RomIt Manual

CONTENTS

6. Romlt Commands

Appendices

1. INTRODUCTION

1.1 The Features of Romlt

Romlt is an exciting new Rom for the BBC micro which provides the user with a brand new filing system - the Ram Filing System or RaFS for short. This allows you to save and load programs and data to sideways Ram acting like a so-called silicon disc; whether this be in Basic, machine code, or from commercially available software such as Wordwise Plus. This is particularly useful for cassette users because its speed of operation is comparable to that of a disc drive.

But more than just offering a speed advantage over cassette, the RaFS has many useful features which will appeal to disc and cassette users alike. These are further enhanced with the addition of battery back-up for its Ram workspace, so allowing permanent storage, and with it, the full benefits of a silicon disc system.

Romlt allows you to save programs and data on to the silicon disc, and then a special feature enables this to be put into eprom. This means that your favourite, and most frequently used programs may be rommed so that they are always resident in your machine; and if you do not have access to an eprom programmer, Beebugsoft provide a special blowing service for registered users of Romlt. See your Romlt registration card, or telephone Beebugsoft on 0727 40303 for further details.

Moreover, once your software is rommed, you no longer need Romlt to run it, so you can distribute your own Rom software among friends and colleagues, or even market it commercially.

A special "Snapshot" facility allows you to take 'snapshots' of any screen (modes 4-7) at the press of a key, and store them to the RaFS for later retrieval.

Romlt also allows you to autoboot Ram or Roms that you have created - you can even make them produce a start-up message when BREAK is pressed, and respond to the *HELP command.

A further feature allows your sideways Ram to be used as a massive 16K printer buffer, so releasing the computer during printing sessions.

In short Romlt is a fascinating product - and one that is extraordinarily easy to use, as you will see when you work through this manual.

The first 5 sections of this manual deal with the various features of Romlt with step-by-step instructions on their use. Section 6 treats each of Romlt's star commands in an alphabetical sequence, and will form a useful reference section as you get to grips with the product. As an additional aid, a command summary appears as the last appendix.

1.2 Conventions used in this manual

In this manual specific keystrokes required (such as the 'RETURN', 'SHIFT', and cursor keys) are indicated : RETURN, SHIFT and ? ? ? ?. All parameters are shown in this manual enclosed in angled brackets. Single sets of brackets <> are used to indicate essential parameters, while double brackets <<>> indicate optional parameters.

1.3 Fitting Romlt

Romlt is supplied on a sideways Rom and can be fitted either in the main circuit board of your BBC micro or in any kind of sideways Rom board. Please refer to appendix (i) for instructions on how to fit Romlt.

Once Romlt is installed in your machine you may use your computer as normal, but the added commands of Romlt are now available whenever needed.

1.4 Sideways Ram

To get the most out of Romlt you will need to have sideways Ram fitted to your machine. This can be purchased (in the form of a pair of 8K static Rams) for a relatively small sum. Sideways Ram cannot however be plugged directly into your machine. It must be used with a carrier board such as the ATPL Rom board; and we thoroughly recommend this particular board.

The ATPL and other boards also provide a facility known as battery backup. This means that the sideways Ram containing your programs and data will not be cleared whenever your micro is turned off. This makes Romlt an even more powerful tool, as controller of a true silicon disc system.

If you do not have sideways Ram fitted, you may still use Romlt effectively. This is because Romlt can designate part of the Beeb's own user Ram as RaFS workspace (see the commands *RAM+ and *RAM- in section 6 of this manual for further details). If you do use this option you will of course lose the use of either 8 or 16K of user Ram; though if you keep in Mode 7, there is still a good deal of room for your programs. To find out how much room you have at your disposal at any time, either use *FREE if you have Beebugsoft's TOOLKIT or type:

P. HIMEM - PAGE RETURN

1.5 Using Romlt Commands

All the facilities of Romlt are accessed using 'star' commands. Each facility has one or more command words associated with it.

If you type the following:

```
*HELP RAFS RETURN
```

you will see a list of the commands available in Romlt.

Command words may be entered in either upper or lower case letters or even a mixture of both. All of the following commands are legal with Romlt.

```
*RAM RETURN
*ram RETURN
*RAm RETURN
```

The Romlt commands may also be entered in an abbreviated form. The shortest abbreviation that you can use will depend on the types and priorities of other Roms fitted in your computer. Each command has a minimum abbreviation that will allow Romlt to recognise it. In this manual the minimum abbreviation given is the command that will call the Romlt routine when Romlt is the highest priority Rom, apart from Basic and the DFS. For the command, *RAM this minimum abbreviation is *RA. so *RAM could also be entered by typing:

```
*RA. RETURN
```

or even:

```
*ra. RETURN
```

though in this particular case the abbreviation is no shorter than the original command!

1.6 Avoiding Command Name Clashes

Although each Romlt command has its own unique command word, you may find that some of these clash with the command words of other Roms which you may have fitted to your machine. Romlt has a special feature to avoid command name clashes. If any command name clashes, when the Romlt command is required, simply preface the command name with a "B" (for Beebugsoft) eg type:

```
*BRAM RETURN
```

instead of just:

```
*RAM RETURN
```

This will ensure that the command is intercepted by Romlt rather than any other Rom. There are no clashes of command words with any other Beebugsoft Roms.

If by contrast there is a conflict of command names, and Romlt intercepts (and executes) a command destined for some other Rom, this may be prevented as follows. A "B" is again used to preface the command name, but this time the "B" must be in a different case to that of the first letter of the command name following it. When Romlt detects this case change, it strips off the "B" (or "b") and passes on the rest of the command to other Roms down the line.

To clarify these conventions by way of example, the following commands are all legal with Romlt and have differing effects. The example presupposes that two Roms, Romlt and some other, both use the command name *RAM.

`*RAM` will execute the Romlt routine, if Romlt is in a higher priority to the clashing Rom.

`*ram` will execute the Romlt routine if Romlt is in a higher priority to the clashing Rom.

`*BRAM` will execute the Romlt command regardless of priority.

`*bram` will execute the Romlt command regardless of priority.

`*bRAM` will execute the routine of the clashing Rom if it is in a lower priority to Romlt.

`*Bram` will execute the routine of the clashing Rom if it is in a lower priority to Romlt.

1.7 Parameters

Some of the Romlt commands require parameters to control their action. When parameters are being entered with a command, they should follow the command directly, but may each be preceded by a space for clarity.

For example, the Romlt command to copy a file from the RaFS to disc is:

```
*FILE R DISC <filename>
```

Thus to copy the file "PROG" from Ram to disc you would type:

```
*FILE R DISC RETURN
```

1.8 Optional Parameters

Some parameters, such as that used with the "BUFFER command, are optional. Optional parameters are indicated in this manual with a double angle bracket thus:

```
*BUFFER <<number>>
```

1.9 Getting Started

With Romlt fitted to your machine the RaFS can be initialised by simply issuing the command *RAM thus:

```
*RAM RETURN
```

If you do not have sideways Ram fitted to your machine this will come back with the error message No RAM. To overcome this enter:

```
*RAM+ RETURN
```

and then do as prompted. This selects user Ram rather than sideways Ram for RaFS use.

Whenever you call *RAM (or any of its variants), Romlt will display the free space available in the RaFS in both hexadecimal and decimal.

Once you have called *RAM (or *RAM+ or *RAM-), each time you save a program it will be saved to Ram, used as a silicon disc. You can catalogue the silicon disc with *CAT or (*.) and load in programs with the LOAD command; and to return to your disc or cassette filing system simply type:

```
*Disc RETURN
```

or

```
*TAPE RETURN
```

as appropriate. A more detailed discussion of the RaFS is given in the next section.

You should note that pressing BREAK will always take you out of the RaFS, and leave your machine in its default filing system, which will probably be disc or cassette. The following command will however put you back into the RaFS each time that BREAK is pressed; though it will not work after CTRL BREAK:

```
*KEY10*RAM¦M RETURN
```

Note that the ¦ character used above is on the key to the left of the CURSOR LEFT key.

2. THE RAM FILING SYSTEM.

2.1 Introduction

The majority of commands found in Romlt are used to implement a new filing system known as the Ram filing system, or RaFS. This filing system works either in sideways Ram (if fitted), or directly into a reserved area in user Ram.

The structure of this filing system owes much to the disc filing systems employed by the BBC micro; and you may look on the RaFS as providing an alternative filing system to cassette or disc, on which you can save and load programs and data, and perform other allied tasks; hence the term "Silicon Disc".

You will find in fact that all of the commands used with the cassette filing system will also operate within the environment of the RaFS. These are summarised below, and the user is referred to the User Guide for descriptions of each, since each works with the RaFS in the same way as described there. As a further guide, you are also referred to appendix (ii) which gives User Guide page references, and some additional notes.

```
BGET# BPUT# *CAT CHAIN CLOSE# EOF# *EXEC EXT# INPUT# LOAD *LOAD
OPENIN OPENOUT OPENUP *OPT PRINT# PTR# *RUN SAVE *SAVE
```

If your machine has a DFS (disc filing system) fitted you will find that there are four extra DFS commands which may be used with the RaFS. These are:

```
*BUILD *DUMP *LIST *TYPE
```

These are also treated in appendix (iii).

Further to this Romlt provides an extensive set of commands for use with the RaFS. These latter are treated in detail in section 6 of this manual in an alphabetical sequence. Many are also treated in the foregoing sections dealing with the various applications of Romlt. They are listed here for reference:

```
*ACCESS *APPEND *BLOCK *DELETE *ENABLE *FILE *INFO *MEND *OVERWRITE
*RBASIC *RBOOT *RELOAD *RENAME *RESTRICT *REXEC *RHELP *RTITLE *SNAP
*SPACE *UNWIPE *WIPE
```

For every one of the three groups of commands described above to work on the RaFS, the command "RAM (or one of its variants) must have been issued prior to their use, and must not. have been cancelled by calls to any other filing system (eg *DISC etc), or by pressing BREAK.

