INTRODUCTION . . .

Thank you for purchasing Ross Software's Utility 4K Eprom. This has been programmed to the highest standards . . .

It has the following enhancements:

1200,600 Baud Cassette Operating System, with visible and audible indication of load and save, flashing cursor,

and 33 new commands.

HOW TO INSERT THE 2532 EPROM

The '2532 Eprom' comes in protective material,

. . .DO NOT TAKE IT OUT OF THIS YET. . .

1) First thing to do is switch off, turn the Atom upside down with the keyboard towards you and remove the bottom by undoing the two screws.

2) Now find 'IC 24' –
This is a 24 pin socket in the centre of two large 40 pin chips (6502) and (8255), and above 'IC 24' are two small 16 pin chips (74LS138) and (7445).

3) When located, insert the '2532 Eprom' with the round notch on the '2532 Eprom' away from you. Be careful not to bend the pins out of line. If this does happen straighten them carefully with pliers.

4) Now replace Acorn Atom bottom and screw back down.

HOW TO START UTILITY EPROM

Simple press break, then enter link 44992.

The cursor will now flash showing that the Eprom is working.

All the new commands are now available and the cassette high speed of 1200 is now ready.

NOTE: The link 44992 must be entered every time break is pressed, otherwise the Atom will be in its original state.

*STRG X,Y,"STRING"

The *STRG command is used to print a string of characters in graphic modes.

X,Y are the coordinates at which string is to be printed, followed by a string in quotes of what to print.

Example:

```
100 FOR I=4 TO 0 STEP -1 ; CLEAR I
110 *STRG 4,28,"* HELLO *"
120 *TONE 3000,50 ; *WAIT 90
130 NEXT I ; END
```

This will display "* HELLO *" in mode 4 to 0

*BSTRG X,Y

This command is as above, but the current string buffer is printed. (String buffer = Memory Location #140)

Example:

```
100 CLEAR 0 ; DIM A(8) ; $A="BUFFER"
110 $#140=$A ; REM LOAD STRING BUFFER WITH $A
120 *BSTRG 0,28 ; END
```

The above shows how to print a variable using *BSTRG.

*CHAR C,X,Y

The *CHAR is used to print a ASCII character corresponding to the value of 'C' at X,Y coordinates. Where 'C' can have a value of 32 to 127.

Example: using *CHAR to print ASCII character set

```
100 CLEAR 2 ; X=0 ; Y=88
110 FOR I=32 TO 127
120 *CHAR I,X,Y ; X=X+8 ; IF X>127 X=0 ; Y=Y-15
130 NEXT I
140 GOTO 140
```

*KBEP

This command generates a bleeping out of the atom speaker until a key is pressed.

Here is a simple musical keyboard program:

```
100 DIM A(1) ; PRINT $12
110 *KEY $A ; IF $A=""GOTO 110
120 *TONE 9000,CHR$A ; PRINT $A ; GOTO 110
```

And here's an example to input a number using *Key

```
100 PRINT $12,"LEVEL 0--9 ?"
110 *KEY L ; IF L=10 GOTO 110
120 PRINT L ; END
```

*RENUMBER X,Y

This command renumbers a basic program, starting with 'X' and in steps of 'Y'.

The *Renumber changes all the "GOTO'S", "GOSUB'S" and "*RESTORE'S" to match the line to which they refer.

The two errors that may occur are:

ERROR 106 This error means the renumber table has run out of memory. The program will not have been renumbered.

ERROR 184 Line number renumbering too large (line number 32767 maximum)

If Error 184 does occur, then renumber the program again so that the numbers don't go over 32767, by lowering the start and step numbers.

NOTE: The renumber uses memory starting at #8200 for a table of references for GOTO'S, GOSUB'S and *RESTORE'S. Two bytes are used for each.

*AUTO X,Y

This command provides automatic line numbering for the convenient entry of basic programs. The user has to specify the start line number 'X' and the increment of the line number 'Y'.

