230	P=#3A:GOSUB 3500	VOICE SUBROUTINE
240	P=#0E:GOSUB 3500	VOICE SUBROUTINE
250	P=#3C:GOSUB 3500	VOICE SUBROUTINE
260	P=#21:GOSUB 3500	VOICE SUBROUTINE
270	P=#29:GOSUB 3500	VOICE SUBROUTINE
280	P=#1D:GOSUB 3500	VOICE SUBROUTINE
290	P=#15:GOSUB 3500	VOICE SUBROUTINE
300	P=#00:GOSUB 3500	VOICE SUBROUTINE
310	P=#29:GOSUB 3500	VOICE SUBROUTINE
320	P=#0F:GOSUB 3500	VOICE SUBROUTINE
330	P=#3E:GOSUB 3500	VOICE SUBROUTINE
340	P=#02:GOSUB 3500	VOICE SUBROUTINE
350	P=#01:GOSUB 3500	VOICE SUBROUTINE
360	P=#19:GOSUB 3500	VOICE SUBROUTINE
370	P=#03:GOSUB 3500	VOICE SUBROUTINE
380	P=#1F:GOSUB 3500	VOICE SUBROUTINE
390	P=#0B:GOSUB 3500	VOICE SUBROUTINE
400	P=#0D:GOSUB 3500	VOICE SUBROUTINE
410	P=#2A:GOSUB 3500	VOICE SUBROUTINE
420	P=#02:GOSUB 3500	VOICE SUBROUTINE
430	P=#00:GOSUB 3500	VOICE SUBROUTINE
440	P=#1B:GOSUB 3500	VOICE SUBROUTINE
450	P=#0B:GOSUB 3500	VOICE SUBROUTINE
460	P=#09:GOSUB 3500	VOICE SUBROUTINE
470	P=#1E:GOSUB 3500	VOICE SUBROUTINE
480	P=#1A:GOSUB 3500	VOICE SUBROUTINE
490	P=#02:GOSUB 3500	VOICE SUBROUTINE
500	P=#00:GOSUB 3500	VOICE SUBROUTINE

```
Stmt    Tiny BASIC Text                          Meaning of Text

 510     P=#0D:GOSUB 3500                         VOICE SUBROUTINE
 520     P=#2A:GOSUB 3500                         VOICE SUBROUTINE
 530     P=#2B:GOSUB 3500                         VOICE SUBROUTINE
 540     P=#35:GOSUB 3500                         VOICE SUBROUTINE
 550     P=#37:GOSUB 3500                         VOICE SUBROUTINE
 560     P=#0E:GOSUB 3500                         VOICE SUBROUTINE
 570     P=#15:GOSUB 3500                         VOICE SUBROUTINE
 580     P=#23:GOSUB 3500                         VOICE SUBROUTINE
 590     P=#2A:GOSUB 3500                         VOICE SUBROUTINE
 600     P=#3F:GOSUB 3500                         VOICE SUBROUTINE
 610     REM LABEL THIS LINE BEGINNING            LABEL THE LINE
 620     REM START A LOOP                         BEGIN A LOOP
 630     Y=@#7800                                 TEST FOR BUMPER CONTACT
 640     IF Y<255 GOTO 660                        EXIT IF ANY CONTACT
 650     GOTO 620                                 REPEAT THIS LOOP
 660     REM EXIT TO HERE
 670     X=#40                                    TURN ON FLASHING LIGHTS
 680     GOSUB 3100                                GO TURN ON A BIT
 690     @#7802=#09                               GO FORWARD
 700     REM START A LOOP                         BEGIN A LOOP
 710     Y=@#7800                                 TEST FOR BUMPER CONTACT
 720     IF Y<255 GOTO 770                        EXIT IF ANY CONTACT
 730     D=0                                      ZERO DISTANCE
 740     LINK #1800                               CALL SONAR
 750     IF D<95 GOTO 770                         EXIT IF LESS THAN VALUE
 760     GOTO 700                                 REPEAT THIS LOOP
 770     REM EXIT TO HERE
 780     @#7802=0                                 STOP ALL MOTION
 790     S=TOP+500:LINK #2F50:REM SPEAK           SPEAK A PHRASE
 800     @#7802=#05                               SPIN CLOCKWISE
 810     T=2                                      NUMBER OF WHOLE SECONDS
 820     GOSUB 3000                               GO WAIT
 830     DELAY 649                                MILLISECONDS TO WAIT
 840     @#7802=0                                 STOP ALL MOTION
 850     @#7802=#09                               GO FORWARD
 860     REM START A LOOP                         BEGIN A LOOP
 870     Y=@#7800                                 TEST FOR BUMPER CONTACT
 880     IF Y<255 GOTO 900                        EXIT IF ANY CONTACT
 890     GOTO 860                                 REPEAT THIS LOOP
 900     REM EXIT TO HERE
 910     @#7802=#05                               SPIN CLOCKWISE
 920     T=2                                      NUMBER OF WHOLE SECONDS
 930     GOSUB 3000                               GO WAIT
 940     DELAY 649                                MILLISECONDS TO WAIT
 950     @#7802=0                                 STOP ALL MOTION
 960     X=#BF                                    TURN OFF FLASHING LIGHTS
 970     GOSUB 3120                                GO TURN OFF A BIT
 980     GOTO 610                                 JUMP TO LABEL BEGINNING OF PAR
2999     GOTO 3400                                GOTO END
3000     FOR S=1 TO T                            .WAIT T SECONDS SUBROUTINE
```

```
Stmt     Tiny BASIC Text                        Meaning of Text

3005      DELAY 1000
3010      NEXT S
3015      RETURN
3100      U=@#7801                               TURN ON A BIT
3105      U=U OR X
3110      @#7801=U
3115      RETURN
3120      U=@#7801                               TURN OFF A BIT @#7801
3125      U=U AND X
3130      @#7801=U
3135      RETURN
3500      V=TOP+500+A:@V=P                       STORE THE PHONEME
3510      A=A+1
3520      RETURN
3600      REM END OF PROGRAM
```

** End of the Program **

APPENDIX 2

Primary Tasks Used to Compose Robot Task "PARTY"