If the RaFS is not active when any of these commands are called, the first two groups will operate as they would normally do without Romlt installed, while commands in the last group will not be recognised.

2.2 Using the RaFS

In this section we will work through some of the RaFS commands to give you a feel for the way in which they operate, and to show you how to get "instant" results.

As you will soon discover, the RaFS is very easy to use. This is because it behaves just like other filing systems on the BBC micro with which users will already be familiar.

First of all switch on your machine, and select the RaFS by issuing the command:

```
*RAM RETURN
```

or by using one of its variants, *RAM+ or *RAM-, (treated in section 6) if you have no sideways Ram fitted in your machine.

This will produce a message giving the amount of free memory available to the RaFS.

Now type in a few program lines. If you save your program as usual, it will be saved not to cassette or disc but to the silicon disc. Type for example:

```
SAVE"MYPROG" RETURN
```

The RaFS (like the cassette filing system) allows filenames of up to 10 characters, but does not support directories of the kind implemented on the DFS.

Having saved "MYPROG" to the silicon disc, you can catalogue the disc with:

```
*CAT RETURN
```

or

```
*. RETURN
```

and you should see that your program now resides in the RaFS Ram. Just to prove it you could clear the Basic workspace with:

```
NEW RETURN
```

Then load your program back in using:

```
LOAD"MYPROG" RETURN
```

If you type:

```
LIST RETURN
```

you should find that the program has been loaded back into user memory. Alternatively you will find that you can also use CHAIN to load and run your program, just as from cassette or disc. To do this type:

```
CHAIN"MYPROG" RETURN
```

You can save any number of Basic programs using the SAVE command as described above, the only limit being the total amount of Ram available to the RaFS. To find out how much free memory remains, you can use the *SPACE command. Just type:

```
*SPACE RETURN
```

More detailed information may be obtained by using the *INFO command, which operates in much the same way as with the DFS. Type:

```
*INFO RETURN
```

or

```
*IN. RETURN
```

for short. Having done this you should get a display that looks something like this:

```
RAM FILING SYSTEM (RaFS)
                     !BOOT code:   NO
SOCKET: F 16K        !HELP code:   NO
MEMORY: &8000-&BFFF  !
                     TITLE code:   NO
End of RAM at..................  805C
Free space in RaFS..... (16291) 3FA3
```

Do not worry if all this does not mean a lot to you now. The *INFO command is simply giving you information on the RaFS, and the information given shows you the number of the socket containing the sideways Ram, it's length (whether 8 or 16K) and memory map, as well as the number of bytes available to you in the RaFS, and the address at which code in sideways Ram ends. The !BOOT, !TITLE and !HELP code messages serve to indicate whether or not three special features of Romlt have been initialised.

A further way of getting information about files stored in the RaFS is to type:

```
*OPT1,2 RETURN
*CAT RETURN
```

The user is referred to the User Guide for a full explanation. OPT1,2 also gives information when saving and loading files. Its effect may be turned off with:

```
*OPT RETURN
```

Returning to our example, you may like to experiment with saving and loading further programs on the RaFS. If you wish to delete any saved program, the command *DELETE will achieve this. Thus to delete the program "MYPROG", just type:

```
*DELETE"MYPROG" RETURN
```

A similar command, *WIPE, is used to clear the whole RaFS area - though if you change your mind before saving new programs to the wiped memory, you can use *UNWIPE to undo its effects.

Because the action of *WIPE is so drastic, it must, as a precaution, first be enabled with *ENABLE (or *EN.). So to wipe your RaFS Ram, proceed as follows:

```
*EN. RETURN
*WIPE RETURN
```

In common with the DFS the RaFS also provides a Rename facility. If you have saved a program called "MYPROG", you can change its name to "NEWNAME" as follows:

```
*RENAME MYPROG NEWNAME RETURN
```

Program and data files may also be locked to prevent accidental erasure. To lock "MYPROG", type the following:

```
*ACCESS MYPROG L RETURN
```

If you now catalogue the silicon disc, you will see that there is a letter "L" adjacent to the filename "MYPROG".

All attempts to Rename, overwrite or delete this file in any way except using the global *WIPE command, will produce the message:

```
FILE LOCKED
```

To unlock the file, type:

```
*ACCESS MYPROG RETURN
```

You may have noticed earlier that as you load programs in from the RaFS, a number of things appear on the screen. The words Searching and Loading followed by the program title MYPROG will have appeard, just as when loading a program from cassette. A block count is also printed, though if you have a very short program, you may not notice this. There will also be a beep when loading is complete.

If you wish to disable these visual and audio signals, just enter the following:

```
*OPT1,0 RETURN
```

Pressing BREAK will cancel the effect of the OPT command, as will typing:

```
*OPT RETURN
```

To obtain more detailed loading information, you may like to try using *OPT1,2 before loading a program.

It is suggested that you experiment further with the general RaFS commands provided by Romlt. To assist you, we list below a collection of the most general commands with a brief description. Full details on each are given in section 6 of this manual, and a full command summary appears in appendix (vi). You are also urged to experiment with the use of the cassette filing system commands. These are listed with notes in appendix (ii).

*ACCESS	Permits locking and unlocking of files
*APPEND	Allows storage of more than one file with same filename
*BLOCK	Transfers a block (8 or 16K) to or from RaFS Ram

*DELETE	Deletes a file
*ENABLE	Used with *WIPE and *BUFFER
*FILE	Copy a file to or from RaFS Ram
*INFO	Information on RaFS Roms and Ram
*MEND	Mend broken files
*OVERWRITE	Cancel the effect of *APPEND
*RBASIC	Used in connection with *RESTRICT
*RBOOT	Used to enable BOOT option
*RELOAD	Change the load address of a file
*RENAME	Rename a file
*RESTRICT	Enable copy-protection
*REXEC	Change execution address of a file
*RHELP	Used to enable HELP option
*RTITLE	Used to enable TITLE option
*SNAP	Activate auto-screen-dump
*SPACE	Return space remaining on RaFS Rom or Ram
*UNWIPE	Reverse the effect of *WIPE
*WIPE	Wipe the whole RaFS Ram

3. SPECIAL FEATURES OF THE RAFS

!BOOT, !TITLE, !HELP and *SNAP

Romlt allows you to store program and data files to Ram using its Ram filing system, and in the following section we will explain how to galvanise the contents of your Ram into Rom (or eprom), so that your files will not be lost once your machine is switched off.

But Romlt also allows the creation of three special types of file not yet discussed. These files always take the names !BOOT, !TITLE and !HELP. Very briefly the !BOOT facility allows an auto-boot option from files stored in Ram or Rom (in a similar way to the DFS); the ITITLE option can put a title or other message on the screen every time that BREAK is pressed; while !HELP allows you to set up your own HELP option which will respond to the *HELP call regardless of whether the RFS or the RaFS are enabled or not.

These three features are now treated in some detail, and are followed in this section by a description of the Snapshot screen dump option.

3.1 !BOOT Files

The !BOOT option allows you to set up a program which automatically runs any time that you press SHIFT R BREAK (ie hold down the SHIFT and R keys, and tap BREAK).

Doing this, effectively performs a *EXEC !BOOT to the RFS. The !BOOT file is then EXECed into memory and run. The !BOOT file itself will typically be a short program to call other programs from the Rom or Ram. For example it might take the form:

```
*TV255
CHAIN"MENU"
```

To set up the BOOT option you need to create a special file called !BOOT, and then to initialise the feature with the command *RBOOT.

The format of the !BOOT file follows exactly that of the !BOOT files used by the DFS. It contains no line numbers, and may be created in one of three special ways.

1. If you have a DFS fitted, then you can use the *BUILD option. This puts line numbers on to the screen, but saves the !BOOT file without them. To use this facility, type:

```
*RAM RETURN
*BUILD !BOOT RETURN
*TV255 RETURN
CHAIN"MENU" RETURN
ESCAPE
```

and the job is done.

Unfortunately the Watford DFS does not allow you to create named *BUILD files directly. If you are using this DFS you should proceed as follows:

```
*BUILD DUMMY RETURN
*TV255 RETURN
CHAIN"MENU" RETURN
ESCAPE
*RENAME,!BOOT RETURN
```

As you can see, this is a little more involved; we have had to create an un-named *BUILD file on disc, then rename it !BOOT, and finally transfer it to the RaFS.

2. You may also use a wordprocessor such as Wordwise Plus. To do this using Wordwise Plus, proceed as'follows:

```
*WORD. RETURN
*RAM RETURN
ESCAPE (to get into editing mode)
*TV255 RETURN
CHAIN''MENU" RETURN
ESCAPE (to return to the menu)
```

Then select option 1, and save the file under the name IBOOT.

If you have Wordwise rather than Wordwise Plus, you will not be able to call *RAM from within the wordprocessor. In this case, proceed as above, except that you should not call *RAM, so that you will save to tape or disc.

You then need to use *FILE to transfer your !BOOT file created on tape or disc, to the RaFS.

To do this, proceed as follows:

```
*FILE DISC R !BOOT RETURN
```

or

```
*FILE TAPE R !BOOT RETURN
```

3. If you do not have a DFS or a suitable wordprocessor, you may create a !BOOT file using a Basic program. The following program will create a !BOOT file identical to that discussed above:

```
10*RAM
20*SPOOL !BOOT
30PRINT"*TV255"
40PRINT"CHAIN";CHR$34;"MENU";CHR$34
50*SPOOL
```

The subterfuge in line 40 is just to persuade the PRINT statement to print the pair of quotation marks around the filename. If you now run this program, it will place a file called !BOOT directly into the RaFS containing the two lines:

```
*TV255
CHAIN"MENU"
```

Assuming that you have created the !BOOT file by one of the above three methods, it only remains to execute *RBOOT to initialise the option. Just type:

```
*RBOOT RETURN
```

You should note that *RBOOT should not be set when no corresponding !BOOT file exists, as this may hang the machine under certain circumstances.

