TOOLKIT

BASIC
PROGRAMMERS
AID

SUPPLIED ON EPROM

FOR THE
BBC
MICRO

TOOLKIT

BEEBUG UTILITY ROM

FOR THE BBC MICRO MODEL B

By Mark Tilley

CONTENTS

Published by BEEBUG Publications Ltd.,PO BOX 50, St Albans, Herts, England.

1. INTRODUCTION

TOOLKIT is a set of utilities held in ROM, for use in conjunction with Basic. It will not work if Basic is not present.

The commands are all entered with an asterisk '*' at the start, in the same way as the built in operating system commands. Most commands can be entered directly without a parameter, for example, *EDIT. Some take an optional parameter, e.g. *EDIT 100. A few have a compulsory parameter, such as *MERGE which requires a filename to be specified.

Commands may be entered in upper or lower case; TOOLKIT does not differentiate between the two. They may also be abbreviated in the usual way by giving the first letter, or first few letters, followed by a full stop. The minimum abbreviation required will depend on the other ROMs and the positioning of the TOOLKIT ROM in relation to them.

It may be that other ROMs have commands with names the same as TOOLKIT commands (for example EDIT and MOVE are quite common). This means that if such a ROM has higher priority than TOOLKIT it will intercept the command and TOOLKIT will not receive it. To get around this problem TOOLKIT has a facility for entering the command name in an alternative form in order to specify explicitly that the command is meant for TOOLKIT. This is done simply by prefacing the command name with the letter 'B' (e.g. *BEDIT, *BCHECK). However, if TOOLKIT is used, for example, with a Watford DFS, commands such as *MOVE will always go to TOOLKIT rather than to the filing system. This is because any filing system command has a lower priority than a service ROM command (such as TOOLKIT). The position of the DFS and TOOKIT Roms in the machine will not make any differences in such a situation.

To make full use of this facility, TOOLKIT should be located at a lower priority than (ie to the left of) ROMs with which there is a potential command name clash. This allows the other ROM to accept the full version of the command name, and TOOLKIT the version prefaced with 'B'.

2. STARTING INSTRUCTIONS

(i) Fitting

TOOLKIT is supplied in an 8K EPROM, which should be fitted in one of the "Paged ROM" sockets in the computer. It should be installed to the left of the Basic ROM. See separate fitting sheet for full installation instructions.

(ii) Using TOOLKIT

Once installed, all the commands in TOOLKIT are available immediately whenever the computer is switched on. TOOLKIT has a facility for enhanced error reporting. In order to use this the *ON command must be issued. When in teletext mode, coloured displays will be used by TOOLKIT for greater clarity. The colour is turned off by the command *OFF.

(iii) RESPONSE TO *HELP

When *HELP is entered with no parameters TOOLKIT will display a message similar to:

```
TOOLKIT 1.00
TOOLKIT
INFO
```

The number in the first line is the version number, which should be quoted in any correspondence. The other two lines give the keywords to which TOOLKIT will respond:

*HELP TOOLKIT
gives a list of TOOLKIT commands.

*HELP INFO
gives various useful information, as follows:

1. The values currently set by *FX3, 4, 5, 6, 9, 10, 11, 12.
2. The values currently set by LISTO, WIDTH and @%.
3. ERR, ERL, and the REPORT error message.
4. The same output as the command *FREE (see below).

Abbreviated forms are recognised here, e.g. *HELP TO., *H.TO., *HELP I.

3. TOOLKIT COMMANDS

On the following pages there is a detailed description of each command. The syntax for each command is specified at the top of the section. In these syntax descriptions, items enclosed in parentheses '()' are optional parameters, others are compulsory. Parameters may be separated by spaces in all commands; commas may also be used (except for *MEMORY and *CHECK).

All commands may be used either in direct mode or in a program. Thus it is possible to have some unusual effects; such as a program renumbering itself whilst running, or even jumping into the editor and resuming the program when you leave the editor! The results of such uses of the commands should be regarded as unpredictable.

To send the results of a particular command to a printer, simply enable the printer in the usual way (with Ctrl-B and any necessary *FX calls) before calling the TOOLKIT command.

*CHECK

Function: Verify file contents
Parameters: ⟨filename⟩ (⟨address⟩) (⟨address⟩ +⟨number⟩)

*CHECK will perform a byte-by-byte comparison of memory contents with the contents of a file (either on cassette or disc). If a byte is found to be different an error message will be given and no further checking will be done.