PRIMARY TASKS USED TO COMPOSE ROBOT TASK "PARTY"

```
BEGIN A LOOP (a Task)

 1 Does Z COMPILE the BASIC statement "REM START A LOOP" which means "BEGIN A LO
OP"
 2  and COPY from "BEGIN" to STRUCTURE TYPE
 3  and SAVE new page in LOOPS
 4  and SAVE new page in STRUCTURES
 5  and END   (Task is 5% full.)


BUILD AND LOAD (a Task)

 1 Does CLEAR the page
 2  and OUTPUT from "*** Build and load an RB5X robot task ***" using format DE
AULT
 3  and CARRIAGE return
 4  and CARRIAGE return
 5  and OUTPUT from "Please enter the name of the robot task:  " using format D
FAULT
 6  and INPUT into HERE
 7  and BUILD a program from the robot task HERE
 8  and CLEAR the page
 9  and OUTPUT from "** Build completed, now loading into the RB5X **" using f
mat DEFAULT
10  and LOAD
11  and END   (Task is 16% full.)


CALL this robot task line <1>  (a Function)

 1 Does PASTE the "REM LABEL THIS LINE " in front of <1>
 2  and Z COMPILE the BASIC statement FRONT which means "LABEL THE LINE"
 3  and PASTE the "L" in front of <1>
 4  and COPY from FRONT to STRUCTURE TYPE
 5  and SAVE new page in LOOPS
 6  and END   (Task is 6% full.)


EXIT IF ANY BUMPER TOUCHED (a Task)

 1 Does Z COMPILE the BASIC statement "Y=@#7800" which means "TEST FOR BUMPER CO
NTACT"
 2  and Z COMPILE the BASIC statement "IF Y<255 GOTO EXIT" which means "EXIT
ANY CONTACT"
 3  and COPY from "EXIT" to STRUCTURE TYPE
 4  and SAVE new page in LOOPS
 5  and SAVE new page in STRUCTURES
 6  and END   (Task is 10% full.)
```

```
SPEAK the phrase called:  <1>  (a Function)

 1 Does BEGIN at page indexed by FIRST in folder SPEECH LOAD DATA
 2  and GET the page indexed by <1> in folder SPEECH LOAD DATA
 3  and COPY from SPEECH ADDRESS to HERE
 4  and PASTE the "S=" in front of HERE
 5  and PASTE the FRONT in front of ":LINK #2F50"
 6  and PASTE the FRONT in front of ":REM SPEAK"
 7  and PASTE the FRONT in front of <1>
 8  and Z COMPILE the BASIC statement FRONT which means "SPEAK A PHRASE"
 9  and END   (Task is 9% full.)


SPIN.CLOCKWISE this many degrees: <1>  (a Function)

 1 Does IF the <1>  IS GREATER THAN  0 then
 2   Do    MULTIPLY the <1> by .01472
 3  and    SPLIT the number PRODUCT
 4  and    TAKE this many characters 4 from FRACTION
 5  and    PASTE the INTEGER in front of FRONT
 6  and    GO CLOCKWISE
 7  and    WAIT this many seconds FRONT
 8  and    STOP ALL MOTION
 9  and    END of test
10  and END   (Task is 5% full.)


STOP ALL MOTION (a Task)

 1 Does Z COMPILE the BASIC statement "@#7802=0" which means "STOP ALL MOTION"
 2  and END   (Task is 3% full.)


TURN OFF THE FLASHING LIGHTS (a Task)

 1 Does Z COMPILE the BASIC statement "X=#BF" which means "TURN OFF FLASHING L
HTS"
 2  and Z COMPILE the BASIC statement "GOSUB 3120" which means " GO TURN OFF A
IT"
 3  and Z INSURE PORT 7801 SUBROUTINE IS
 4  and END   (Task is 7% full.)


TURN ON THE FLASHING LIGHTS (a Task)

 1 Does Z COMPILE the BASIC statement "X=#40" which means "TURN ON FLASHING LI
TS"
 2  and Z COMPILE the BASIC statement "GOSUB 3100" which means " GO TURN ON A
T"
 3  and Z INSURE PORT 7801 SUBROUTINE IS
 4  and END   (Task is 7% full.)


X LOAD THE INTRODUCTION (a Task)

 1 Does REMARK that: "RB5X say the introduction - new voice cards only!"
 2  and LOAD THE PHRASE called: "SAY THE INTRODUCTION" with phonemes: "H.EH1.EH
.L.O1.U1.AH1.EH3.I3.Y.PA1.PA1.AE1.EH3.M.THV.UH1.AH1.UH2.ER.B.E1.AY.Y.F.AH1.EH3.
.V.PA1.EH1.EH2.K.PA0.S.I1.N.T.EH1.EH3.L.I1.I3.D.J.EH1.EH3.N.T.R.O1.U1.B.AH1.UH3
T.STOP."
 3  and END   (Task is 25% full.)
```

```
EXIT IF SONAR distance value is less than <1>  (a Function)

 1 Does Z COMPILE the BASIC statement "D=0" which means "ZERO DISTANCE"
 2  and Z COMPILE the BASIC statement "LINK #1800" which means "CALL SONAR"
 3  and SPLIT the number <1>
 4  and PASTE the "IF D<" in front of INTEGER
 5  and PASTE the FRONT in front of " GOTO EXIT"
 6  and Z COMPILE the BASIC statement FRONT which means "EXIT IF LESS THAN VA
"
 7  and COPY from "EXIT" to STRUCTURE TYPE
 8  and SAVE new page in LOOPS
 9  and COPY from YES to SONAR NEEDED
10  and SAVE new page in STRUCTURES
11  and END   (Task is 14% full.)


JUMP to the line called <1>  (a Function)

 1 Does PASTE the "JUMP TO LABEL " in front of <1>
 2  and Z COMPILE the BASIC statement "GOTO GOES HERE" which means FRONT
 3  and PASTE the "J" in front of <1>
 4  and COPY from FRONT to STRUCTURE TYPE
 5  and SAVE new page in LOOPS
 6  and END   (Task is 6% full.)



MOVE FORWARD (a Task)

 1 Does Z COMPILE the BASIC statement "@#7802=#09" which means "GO FORWARD"
 2  and END   (Task is 3% full.)



PREPARE THE ROBOT (a Task)

 1 Does Z COMPILE the BASIC statement "@#7803=#98" which means "INITIALIZE I/
 2  and END   (Task is 3% full.)

PREPARE THE VOICE (a Task)

 1 Does Z COMPILE the BASIC statement "@#780B=#A7:S=TOP" which means "PREPARE .
E VOICE"
 2  and COPY from "YES" to VOICE NEEDED
 3  and END   (Task is 5% full.)



REPEAT THIS LOOP (a Task)

 1 Does COPY from STATEMENT NUMBER to HERE
 2  and Z GET PREVIOUS BEGIN
 3  and DELETE the page indexed by STATEMENT NUMBER from folder LOOPS
 4  and PASTE the "GOTO " in front of STATEMENT NUMBER
 5  and COPY from HERE to STATEMENT NUMBER
 6  and Z COMPILE the BASIC statement FRONT which means "REPEAT THIS LOOP"
 7  and Z COMPILE the BASIC statement "REM EXIT TO HERE" which means ""
 8  and Z PATCH UP EXIT STATEMENTS
 9  and COPY from "REPEAT" to STRUCTURE TYPE
10  and SAVE new page in STRUCTURES
11  and END   (Task is 10% full.)
```