The effect of *RBOOT may be cancelled by typing:

```
*RBOOT- RETURN
```

Now, whenever you now press SHIFT R BREAK (ie hold down the SHIFT and the R keys, and tap BREAK) your !BOOT file will automatically run. In this case it will issue the *TV command to lower the screen display by one line, and then CHAIN a program called MENU, which should also be resident in the RaFS.

It should be noted that if you have a number of RFS Roms in your machine with !BOOT options, the one which will auto-boot will be the one which is in the highest priority socket. Also, since pressing SHIFT R BREAK puts you into Acorn's Rom filing system, you will not be able to write to RaFS Ram until you have executed *RAM.

3.2 !TITLE Files

The purpose of !TITLE is to allow your Rom (or Ram) to produce a screen message whenever BREAK is pressed. In this way your Rom can, if you wish, announce itself in the same way as some commercial Roms; though the message which you create may of course serve any purpose that you wish.

The rules for using this facility are very similar to those for !BOOT described above. You must create a !TITLE file containing the text that is to be printed, and you must enable the option with the command *RTITLE.

As with *RBOOT, *RTITLE must not be implemented when no corresponding !file exists; and the effect of *RTITLE may be cancelled by typing:

```
*RTITLE- RETURN
```

The !TITLE file takes a very similar form to the !BOOT file described above, and the user is referred to the three methods outlined above for its creation.

In the case of !TITLE, the file must contain only the text to be printed (no PRINT statements are required), with a blank line at the end.

To clarify this, if you are using method 3 to create your file, you might use the following program:

```
10*RAM
20*SPOOL !TITLE
30PRINT"My ROM v1.0"
40PRINT"© 1985 Me"
50PRINT
60*SPOOL
```

Remember that because of the way in which *SPOOL works, the PRINT statements that appear in this program will not appear in the ITITLE file created.

If you run this program, and type:

```
*RTITLE RETURN
```

You should see something like the following each time that BREAK is pressed:

```
BBC Computer
```

```
My ROM v1.0
© 1985 Me
Acorn DFS

BASIC
```

You should note that for a screen message to be printed in this way, the Rom or Ram carrying the message must be in a higher priority socket than the DFS, if one is fitted.

If you have more than one RFS Rom in your machine using the !TITLE option, the machine's operating system will confuse which !TITLE file to display for each Rom. This may be avoided by labelling each !TITLE file with a unique code as follows.

When you create the !TITLE file, you may append its name with up to four characters which will uniquely define it. For example, line 20 of the program above might be altered to:

```
20*SPOOL !TITLERom1
```

Any combination of numbers and/or upper and lower case letters may be used for the code, as you wish. Now, when the command *RTITLE is issued, Romlt looks at the !TITLE file to see what code you have used, and uses this as an identifier each time that the TITLE file is accessed.

If for any reason you change your !TITLE code in a given Rom (or Ram), you should re-issue the command *RTITLE, so that Romlt can register your new code. Of course, once you have rommed your code it can no longer be changed.

3.3 !HELP Files

The purpose of !HELP is to allow your Rom (or Ram) to produce a response to the operating system command *HELP. Many commercial Roms use this feature to give a summary of the commands which they provide; and some also use it to give syntax information. This is the case with Romlt itself, as you will see if you type:

```
*HELP RETURN
```

Romlt will announce itself along with other Roms in your machine.

To produce a similar response from your own Rom (or Ram) you will need to create a !HELP file containing the text to be printed; and then enable the HELP facility with the command *RHELP in much the same way as described above for ITITLE. There is just one small difference in the !HELP file. This must contain a blank line at the start of text, rather than at the end as with !TITLE.

Suppose that you wish to produce the HELP message:

```
Screenshot Display Rom
Boot with SHIFT R BREAK
```

If you are using the program method (method 3 above) for generating the !HELP file, the program would read as follows:

```
10*RAM
20*SPOOL !HELP
30PRINT
40PRINT"Screenshot Display Rom"
50PRINT"Boot with SHIFT R BREAK"
60*SPOOL
```

If you run this program, this will save the appropriate !HELP file into Ram. You then need to type:

```
*RHELP RETURN
```

This will initialise the facility, and now every time that a *HELP is issued, whether from within the RaFS or not, your message will be among those appearing on screen.

As with *RBOOT and *RTITLE, the HELP facility can be turned off when so desired. This time the command to use is:

```
*RHELP- RETURN
```

Again, if you have a number of RFS Roms in your machine which have the HELP facility enabled, the operating system will confuse which HELP file belongs to which Rom. To avoid this, the same convention should be used as with !TITLE.

In other words the filename !HELP should be appended with a code of up to four characters which uniquely defines the Rom to which the HELP facility belongs; though you do not have to use the same identifying code as used with !TITLE in the same Rom. When you subsequently call *RHELP, Romlt will take a note of the code, and make use of it each time to ensure the correct destination of *HELP calls.

3.4 Saving Screens with *SNAP

The contents of any screen in modes 4 to 7 may be saved to the RaFS using the *SAVE command as with the cassette or disc filing systems. But Romlt provides a special command *SNAP (and *SNAPR)

to do this automatically. This can be useful for various purposes; and particularly where mode 7 is concerned, since the 16K RaFS could hold 14 complete screens.

To use the SNAP facility proceed as follows:

```
*RAM RETURN
*SNAPR RETURN
```

You may now create the screen which you wish to save. At any point, even if you are not in the RaFS or RFS, pressing:

```
CTRL @
```

(ie hold down CTRL then tap @) should freeze the screen, and await the pressing of one of the number keys (0-9). Suppose that 5 is pressed, Romlt will save the screen to the RaFS under the filename SNAP5.

To load in the screen at any time, you must select the correct mode, then *LOAD the screen data. For example:

```
*RAM RETURN
MODE7 RETURN
*LOAD SNAP5 RETURN
```

If when *SNAP was called, it was called without the "R" following it (R for RaFS), then when the screen was saved on pressing CTRL @ , it would be saved to the currently active filing system; which may or may not be the RaFS.

You should note that when saving and loading screens on the BBC micro, whether using *SNAP or any other means, the screen which you save must not have been scrolled prior to saving. Similarly, you should ensure that a screen to which you load a screen dump, such as that produced by *SNAP, should not have been scrolled once the screen mode has been selected.

4. PRODUCING ROM SOFTWARE

A special feature of Romlt is its ability to generate the protocol necessary for putting programs or data into Rom in the so-called Rom filing system format. If an Eprom is blown following this format, it may be used in any BBC machine regardless of whether Romlt is fitted or not, and it will behave in exactly the same way as the Ram in the RaFS from which it was generated -except of course, that it cannot be written to and, more importantly, it will not be erased when you turn your machine off.

This opens up the possibility of users romming their favourite software, either for their own use or for the use of others. If you do not have an Eprom blower, you may use the special Eprom blowing service operated by Beebugsoft for registered users of Romlt. For details of this service, see your Romlt registration card, or telephone Beebugsoft on 0727 40303.

Suppose that you have created a piece of software that you wish to put onto Eprom. The first stage is to thoroughly test and debug it. When you have done this you should save your program to the RaFS. To do this type:

```
*RAM RETURN
SAVE "MYPROG" RETURN
```

Where "MYPROG" is the filename that you have chosen.

For the moment we will assume that you wish the Eprom to hold just the program "MYPROG". If you wished it to contain additional programs, these should now be saved to RaFS. Also any of the special options such as the !TITLE, !BOOT, !HELP and file locking options (see *RESTRICT in section 6) should also be set up at this point. But once again we will assume for the moment that these are not required.

Once the RaFS Ram contains all that the proposed Eprom is to contain, you proceed in one of two ways depending on whether you are going to blow the Eprom yourself, or whether you are preparing a disc or tape to send to us to Rom for you.

If you are preparing a disc you should use the following:

```
*BLOCK R DISC MYROM RETURN
```

Cassette users should type:

```
*BLOCK R TAPE MYROM RETURN
```

Where "MYROM" is the filename of the disc (or cassette) version of your Rom. This should be sent to Beebugsoft with your registration number, and the appropriate fee.

If you are going to blow the Eprom yourself, you first need to know whether your particular blower requires any unused bytes to be in the form FF hex. If so, issue the following command:

```
*PAD RETURN
```

Even if it does not, it will do no harm to issue a call to *PAD.

Next you will need to relocate the contents of the RaFS into the Beeb's own workspace where your Eprom blower can copy it. To do this type:

```
*BLOCK R M <page> RETURN
```

where <page> signifies the top two bytes of the page boundary address in hex of the start of the workspace which is to carry the relocated code. Disc users will probably relocate to &1900, and will use:

```
*BLOCK R M 19 RETURN
```

By the same token, cassette users will probably use &0E00 (call *BLOCK R M 0E); but you are advised to consult the operating instructions for your blower on this point.

Once the configured code is in the correct area of memory, you can blow your Eprom exactly as instructed in your Eprom programmer manual.

Once the Eprom is programmed, you can plug it into one of the sideways Rom sockets, and autoboot it with SHIFT R BREAK (see * BOOT), or by calling up the Rom filing system with *ROM, and then using LOAD, *LOAD or CHAIN as appropriate.

You will discover that you can access your rommed software after a *RAM or a *ROM call. The RaFS supplied by Romlt has all the necessary routines to control the Rom. If the RaFS is not fitted in the machine in which your newly created Eprom is to be used, you may activate it with *ROM (pressing SHIFT R BREAK has the same effect). This calls up the so-called Rom filing system which Acorn have written into the Beeb's operating system.