The form without the addresses specified will compare the current Basic program (i.e. that at PAGE) with the file. Otherwise a start and end address in memory can be specified. If no end address is given the end will be determined by the length of the file. The '+' option may be used for the end address in the same way as it is used in *LOAD and *SAVE. All numbers are in hexadecimal.

If the end of the file is reached before the end of the specified memory range or before the end of the Basic program, or if no end address was given, a message 'End of file at hhhh' will be displayed, the hex. number being the byte in memory corresponding to the last byte of the file.

If the file is longer than the memory range no error message will be given, so it is possible to check the contents of the first part of a file only.

Examples:

```
*CHECK PROG
*CHECK PROG 1900
*CHECK data 5000 5300
*CHECK file 2000 +1200
```

*CLEAR

Function: Clear variables
Parameters: none

*CLEAR will clear all variable definitions in exactly the same way as the Basic statement 'CLEAR'. Additionally, though, *CLEAR will set each of the static integer variables (i.e. A% to Z%) to zero.

The integer variables A% to Z% are not assigned any definite value on power up, so it is often necessary to clear those required at the start of a program. This may also be useful when running a new program when the variables retain the values set by a previous program.

*EDIT

Function: Enter screen editor
Parameters: (⟨linenumber⟩)

*EDIT will enter the editor. The command has two forms: if a line number is specified, that line will be shown first; if no line number is given the first line of the program will be displayed. On entry to the editor the screen will be cleared and mode 7 display selected. Then a single line of the program will be displayed with the cursor positioned at the first character of the line (note that the line numbers may not be edited, and a space is always left after the line number). The Escape key is used to leave the editor and return to Basic.

MOVING ALONG A LINE

The cursor may then be moved along the line by means of the left and right cursor keys. At any position in the line any characters typed will be inserted into the line, so the line may be edited as required.

Presing the left or right cursor keys and Shift at the same time causes the cursor to move to the beginning or the end of the line. Note that in this context a 'line' refers to a complete Basic program line which may occupy several lines on the screen.

INSERT OR OVERWRITE

When the editor is entered it will be in 'insert mode' and will function as described above. However, you may wish to overwrite characters in the line instead of inserting. This is done by switching to 'overwrite mode', by pressing the Tab key. When Tab is pressed a soft beep will be sounded as a warning, and an asterisk (*) will be displayed next to the line number. This is simply to let you know that you are in 'overwrite mode'. To revert to 'insert mode' press Tab again; the asterisk will disappear. Note that Shift cursor-right will position the cursor past the last character of the line and cause 'insert mode' to be selected. This is then used for adding to the end of a line.

DELETING

The Delete key will always function as normal, i.e. the cursor is moved back one space and the character there deleted. While editing it is often necessary to delete the character at the position of the cursor, and move the rest of the line back one space. This is achieved by pressing Shift and Delete together. If every character of a line is deleted the complete line will be deleted when you move to another line.

MOVING TO ANOTHER LINE

Any changes made while editing will not be effected in the program until you move off that line. To move to the next line press either the cursor-down key or Return. In the first case the cursor will remain at the same position in the next line (unless the next line is shorter in which case it will be at the end); Return moves the cursor to the beginning of the next line. The cursor-up key will similarly move to the previous line. Pressing Shift together with cursor-up or down will move the cursor to the first or last line of the program.

INSERTING LINES

There is a facility to insert new lines into the program. This is done by typing Ctrl and Tab together. This will cause a line to be added to the program with a line number one higher than that of the current line. If a line of this number already exists a warning beep will be sounded and the editor will merely move to that line.

When a line is inserted it will consist of a single space. This may then of course be edited in the normal way, but if no changes are made the single blank line will remain in the program. To cancel the effect of Ctrl-Tab press Shift-Delete to delete the space from the line, and move to the next line.

LEAVING THE EDITOR

To finish editing press Escape. Note that any changes made to the current line will be ignored, so after editing a line press Return (for example) before Escape. This means that if you make a mistake while editing you can, by pressing Escape, cancel the changes to the current line.