If the line already exists, the fact is flagged by a bleep. This is in case the user does not wish to change that line.

To stop the automatic line numbering press the ESC key. The number displayed will not be put into the program.

*BSAVE "FILE NAME"

This command saves a basic program so that it can be loaded and run with *RUN.

This command is the same as the 'SAVE' command, but it saves the auto run address.

NOTE: Only works with named cassette files.

*VAR

This command prints out the values of the integer variables A to Z.

This can be very useful when debugging programs.

*DATA AND *READ STATEMENTS

Data and Read Statements are used together to assign values to variables within a program. Every time a *Data Statement is encountered, the values in the argument field are assigned sequentially to the next available position of a data buffer.

All Data Statements, no matter where they occur in a program, are combined into a continuous list.

*Read Statements cause values in the data buffer to be accessed sequentially and assigned to the variables named in the *Read Statement. They start with the first *Data Element from the first Data Statement, then the next element, and so on to the end of the first Data Statement, and then to the first element of the second Data Statement, etc.

Each time a *Read command is encountered, it reads the next data value that has not been assigned to a variable.

If a *Read is executed and there is no more data, an Error 1 is generated.

Numeric and String Data may be intermixed. However, they must be used in the appropriate order to assign the data to the appropriate variables.

*Data and *Read Statements may be placed anywhere within the program. For example:

```
10  DIM M(9),N(12)
20  *DATAJANUARY,17,1982
30  *DATASMITH
40  *RESTORE 20
50  *READ $M,D,Y,$N
```

The statements shown above are equivalent to the following:

```
10  DIM M(9),N(12)
20  $M="JANUARY"
30  D=17
40  Y=1982
50  $N="SMITH"
```

*RESTORE N

The *Restore Statement sets the data buffer to the line number specified.

The *Read will start looking for data from that point.

N = line number to restore data buffer to.

*BLOCK N X,Y,X1,Y2

This command will fill a graphic block.

N = 1, 2 or 3 set, invert or clear block
X,Y = The coordinates of bottom left hand corner
X1 = The X length from X point
Y1 = The Y length from Y point.

Example:

```
10 CLEAR 0 ; *BLOCK 1 30,22,4,4 ; END
```

This will draw a square graphic block of 4 * 4 pixels, in the middle of the screen.

*SHAPE N X,Y *TABLE T

The *Shape command is used together with the *Table to draw complex shapes anywhere on the screen in any graphic mode.

Where 'N' is the shape number the user wants to draw.

X,Y are the coordinates at which shape is to be displayed.

T is the memory address of the shape table.

SHAPE TABLE DATA

The shape command needs data to draw each shape. The data is held in memory locations of which each byte holds two vectors.

The way *Shape works is this, it reads the first vector, plots a point at its current location and moves in a direction represented by the vector.

The data can represent in movement in one of 8 directions:

0 = No move (do nothing)
1 = Move up
2 = Move down
4 = Move left
5 = Move up left
6 = Move down left
8 = Move right
9 = Move up right
A = Move down right
3, 7, B, C, D, E and F are all not used.

To change the point locationwithout drawing, a zero byte should be entered.

To change back to draw mode again enter another zero byte.

To indicate the end of shape data, enter two zero bytes.

An overlap flag has been set up at location #DE, it will be set to zero if the shape draws over any point that was already set.

NOTE: Flag only works in modes 1, 2, 3 and 4.

SHAPE TABLE

The shape table can be put into any convenient part of memory.