To check that your Rom is correctly installed, you could catalogue the Rom (or Ram) filing system as follows:

```
*RAM RETURN
```

```
*CAT RETURN
```

This will list all the filenames of the files stored in RFS and RaFS format. If you have more than one Rom or Ram used in this way, you will see that the filenames are grouped together in blocks on the screen. Each block represents one Rom or Ram.

If on performing a *CAT you discover groups of filenames that you did not expect, it may be that you have in your machine one of the few commercial Roms (such as WordEase) which make use of the RFS format. In this case you must be careful not to set up auto-boot options in your own Roms which may prevent aother Rom from booting up correctly (especially when the latter is placed in a lower priority socket than your own).

5. USING RAM AS A BUFFER

An important feature of Romlt is its ability to provide the user with an extensive buffer area which can be used to extend the BBC micro's internal buffers. These include the keyboard, sound, speech and RS423 buffers. But probably the most useful is to set up sideways Ram as a printer buffer.

This will give you more than 15K of buffer area (with a 16K Ram), which will be sufficient to release your micro very quickly when printing even substantial lengths of text.

The command which makes all this possible is *BUFFER. It has an optional numeric parameter to select the function to which the buffer is to be put. This may take the value 0-8 as in the accompanying table. So that *BUFFER 8 would select the speech buffer, and so on.

It is a feature of the BBC micro that pressing ESCAPE flushes, or clears, all internal buffers. Since this may be inconvenient when using the special Ram buffer described here, this feature has been disabled.

If however you wish ESCAPE to flush the designated buffer, you should use as numeric parameter the number in parentheses in the table below (with values in the range 128-136).

Thus *BUFFER 130 would select the RS423 output buffer; and the buffer could be flushed by pressing ESCAPE .

Parameter	Buffer type
0 (128)	Keyboard buffer
1 (129)	RS423 input buffer
2 (130)	RS423 output buffer
3 (131)	Printer buffer
4 (132)	Sound channel 0
5 (133)	Sound channel 1

6 (134) Sound channel 2

7 (135) Sound channel 3

8 (136) Speech buffer

It is important to note that when the buffer facility is used it will erase the whole contents of the sideways Ram, which may contain RaFS files.

To avoid the loss of RaFS files when using *BUFFER you may easily save the current contents of the Ram as follows:

```
*BLOCK R DISC name RETURN
```

or

```
*BLOCK R TAPE name RETURN
```

This will store the whole contents of RaFS Ram under the filename "name". To load it back into Ram, type:

```
*BLOCK DISC R name RETURN
```

or

```
*BLOCK TAPE R name RETURN
```

Because use of the buffer has the destructive effect described, the facility must, as a precaution, be first enabled with the command *ENABLE (or *EN.).

Thus, to set up sideways Ram as a printer buffer which will not be cleared on pressing ESCAPE , use the following:

```
*EN. RETURN
*BUFFER 3 RETURN
```

Because it is assumed that this will be the most frequent use of the Buffer command; this particular option has been installed as the default. This means that if you require to use the printer buffer, with ESCAPE not clearing the buffer, you may simply type:

```
*BUFFER RETURN
```

It should be remembered that *BUFFER should be called from within the Ram filing system. If any filing system commands are required when using the buffer, the filing system required should be first initialised (eg type *DISC once *BUFFER has been called).

To return to RaFS use when you have finished with the buffer facility, type

6. ROMIT COMMANDS

This section contains a full description of each Romlt command. For a discussion of the use of star commands with Romlt and for a discussion of parameters used with them, see sections 1.5-1.8 of this manual.

*ACCESS

Syntax:	`*ACCESS <filename> <<L>>`
Minimum Abbreviation:	`*AC.`
Function:	Locks or unlocks the named file.

Use of this command will prevent the named file from being deleted with the command *DELETE, renamed with *RENAME, or being overwritten with

*SAVE. To so protect a file use the following form of the command:

```
*ACCESS <filename> L RETURN
```

When a file has been locked in this manner the INFO (see *INFO) of the file will show the letter L under the heading Flag.

To unlock the file so that it may be deleted or overwritten use the same command but without the second parameter, L:

```
*ACCESS <filename> RETURN
```

IMPORTANT

Locking files does not prevent them from being destroyed by the use of

*WIPE although if this command is accidentally issued it may be undone by using *UNWIPE .

*APPEND

Syntax:	`*APPEND`
Minimum Abbreviation:	`*AP.`

Function:	Enables the RaFS to hold more than one program having the same filename.

Normally, saving a file into the RaFS will cause any old file having the same name to be overwritten. However, by issuing *APPEND the RaFS can be made to behave in a manner not unlike a the cassette filing system, thus allowing the same filename to be used without deleting any old versions. The new version is saved at the end of the RaFS so you know which is the latest version.

This facility might at first not seem very useful, but during program development you might like to keep all your old versions, just in case your latest crashes. This can easily be done by programming a function key thus:

```
*KEY0 SAVE"PROGRAM"¦M RETURN
```

Simply pressing function key 0 will then save your latest version with the filename PROGRAM at the end of the RaFS, space permitting. If the message 'RAM full' occurs you can always *DELETE the oldest version, as all commands such as LOAD, *DELETE, *ACCESS, *RENAME etc., always operate on the first found occurrence of the specified filename.

To return to the default system whereby saving deletes a file of the same name you use the command *OVERWRITE .

*BLOCK

Syntax:	`*BLOCK <source> <destination> <page>`
or	`*BLOCK <source> <destination> <filename>`
Minimum Abbreviation:	`*BL.`
Function:	To copy the entire block of sideways Ram area either:

1) To or from a specified page in the BBC micro's user Ram area.

2) Between sideways Ram and a selected filing system.

```
*BLOCK <source> <dest.> <page>
```

If you wish to blow the current contents of your RaFS Ram into a Rom the

*BLOCK command can be used to shift the entire contents of the RaFS, whether it is in user or sideways Ram, to a selected position in user Ram.

The parameter <page> signifies the top two bytes of the page boundary address in hex of the start of the workspace which is to carry the relocated code. For example, you would use 19 if you wanted to specify &1900.

The <source> and <destination> refer to the location of the block of sideways Ram. The single letter parameter represents the following:

R Ram
M Memory

The normal default page for cassette systems is &0E00. Hence to shift the entire contents of the RaFS from sideways Ram into memory that starts at page &E (decimal 14) you would enter:

```
*BLOCK R M 0E RETURN
```

Similarly disc users may want to shift the RaFS code from the RaFS to memory starting at &1900, and would enter:

```
*BLOCK R M 19 RETURN
```

To perform the reverse operation, transfering the block of code from memory to the RaFS, requires the M and R paramters to be reversed. The <page> parameter now signifies from where the block of code is to be transferred. For example, to move a block of code from &1100 to the RaFS you would enter:

```
*BLOCK M R 11 RETURN
```

The *BLOCK command shifts everything in the RaFS including all files, headers and service code. Do nor therefore try to run anything that has been shifted as you will probably only succeed in crashing the computer. This command is primarily intended for moving code from the RaFS to a position in user memory so that it may then be blown into a Rom. The type of Eprom programmer in use will determine the page address to be used, so please consult the manual of your blower to find this out. If you do not have access to an Eprom blower, but still want to have your code blown into a Rom, Beebugsoft will undertake this for you.

```
*BLOCK <source> <dest.> <filename>
```

The second use of this command is to transfer the complete contents of the RaFS Ram directly to or from disc, tape, or any other filing system.

The <source> or <destination> may be R for Ram, and also the filing system specified by name.

For example, to save the contents of the RaFS to disc under the filename 'DATA' you would issue the following:

```
*BLOCK R DISC DATA RETURN
```

Similarly, to load a previously saved RaFS Ram area back into sideways Ram from tape at 300 baud, enter:

```
*BLOCK TAPE3 R DATA RETURN
```

Either the first or second parameter must be an R. Other filing systems that may be used with this command include:

ADFS	Advanced Disc Filing System
DISC	Disc filing system (or DISK)
IEEE	IEEE 488 filing system
NET	Econet filing system
RAM	Ram filing system
ROM	Rom filing system
TAPE	Tape filing system (1200 baud)
TAPE3	Tape filing system (300 baud)
TAPE12	Tape filing system (1200 baud)
TELESOFT	Teletext filing systems

*BUFFER <<number>>

Syntax: `*BUFFER <<number>>`
Minimum Abbreviation: `*BUF.`

Function:	To turn Ram (sideways or user) into a buffer area. Because using sideways Ram as a buffer will destroy any files stored with the RaFS, this command must be enabled with the *ENABLE command, as a precaution against accidental use.

Issuing the *BUFFER command without the number parameter will turn the Ram used by Romlt, whether it is sideways or user Ram, into a printer buffer. Therefore, to list a program out to a printer you would issue the following commands:

```
*ENABLE RETURN
*BUFFER RETURN
VDU2 RETURN (or CTRL-B)
LIST RETURN
VDU3 RETURN (or CTRL-C)
```

With the number parameter being employed the Ram may be allocated as any of the following buffers:

Parameter	Buffer type
0 (128)	Keyboard buffer
1 (129)	RS423 input buffer
2 (130)	RS423 output buffer
3 (131)	Printer
4 (132)	Sound channel 0
5 (133)	Sound channel 1
6 (134)	Sound channel 2
7 (135)	Sound channel 3
8 (136)	Speech buffer

The numbers in brackets may be used if you want the buffer to be flushed when ESCAPE is pressed. If you use the lower number the contents of the buffer will be preserved when ESCAPE is pressed. This is of particular importance when using the printer buffer.

When *BUFFER is called without any parameters it defaults to a printer buffer in such a way that ESCAPE will not flush the contents, (ie *BUFFER is the same as *BUFFER13).