APPENDIX 3

Provided RCL Tasks

```
 TASKS
 Name        (Status)

 ASSIGN A MOTOR CODE
*BEGIN A COUNTED LOOP called <1> beginning at <2> ending at <3>
*BEGIN A LIMITED LOOP
*BEGIN A LOOP
*BUILD a program from the robot task <1>
*BUILD AND LOAD
*CALCULATE variable <1> = <2> (+,-,*,/) <3>  <4>
*CALL this robot task line <1>
 CLEAR ALL ITEMS
*DEFINE A VARIABLE called <1>
 END HERE
 END OF PROGRAM
*EXIT IF ANY BUMPER TOUCHED
*EXIT IF FRONT+REAR BUMPER PRESS
*EXIT IF SONAR distance value is less than <1>
*EXIT IF THE BATTERY IS LOW
*EXIT IF THE CHARGER IS TOUCHED
*EXIT IF THE TAPE IS SENSED
*EXIT IF THIS BUMPER is touched <1>
*EXIT THIS LOOP
*FOLLOW TAPE
*GO CLOCKWISE
*GO COUNTERCLOCKWISE
*HONK the horn for this many seconds <1>
 INITIALIZE MEMORY
 INITIALIZE VARIABLES
*JUMP to the line called <1>
*LIGHTS ROUTINE
*LIST THE PROGRAM
*LOAD
*LOAD THE PHRASE called: <1> with phonemes: <2>
*MAINTAIN CHARGE
*MOVE BACKWARD
*MOVE DISTANCE BACKWARD for this many feet: <1>
*MOVE DISTANCE FORWARD for this many feet: <1>
*MOVE FORWARD
*MOVE FORWARD TIL TAPE NOT SENSED
*MOVE RANDOMLY
*MOVE THE ARM FROM SHOULDER (UP,DOWN,IN,OUT): <1> for this many degrees <2>
*MOVE THE FOREARM (IN,OUT): <1> for this many degrees <2>
*MOVE THE HAND (OPEN,CLOSE): <1> for this (1-100) %: <2>
*MOVE TIMED BACKWARD for this many seconds <1>
*MOVE TIMED FORWARD for this many seconds <1>
*MOVE WITH BETA INTELLIGENCE
 OTHERWISE DO
*PICK A RANDOM DIRECTION
*PIVOT ON LEFT CLOCKWISE
*PIVOT ON LEFT COUNTERCLOCKWISE
*PIVOT ON RIGHT CLOCKWISE
*PIVOT ON RIGHT COUNTERCLOCKWISE
*PREPARE THE ROBOT
*PREPARE THE VOICE
 REMARK that: <1>
*REPEAT THE LIMITED LOOP unless <1> is (<,>,=,<>) <2> to <3>
*REPEAT THIS COUNTED LOOP for the counter <1>
*REPEAT THIS LOOP
 REPORT ASSIGN ERROR
```

```
 TASKS
 Name        (Status)

 REPORT DELETE ERROR
 REPORT GET ERROR  (Undefined)
 REPORT LOAD ERROR
 REPORT MATH ERROR
 REPORT REPLACE ERROR
 REPORT SAVE ERROR
*RUN
*SET the variable <1> equal to <2>
 SHOW ERRORS
 SHOW THE PHONEME DICTIONARY
*SPEAK the phrase called:  <1>
*SPIN AROUND CLOCKWISE this many times: <1>
*SPIN AROUND COUNTERCLOCKWISE this many times: <1>
*SPIN CLOCKWISE this many degrees: <1>
*SPIN COUNTERCLOCKWISE this many degrees: <1>
*SPIN LEFT 90 DEGREES
*SPIN RIGHT 90 DEGREES
 STARTUP
*STOP ALL MOTION
*TEST IF the variable <1> is (=,<,>,<>) <2> compared to <3>
*TURN OFF LED number <1>
*TURN OFF THE FLASHING LIGHTS
*TURN OFF THE HORN
*TURN OFF THE INFRARED LED
*TURN ON LED number <1>
*TURN ON THE FLASHING LIGHTS
*TURN ON THE HORN
*TURN ON THE INFRARED LED
*TURN THE WRIST (CW, CCW): <1> for this many degrees <2>
*WAIT this many seconds <1>
*WAIT RANDOMLY up to this many seconds <1>
*X ALPHA
*X ALPHA W/SONAR
*X BETA
*X BETA W/SONAR
*X BUMPER ACTIVATED MOTIONS
*X CHARGER FINDER
*X LOAD THE INTRODUCTION
*X SIMPLE SIMON
 Z COMPILE the BASIC statement <1> which means <2>
 Z COMPUTE 2 TO THE N-1 , with N = <1>
 Z DELAY for this many loops:  <1>
 Z DO CHECK if item <1> is in folder <2>
 Z DO NUMBER TEST on <1>
 Z EMPTY THE PROGRAM
 Z ERROR CHECK IF/UNTIL STATEMENT
 Z ERROR CHECK MATH FUNCTION
 Z ERROR CHECK STRUCTURES
 Z ERROR CHECK THE FOR STATEMENT
 Z ERROR CHECKING?
 Z FIND the label called <1>
 Z GET PREVIOUS BEGIN
 Z GET PREVIOUS IF OR ELSE
 Z INCLUDE at <1> the statement <2> which means <3>
 Z INCREMENT the <1>
 Z INCREMENT STATEMENT NUMBER
 Z INITIALIZE LEGAL VARIABLES
```

```
TASKS
Name        (Status)

Z INSURE ARM PULSE SUBROUTINE IS
Z INSURE BETA SUBROUTINE IS
Z INSURE END OF PROGRAM IS THERE
Z INSURE PORT 7801 SUBROUTINE IS
Z INSURE RANDOM TURN SUBROUTINE
Z INSURE RANDOM WAIT SUBROUTINE
Z INSURE VOICE SUBROUTINE IS
Z INSURE WAIT SUBROUTINE IS
Z LOAD SONAR
Z LOAD VOICE SUBROUTINE
Z LOOK FOR BEGIN
Z LOOK FOR MATCHING STATEMENT
Z OUTPUT PAGE NUMBER
Z OUTPUT PROGRAM HEADING
Z PATCH UP END STATEMENTS
Z PATCH UP EXIT STATEMENTS
Z PATCH UP JUMP STATEMENTS
Z SAVE FOLDER ERROR
Z SAVE NUMBER OR FOLDER ERROR
Z SAVE STRUCTURE ERROR missing the statement <1> to go with the statment <2>
statement number <3>
ZZ AVAIL TASK
ZZ AVAIL TASK #1
ZZ AVAIL TASK #10
ZZ AVAIL TASK #11
ZZ AVAIL TASK #12
ZZ AVAIL TASK #13
ZZ AVAIL TASK #14
ZZ AVAIL TASK #2
ZZ AVAIL TASK #3
ZZ AVAIL TASK #4
ZZ AVAIL TASK #5
ZZ AVAIL TASK #6
ZZ AVAIL TASK #7
ZZ AVAIL TASK #8
```