This is how the shape table is constructed, using memory at #2800.

```
2800 04    – Number of shapes on table
2801 2809 – Address of shape 0
2803 280D – Address of shape 1
2805 2810 – Address of shape 2
2807 2815 – Address of shape 3

     Shape 0 Data
2809 14 22
280B 00 00 (end of shape 0)

     Shape 1 Data
280D 66
280B 00 00

     Shape 2 Data
2810 89 A1 85
2813 00 00

     Shape 3 Data
2815 22 AA 88 14
2819 00 00
```

Now here an example program using the *Shape command.

```
100 CLEAR 0 ; M=#2800
110 *RESTORE 200 ; *TABLE M
120 *READ N ; IF N=-1 GOTO 140 ; REM PUT SHAPE TABLE
130 ?M=N ; M=M+1 ; GOTO 120 ; REM INTO MEMORY
140 S=RND & 3 ; X = S * 12 + 8
150 FOR Y=150 TO -5 STEP -1
160 *SHAPE S X,Y ; *TONE 999,Y ; REM DRAW SHAPE & SOUND
170 *SHAPE S X,Y ; NEXT Y ; REM CLEAR SHAPE
180 PLOT 13,(RND&63),(RND&31)
190 GOTO 140
200 *DATA 4,#28,#09,#28,#12 ; REM NUMBER OF SHAPES &
210 *DATA   #28,#1C,#28,#25 ; REM ADDRESS OF SHAPES
220 *DATA #91,#8A,#15,#9A,#28 ; REM SHAPE 0
230 *DATA #2A,#00,0,0
240 *DATA #19,#91,#49,#88 ; REM SHAPE 1
250 *DATA #A4,#68,#AA,#28,0,0
260 *DATA #99,#55,#9A,#89 ; REM SHAPE 2
270 *DATA #A6,#6A,#A8,0,0
280 *DATA #81,#15,#19,#82,#68 ; REM SHAPE 3
290 *DATA #92,#91,#8A,#24,#4A
300 *DATA #22,#44,0,0,-1
```

Follows are nine memory bytes used by the Eprom commands.

```
#023D,#023E,#023F  Used by *Key
#00AE , #00AF      Used by *Shape (sc. table pointer)
#00AC , #00AD      Used by *Read (sc. data pointer)
#00AA , #00AB      Used by *Auto (sc. auto flag)
```

*MC

The *MC command is a machine code monitor entry.

The monitor has three commands, memory change, hexdump and textdump.

MEMORY CHANGE

This command lets the user change memory locations very easily as in the following example:

*MC	Machine code monitor entry
CMD ? M2800 -----	Enter memory change command 'M' Followed by memory location
2800 C 43	Press space key to look at next byte
2801 A 41 $T	Press '$' key then 'T' key to store character 'T' in memory 2801
2802 C 0D -	Press '–' key to step back memory location
2801 T 54 80	To change contents of Loc. 2801 to 80 enter 80
2802 C 0D	Press 'ESC' key to escape from memory change command

The memory change command prints out in the following format:

Memory location, ASCII character then data byte.

NOTE: If the ASCII code is a control character an inverted 'C' is printed.

HEXDUMP

The Hexdump prints out 64 memory locations as follows:

```
CMD ?  HA000
A000  53 48  41  50  45  A4 96  42
A008  4C 4F  43  4B  A9  D6 54  53      Only 32
A010  45 43  AF  FO  54  4D 47  4F      bytes shown
A018  AF FO  4B  45  59  A2 2B  54
```

TEXTDUMP

The Textdump prints 64 memory locations as characters:

```
CMD ? T8400
8400  T H I S    I S
8408  W H A T    T E X          Only 32
8010  T D U M P    I S          bytes shown
8018  L I K E    * * *
```

*APPEND

This command loads a basic program at the end of a current program, and resets the top to reflect the size of the new, enlarged program.

The *Append works the same as load, other than the program that is currently in the atom is not removed.

NOTE: When used, line numbers of program to append to current program, should have higher line numbers, for the program to run correctly.

*VERIFY

The *Verify command allows the user to verify programs that have been recorded correctly on tape.

The *Verify command is used the same as load, except for named files where each file name is printed (same as *Cat format).

The two match errors are 35 and 99 for unnamed files and named files respectively, this means the data stored on tape is not the same as stored in memory.

i.e. an error has been recorded.