When the buffer is in use all buffer related commands work as expected. Therefore, ADVAL(-(N+1)) will return the free space in the buffer, *FX15 will flush it as will *FX21, *FX138 will insert a character into the selected buffer and *FX145 will remove a character. Details of all of these commands may be found in the BBC User Guide.

To disable the buffer simply issue the command *RAM.

IMPORTANT.

Use of the BUFFER facility offered by Romlt will destroy any files held in the RaFS.

*DELETE

Syntax:	`*DELETE <filename>`
Minimum Abbreviation:	`*DE.`
Function:	To delete the named file.

This command will delete the named file from the RaFS and then close up the space occupied by that file. This is the same as doing a *DELETE followed by a *COMPACT on the DFS. The command will only operate on a file that has not been locked with *ACCESS. If the file has been locked the error message 'File locked' will be issued.

*ENABLE

Syntax:	`*ENABLE`
Minimum Abbreviation:	`*EN.`
Function:	To enable the use of the commands *WIPE and *BUFFER.

Since the commands *BUFFER and *WIPE can cause the entire contents of your Ram to be lost, the command *ENABLE must be issued immediately before, in order to prevent accidental misuse. For example:

```
>*WIPE RETURN
Not enabled

>*ENABLE RETURN
>*WIPE RETURN
Free bytes in RAM E .....(16291) &3FA3
>
```

*FILE

Syntax:	`*FILE <source> <destination> <filename>`
Minimum Abbreviation:	`*FI.`
Function:	To transfer a program to the RaFS from another filing system or vica versa.

This command will transfer the specified file between the RaFS and any other specified filing system. This command is of particular use to those who have to employ the RaFS in user Ram, and therefore do not have sufficient room in memory for both the program and the RaFS.

During the transfer all headers are added to the program being transferred, and the correct load and execution addresses are added.

The <source> or <destination> may be R for RaFS Ram (whether in user Ram or sideways Ram), and also the filing system specified by name fie by its proper title).

For example, to transfer the file called PROGRAM from the RaFS to disc you would issue the following:

```
*FILE R DISC PROGRAM RETURN
```

Similarly, to load a previously saved file back into sideways Ram from tape at 300 baud, enter:

```
*FILE TAPE3 R PROGRAM RETURN
```

Either the first or second parameter must be an R.

For other filing systems that may be used refer to the *BLOCK command. In each case the specified source or destination should be the approved filing system name, For example, NET for Econet, TAPE or TAPE12 for the cassette filing system.

*HELP

Syntax:	`*HELP`
or	`*HELP RFS`
or	`*HELP RAFS`
or	`*HELP UTILS`
Minimum Abbreviation:	`*H.`
Function:	To provide information on commands and their syntax.

Typing *HELP by itself will cause a display similar to that below to be shown:

```
*HELP

RAM Filing System 1.00
RFS
RAFS
UTILS

TOOLKIT PLUS 2.00
TOOLKIT

SLEUTH 1.06

DFS 0.90
DFS
UTILS

OS 1.20
```

Since this simply lists the resident Roms in your machine, the display might not be exactly as that shown. Typing *HELP followed by one of the three abbreviations RFS, RAFS and UTILS, produces the following displays:

```
*HELP RFS

     BGET# <handle>
     CHAIN <f ilename>
     CLOSE# <handle>
     EOF# <handle>
```

```
     INPUT# <handle>,<variable list>
     LOAD <filename>
     OPENIN <filename>

     *CAT
     *EXEC <filename>
     *LOAD <filename> <<load address>>
     *OPT <number>,<number>
     *RUN <filename>

*HELP RAFS

     *ACCESS <filename> <<L>>
     *DELETE <filename>
     *RAM <<socket>> <<S>>
     *RAM <<+>> <<S>>
     *RAM <<->> <<S>>
     *RBOOT <<->>
     *RHELP <<->>
     *RTITLE <<->>
     *RENAME <oldname> <newname>
     *RESTRICT <filename> <<R>>
     *REXEC <filename> <address>
     *RELOAO <filename> <address>

     BPUT# <handle>,<number>
     OPENOUT <filename>
     PRINT# <handle>, <variable list>
     SAVE <filename>
     *SAVE <filename> <start> <end>
     *SPOOL <filename>

*HELP UTILS

     *APPEND
     *BUFFER <<number>>
     *BLOCK <source> <dest.> <page>
     *BLOCK <source> <dest.> <filename>
     *ENABLE
     *FILE <source> <dest.> <filename>
     *INFO
     *MEND
```

```
*OVERWRITE
*PAD
*RBASIC
*SNAP <<R>> <<D>>
*SPACE
*UNWIPE
*WIPE
```

Please note that if you have a DFS fitted that also responds to *HELP UTILS, use of this will put up the information shown above as well as the DFS information. As this is so, it is probably advisable to set the screen to the paged scrolling mode via CTRL-N (ie press CTRL and tap N), before issuing the *HELP, as this will enable you to read the commands without them scrolling off the top of the screen.

*INFO

Syntax:	*INFO
Minimum Abbreviation:	*IN.
Function:	To provide information on the RaFS and any files therein.

Similar in many respects to the *INFO commands of the DFS, this command provides information about the files held in the RaFS, and data on the state of the RaFS itself. Typing *INFO will produce a display similar to that below:

```
RAM FILING SYSTEM (RaFS)
                     !BOOT code:     NO
SOCKET: F 16K        !HELP code:   game
MEMORY: &8000-&BFFF  !
                     TITLE code:     NO
Name      Flag   Load    Run Length Start
TEST        LR FF1900 FF8023   0100  806C
!HELPgame    L FFFFFF FFFFFF   0067  816C
End of RAM
at...................                81D3
Free space in RaFS......
(15917)                              3E2D
```

This example simply shows the type of display achieved, and of course will differ unless you have a file named TEST saved in the RaFS.

The information given is as follows:

SOCKET	Indicates the currently selected Ram socket used by the RaFS and size of the current RaFS in kilobytes. If this is user Ram the letter displayed would be U.
MEMORY	Gives the memory map (area covered) of the RaFS.
!BOOT code	This will indicate the four character identifier if the command *RBOOT has been used.
!HELP code	This will indicate the four character identifier if the command *RHELP has been used.
!TITLE code	This will indicate the four character identifier if the command *RTITLE has been used.
End of Ram	Gives the physical address of the byte following the last used byte in the RaFS.
Free space	Gives the remaining number of bytes in the RaFS. This is given in both decimal (in brackets), and in hex.

The information heading the program TEST is as follows:

Flag:	If the letter L is shown here the file has been locked with *ACCESS. If the letter is R then the file is restricted with *RESTRICT.
Load:	The load address of the file.
Run:	The execution or run address of the file.
Length:	The length of the file in bytes. The figure is in hexadecimal.
Start	Gives the physical address at which the file has been saved within the memory map of the RaFS

Filenames used with the RaFS may be up to 10 characters in length. Unlike the DFS you may not specify directories or libraries within the filename. This means that if you try to save a file with the filename S.TEST it will be saved as S.TEST, and not as the file TEST in directory S.

*MEND

Syntax:	*MEND
Minimum Abbreviation:	*ME.
Function:	Attempt to restore the RaFS following a Bad ROM error.

Following a 'Bad ROM' error subsequent commands operating on the RaFS will also present the same error. Use of this command will attempt to rectify this matter by removing the last file from the end of the RaFS, as this is the area where corruption is most likely to have occurred.

We must stress that this command only attempts to rectify the situation, but is successful in the majority of cases; in the remainder, the only effective answer is to issue the commands *RAM followed by *WIPE. If that doesn't work the only answer is to switch the machine off and then back on again. This is guaranteed to rectify even the most severe corruption of the RaFS! If you have battery backup, you may need to switch it off and even remove the Ram chips so they can discharge their memory.

*OVERWRITE

Syntax:	*OVERWRITE
Minimum Abbreviation:	*OV.
Function:	To reset RaFS to default where saving a program overwrites an earlier saved file with same name.

If you have issued the *APPEND command the *OVERWRITE command may be used to revert the RaFS back to it's default position. In the default condition, or after the *OVERWRITE command has been used to cancel

*APPEND, saving a file will first try to delete a file which has the same name.

*PAD

Syntax:	*PAD
Minimum Abbreviation:	*PAD
Function:	To pad the free area of the RaFS ready for Eprom blowing.

Some makes of Eprom blower work more efficiently if all unused bytes are set to hex &FF (decimal

255). This command simply does this for you by placing the value &FF in every unused byte in the RaFS. If the Eprom blower you are using does not require this operation then you can ignore this command.

*RAM

Syntax:	`*RAM <<socket>> <<S>>`
or	`*RAM+ <<S>>`
or	`*RAM- <<S>>`
Minimum Abbreviation:	`*RAM`
Function:	To select the RaFS and the area of Ram in which Romlt has control.

If the command *RAM is issued Romlt searches through the sideways sockets from number 15 down to 0 until it finds sideways Ram. When Ram is found the socket number is allocated to the RaFS and the free space in the RaFS displayed. The number displayed when *INFO is typed is the socket allocated in this manner. The command *RAM with no parameters selects the highest priority Ram located.

If no Ram is found in the sideways sockets 15 to 0 the error message 'No RAM' will be issued. To activate the RaFS when no sideways Ram is present, either *RAM+ or *RAM- may be issued (see below).

If a socket number is specified, i.e. *RAM E, the RaFS will be allocated the Ram located in that socket, provided of course that Ram actually resides there. If it does not, the 'No RAM' error message will be given. The socket number must be issued in hexadecimal, so *RAM E would activate Ram in socket 14 if Ram is found there.

`*RAM+ *RAM-`

If you do not have sideways Ram resident in your machine, the RaFS may be assigned either 8K or 16K of user Ram. To select an 8K RaFS issue the command *RAM-, and to select the 16K version issue the command *RAM+.

