ATOM
UTILITY ROM

INSTRUCTION MANUAL

A T O M

U T I L I T Y R O M

I N S T R U C T I O N M A N U A L

WILLOW SOFTWARE JULY 1981. V1.0

C O N T E N T S

Introduction	2
Installation	3
Initialisation	4
*ALERT	5
*APPEND	5
*AUTO	6
*BDUMP	7
*BLIP	7
*BSAVE	8
*BZERO	8
*DEL	8
*DIS	9
*FIND	9
*KEY	11
*KRUNCH	12
*MDUMP	12
*REN	13
*SIZE	14
*SPACE	14
*XREN	14
Register Dump	15
Memory Usage	15
Errors	16

The Willow Software Utility ROM contains 4K bytes of utility programs and subroutines which are always available for use, whenever the Atom is turned on.

The Utilities provide powerful tools for writing, developing and debugging programs in Basic and Assembler. In addition a number of features have been included to make your Atom easier to use.

The Utility ROM works with any size of Atom, and does not interfere with the standard facilities of the machine. No text space RAM is required.

This manual contains descriptions of each of the 17 new commands provided by the Utility ROM, with sections on memory usage and errors. Please read through the manual first, particularly the section on errors.

The manual assumes that you have a good knowledge of the excellent Atom manual, and that you have some experience of programming. Some of the commands are very easy to understand and use, others are more complicated, and some will only be of use when you have done some Assembly language programming. Don't worry if you do not understand all the commands at first. You will soon be using them all, as your knowledge of programming the Atom grows.

All the commands may be abbreviated in the normal manner. The minimum abbreviation for all commands is two letters. For example, *APPEND may be abbreviated to *AP. *APP. *APPE. or *APPEN. .

The best way to learn how to use your Utility ROM is to try the commands on your Atom. Experiment with the commands using small programs.

Some of the information in this manual is necessarily fairly complex, and will be much more readily understood by practical experience.

INSTALLATION

The Utility ROM is a MOS device that can be damaged by static charges which occur all around us, even in the human body. Installation is however, straightforward, if the instructions given here are followed carefully. If you are nervous about installing the ROM, take it along with your Atom to your local computer or electronics shop who will do the installation for you - usually for only a nominal charge.

1. Disconnect the Atom from its power supply, and remove all the connectors (cassette etc) from the back.

2. Place the Atom upside down on a soft surface such as a cushion, to avoid damaging the keyboard. Undo the two screws underneath, and remove the bottom of the case.

3. Locate the socket for the Utility ROM. It is marked IC24, and will be found near the centre of the PCB between two large ICs - IC22 (6502) and IC25 (8255).

4. Before handling the ROM - EARTH YOURSELF to discharge any static. This is easily done by touching the metal frame of some earthed mains equipment, such as an electric fire. Avoid working on a nylon carpet.

5. Carefully remove the ROM from its packing. Avoid touching the pins. Insert the ROM into the socket marked IC24. Be careful to get it the right way round. Pin 1 of the ROM is identified by a notch and/or a small dot. The ROM should be inserted the same way round as all the other ICs on the board ie pin 1 pointing towards the back of the Atom. It may be necessary to straighten the pins by pressing the edge of the ROM on to a hard surface - don't overdo it!

6. Check that all the pins are located correctly in the socket, and that no pins have been bent out, or bent underneath. Ensure that the ROM is pressed fully down in the socket. If you have bent any pins, carefully lever the ROM out of the socket using a small screwdriver or nailfile at both ends. Straighten the pins using small pliers, and try again. The pins are not very strong, and can easily be broken - so be careful.

7. Check everything again, particularly that it is the right way up. It is easy to correct mistakes at this point; after the power is connected it may be too late. Finally reassemble the Atom and reconnect. If the initialisation (see next section) does not work, switch off the power immediately, and check you have done everything correctly.