APPENDIX 4

Your Own Application

YOUR OWN APPLICATION

Now that your have used this Tutorial and have seen how to go about writing a robot task for your RB5x, we hope you will use the form below for creating your own robot tasks.

Scenario

Building Blocks	Programming

ROBOT CONTROL LANGUAGE WITH SAVVY
PROGRAMMER'S MANUAL

TABLE OF CONTENTS

A. Overview.....A-1

B. Structure.....B-1

- Action Commands.....B-1
- Programming Commands.....B-1
- Parameter-Passing Functions.....B-1
- Error-Checking and Compiler Routines.....B-1
- Vocabulary Elements in Robot Tasks.....B-2
- RCL and Its Relationship to Tiny BASIC.....B-3
- Compiling RCL into Tiny BASIC.....B-5

C. Implementation.....C-1

- Instructions for Keying in the Sample Robot Task.....C-1
- Writing New Robot Tasks.....C-3
- Extending RCL and Defining New Actions.....C-5
- Modifying Existing RCL Tasks.....C-6

D. Debugging and Error Trapping.....D-1

E. Miscellaneous Programmer's Notes.....E-1

F. Appendices.....F-1

- Appendix 1: Provided RCL Tasks
- Appendix 2: Detail of Provided RCL Tasks

Figures

1. Hardware and Language Relationships.....B-3
2. Language Structures and Relationships.....B-4

Copyright 1984 All rights reserved.

A. OVERVIEW

Robot Control Language(tm) with Savvy(R) -- RCL(tm) for short -- provides a simple, straightforward method of programming the RB5X(tm) to perform many functions without using the terse Tiny BASIC code that ultimately controls the robot's actions.

RCL users instruct the RB5X using "robot tasks" like MOVE FORWARD, GO CLOCKWISE, TURN THE WRIST, TURN ON THE HORN, etc. RCL then automatically cross-compiles the robot task, line by line, into Tiny BASIC code and then downloads the code into the robot's on-board memory by means of an RS-232 serial communications interface.

In addition to commands that initiate and control RB5X's action, RCL also includes facilities for error trapping and compilation of loops. Each element of RCL is as meaningful to Savvy as any Savvy primary language command such as ADD, EDIT, or SAVE. Furthermore, the system is flexible, to allow the addition of new RCL commands, and the editing and modification of all RCL elements and associated Savvy tasks. Since RCL is Savvy-based, all robot vocabulary can be extended with synonyms for existing commands.

This document presumes the reader has gone through the RCL Tutorial, has reviewed the Savvy Personal Language Reference Folio, has fundamental Savvy task development skills, and possesses at least a superficial knowledge of NSC Tiny BASIC. (If you need more information about how to use Tiny BASIC, consult the RB5X Reference Manual.)

Copyright 1984 All rights reserved.

B. STRUCTURE

Accompanying this document is a complete directory of all RCL vocabulary. The vocabulary has four main elements: action commands, programming commands, parameter-passing functions, and error-checking and compiler routines.

ACTION COMMANDS

These commands instruct the robot to perform an activity.

Examples:

```
FOLLOW TAPE
GO CLOCKWISE
MOVE BACKWARD
```

PROGRAMMING COMMANDS

These commands are used for initialization, error checking, looping, and other programming conventions.

Examples:

```
LOAD
REPORT ASSIGN ERROR
REPEAT THIS LOOP
```

PARAMETER-PASSING FUNCTIONS

These functions pass specific parameters to the robot, sometimes in the form of distances to move, degrees to rotate, phrases to speak, etc. Other parameter-passing functions are used strictly as programming conventions, such as counts for a loop.

Examples:

```
Parameter-passing functions as robot
instructions:

MOVE DISTANCE BACKWARD for this many feet: <1>
MOVE TIMED FORWARD for this many seconds: <1>

Parameter-passing functions for programming:

DEFINE A VARIABLE called <1>
REPEAT THIS COUNTED LOOP for the counter <1>
```

ERROR-CHECKING AND COMPILER ROUTINES

All RCL vocabulary elements that are preceded by a 'Z' are for error checking and compilation.

Copyright 1984 All rights reserved.

Examples:

```
Z ERROR CHECK MATH FUNCTION
Z LOOK FOR MATCHING STATEMENT
Z LOAD SONAR
```

There are several elements in the directory preceded by an 'X'. These are sample RCL robot tasks (i.e., X Simple Simon).

Each element of the RCL vocabulary -- action commands, programming commands, parameter-passing functions, and the error-checking and compiler routines -- has a corresponding Savvy task or function that actually executes the task. These tasks and functions may also call other tasks and functions, and may in turn be called by other tasks. Their operation is completely transparent to the end-user. (Appendix 2 contains the complete text of all RCL tasks and functions.)