Summary of *EDIT functions

	⟨cursor-left⟩	move one space left
	⟨cursor-right⟩	move one space right
	⟨cursor-up⟩	move to previous line
	⟨cursor-down⟩	move to next line
	Return	move to start of next line
Shift	⟨cursor-left⟩	move to start of line
Shift	⟨cursor-right⟩	move to end of line and enter insert mode
Shift	⟨cursor-up⟩	move to first line
Shift	⟨cursor-down⟩	move to last line

	Delete	backspace and delete
Shift	Delete	delete current character
	Tab	toggle insert mode on/off
Ctrl	Tab	insert line
	Escape	leave editor,ignoring changes on current line

NOTE that the Editor is only entered, on calling *EDIT, if there is a program in the machine to be edited. Otherwise you will be returned to immediate mode in Basic. If you have pressed Break, you will need to reinstate the program with OLD or *OLD before you can use *EDIT upon it.

*FREE

Function: Display Basic memory variables
Parameters: none

This command displays the number of bytes of free memory, the size of the current program, the next free memory location (after the variable storage space), and the values of PAGE, TOP, LOMEM, and HIMEM. Below is an example printout.

```
*FREE

Free memory = 14963 bytes
Program size= 10381 bytes
Next free location= &418D
PAGE= &1900  LOMEM= &418D
TOP = &418D  HIMEM= &7C00
```

*MEMORY

Function: List contents of memory
Parameters: (⟨startaddress⟩) (⟨endaddress⟩ +⟨number⟩)

The *MEMORY command will give a hexadecimal dump of the memory contents between the two specified addresses. The ASCII representation of the memory contents is displayed at the right of the screen.

Paged mode will be engaged so press Shift as usual to get the next page. This selection of paged mode does not affect the setting of paged mode otherwise (i.e. if paged mode was off before using *MEMORY it will remain off afterwards).

Each of the parameters is optional. If the end address is omitted it will default to &FFFF; if both are omitted the listing will start at PAGE. The second parameter may be given as an offset from the first, using the +⟨number⟩ form as in the *SAVE and *LOAD commands. Both parameters must be given in hexadecimal.

Examples:

*MEMORY	list from PAGE.
*MEMORY 1900	list from &1900.
*MEMORY 1900 1BFF	list from &1900 to &1BFF inclusive.
*MEMORY 1900 +300	same as above.

*MERGE

Function: Merge program file with program in memory.
Parameters: ⟨filename⟩

This provides a useful command that BBC Basic lacks. The effect is to merge a program stored on disc or cassette with the program currently in memory to produce a single program in memory. The lines of the program are read from the program file and added to the program in memory. If it happens that the two programs each contain a line of a given number, the line from the file will overwrite the line already there, and a warning message will be given to this effect.

As an example of the use of *MERGE, it is convenient to save frequently used procedures as separate files. They may then be appended to development programs using *MERGE.

If you attempt *MERGE using a file which is not in Basic (eg using a text file, data file or machine code) the message 'Bad File' will be issued.

Note that on cassette systems it is helpful to use motor control when merging a program. This will prevent the tape recorder from reading data before TOOLKIT has finished merging the last block.

*MOVE

Function: Relocate a Basic program
Parameters: ⟨address⟩

This command moves the current Basic program (i.e. the program at PAGE) to the highest page boundary not greater than the specified address, and changes PAGE to this address. It also alters all the necessary internal pointers so that the program will run at its new location.

The address must be in hexadecimal, but may be given in two forms: either the complete address or just the high byte.

*MOVE is most useful for moving a program down in memory after loading from disc, if there is not enough memory for the program to run at &1900. If a program is being moved into the disc workspace (&E00 to &18FF) it is safest to issue the command *TAPE before *MOVE. This prevents the program being corrupted. To resave a program relocate it back up with *MOVE 1900, select *DISC, and then save.

Example of disc use:

```
*TAPE
*MOVE E00
```

Example of cassette use:

```
*MOVE D00
```

The following are all equivalent:

```
*MOVE E00
*MOVE E49
*MOVE E
```

Note that: (1) If you have moved a program into the disc workspace, pressing Break will corrupt it.

(2) If you have a Watford DFS, in the old 1.2 version you should not attempt to relocate Basic programs to below &1400

*NEW

Function: Clear program
Parameters: none

*NEW is identical in effect to the Basic command 'NEW', i.e. it will reset Basic's pointers to show that there is no program in memory. The only difference is that *NEW may be used within a program.