INITIALISATION

Before you can use any of the facilities described in this manual, it is necessary to let the Atom know that the Utility ROM is present. This can be done by typing the statement:

```
LINK #A000
```

This must be done each time you switch on the Atom, and each time you use the BREAK key.

If all is well, the screen will be cleared, and the message "WILLOW SOFTWARE UTILITY ROM" will appear at the top of the screen, together with a version number (eg V1.0) which identifies the software in your ROM.

Sometimes you may want to initialise the ROM, without having the screen cleared, or the message. This may be required if the initialisation is done within a program. To do this change the LINK address to #A024.

It is only necessary to initialise the ROM once, it remains active until you turn off the power or type BREAK.

If it becomes necessary to reinitialise the ROM at any time, you must press BREAK first.

*ALERT

*ALERT provides an indication of the completion of cassette input and output commands by bleeping, so that you do not have to watch the screen for the return of the > prompt.

This command works like a switch. The first time you type it the facility is turned on. When you type it again, the facility is turned off.

It will work with all cassette commands - LOAD, SAVE, *LOAD, *SAVE, *RUN, *BSAVE, and *APPEND. The normal operation of these commands is otherwise unaffected.

*APPEND "filename"

*APPEND is used to append extra files on to the end of the program in the current text space.

The format of the command is the same as the normal LOAD command.

The Atom manual tells you how to do this without using a special command, but *APPEND makes it much easier, since it automatically works out where to load the file and also resets TOP.

Any number of files may be appended using this command. The resulting text may be renumbered using *REN or *XREN.

Example:

```
>*ALERT
>LOAD "MAINPROG"
>*APPEND "SUB1"
>*APPEND "SUB2"
```

This will load the three files MAINPROG, SUB1 and SUB2 from cassette into the current text space, one after another.

*AUTO [start line] [,increment]

*AUTO provides automatic line numbers for entering programs.

If the command is given without any parameters, the start line will be 100, and successive lines will be numbered in steps of 10 (increment). If there is already a program in the current text space, the default start line will not be 100 but will be one increment after the last line in the program. If the start line or increment are specified, these values will override the defaults. Either or both the parameters may be given.

The line numbers appear on the screen, and are entered into the input line buffer just as if you had typed them. It is therefore possible to use the delete key to change the given line number if required. This can be useful to insert an extra line earlier in the program, without leaving auto mode.

It is quite possible to overwrite existing program lines when using *AUTO. When a line number is given that already exists in the program, a warning is given in the form of an exclamation mark before the line number. If you wish to retain the existing line, without leaving auto mode, use delete to remove the line number from the screen and press return.

When in auto mode all keys may be used in the normal way, including the cursor movement and COPY keys. If you wish to cancel a line CTRL X may be used. In this case the line number will be repeated for you to retype the line.

To leave auto mode simply type return in response to the line number prompt, or type ESCAPE at any point on the line. You will notice that the current line will disappear from the screen when you leave auto mode. This is so that what you see on the screen is what has been stored in the program. Note that when you type return in response to a line number prompt, that line is not removed from the program, if it already exists.

Example:

```
>*AUTO
 100 INPUT X
 110 PRINT X,X*X'
 120(return)
>*AUTO
 120 END
 130(return)
>*AUTO 100,5
!100 PRINT "NUMBER SQUARE"'
 105 @=5;FOR X = 1 TO 10
!110(delete)5 NEXT X
 115(return)
>LIST
  100 PRINT "NUMBER SQUARE"'
  105 @=5;FOR X = 1 TO 10
  110 PRINT X,X*X'
  115 NEXT X
  120 END
```

*BDUMP

*BDUMP prints out the names and values of all the integer Basic variables - @ and A -> Z, except those that have the value zero.

The command may be used in immediate mode (in response to the > prompt) or it may be used within programs.

To use it for debugging programs, all variables should first be set to zero using *BZERO. *BDUMP statements may then be inserted into the program where required, and will give the current values of any variables used.