VOCABULARY ELEMENTS IN ROBOT TASKS

A user composing a robot task selects action commands, programming commands, and parameter-passing functions to form a robot task, which is also an actual Savvy task. The task can be edited with the Savvy editor and is saved on disk just like any other Savvy task. The task name becomes part of RCL. The X FIND CHARGER ROUTINE listed as Sample 1 is an example of a simple task. It has looping, parameter passing, and other programming conventions.

Sample 1

```
X CHARGER FINDER (a Task)

1 Does WAIT this many seconds 20
2  and PREPARE THE ROBOT
3  and INITIALIZE MEMORY
4  and BEGIN A LOOP
5  and PREPARE THE ROBOT
6  and CLEAR ALL ITEMS
7  and BEGIN A LOOP
8  and MOVE FORWARD
9  and EXIT IF ANY BUMPER TOUCHED
10  and EXIT IF THE TAPE IS SENSED
11  and REPEAT THIS LOOP
12  and TEST IF the variable "R" is (=,<,>,<>) "=" to 0
13  and EXIT THIS LOOP
14  and END HERE
15  and MOVE WITH BETA INTELLIGENCE
16  and REPEAT THIS LOOP
17  and FOLLOW TAPE
18  and MAINTAIN CHARGE
19  and END  (Task is 7% full.)
```

 Copyright 1984 All rights reserved.

From this set of instructions, RCL compiles the Tiny BASIC text necessary to accomplish the specified activity or function. The error-checking and compiler routines are used as necessary in the actual Savvy tasks that correspond to RCL elements.

RCL AND ITS RELATIONSHIP TO TINY BASIC

Programming the RB5X in RCL first requires communication between two different hardware configurations, the 6502-based Apple II and the INS8073-based robot. The diagram in Figure 1 illustrates the hierarchy of languages that perform this communication and their relationship. (FORTH is the machine language that interfaces Savvy with the 6502.)

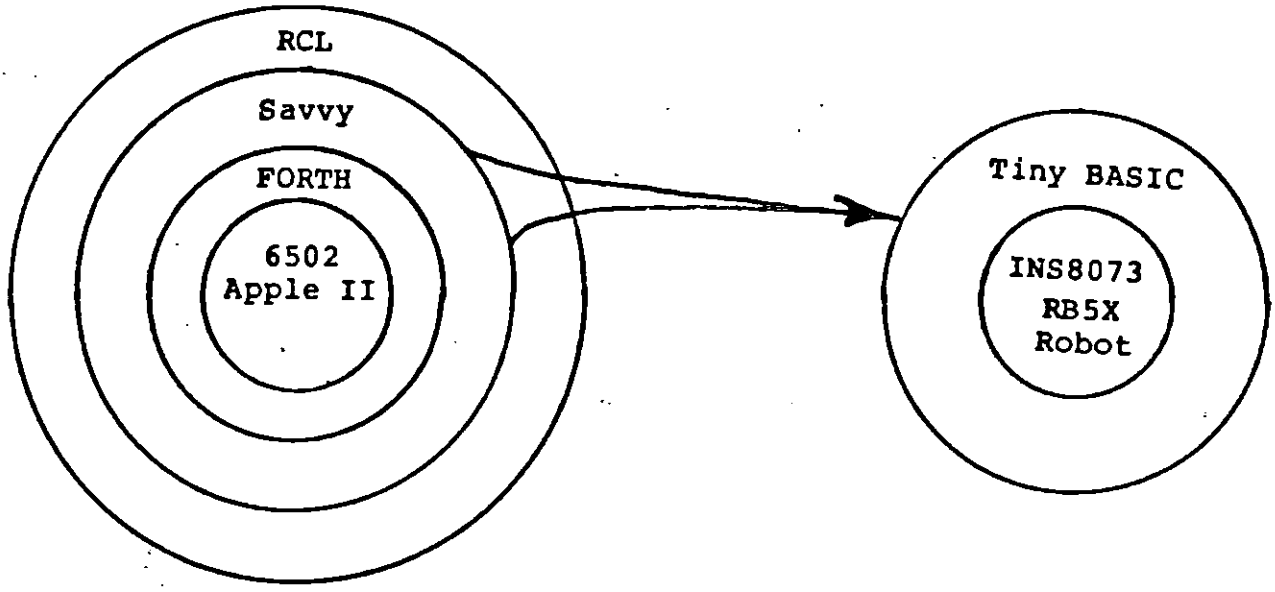

Figure 1.
Hardware and Language Relationships

Copyright 1984 All rights reserved.

As illustrated in Figure 1, Tiny BASIC communicates with the robot hardware. Because of its machine code-like nature, many Tiny BASIC instructions are necessary to implement an RCL command. Figure 2 illustrates the language structures and relationships.