Issuing either of these commands necessitates that the Ram area is protected from other use, and to do this you should press the letter R and BREAK as prompted. The area allocated to the RaFS is located from OSHWM upwards, and PAGE is reset accordingly.

Do not forget that even in mode 7 the use of a 16K RaFS in user Ram will limit what else can be held in the memory. If you have an Acorn DFS fitted, the actual space left will be about 9.25K (9472 bytes, &2500 in hex) if the 16K RaFS is selected. With cassette based systems the 16K RaFS will leave about

11.5K of useable Ram (11776 bytes, &2E00 in hex).

```
*RAM S
```

If you wish to suppress the screen display each time the RaFS is called up, you may use the S parameter (S for suppress). To do this replace:

*RAM with *RAM S
*RAM + with *RAM+ S
*RAM- with •RAM- S

*RBASIC

Syntax:	`*RBASIC <filename>`
Minimum Abbreviation:	`*RB.`
Function:	Renders a BASIC program in a form that may be *RUN.

Use of this command will render a program already held in the RaFS under the filename specified in a form that means that when it has been restricted with *RESTRICT the program may be *RUN as if it were a machine code program. This command should therefore be used on all BASIC programs that are to be blown into an Eprom and which you want to protect from prying eyes.

When this command has been used, some special code is attached to your program to make it respond to *RUN as if it were a machine code file. By then restricting the file with *RESTRICT it may not be run by any command other than *RUN. This therefore provides your software with a good degree of protection.

Cassette users who modify software with the *RBASIC command and then intend it to be used on a disc system, should emulate the disc enviroment by ensuring that page is set to &1900. Before writing or loading programs, you should enter:

```
PAGE=&1900 RETURN
NEW RETURN
```

*RBOOT

Syntax:	`*RBOOT<<->>`
Minimum Abbreviation:	`*RBO.`

Function:	To implement or disable a facility that allows a Rom to be auto-booted.

By issuing the command *RBOOT, special code is added to that held in the RaFS so that it is possible to automatically exectute the commands in a file called !BOOT when you press the keys R SHIFT BREAK in the following sequence:

```
Press SHIFT
Press R
Press BREAK
Release BREAK
Release R
Release SHIFT
```

This is called 'booting' a program.

The file called !BOOT holding the instructions must be made and saved in Ram. For example, to automatically select graphics mode 7, and then run a program called MENU, it would be necessary to create a file consisting of the Basic commands:

```
MODE 7
CHAIN"MENU"
```

If you are unfamiliar with creating such a file, you should refer to section 3.1.

```
*RBOOT-
```

The special code added to the RaFS to look for a !BOOT can be removed by issuing the command *RBOOT-.

*RELOAD

Syntax:	`*RELOAD <filename> <address>`
Minimum Abbreviation:	`*RE.`
Function:	To change the load address of a file held in the RaFS.

This command will directly change the load address of a file that is already held in the RaFS under the filename specified to the new address you require. The address you specify here must be in hexadecimal but the leading '&' is not required.

The address you give may be upto 32 bits in length (8 digits), and, although only 6 digits are displayed via *INFO, all 8 are significant and are stored. This command may be of little use to BASIC programmers, but will be of great use to machine code programmers.

*RENAME

Syntax:	`*RENAME <oldname> <newname>`
Minimum Abbreviation:	`*REN.`
Function:	To rename a file held in the RaFS.

Provided the file specified as <oldname> is stored in the RaFS and has not been locked via *ACCESS this command will change the filename to that specified as <newname>. If the file has been locked the error message 'File Locked' will be given. If there is no file under the filename <oldname> held in the RaFS the error message 'File not found' will be issued.

For example, to change the name of a file called TEMP to EXAMPLE, enter:

```
*RENAME TEMP,EXAMPLE RETURN
```

*RESTRICT

Syntax:	`*RESTRICT <filename> <<R>>`
Minimum Abbreviation:	`*RES.`
Function:	To prevent any file from being accessed by any command other than *RUN. This provides a means of software protection.

Use of this command restricts the named file so that it can only be *RUN and not be operated on by any other 'read' command. Use of this command in the RaFS stops normal loading or chaining of programs.

If a Rom is blown containing nothing but restricted files other users can then only *RUN those files. This therefore provides an extremely useful form of software protection. This is of course of immediate use to machine code files, but is of little use to Basic files until they have been modified by the 'RBASIC command of Romlt.

Please note that it is not advisable to restrict any data files as they cannot then be accessed via OPENIN, BGET#, *LOAD etc, or files that are used with *EXEC such as !BOOT, !TITLE and !HELP.

If the 'read' commands other than *RUN are attempted on a restricted file you will receive the error message: Locked

For example, to restrict a file called MY.PROGRAM so that it can not be loaded or chained, enter:

```
*RESTRICT MY.PROGRAM R RETURN
```

*REXEC

Syntax:	`*REXEC <filename> <address>`
Minimum Abbreviation:	`*REX.`
Function:	To change the run address of a file held in the RaFS.

This command is very similar to *RELOAD and will directly change the execution, or run address of a file that is already held in the RaFS under the filename specified, to the new address you require. The address you specify here must be in hexadecimal but the leading '&' is not required.

The address you give may be upto 32 bits in length (8 digits), and, although only 6 digits are displayed via *INFO, all 8 are significant and are stored.

This command may be of little use to BASIC programmers as BASIC programs automatically load to PAGE and the run address is then ignored.

*RHELP

Syntax:	`*RHELP<<->>`
Minimum Abbreviation:	`*RH.`
Function:	To implement or disable a *HELP facility in your code.

By issuing the command *RHELP special code is added to that held in the RaFS which will respond to the *HELP command by printing the contents of the text file !HELP.

The command *RHELP- will remove the code that *RHELP added to the RaFS. Either command may be used at any time without affecting anything already held in the RaFS, but if there is insufficient room to add the required code a 'RAM full' error message will be given.

There are three ways to construct the Ascii file !HELP for use with this command, and all are fully covered in section 3.1 to this manual. However, the simplest method is for users with a DFS fitted to

enter the following:

```
*BUILD !HELP RETURN
1 <blank line> RETURN
2 First line of text RETURN
3 Second line of text RETURN
4 etc., etc., RETURN
5 ESCAPE
```

This will then give you the *HELP facility in your code so that when *HELP is issued your text is printed out. You may of course have as many or as few lines as you like, just so long as there is room in the RaFS to accomodate it all.

Suppose you have blown several Roms, each of which have the special code and corresponding !HELP file. Issuing the command *HELP will print the text in the !HELP file in the highest priority Rom socket only.

This may be avoided by using a filename that starts with the characters !HELP, but contains up to four additional characters as an identifier. For example, the following filenames are valid:

```
!HELP2
!HELPFour
!HELP+
```

When the command *RHELP is issued, it looks for a file that starts with the characters !HELP, and remembers the characters followng it.

If at any time you rename your !HELP file, you must enter *RHELP again so that the special code can be informed of the change.

The four characters following the !HELP filename which the special code currently identifies may be found by entering the *INFO command.

*RTITLE

Syntax:	`*RTITLE<<->>`
Minimum Abbreviation:	`*RT.`
Function:	To add or remove service code to print a title message when BREAK is pressed.

By issuing the command *RTITLE special code is added to that held in the RaFS which means that you

can then create a file with the filename !TITLE which will be printed on the screen whenever the BREAK key is pressed.

The command *RTITLE- will remove the code that *RTITLE added to the RaFS. Either command may be used at any time without affecting anything already held in the RaFS, but if there is insufficient room to add the required code a 'RAM full' error message will be given.

Please note that in order for the title to be printed on the screen once your code has been blown into an Eprom this Rom must be in a higher priority socket than the DFS, as this Rom inhibits any further Rom titles from being printed. Therefore, with the DFS in socket 14 and your Rom in socket 13 this function will not operate. However, reverse the position of the two Roms and the title will appear when BREAK is pressed.

The way in which a !TITLE file is built is exactly as given for *RHELP above. However, this time the information stored in that file will only be printed when the BREAK key is pressed. If your code contains several commonly used procedures to help you in your programming you could put the following in the !TITLE file. Again this shows the use of *BUILD, so users who do not have a DFS fitted should refer to section 3.1 which gives other methods for constructing this file.

```
*BUILD !TITLE RETURN
1 Graphics Procedures RETURN
2 *HELP gives names RETURN
3 <blank line> RETURN
4 ESCAPE
```

Suppose you have blown several Roms, each of which have the special code and corrosponding !TITLE file. Pressing BREAK will print the text in the !TITLE file in the highest priority Rom socket only.

This may be avoided by using a filename that starts with the characters !TITLE, but contains up to four additional characters as an idenifier. For example, the following filenames are valid:

```
!TITLErom7
!TITLE!!!
!TITLE-IAN
```

When the command *RTITLE is issued, it looks for a file that starts with the characters !TITLE, and remembers the characters following it.

If at any time you rename your !TITLE file, you must enter *RTITLE again so that the special code can be informed of the change.

The four characters following the !TITLE filename which the special code currently identifies may be

found by entering the *INFO command.

*SNAP

Syntax:	*SNAP<<R>>
Minimum Abbreviation:	*SN.
Function:	To save the current screen to RaFS under the filename SNAP.

Any time after issuing this command, pressing the CTRL key and the @ key together, releasing them and then pressing any other key will save the screen display to the RaFS under the filename SNAP plus the character of the key you pressed. For example:

```
*SNAPR
CTRL @
9
```

would save the current screen under the filename SNAP9 when the 9 is pressed.