If a large number of variables are to be dumped, the values of the first ones will disappear off the top of the screen before you have a chance to read them. This can be avoided by typing CTRL N first, to put the Atom into page mode.

Note that when *BDUMP is used within programs, it must be the only statement on a line, or the last statement on a multi-statement line. Any characters on a line after *BDUMP will be ignored. This also applies to all other Utility ROM commands which may be used within programs. Note also that the value of the print field width (@) is set to 8 by this command.

Example:

```
10 *BZERO
20 A=2;B=3;*BDUMP
30 C=44
40 *BDUMP
50 A=4;PRINT'"DUMP3";*BD.
60 END
```

*BLIP

*BLIP causes the Atom to give a short unobtrusive 'blip' sound for each key pressed. It is intended as an aid to typing, and to help detect the double entry of characters which sometimes occurs due to keybounce on the Atom keyboard.

Like *ALERT, this command acts like a switch. Type it once to turn it on, and again to turn it off.

*BSAVE "filename"

*BSAVE is a special version of the SAVE command which is used to save Basic programs on cassette, so that they may be automatically loaded and run using the standard *RUN command which normally only works for machine code files. The format of the command is the same as the normal SAVE command.

Example:

```
*BSAVE "PROG"
*RUN "PROG"
```

If you use the optional start address on the *RUN command to relocate the program, be sure to reset the text space pointer (?18) to the new address first.

There have been other published methods of achieving this facility, using the address #CE86. These methods do not work if the program contains DIM statements. *BSAVE works for all programs; TOP and the free space pointer will be set to the correct values when the program is run using *RUN.

*BZERO

*BZERO will set all integer Basic variables to zero. The least significant byte of the print field width variable (@) is not changed.

Like *BDUMP, this command may be used in direct mode, or within programs.

*DEL [line1] [,line2]

*DEL will delete a range of statements from a program.

The line number range works in the same way as the standard LIST command. If only one line number is given only that line is deleted. If either the first or second parameter is given, with the comma, the default values of the start or end of the program are used as with LIST. Unlike LIST however, if no parameters are given the command will be ignored. This is to avoid accidental deletion of the whole program. If you do want to delete the whole program, use NEW.

Examples:

```
*DEL 20,40
*DEL ,100
*DEL 197,
*DEL 125
```

*DIS address

*DIS is a disassembler program, or assembler in reverse. It translates machine code, starting at any address in memory, into symbolic assembler source statements in Atom format. Absolute addresses are used throughout.

This utility program may be used to understand the Atom operating system, or even the utility ROM itself. It may also be used with your own machine code programs, and to decipher programs published in 'memory dump' format.

The address parameter gives the address at which the disassembly is to start. If you wish to specify the address in hexadecimal, the number should be preceded by #. The disassembly may be abandoned at any time by typing ESCAPE. The utility will work in page mode, in the usual way. The format of the listing is very similar to the listing produced by the Atom assembler. Invalid op codes are printed as ??? .

Examples:

```
*DIS #CBFF
*DIS 43222
```

*FIND delim string delim [line1,line2]

*FIND will list lines which contain a specified string of characters.

Example:

```
*FIND "FOR"
```

This will print out all the lines in the program which contain the string FOR. In this example, the delimiter (delim) is ". In fact any character which does not occur in the string may be used as the delimiter, except the space character. This is so that you can find any string you want to, by choosing a suitable delimiter.

Example:

```
*FIND/"/
```

In this case / is the delimiter. All lines containing " will be printed.

Note that *FIND will only find strings of characters which occur within lines, it will not find the line number itself since line numbers are not stored as characters in the Atom.

If the line range is omitted as in the examples above, then the whole program will be searched. The search may be limited to a section of the program by specifying a line range which works in exactly the same way as it does for the LIST command.

Example:

```
*FIND :LINK:120,250
```

This will print out all the lines in the range 120-250 which contain the string LINK.

The string may contain any characters including spaces, and may be as long as you can fit on the command line.