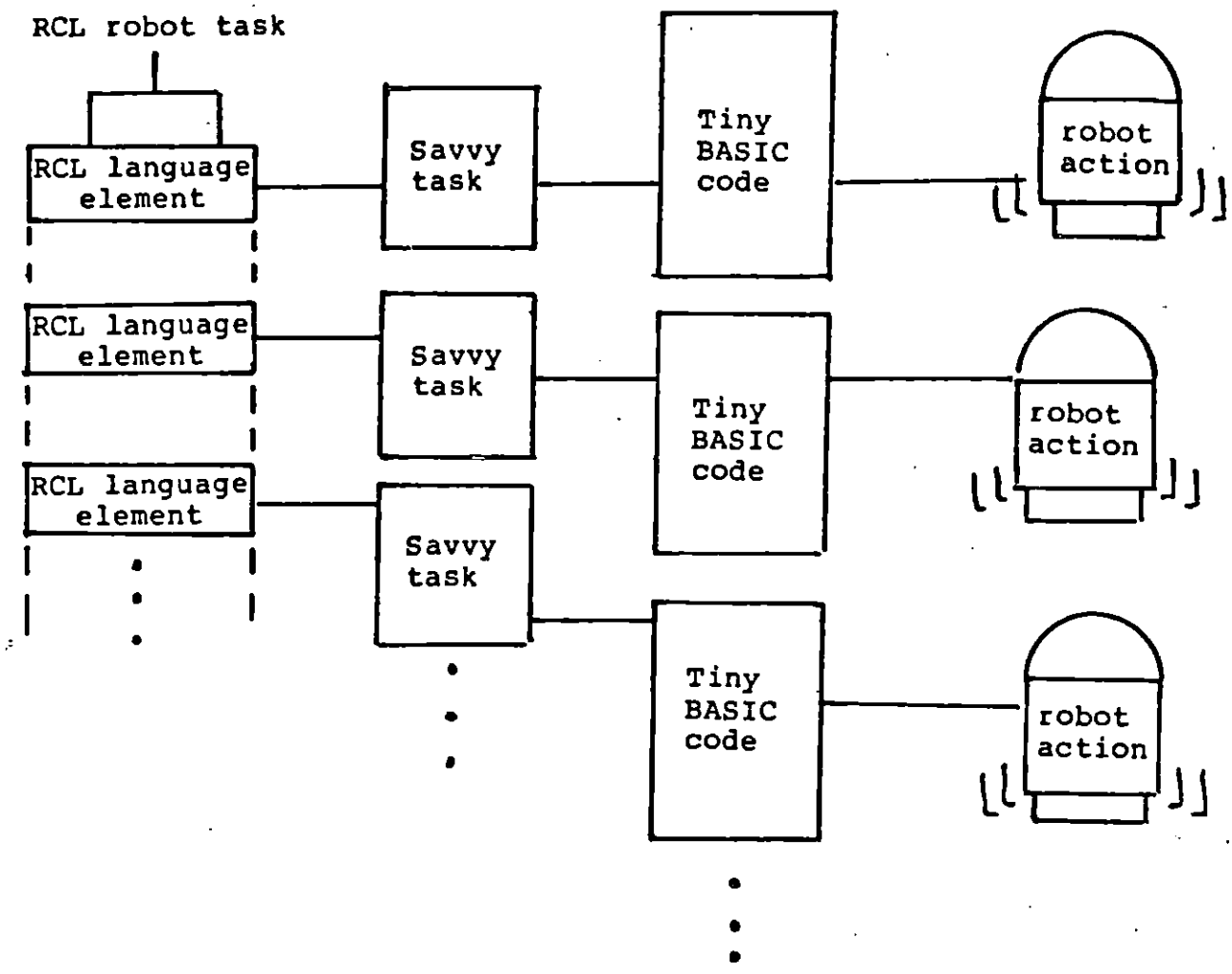

Figure 2.
Language Structures and Relationships

 Copyright 1984 All rights reserved.

COMPILING RCL INTO TINY BASIC

The Tiny BASIC program text is stored in a Savvy folder named SOURCE. It is indexed (keyed) by the item STATEMENT NUMBER and the two items BASIC TEXT and MEANING. BASIC TEXT is the Tiny BASIC text, a combination of symbols and numbers, such as @#7803=#98, that specify action, define addresses, turn bits on and off, etc. MEANING is the documentary description that accompanies the code.

The folder structure for stored Tiny BASIC text in RCL is:

```
SOURCE (a folder)

1 Id STATEMENT NUMBER
2 Item BASIC TEXT
3 Item MEANING
```

A Savvy task and function perform the compilation according to the following process before the data is actually stored in the SOURCE folder.

1. Increment the current line number to form the STATEMENT NUMBER for this statement

2. Copy the appropriate piece of text to the BASIC TEXT

3. Copy any textual description to MEANING

4. Save this page of data in the SOURCE folder

The following tasks compile RCL according to the preceding algorithm:

Sample 2

Z COMPILE the BASIC statement <1> which means <2> (a Function)

```
 1 Does Z INCREMENT STATEMENT NUMBER
 2  and COPY from <1> to BASIC TEXT
 3  and COPY from <2> to MEANING
 4  and SAVE new page in SOURCE
 5  and END  (Task is 2% full.)
```

Z INCREMENT STATEMENT NUMBER (a Task)

```
 1 Does ADD the STATEMENT NUMBER and STATEMENT NUMBER INCREMENT
   VALUE
 2   and COPY from SUM to STATEMENT NUMBER
 3   and END  (Task is 1% full.)
```

Copyright 1984 All rights reserved.

C. IMPLEMENTATION

RCL is flexible and extensive enough to allow users to program the RB5X to perform countless activities. The best way to understand how the RB5X works and how robot tasks are composed is to actually key one in. By entering the following robot task (Sample 3), you learn how to hook up the robot, how to prepare its memory, how to download the robot task from your Apple to your RB5X, and other aspects of "training" the RB5X.

INSTRUCTIONS FOR KEYING IN THE SAMPLE TASK

Composing a robot task is just like defining a regular Savvy task. (Again, these instructions assume you have a basic familiarity with Savvy and understand the way Savvy prompts for input.) In the following dialogue, the user's response is in upper case and Savvy's prompts are in lower case, just as in actual use if you have an 80-column card or an Apple IIe.

1. Begin by defining a task called TEST PROGRAM.

   DEFINE a task called TEST PROGRAM

2. Type the text, beginning with line 1, just as it is listed in Sample 3 on page C-2. Remember that all lower case words in the script are Savvy prompts that RCL automatically inserts after you press the RETURN key.

3. Check the task for errors and correct with the Savvy editor.

4. If the robot task appears satisfactory, instruct Savvy to BUILD the source task. Savvy prompts for the task name.

   BUILD a program from the robot task "TEST PROGRAM"

   First the system displays the following message:

   Would you like your robot task checked for errors? (Y/N).

   After your response, the system displays the message:

   Now building robot task TEST PROGRAM

   ...one moment, please.

   To see the Tiny BASIC code that the BUILD command creates, type LIST THE PROGRAM and you see the Tiny BASIC with comments.

Copyright 1984 All rights reserved.

5. Plug the Apple into the RB5X via the RS-232 serial interface and switch the robot on. Type:

   LOAD

   The drive spins and RCL displays the message:

   ** Now loading...One moment please **

   The system then displays the following message:

   Now loading your robot task...###

6. After RCL downloads the completed robot task, the RB5X executes the TEST PROGRAM. If the drives do not spin and the robot does not respond, the baud rate may be incorrect (it must be set at the same rate on both robot and serial interface), the serial card may not be connected properly, or the robot may be inadequately charged.