*OLD

Function: Restore program.
Parameters: none

*OLD has the same effect as Basic's 'OLD', but again may be used within a program.

*ON

Function: Enable TOOLKIT error reporting and colour displays (if previously cancelled with *OFF).

*ON will initialise TOOLKIT so that the enhanced error reporting facilities will be used. It will also enable various messages etc. to appear in colour when in mode 7 (if previously cancelled with *OFF).

(i) Error reporting:
After issuing the command *ON, TOOLKIT will intercept the error vector. Any error that occurs while a program is running will then first cause the usual error message, e.g. 'Mistake at line 120'. Secondly the screen editor will be entered at the current line, with the cursor at the approximate position of the error. The error can then be corrected in the usual way (see *EDIT above), and the program re-run.

Note: this facility makes error correction considerably easier, but has its limitations. The cursor position may not be at the exact position of the error. This depends on the type of statement; in some cases Basic will have read past the error before the TOOLKIT error handling routine is called, so TOOLKIT has no way of knowing the exact position of the error.

(ii) Teletext colours:
This command also controls the use of colour. Once *ON has been selected all error messages will appear in red, some warning messages in yellow, and other messages in various colours.

Note that if the output of TOOLKIT commands is sent to a printer the teletext colour codes will corrupt the printout, so enter *OFF before printing.

Notes:

(a) After power-on or Break a compromise between *ON and *OFF states is initiated. Toolkit screens appear in colour, but the special error handling facility is not enabled. *ON will enable this, and *OFF will cancel the colour.

(b) *ON and *OFF may both be used either in direct mode or in a program, so it is possible to use TOOLKITs error handling for just a section of a program.

(c) If the memory from &A00 to &AFF is overwritten (e.g. by using the RS423 for input) while *ON is selected, the effects are unpredictable since this area is used by TOOLKIT. If this occurs simply press Break and don't use TOOLKIT until you have finished using this memory for other purposes.

(d) To allow TOOLKIT to intercept the error vector, there must be no "ON ERROR" statements in the program being run.

*ONF

Function: As *ON but sets function keys f0-f1

*ONF is identical to *ON except that two function keys are set as follows to allow single key access to *EDIT and *UTIL:

f0 - *BEDIT
f1 - *BUTIL

*OFF

Function: Cancel *ON. Cancel colour displays.

*PACK

Function: Compact a Basic program.
Parameters: (⟨options⟩)

*PACK will compact a Basic program by removing all spaces that are not strictly necessary, and by removing all REM statements and comments after '\' in assembly language sections. A message is given detailing the length of the program after compacting, and the number of bytes saved by the process.

Although all vital spaces are left in (e.g. in 'IF A=B A=C'), spaces between a variable name and a keyword are removed. These spaces are necessary when entering a program from the keyboard, since the keyword would otherwise not be recognised, but since the keyword has already been tokenised this is all right. However if a compacted program is now edited by using the Copy and cursor keys the keyword will not be recognised and an error will ensue when the program is run. To overcome this use *EDIT; this automatically reinserts those vital spaces in any editing which you subsequently perform.

There are three forms of the compacting command:

*PACK	removes spaces, REMs and comments.
*PACK S	just removes spaces.
*PACK R	just removes REMs with their suceeding text, and assembler comments.

Note that if you use *PACK to remove REMs, your program should contain no GOTO statements whose destination is a REM line. If this does occur, you will get a "No such line" error message when you run the program. It is generally wise to take a copy of a program before using *PACK for such reasons.

*RECOVER

Function: Recover bad program
Parameters: (+) (⟨address⟩)

The 'Bad program' error message means that a program has somehow become corrupted. However it is often the case that only a small part of it has actually been affected and this could easily be corrected if only the program could be listed and edited. *RECOVER will achieve this in most cases.

How to use *RECOVER:

When you receive the error message 'Bad program' use the following procedure:

(i) Enter *RECOVER. If control does not return to Basic press Escape and see (iv), below. Otherwise list the program and depending on the result, apply step (ii) or step (iii).

(ii) If the program is all there, correct the parts that were corrupted. If any line numbers are out of sequence it will be necessary first to renumber the program; otherwise it will be impossible to edit some of the lines. When the program was originally corrupted it may be that control codes (i.e. bytes of value less than 32) were inserted into the program. In order that the program lists correctly, and to make it easy to pick out these bytes, *RECOVER changes them to # symbols.