To save typing, if no line number range is to be given, the second delimiter may be omitted.

Example:

```
*FIND/CATS AND DOGS
```

There is also a default facility for the search string. This is provided for use in long programs where you may need to find all the occurrences of a string, and edit the lines on screen. The default string is the last string that was specified, it is obtained by typing two delimiters together.

Examples:

```
*FIND /PRINT/ ,200
*FIND // 210,300
*FIND // 310,
```

Be careful not to forget the delimiter when specifying a string. Since any character may be a delimiter, the first non-blank character on the line is always taken as the delimiter.

Example:

```
*FIND CAT
```

This will take C as the delimiter and look for the string AT.

*FIND will work in page mode. The listing may be abandoned at any time by typing ESCAPE.

*KEY

*KEY is intended for use within Basic programs. It provides the true keyboard scanning facilities lacking in the standard Atom system. The command makes it possible to scan the keyboard to discover if a key has been pressed, without having to wait for a key to be pressed.

As with other Utility ROM commands, *KEY must be either the only statement on a line, or the last statement in a multi-statement line.

*KEY scans the keyboard, and puts the ASCII value of any key pressed into the string $K. If no key has been pressed, the string $K will be set to the null string (""). In the program, keys may be checked for by using normal string comparison statements. Note that the string $K is set up by the command. The variable K should not be used for any other purpose in the program.

Example:

```
10 *KEY
20 IF $K = "" PRINT "NO KEY PRESSED"' ; GOTO 10
30 IF $K = "E" GOTO 50
40 PRINT "                KEY " $K " PRESSED"' ; GOTO 10
50 END
```

Try this program to get some idea of how the command works.

*KEY will detect all normal keys including shifted and control characters, but the use of the cursor movement or LOCK keys will cause the command to 'hang up' until another key is pressed. The return key cannot be detected by this command since it gives the same result as no key being pressed, the null string.

This command should prove useful for writing games and other interactive programs, since it will detect a key held down (moving a laser base to the left or right?) as well as a single key (the 'fire' button for the laser cannon?).

When looking for a single key depression, it may be necessary to include a delay loop, as the routine is fairly fast.

*KEY may be used within assembly language programs. The routine is located at address #AFD6. It may be called as a subroutine using JSR. On return, location #21C will contain the ASCII value of any key pressed; this will also be in A. If no key has been pressed, #21C will contain CR (13); A will contain #DF.

Example:

```
10 DIM LL1;FOR N=1 TO 2;DIMP-1;[
20:LL0 JSR#AFD6;CMP@#DF;BNE LL1;LDA@42
30:LL1 JSR#FFF4; JMP LL0
40];NEXT N;LINK LL0
```

*KRUNCH

*KRUNCH compresses a program by removing all REM statements and all spaces, except those that occur within strings in quotes.

Compressing a normally written Basic program can reduce the size dramatically and also speed up the running of the program. It is recommended that programs are first written well spaced out, with plenty of comments, so that the program is easy to read and understand. Two versions of the program can then be saved on cassette, one normal, and one compressed by *KRUNCH. The compressed version should be used for normal running. It will take up less memory, take less time to load, and run faster.

There are some places in Atom Basic where spaces are essential. The most common example of this is in the statement:

```
FORA=B TOC STEPD
```

The spaces after the B and after the C are required or an error will result. If your compressed program contains statements of this form, you can use *FIND to locate them, and edit them on screen to insert the essential spaces.

There are also other rarer examples of statements which require essential spaces. For example, in the following two statements the space after the B is required:

```
IFA=B PRINT"EQUAL"
IFA<B ANDC=B GOTO 10
```

Note that the time taken by *KRUNCH depends on the number of spaces in the program, and the size of the program. In most cases it will only take a few seconds, but for a 5K program with spaces in every line, it could take 20 seconds or so, since it has to move the whole program in memory, for every line that contains a space.