7. To run again, type:

   RUN<RET>

Sample 3

```
TEST PROGRAM (a Task)

 1 Does PREPARE THE ROBOT
 2  and TURN ON THE FLASHING LIGHTS
 3  and MOVE DISTANCE FORWARD for this many feet 2
 4  and SPIN LEFT 90 DEGREES
 5  and SPIN RIGHT 90 DEGREES
 6  and MOVE TIMED BACKWARD for this many seconds 5
 7  and PIVOT ON LEFT CLOCKWISE
 8  and WAIT this many seconds 4
 9  and PIVOT ON LEFT COUNTERCLOCKWISE
10  and WAIT this many seconds 4
11  and PIVOT ON RIGHT CLOCKWISE
12  and WAIT this many seconds 4
13  and PIVOT ON RIGHT COUNTERCLOCKWISE
14  and WAIT this many seconds 4
15  and STOP ALL MOTION
16  and HONK the horn for this many seconds 2
17  and END  (Task is 8% full.)
```

Remember that you type Q<RET> to complete the robot task and then RCL replies with the END line.

 Copyright 1984 All rights reserved.

WRITING NEW ROBOT TASKS

As mentioned in the preceding instructions, creating a new robot task is just like defining a new Savvy task. However, instead of using Savvy commands and primaries directly, RCL is used to specify instructions to the robot. Each robot task must begin with the directive:

```
PREPARE THE ROBOT
```

This initializes the RB5X hardware and prepares it to accept the robot task that has been compiled into Tiny BASIC. After the PREPARE THE ROBOT command has been entered, other RCL language elements can then be combined to define a robot task.

Simple robot tasks such as the TEST PROGRAM illustrated in Sample 3 do not include loops or conditional tests, nor do they require the use of the special compiler routines. Other, more complicated robot tasks use these conventions. When composing robot tasks with looping conventions, remember that loops may be nested no more than eight deep in Tiny BASIC.

Sample 4 is a robot task that is more complex than TEST PROGRAM and illustrates looping conventions. Try it out to begin to get a feel for the way the language works.

Copyright 1984 All rights reserved.

Sample 4

```
TEST LOOPS (a Task)

 1 Does CALL this line "START OVER"
 2  and PREPARE THE ROBOT
 3  and BEGIN A LOOP
 4  and DEFINE A VARIABLE called "L"
 5  and EXIT IF ANY BUMPER TOUCHED
 6  and REPEAT THIS LOOP
 7  and BEGIN A COUNTED LOOP called "L" beginning
        at 1 ending at 5
 8  and TURN ON THE FLASHING LIGHTS
 9  and WAIT this many seconds 1
10  and TURN OFF THE FLASHING LIGHTS
11  and wait this many seconds 1
12  and REPEAT THIS COUNTED LOOP for the counter "L"
13  and BEGIN A LOOP
14  and MOVE FORWARD
15  and EXIT IF SONAR distance value is less than 95
16  and REPEAT THIS LOOP
17  and MOVE BACKWARD
18  and SET the variable "L" equal to 0
19  and BEGIN A LOOP
20  and CALCULATE variable "L" = "L" (+,-,*,/) "+" 1
21  and HONK the horn for this many seconds 1
22  and WAIT this many seconds 1
23  and TEST IF the variable "L" is (=,<,>,<>) "=" compared
        to 4
24  and REPEAT THIS LOOP
25  and STOP ALL MOTION
26  and JUMP to the line called "START OVER"
27  and END HERE
28  and END  (Task is 19% full.)
```

After new tasks are written, you may go through the BUILD procedure described in the section on "Instructions for Keying in the Sample Task." However, in certain cases you may want to skip the two-part BUILD and LOAD procedure and just type:

```
BUILD AND LOAD
```

This RCL primary task performs the BUILD and the LOAD operations consecutively after the robot task title is entered, without querying again. The BUILD AND LOAD routine does not, however, have the same error-checking faculties as the separate BUILD and LOAD tasks, and we do not advise using it on new tasks.

 Copyright 1984 All rights reserved.

EXTENDING RCL AND DEFINING NEW ACTIONS

Extending RCL and defining new actions, either to expand the language or to accommodate new hardware, requires a knowledge of Tiny BASIC. The proper Tiny BASIC code must be inserted in the source folder to provide communication with the hardware.

Implementing new RCL primary tasks begins with design at the Tiny BASIC level. When the fundamental instructions are coded, they may be incorporated into Savvy tasks. Certain existing RCL elements make it easy to write one-liners. These are Z INCREMENT STATEMENT NUMBER and COMPILE the BASIC statement <1> which means <2>. (The text of these routines is listed in the section "Compiling RCL into Tiny BASIC.")

For example, a task such as STOP ALL MOTION is done like this:

```
STOP ALL MOTION (a task)

1 Does COMPILE the BASIC statement "@#7802=0" which means
  "STOP ALL MOTION"
```

Some new primary tasks require parameterized subroutines. For example, if you were implementing a siren, you would write a routine similar to TURN ON THE HORN listed in Sample 5.

The first line compiles the Tiny BASIC code; the second is a subroutine to turn on a bit. It is necessary to ensure that the target subroutine is there, so a task is Z INSURE PORT 7801 SUBROUTINE IS. The text of this task is listed as Sample 6.

Sample 5

```
TURN ON THE HORN (a Task)

 1 Does Z COMPILE the BASIC statement "X=#80" which means
   "TURN ON HORN"
 2  and Z COMPILE the BASIC statement "GOSUB 3100" which
    means "GO TURN ON A BIT"
 3  and Z INSURE PORT 7801 SUBROUTINE IS
 4  and END  (Task is 6% full.)
```

Copyright 1984 All rights reserved.

Sample 6

```
Z INSURE PORT 7801 SUBROUTINE IS (a Task)

 1 Does WHEN folder SOURCE DOES NOT CONTAIN 3,100 then
 2  Do COPY from STATEMENT NUMBER TO HERE
 3  and Z INCLUDE at 3,100 the statement "U=@#7801"
        which means "TURN ON A BIT"
 4  and Z INCLUDE at 3,105 the statement "U=U OR X"
        which means ""
 5  and Z INCLUDE at 3,110 the statement "@#7801=U"
        which means ""
 6  and Z INCLUDE at 3,115 the statement "RETURN"
        which means ""
 7  and Z INCLUDE at 3,120 the statement "U=@#7801"
        which means "TURN OFF A BIT"
 8  and Z INCLUDE at 3,125 the statement "U=U AND X"
        which means ""
 9  and Z INCLUDE at 3,130 the statement "@#7801=U"
        which means ""
10  and Z INCLUDE at 3,135 the statement "RETURN"
        which means ""
11  and COPY from HERE to STATEMENT NUMBER
12  and END of test
13  and END  (Task is 23% full.)
```

Since RCL is Savvy-based, the existing commands can be "extended with synonyms" using the Savvy ASSOCIATE command. These might be abbreviations of certain frequently used commands to speed up data entry, such as TOH for TURN ON THE HORN. And, because of Savvy's Adaptive Pattern Recognition Processing (APRP), the more synonyms added to RCL, the more Savvy is able to "help out" in cases of misspelling or improperly worded phrases.