To reload and display the screen image just saved, for example, SNAPS, it necessary to enter:

```
*LOAD SNAP9 RETURN
```

To use this SNAP-shot facility *RAM must have been previously issued. After issuing *SNAP any program may be run, and, if it is not protected, you may save any screen to the RaFS simply by pressing CTRL @ and another key.

Please note that this will not work with modes 0 to 3, as they are all too long to be stored in the RaFS. However, modes 4 to 7 may be saved in this way, and upto fourteen mode 7 screens may be held in the RaFS.

```
*SNAP
```

If *SNAP is called without the trailing R (as in *SNAPR) above), the screen will be saved to the currently activated filing system. This may or may not be the RaFS. (The trailing R selects the RaFS).

It should be noted that a screen that is to be saved with this method should not have been scrolled, as this can upset its internal start position.

For further details on the *SNAP command, refer to section 3.4.

*SPACE

Syntax:	`*HELP`
Minimum Abbreviation:	`*SPA.`
Function:	To display free space left in the RaFS.

When this command is issued the following display will result:

```
Free bytes in Ram E .... (16291) &3F93
```

The actual number displayed will of course depend on what you have in the RaFS, and that shown is how the command responds to a 16K RaFS with nothing having been saved to it.

Please note that the actual number of bytes given does not mean that a program of that length may be saved into the RaFS. This is because of the way the RaFS is handled. Each file saved is split into 256 byte blocks, and each block has a header. This header occupies a few extra bytes, so the actual space left to programs is less than that displayed. Note also that issuing either *RBOOT, *RHELP or *RTITLE also uses up space in the RaFS.

*UNWIPE

Syntax:	`*UNWIPE`
Minimum Abbreviation:	`*UN.`
Function:	Undo the action of *WIPE.

If the command *WIPE has been issued *UNWIPE will restore the contents of the RaFS. This however will only work if nothing has been saved to the RaFS since the *WIPE command was issued. *UNWIPE is equivalent to using OLD on a NEWed Basic program.

*WIPE

Syntax:	`*WIPE`
Minimum Abbreviation:	`*WI.`
Function:	To clear the RaFS of all files.

Issuing *WIPE has exactly the same effect on the RaFS as NEW does on a BASIC program. It may have its action reversed by using *UNWIPE.

*ENABLE must be issued immediately before *WIPE since the command will destroy the entire contents of the RaFS including those files that have been locked via *ACCESS. However, like NEW in Basic this is not fatal as it can be undone with *UNWIPE.

For example:

```
*ENABLE RETURN
*WIPE RETURN
Free bytes in RAM E .... (16291) &3FA3
>
```

APPENDIX I

ROM FITTING INSTRUCTIONS

WARNING!

Roms ore very sensitive devices and should ho installed with great care, ensuring that you do not touch the metal legs.

GAINING ACCESS TO YOUR ROM SOCKETS

1. Disconnect the computer from the mains.

2. Unscrew the pair of screws at the top of the back panel which may be labelled 'FIX'.

3. Unscrew the pair of screws located underneath and towards the front of the computer which may be also labelled 'FIX'.

4. Lift the lid off the computer.

5. If you are installing Roms in the model B+ BBC micro, you should now proceed to instruction 20.

6. If you are installing the Roms in your own expansion board, you should now proceed to instruction 24.

MODEL B INSTALLATION

7. Carefully unscrew the two (maybe three) nuts and washers holding the keyboard assembly to the computer case. These screws are located about five centimetres behind the ones removed in stage 3.

8. Lift the front of the keyboard, rotate it backwards, and carefully lay it on the back part of the computer. There is no need to remove the multi-wire ribbon cable attached to the main board, or the speaker wire.

9. On the very bottom right of the circuit board there are five large sockets. The one on the far left holds the operating system Rom and must not be interfered with. On the circuit board itself to their immediate left and belween the four rightmost sockets are the identifying marks (from left to right): IC52, IC58, IC100, IC101.

10. One of those sockets holds the Basic Rom. It ran be identified by a serial number on its surface ending with either BO1 or BO5. If you have a disc filing system, then a Rom with a sticker will identify it as a 'DFS'.

11. If there! is a vacant socket to the LEFT of the socket holding the Basic Rom. then your new Rom should be inserted there, in a manner described from instruction 15.

12. If there are no vacant sockets to the right of your Basic Rom then there is no more room in your computer for additional Rom software. You will either have to remove one of the less important Roms, or consider purchasing an expansion board such as the ATPL Sidewise or Watford expansion hoards.

REMOVING A ROM

13. Remove then Basic Rom by prizing up each end a liltle at a time, by pushing| a small screwdriver tip under an end and gently twisting. Be careful not to to scratch thn printed circuit board or bend any of the pins. Always pull the chip out vertically and not from one end as this can fracture the pins at the other end. A special Rom extractor tools will help in the removal proress.

14. The Basic Rom should be inserted into the right-most vacant socket available as described below.

INSERTING A ROM

15. Before inserting your Rom, identify a small notch at its end. Holding the ends of the Rom between thumb and forefinger line up the legs of it with the holes in the socket, ensuring that the notch is pointing AWAY from you towards the back of the computer.

16. New Roms often have their legs splayed out so they are too wide to line up with the holes in their socket. If you find this to be the case, hold the Rom at both ends between thumb and forefinger, and place it on its side so that the legs are flush with a hard surface. Carefully apply pressure to bend the legs so that they appear more angled. If you over bend the legs, use the edge of a table to bend them back again.

17. Apply film pressure, ensuring that all the pins have entered their respective holes, and none are bent out or underneath.

18. if you have just relocated your Rasic Rom. you will now want to insert your new Rom as described from instruction 15.

19. Put the keyboard back to its normal position (do not do the screws up yet) and continue at instruction 25.

MODEL B+ INSTALLATION

20. Locale the eight large sockets towards the top right of the circuit board. The two bottom sockets on their own (marked on the circuit board as IC29 and IC37) are reserved for the speech system and do not concern us here.

21. The top right socket (labelled IC71 between itself and the socket below) is taken by the combined Operating System and Basic Rom, and is labelled with Acorn's copyright. One of the remaining five sockets will contain the disc filing system Rom and will be labelled 1770 DFS.

22. Your new Rom should be inserted in any one of the vacant five remaining sockets (labelled on the circuit board as IC35, IC44, IC57, IC62, and IC68) as described from instruction 15.

23. Continue from instruction 25.

EXTENSION ROM BOARD INSTALLATION

24. Refer to the manafacturer's Rom fitting instructions.

TESTING THE ROM

25. Before reassembling the machine, test that the Rom has been correctly installed as described below.

26. Reconnect mains power and switch on taking care to avoid contact with the circuit board. The machine should function as normal, however if it does not, switch it off and check that the Rom is the correct way round and all the pins on the Rom are inserted correctly. If the message 'Language?' appears it means that your Basic Rom is not being recognised and should be inspected.

27. When you are satisfied with the installation, reassemble your machine in reverse order!

28. Turn the machine on, and enter: *HELP RETURN

This will display the title and version number of all the Roms in your machine (Except Basic). Your new Rom should be amongst them.

PROBLEMS

29. If you have inserted you Rom into the computer the wrong way round and it has become permanently damaged, we offer a replacement service for a nominal charge.

APPENDIX II

ROMIT COMPATIBLE COMMANDS

This appendix deals with those filing system commands that are supplied by either the in-built cassette and Rom filing systems, or by any fitted disc filing system. Each is listed together with the relevant pages in the BBC micro User Guide, and the Advanced Uner Guide. If any of the commands require further clarification a number in square brackets is given, and that indicates the relevant paragraph following this list of commands.

CASSETTE/ROM filing system commands

COMMAND	Standard User Guide	Page number in Advanced User Guide	Further Information in Paragraph
ADVAL(-N)	202	175	[1]
BGET#	212	184	[2]
BPUT#	213	185	[2]
*CAT	391	361	
CHAIN	216	188	
CLOSE#	219	192	[2]
EOF#	249	220	[2]
*EXEC	394	364	
EXT#	256	227	[2]

INPUT#	279	249	[2]
LOAD	292	261	
*LOAD	393	363	
OPENIN	311	279	[2]
OPENOUT	313	281	[2]
OPENUP		282	[3]
*OPT	398	368	[4]
PRINT#	328	299	[2]
PTR#	330	301	[2]
*RUN	392	364	
SAVE	344	314	
*SAVE	392	362	
*SPOOL	402	371	

Pages 391 to 390 of the BBC User Guide should bo read if you are unclear on the manner in which the cassette filing system works, as the operation of those commands within the RaFS is virtually identical.

[1| The use of the negative ADVAL command is only of use if you are using the RomIt buffer facilities. For further information on this see the command *BUFFER.

[2] All random access filing commands work in a manner that is not dissimilar to that employed by a DFS, but within the constraints of a serial file environment. This means that once a file has been closed it may only be extended up to the point where the next file header is located. If you try to extend the file past this point the error message 'Can't extend' will be issued. If the file is the last held in the RaFS this problem should not occur. Pages 188 to 193 of the BBC User Guide should be rend in connection with

these commands. The filing handle assigned to the RaFS for write operations is hex &4F, and for read operations hex &4E.

[3] This is a BASIC II command, so users with BASIC I will not be able to use it. It opens up a file for both reading and writing.

[4] To gain access to a restricted form of the *INFO command you may use *OPT1,2 followed by *CAT which will give you the file name along with number of blocks in the file, the file length and the load and run addresses. The Flag status is not given as this is Romlt-specific, and supplied by the commands *ACCESS and *LOCK . Note that the load and run addresses are given as the full 32 bits (8 digits) and not as the 24 bits (6 digits) given via *INFO.

DISC filing system commands

There are only four DFS specific commands that might be used with the RaFS, and these are:

```
*BUILD
*DUMP
*LIST
*TYPE
```

Please refer to the relevant page of your DFS manual for further details on each command. All four commands work exactly as if they were working with discs. However, *BUILD does require some attention here.