*MDUMP address

*MDUMP prints out a formatted memory dump starting at any specified address. The dump is given in hexadecimal, with ASCII characters below. 8 bytes are printed on each line, with the address in hexadecimal on the left.

The starting address may be given in decimal or hexadecimal (preceded by #). The command will work in page mode. The dump may be abandoned at any time by typing ESCAPE.

Example:

```
*MDUMP #C000
*MD. 123
```

*REN [start line] [,increment]

*REN will renumber a Basic program, including all GOTOs and GOSUBs, and their abbreviations.

The command may be used without any parameters. In this case the renumbered program will start at line 100, with the line numbers going up in steps of 10. Either the start line number or the increment may be omitted, in which case the default value of 100 or 10 will be used.

Examples:

```
*REN
*REN 500,5
*REN ,100
*REN 1000
```

*REN works in two passes, in other words, it reads the program twice. The first time it checks the program, working everything out, but not actally making the changes. If any errors occur on the first pass, an error message is given, and the program is not changed in any way. If all is well the whole process is repeated on the second pass, and the program is then actually renumbered.

Possible failures on the first pass are: GOTOs or GOSUBs without destination lines, lines that become too long, line numbers which become greater than 32767, and too many GOTOs and GOSUBs. For further information see the section on errors.

*REN will usually work almost instantaneously. If a program contains a large number of GOTOs or GOSUBs it might take a few seconds.

*SIZE

*SIZE prints the size of the current program in bytes. It may be used in direct mode, or within programs.

Examples:

```
>*SIZE
10 PRINT "PROGRAM SIZE IS "; *SIZE
```

*SIZE does not take any account of arrays etc. It is simply the number of bytes between the start of the text space and TOP. The correct size if no program is present will be 2 bytes.

*SPACE

*SPACE prints the amount of free space there is available after the program in the current text space.

It does not take any account of arrays etc. It is simply the number of bytes between the end of the program, and the end of the text space. As with *SIZE, the command may be used in direct mode or within programs.

Note that *SPACE overwrites all locations between the end of the program and the end of the text space.

*XREN [start line] [,increment]

*XREN is a special version of *REN which does not change GOTOs or GOSUBs. It also does not perform all the pre-checks that *REN does. The format of the parameters is the same as for *REN. The command only renumbers the line numbers of the program.

*XREN is useful when you do not want the GOTOs and GOSUBs changed, for example, after using *APPEND, where the resulting line numbers may not be in order, and *REN would not work correctly.

REGISTER DUMP

This is an extra facility which provides a powerful debugging aid for assembly language programmers. It is a subroutine within the Utility ROM which may be called as often as required in a program by "JSR #AAC5". It is not necessary to initialise the ROM before using this routine.

The subroutine prints out the address from which it was called (which it gets from the stack), the contents of the A, X, and Y registers, the stack pointer and the status register, in that order. It then returns to your program after restoring everything. Therefore the subroutine does not affect the program in any way.

Since it prints out the address from which it was called, the subroutine may be called many times in a program without confusion.

Note that zero page locations #99 to #9F are used by the routine for temporary storage.

MEMORY USAGE

In order to operate correctly, the Utility ROM has to maintain some permanent storage in page zero. It uses locations #A8 to #AF for this purpose. These locations should not be changed after the ROM has been initialised.

The Utilities also make use of various locations from #80 to #A7 while they are actually in use. These locations are only used for workspace, so they are still available for use within programs.

When utilities are used within programs that make use of any of the spare RAM locations at #80 and #21C onwards, it is necessary to ensure that you do not use locations that are used by the ROM. The calling routine which is used for all *commands, including standard COS commands, changes locations #90,#91,#95,#96, and #A4 to #A7. In additon, *KEY uses #21C, #21D, and Basic variable K, *BDUMP uses #93, *BZERO, *SPACE, and *SIZE use no other locations.

*REN and *FIND use location #21C onwards for workspace. In addition, some of the utilities change the values of integer Basic variables, including @.