MODIFYING EXISTING RCL TASKS

The Savvy tasks that actually implement the RCL commands may be modified to suit your needs.

For example: If you find it tiresome to begin each robot task with PREPARE THE ROBOT, you can insert the Tiny BASIC initialization code (@#7803=#98) in the LOAD routine so that RCL automatically does this initialization. If you don't like using periods as delimiters in the phonemes, you can change them to slashes (/) by instructing Savvy to COPY from "/" to DELIMITER. You can change display messages with Savvy's editor or modify the RUN task to allow you to communicate "interactively" with the robot. (Add wait states so you must type RUN at the keyboard after each command as a useful debugging facility.)

Of course, before you begin modifiying the Savvy tasks, be sure you have a recent backup of your disk so you can restore the task to its original state if there are any problems.

 Copyright 1984 All rights reserved.

D. DEBUGGING AND ERROR TRAPPING

Debugging and error trapping in new robot tasks and primary tasks takes place at three levels:

Robot task level (RCL level)
Savvy task level
Tiny BASIC level

At the highest level, the robot task level, there are error-trapping routines in Savvy itself that check new scripts for language errors, matching statement errors, errors of correspondence between arguments and functions, etc. These routines do not check for logic errors. When RCL detects an error, it saves the error and displays the number of the offending line.

Some errors that are returned may not interfere with the purpose of the script. In fact, some errors returned may be things you intended to do. Endless loops are usually considered a bug in most programming situations; however, endless loops to carry on a specified motion may be something you intend the RB5X to do.

When a Savvy script is modified and then does not function as intended, check all other routines that are called by and from the script to check for errors. Because of the reentrant nature of these tasks, bugs may not necessarily show up in the modified task. If an unmodified task that previously ran without problems begins to behave in a peculiar fashion, check to see if the tasks it calls or is called by have been recently modified.

Debugging new tasks that extend RCL must begin at the Tiny BASIC level. Be sure the Tiny BASIC code works before introducing it into the task that compiles it. If it works at that level, test at the Savvy task level, and finally at the robot task level. Debugging Tiny BASIC is not an activity for the casual or inexperienced programmer.

Copyright 1984 All rights reserved.

E. MISCELLANEOUS PROGRAMMER'S NOTES

Listed below are various notes concerning language use and conventions and other programming information.

1. JUMP to the line called <1>

   CALL this line <1>

   These may be nested in any loop and the label name must be in quotes.

2. MOVE THE ARM FROM THE SHOULDER (UP, DOWN, IN, OUT): <1> for this many degrees: <2>

   MOVE THE FOREARM (IN,OUT): <1> for this many degrees: <2>

   MOVE THE HAND (OPEN, CLOSE): <1> for this (1-100) %:<2>

   TURN THE WRIST (LEFT, RIGHT): <1> for this many degrees:<1>

   The first argument in all of these functions does not have to be enclosed in quotation marks. If the second argument is incorrect, a warning message appears on the screen and the BUILD stops.

3. PREPARE THE VOICE -- Initializes the voice

4. LOAD the phrase called <1> with phonemes <2>

   For example:

   ```
   LOAD THE PHRASE called "HELLO" with the
   phonemes "H.E1....STOP."
   ```

   Each phoneme must have a period after it, including the last phoneme in the sequence. There can be no spaces in between. If any periods are omitted, the machine goes into an endless loop. The last phoneme must always be STOP.

   The "LOAD THE PHRASE" action should be near the top of your robot task, but after the "PREPARE THE VOICE" command. You should not include the "LOAD THE PHRASE" task with any of your loops since it only needs to be done once. Use the "SPEAK the phrase called <1>" in the body of your task when you want the robot to speak the phrase you previously loaded.

Copyright 1984 All rights reserved.

5. In the following tasks:

   DEFINE A VARIABLE

   BEGIN A COUNTED LOOP

   REPEAT THIS COUNTED LOOP

   CALCULATE

   REPEAT A LIMITED LOOP

   TEST IF

   the variable used must be a legal Tiny BASIC variable (A-Z).

6. CALCULATE

   "CALCULATE" allows the programmer to alter a Tiny BASIC variable by operating upon it with one of the four standard arithmetic functions (add, subtract, multipy, divide) along with either another variable or a constant value. A common use of this task is to increment a variable for use in a loop. For example:

   CALCULATE variable "A" = "A" "+" 1

   In this case "A" is the variable to be operated on. The variable is in quotation marks because Savvy must treat the Tiny BASIC variable as a literal. The numeral, on the other hand, does not require quotes since it is a constant in both Savvy and in Tiny BASIC. Likewise, the operand must be placed in quotes since Savvy does not recognize it as a constant.

7. TEST IF

   "TEST IF" gives the programmer the option of being able to test a variable against another variable or a constant value while in a loop. The result of such a test will be either true or false. If the result is true, then the loop continues; if the result is false, the task branches out of the loop statement immediately following the end of the loop. For example:

```
BEGIN A LOOP
   CALCULATE variable "A" = "A" (+,-,*,/) "+" 1
   TEST IF the variable "A" is (=,<,>,<>) "=" compared
   to 4
REPEAT THIS LOOP
END HERE
```

 Copyright 1984 All rights reserved.

In this procedure, the loop will continue until the variable A is equal to four; when this occurs, the task terminates.

8. When using variables in your tasks, such as for the TEST IF task, the CALCULATE task, or the COUNTED LOOPS, be careful about using the following variables; they are already being used within some of the RCL primary tasks:

   A, D, I, J, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z

   There may or may not be interaction between your choice of variable and the use of that variable within one of the tasks, so it is best to use other variable names unless you check the tasks being used, or look at the LIST THE PROGRAM.

Copyright 1984 All rights reserved.