(iii) If the 'recovered' program is shorter than the original the '+' option can be used to recover further lines. Simply enter '*RECOVER +' repeatedly until the program is fully recovered (this may not always be possible if the program was badly corrupted at the end). Then see (ii), above.

(iv) If *RECOVER fails to terminate it probably means that the end-of-program marker (a byte &FF) is missing. There are two ways to get around this:

(a) insert an end-of-program marker manually, or

(b) use the second option on the *RECOVER command. To do the latter, estimate where the program ends and specify this address (in hexadecimal) as a parameter.

e.g. *RECOVER 3200

The effect of this will be to insert the required byte at the first suitable place after the specified address. If this recovers only part of the program use *RECOVER + as above.

The two options may be used together, so if *RECOVER + fails to terminate normally, press Escape and use something like *RECOVER + 3500.

How *RECOVER works:

Each Basic program line contains a byte which is set to the length of the line, in bytes. Every line starts with carriage return (a byte of value &D). At various times Basic checks that the program is intact by using these line length bytes to follow through the program. Each line length is added to the address of the start of the line, and should then give the address of the start of the next line, which should contain &D. If at any stage the expected &D is not present, Basic gives a 'Bad program' and refuses to do any more.

*RECOVER gets around this by sometimes inserting missing &D bytes at the end of a line, or correcting corrupted line length bytes. Which one it does depends upon which seems to be the most likely cause of the error; the correct one is virtually always chosen. Once recovery has taken place, the program will need careful editing to restore it to its original state. As mentioned above control codes are replaced by # to allow easy correction by editing, and all occurences of # should be inspected and replaced with whatever occupied that place before corruption occured. Note that if a program is badly corrupted, lines may be split or joined together. New lines created by TOOLKIT as it attempts to repair the damage, will be given the line number 0. Renumbering will restore sequential numbering.

Summary and examples:

*RECOVER	will attempt to recover a program so that it may be listed and edited.
*RECOVER +	will remove the present end-of-program marker and attempt to recover further program lines.
*RECOVER 2600	will insert an end-of-program marker at the first suitable position beyond the hex address 2600.
*RECOVER + 2600	will do both of the above.

*RENUMBER

Function: Renumber program or part of program
Parameters: (⟨first no.⟩) (⟨incr⟩) (⟨start line⟩) (⟨end line⟩)

This is an enhanced version of the Basic 'RENUMBER' command, which enables you to renumber any section of a program. *RENUMBER takes 4 parameters, all of which are optional. If the last two are omitted, the effect is exactly the same as using Basic's RENUMBER, i.e. the whole program is renumbered starting with the given ⟨first no.⟩ (10 if this is omitted), and in increments of ⟨incr⟩ (10 if omitted).

The other two parameters are an addition to the Basic command. If present they specify the section of the program to be renumbered. If either is omitted they default to the first and last lines of the program.

Direct references to line numbers (such as GOTO100 for example) in this section will be altered throughout the program.

The parameters of *RENUMBER may be separated by commas so that any may be omitted by typing ',,'. Line numbers must lie between 0 and 32767 inclusive. The increment must be positive, and may take values greater than 255 (unlike the Basic command).

If the parameters are specified in such a way that after renumbering the program would have line numbers out of sequence, an error message is given, and the program is not changed. This can occur if, for example, a section of a program is renumbered, and the last new number in this section is not lower than the number of the following line (which has not been renumbered). The message 'Overlap' will be given in this case. Other errors that can occur are 'Overflow' if the line numbers become greater than 32767 - renumbering is not carried out in this case, and 'Failed at ...' if a reference to a non existent line is found. This latter is just a warning message, and the program is renumbered regardless.

It is possible to insert embedded teletext codes into programs using shift/function keys. Both the normal Basic Renumber and TOOLKIT Renumber command may occasionally encounter problems if these codes are used.

Note: *RENUMBER causes all variables to become undefined (as does the Basic RENUMBER command).