Most of the commands can'cause only syntax errors of various sorts. If a command is mis-spelt or over-abbreviated, it will not be recognised as a utility, and will be passed on to the command line interpreter. The resulting error will be COM? ERROR 48.

Many of the commands will ignore extra characters, and will also ignore you if you ask for something silly such as *DEL 44,22.

Commands that require parameters will give normal Atom error numbers if you give invalid parameters.

For certain kinds of invalid parameter, *AUTO, *REN, *XREN, and *FIND will ignore what you have typed and use their default values. *DEL will also ignore some invalid parameters, for example *DEL 100,ABC will be interpreted as *DEL 100, and delete from line 100 to the end of the program.

*REN has been written to check for most error conditions, before it attempts to renumber the program, but there is still one situation when it can fail and leave you with a partially renumbered program. This occurs when there is not enough memory in the current text space for the renumbered program. It can happen when renumbering with larger line numbers, thus resulting in a larger program. This will give ERROR 248. The program will be partly renumbered, and there may be one line missing. It is therefore wise to have a copy of the program on cassette when dealing with large programs.

One of the errors that may occur in the first pass of *REN is ERROR 2 - Too many GOTOs and GOSUBs. *REN will only handle up to 127 GOTOs and GOSUBs. This arises for various reasons, but the main reason is that the ROM was written so that it does not use any of your valuable text space memory. In practice this is not a serious limitation, since with the amount of memory available, even on the expanded Atom, programs with more than 127 GOTOs and GOSUBs will be rare. There is also the availability of the 'GOTO label' facility on the Atom which should always be used in preference to line numbers, since it is faster. We should like to hear from anyone who finds this limitation a problem (and we should like to see the program!).

Most errors that occur will be found in the Atom Manual. The following error numbers do not appear in the manual, or have slightly different meanings:

ERROR 2 More than 127 GOTOs and GOSUBs in a program.
ERROR 109 Line number greater than 32767.
ERROR 255 Resulting line too long.
ERROR 222 Delimiter missing in *FIND.

If you have used your Atom for any length of time, you will be aware that it is possible for the machine to 'hang up' in certain circumstances, so nothing happens when you type commands. This can occur if you overwrite your program, or if you change to a different text space and forget to type NEW. This can also occur, in similar circumstances, when using the Utility ROM. If this does happen, the only solution is to type BREAK and reinitialise the ROM.

If you experience any problems with the use of the Utility ROM please let us know. Please check this manual first, and ensure that you are not overwriting your program, or locations #A8 to #AF.

If your program uses any of the locations from #80 or #21C onwards, be sure that it does not change any of the locations used by the ROM commands that you are using.

We will require as much information as possible to sort out problems, so please give full details of how a suspected fault may be reproduced. If the problem only occurs with a program longer than a few lines, then please send a cassette with the program recorded twice. This will be returned to you.

We are also willing to answer any questions about the use of the Utility ROM, so please do not hesitate to contact us, whether or not you suspect a fault.

WILLOW SOFTWARE

PO Box 6

Crediton

Devon EX17 1DL

Atom Utility ROM Manual Addenda

*BSAVE

Note that *BSAVE will not work with unlabelled files. This is not a fault in the Utility ROM, but is due to a built in limitation of the Atom cassette operating system which does not allow unlabelled files to have an execution address. (See the description of the *SAVE command on pp140-141 of the Atom Manual). Note also that *BSAVEd programs can only be *RUN on Atoms with the Utility ROM fitted. This is because the execution address used is within the Utility ROM.

*BLIP

The machine code routine which makes the short blip noise for each key pressed, is located at address #A13E. It may be called from Basic by LINK #A13E, and from Assembler by JSR #A13E.

TESTING

If you suspect a fault in the ROM, it may be tested using the program given on page 93 of the Atom Manual. The correct signature which should be obtained is F315.

Willow Software
October 1981



© WILLOW SOFTWARE PO BOX 6, CREDITON, DEVON