NOTE: Each file name printed is verified, so make sure that the file block number starts with zero, and they are displayed as increasing file block numbers.

*POINT X, Y P

This is a graphic command, that tests a point to see if that point is set or clear.

If the point is set A '1' is returned, or if point is clear a '0' is returned.

'X','Y' are the coordinates to test and 'P' is any variable which the number 1 or 0 is stored into.

NOTE: Can only be used in graphic modes 1, 2, 3 and 4.

The *Zero command sets the values of all the integer basic variables to zero.

*WAIT N

The *Wait command provides the user with an easy way to program wait loops.

N is a decimal value 0 to 65535.

The time value of one 'N' is 1/60 second.

So *Wait 600 will delay for 10 seconds. . .

*BLEEP

This command switches on a bleep at the end of all subsequent cassette input and output commands so you know when it is finished.

NOTE: The bleep will keep bleeping until any key is pressed on the keyboard to stop it.

*BPOFF

This command turns off the above bleep.

*TONE D,N

This command generates a tone

'D' is the duration (1 to 65535) of the tone

'N' is the tone pitch (1 to 256).

*TAPE S

This command sets the tape operating speed

Where 'S' is a number – 1,2 or 3
1=300 BAUD, 2=600 BAUD or 3=1200 BAUD.

NOTE: 1200 BAUD is set when the utility is first linked.

*STOP

The *Stop statement causes the program to halt execution, until a key is pressed, the line number containing the *Stop which caused the halt is printed out, i.e. 'STOP NNNNN'.

*POP

This removes references to a current subroutine from the stack, so that you can jump directly from the subroutine to any point in the main program, any number of times.

*DELETE X,Y

This command allows the deletion of a range of line numbers.

Where 'X' is the first number to delete,
and 'Y' is the last number to delete.

*PACK

This command packs the program by removing spaces.

No significant spaces are removed.
i.e. no spaces will be removed in quotes, or after *Data Statements. . .

*KEY $N

The keyboard is scanned once and the character of the key pressed is stored in the designated string.

If no key was pressed the null string is returned ($N"")

NOTE: More than one key can be held down at a time, and each will be returned into string one at a time, as each *Key is executed.

*KEY N

As above, but only numbers 0 to 9 are scanned, and a value of 0 to 9 corresponding to the key presses is returned in the variable specified after the command.

If no number key is pressed 'N' = 10.

*SCREEN X

This command allows the user to set the cursor to any screen memory location.

For example: *Screen 256 would set the cursor to the middle of the screen.

*FIND "STRING TO FIND"

This command prints out all lines which contain a given string.

The string the user requires to find is put in quotes after the find command.

For instance, all 'FOR' statements can be printed out with *FIND "FOR".

NOTE: Anything in between the quotes will be printed out, so for example *FIND """" will print out all the lines with a quote in them.

ERROR CODES USED IN UTILITY EPROM COMMANDS

The following list of errors are generated, by the utility eprom. . .

1 *Read run out of data to read
No data left for *Read to read data from

35 *Verify unnamed cassette file match error

60 *Key with missing variable
The *Key command did not have a string or letter variable to put scanned key into

71 *Restore line number not found
The line number specified in *Restore was not found

73 *STRG quotes missing
One or both of the quotes missing after *STRG

99 *Verify named cassette file match error

103 *Point with missing variable
The *Point command has no variable to store the number returned
NOTE: Variable can only be A – Z or @

106 *Renumber has run out of memory to store table
The *Renumber stores a table (starting at =8200) of GOTO's, GOSUB's and *Restore's address pointers. This table is full and memory has run out

163 *Shape number error
The *Shape number used is not on the current shape table in use

169 *Find quotes missing
One or both of the quotes missing after *Find

180 *Tape number is not 1, 2 or 3
The *Tape speeds can only be 1=300, 2=600 or 3=1200 Baud

184 *Renumber line number overrun (>32767)
If this occurs renumber with lower start and step