Examples:	renumbers:	start no.	incr.
*RENUMBER	all	10	10
*RENUMBER 100	all	100	10
*RENUMBER 100 100	all	100	100
*RENUMBER 100,10,200,300	200-300	100	10
*RENUMBER 20 20 400	400-end	20	20
*RENUMBER ,,1000	1000-end	10	10
*RENUMBER 100,,,499	start-499	100	10

*REPORT

Function: Display error report
Parameters: none

*REPORT is equivalent to the Basic 'REPORT:PRINT" at line ";ERL. The error report will be in red if in mode 7 and colour messages have been enabled (see *ON). *REPORT will give the correct error message no matter where the error occurred since it uses the number of the ROM in which the error occurred. (Basic's REPORT will produce nonsense if the last error occurred in, for example, TOOLKIT).

Example of use:

```
IF ERR=17 THEN RUN ELSE *REPORT
```

*SCREEN

Function: Save screen display to a file.
Parameters: ⟨filename⟩

*SCREEN saves the current screen memory to a file with the specified name on cassette or disc. It works in any mode. If using cassette, no messages are given so the screen is not corrupted. When a beep is heard 'play' and 'record' should be pressed, then press any key to save. The screen display can be loaded back in at any time by the statement *LOAD ⟨filename⟩, having once entered the appropriate mode.

This command is designed to be used within a program, e.g. after creating a screenful of graphics.

Note: *SCREEN should only be used if the screen has not scrolled since the last mode change, and should also be reloaded to an un-scrolled screen. To achieve this, clear the screen with a mode change before loading.

*UTIL

Function: Enter utility editor
Parameters: (⟨digit⟩)

The utility editor provides 9 separate utilities which streamline the debugging and editing of programs. When *UTIL is typed, a menu will be displayed and at the top will be the first and last line numbers of the program currently in memory (see below). The menu lists the 9 commands which can be chosen, as well as an option to move directly into the line editor (see *EDIT). The prompt 'Enter option: ' simply requires an option number (you do not need to press Return). After each command you will be prompted for another; pressing Return instead of a number here will return you to the menu.

To exit the utility editor and return to Basic simply press Escape when the 'Enter option: ' prompt is displayed. Pressing Escape while a command is executing will stop that command and prompt for another. If an error occurs during a command, control will return to Basic.

Each option can be used without the menu being displayed, simply by entering, for example, *UTIL 1. This form should be used when printing the output of a command, so that the menu is not printed.

Paged mode is used for all TOOLKIT commands, so press Shift when the display pauses. This does not affect the status of paged mode (ON or OFF) as previously set.

Note: *UTIL output normally uses colours, so if a printout is required enter *OFF first.

*UTIL 6, 7, and 8 will not produce any results until the variables have been actually created by first running your Basic program.

There follows an explanation of each option. In each case any required input will be explained on screen.

1. String search
All occurrences of the required string in the current Basic program will be found; and the complete line where each occurred will be listed with the search string highlighted in cyan (unless *OFF has been entered).

The literal string will be used so keywords will not be tokenised unless the string is prefixed with a '£', when the entire string will be tokenised as if it were a Basic program line. Note that searching for part of a keyword is not possible. Also, the following keywords must be followed by an opening bracket: INSTR(, LEFT$(, MID$(, POINT(, RIGHT$(, STRING$(, and TAB(.

The basic keywords HIMEM, LOMEM, PAGE, PTR and TIME are each stored as two different tokens by Basic, depending upon which side of the expression the keyword is used. For example in the statements 'NOW%=TIME' and 'TIME=NOW%', different tokens would be used for TIME. *UTIL 1 and 2 will always locate the left hand keyword, to search for the right hand keyword the '=' sign must also be entered in the search string. eg'=TIME'.

Wildcards may be used; if the character '@' appears in a string it can be matched by any character. Thus for example, searching for FILE@NAME will match both FILE NAME and FILE-NAME. This facility only applies to strings of at least 3 characters. This enables '@' itself to be searched for (e.g. to find all uses of the variable '@%'.)

The search can be restricted to a section of the program by using option 9, below.

2. Search and replace
The first part of this command is identical to the above command (with keywords tokenised, wildcards, etc). But in this case, when each string is found, it is replaced by the second string (which may be the same length, shorter, longer, or even of zero length). The new line is then listed. If a string is replaced by a longer string any variables will become undefined since the program will encroach into the variable space.

To replace occurrences only in a part of a program use option 9 to restrict the range.

3. Move lines.
This option moves a section of a Basic program to a position elsewhere within the program. The program will then have line numbers out of sequence, and must be completely renumbered before doing anything else. Three line numbers will be requested; the first two specify the section of the program to be moved, and the last is the line number after which the moved lines will be placed. The latter must not lie between the first two, for obvious reasons. If the last line number does not exist the lines will be moved to the place where the line number would be if it existed.

4. List procedures and functions.
This will search a program for DEFPROC and DEFFN statements and list all program lines where these occur. So this gives a complete list of all procedures and functions, together with their formal parameter lists. It does not matter if DEF and PROC are separated by one or more spaces or if they are adjacent.

5. List A% to Z%
The current values of the static integer variables A% to Z% will be listed, both in decimal and hexadecimal.

6. List other numeric variables.
All currently defined numerical variables (other than @% to Z%) will be listed, with their values in decimal and hexadecimal.

7. List string variables.
As above for string variables, listing the contents of the string.

8. List arrays.
Lists all currently defined arrays, with their dimensions. The values assigned to their various elements are not displayed.

9. Change edit range
This will select a part of the program to be used by options 1 and 2. If one of the line numbers is not entered (i.e. only Return is pressed) it will remain unaltered. If neither is entered, the entire program will be selected.

After using this option, the menu will be displayed, with the range of lines at the top. If option 1 or 2 is then selected the search will be restricted to the new edit range. This is cancelled by leaving the utility editor. If *UTIL 1 is used (i.e. without using the menu) the whole program will be used for the search.

4. MEMORY USAGE

TOOLKIT has been designed to be fully compatible with BASIC and EXMON. The main uses of the lower part of the memory are summarised here.

PAGE		
0	0-&4F	BASIC (and EXMON) workspace.
	&50-&6F	TOOLKIT workspace.
	&70-&8F	Free for user.
	&90-&FF	MOS/DFS workspace.
1	Processor stack.	
2-3	MOS workspace.	
4	BASIC static variables and variable pointers.	
5	BASIC FOR/REPEAT/GOSUB stack. TOOLKIT utility editor workspace. (EXMON processor stack save space).	
6	BASIC string buffer. TOOLKIT edit buffer, input buffers. (EXMON string search buffer).	
7	BASIC input buffer. (EXMON workspace).	
8	Sound workspace.	
9	Cassette workspace. RS423 transmit buffer.	
A	RS423 receive buffer. TOOLKIT workspace.	

The memory from &A00 to &AFF should be reserved for TOOLKIT ROM, though if it is overwritten, using any TOOLKIT command will restore it.

The other workspace (e.g. &50 to &6F) may be used, but will be overwritten by using TOOLKIT commands.

5. COMMAND SUMMARY

*B____	All commands can be entered prefixed by a 'B'. This enables commands to be used if another ROM intercepts it first.
*CHECK ⟨fsp⟩ (⟨addr⟩) (⟨addr⟩)	Verify byte by byte a program (or data) in memory against the contents of a file.
*CLEAR	Clear all variables, A%-Z% to zero.
*EDIT (⟨line⟩)	Enter the screen editor.
*FREE	Display Basic memory variables.
*MEMORY (⟨addr⟩) (⟨addr⟩)	Hex. & ASCII memory dump.
*MERGE⟨fsp⟩	Merge a program from file with a program in memory.
*MOVE ⟨addr⟩	Move and relocate a BASIC program.
*NEW	As 'BASIC 'NEW' (can be used in program).
*OFF	Reverses *ON.
*OLD	As BASIC 'OLD'.
*ON	Enables TOOLKIT error reporting and use of colours in teletext mode.
*ONF	As *ON, plus defines f0 as '*EDIT' and f1 as '*UTIL'.

*PACK (R S)	Compact a program, removing unnecessary spaces and/or comments.
*RECOVER (+) (⟨addr⟩)	Recover a bad program.
*RENUMBER (⟨line⟩) (⟨incr⟩) (⟨line⟩) (⟨line⟩)	Renumber a program.
*REPORT	Equivalent to 'REPORT:PRINT" at line ";ERL'
*SCREEN ⟨fsp⟩	Save the screen display to a file.
*UTIL	Enter the menu-driven utility editor.
*UTIL1	String search.
*UTIL2	Search and replace.
*UTIL3	Move program lines.
*UTIL4	List procedures and functions.
*UTIL5	List variables A% to Z%.
*UTIL6	List other numeric variables.
*UTIL7	List string variables.
*UTIL8	List arrays.
*UTIL9	Select subrange for use by UTIL1, UTIL2.