***BUILD**

Syntax: *BUILD <filename>
Function: To build an Ascii file.

You will already be aware of how *BUILD works in conjunction with your DFS, and working with the RaFS this remains the same. However, it is worth pointing out that if you build a file under the name of ! BOOT you can then boot your RaFS code once it has been blown into an Eprom by pressing SHIFT R BREAK This is exactly the same as pressing SHIFT BREAK to boot a disc.

However, not all makes of DFS use *BUILD as expected. For example, Watford DFS version 1.2 onwards omits the filename, and saves the file with

a null filename. This can quite easily be rectified by using the *RENAME command. Tor instance, if you are building the !BOOT file you would rename the un-named file with:

```
*RENAME ,!BOOT
```

Here the comma acts as a delimiter and specifies that the first filename is in fact null, or empty.

If you do not have a DFS fitted to your machine there are various other ways of creating a !BOOT file, and these may he found in Section 3.2 to this manual.

NOTE

You DFS manual will also contain details of how to use random access files, and this should be read in conjunction with the information found in the BBC User Guide.

APPENDIX III

ERROR MESSAGES

The following is a list of error messages that may occur when using Romlt. Numbers in brackets indicate the error numbers generated and held in the variable ERR. Where there is no number it means that this is a Romlt-spocific error which, although operating in the normal manner, may not bo handled through the pseudo-variable ERR.

Bad address (252)

Signifies that the address attempted is incorrect.

Bad attribute (207)

The second parameter with *ACCESS can only be L, so this error indicates that this parameter is incorrect.

Bad command (254)

Either the leading * has been omitted or the command has not been recognised by any Rom in the computer.

Bad filename (204)

An attempt has been made to use a filename of more than 10 characters in length.

Bad Hex (28)

An attempt has been made to use a hex number that cannot be recognised. i.e &12S4

Bad option (203)

A mistake has been used in the first parameter following the command *OPT.

Bad PAGE

The page address required is incorrect, or might have been omitted.

Bad ROM (215)

This message appears if the RaFS is corrupted in any manner.

Can't Extend (191)

An attempt has been made to extend a random access file where there is insufficient room immediately after it to do so.

Channel (222)

Channel used has not previously been opened.

Data? (216)

This indicates a Cyclic Redundancy Check error. Try *MEND to rectify this error.

Eof (223)

An attempt has been made to read past the end of a file.

File locked (195)

The tile has been locked so cannot be operated on by anything other than *RUN.

File not found (214)

The specified file cannot be located.

File open (194)

An attempt has been made to open a file that is already open.

Header? (217)

This indicates a Cyclic Redundancy Check error. Try *MEND to rectify this error.

No RAM

Signifies that the RaFS cannot locate sideways Ram.

RAM full

Signifies that there is insufficient space in the RaFS to add what has been attempted.

APPENDIX IV

ACCESSING THE RaFS FROM MACHINE CODE

Section 43 of the BBC User Guide between pages 442 and 466 contains most of the information you will need in order to be able to use the RaFS in assembler. In addition any information contained within your DFS manual should also be read. All of these OS routines operate exactly as detailed in these manuals, but to clarify matters the information given here should be of some help. There are however, some: differences that are peculiar to the RaFS and these are detailed in this appendix.

OSFIND

Entry address: &FFCE Indirection vector: &21C

Exactly as given in the BBC User Guide. On exit Y will contain a channel number if the file has been succcesfully opened, or 0 if it was unable to open the file. However, the RaFS requires that Y is assigned a unique filing handle for the read and write operations. Details are given under USBPUT, OSBGET and OSGBPB.

OSARGS

Entry address: &FFDA Indirection vector: &214

Exactly as given in the BBC User Guide. The RaFS has however been officially allocated its own unique filing system identification number, which is 12. Therefore, with A=0 and Y=0, use of OSARGS will return in A the number 12 if the RaFS is active. At the same time it also places the Rom number in zero page location &F5.

If Y is non-zero the value of A determines the action to be taken on the file whose handle (see OSBPUT, GSBGET and OSBPB) is in Y. The BBC User Guide however is not very clear on this, so the following should help.

A=0 Read sequential pointer = Read PTR#
(BASIC; A%=PTR#<channel>)

A=1 Write sequential pointer = Write PTR#
(BASIC; PTR#<channel>=A%)

A=2 Read sequential file length = Read EXT#
(BASIC; A% = EXT#<channel>)

In all other respects OSARGS is implemented exactly as in the BBC User Guide.

OSFILE

Entry address. &FFDD Indirection vector: &212

Exactly as in the BBC User Guide. However, to clarify the action taken according to the value of A consider the following:

A=0 SAVE, *SAVE

A=1 Write the catalogue information for the named file.

A=2 Write load address only = *RELOAD

A=3 Write execution address only = *REXEC

A=4 Write attributes only = *ACCESS

A=5 Read named file's catalogue information, place file type in A.

A=7 Delete the named file = *DELETE

A-&FF LOAD, *LOAD

OSBGET

Entry address: &FFD7 Indirection vector: &216

Exactly as per the BBC User Guide except that the channel number allocated via OSFIND should be safely stored away somewhere in memory and the value of Y should be set to &4E (decimal 78). If this is not done the 'Channel' error will occur. The value of Y (&4E) is a unique filing handle allocated to the RaFS for read operations.

OSBPUT

Entry address: &FFD4 Indirection vector: &218

As above, but now the required filing handle is &4F (decimal 79). This is the filing handle allocated to the RaFS for all write operations.

OSGBPB

Entry address: &FFD1 Indirection vector: &21A

This facility is not implemented in the cassette filing system, but will be known to disc users. For the benefit of both the following should be of some use.

OSGBPB stands for OS Get Byte Put Byte, and is a general random access command. It allows files to be OPENed for both reading and writing. The BASIC 2 keyword OPENUP performs the same task. The routine will read or write a byte (or group of bytes) to or from a specified and previously opened file. The option to read or write depends on the value of A and the filing

handle used. The length of the data to be passed and its locution are specified in a control block located somewhere in memory.

On entry X (lo-byte) and Y (hi-byte) point to this control block which is set up as follows.

Offset	
XY+0	Filing handle
XY+1	Pointer to data (address)
XY+5	Number of bytes to be passed
XY+9	

Byte offset in file (if used)

XY+13

The value of A determines the operation to be performed.

A=1 PUT byte(s) using byte offset.

A=2 PUT byte(s) ignoring byte offset.

A=3 GET byte(s) using byte offset.

A=4 GET byte(s) ignoring byte offset.

A=8 Read files names.

With A=1 or A=2 the filing handle at XY+0 should be &4F, and with A=3 or A=4 this should be &4E.

On exit C clear signifies a successful transfer. C set indicates the the end of file was reached before the transfer had been completed. In this case the number of bytes and the byte offset (if used) are altered to indicate just how much data has been passed, and the new pointer value is the old value plus the number of bytes passed.

OSFSC

Entry address: See below Indirection vector: &21E

This routine is not detailed in either the BBC User or Disc system guides. There is no actual entry address for use with this routine which is known as the File system control entry routine, and is responsible for a host of filing system commands.

As there is no direct entry address a subroutine should be used to call the routine, this means that once it has completed its designated task it will return in an orderly fashion. To do this proceed as follows:

```
LDA#<value>:JSR fsc:.....rest of program
.fsc JMP(&21E)
```

In this way the return from the routine will be to the point following the JSR fsc instruction.

The value of A again determines the action to be carried out.

A=0 *OPT X = 1st parameter, Y=2nd parameter.

A=1 EOF# Y=filing handle.

A=2 *RUN, CHAIN X and Y point to filename in Ram.

A=3 As above.

A=4 As above.

A=5 *CAT

A=6 CLOSE spool, exec files.

A=7 Returns X=&4E , Y=&4F

A=8 *ENABLE

A=8 does not apply to the RaFS, but may be of some use to disc users.

APPENDIX V

Assembling Machine Code Directly into Sideways Ram

One particularly useful feature of Romlt is the ability to pass assembler code directly to sideways Ram if fitted. To do this simply issue *RAM and set P% to &8000 in your assembler program. When run this will then assemble your code directly in sideways Ram. To save the code so assembled simply use:

```
*BLOCK R DISC <filename> RETURN
```

or

```
*BLOCK R TAPE <filename> RETURN
```

Please note that doing this renders the RaFS incapable of operation, and all RaFS commands will return the Bad ROM error. It is also worth pointing out that issuing *RAM will wipe the contents of the Ram to which youi have assembled your code.

Whilst your assembler is running the screen display will not give the correct opcodes. Instead, the opcodes for the appropriate section of the Basic Rom will be shown. This has got nothing to do with RomIt and is simply a quirk of the way in which the assembler operates.

Also, if your assembler program contains any assembler specific errors, such as Byte, these will cause the computer to hang. Again, this is simply a quirk of the assembler and has got nothing to do with Romlt.

APPENDIX VI

ROMIT COMMAND SUMMARY

*ACCESS

Syntax	`*ACCESS <filename> <<L>>`
Function	Locks or unlocks the named file.

*APPEND

Syntax	`*APPEND`
Function	Enables the RaFS to hold more than one program having the same filename.

*BLOCK

Syntax	`*BLOCK <source> <destination> <page>`
or	`*BLOCK <source> <destination> <filename>`
Function	To copy the entire block of sideways Ram area either: 1) to or from a specified page in the BBC micro's user Ram area. 2) between the RaFS and a selected filing system.

*BUFFER

Syntax	`*BUFFER <<number>>`
Function	To turn Ram (sideways or user) into a buffer area. *ENABLE must be issued before this command.

*DELETE

Syntax	`*DELETE <filename>`
Function	To delete the named file.

***ENABLE**

Syntax	`*ENABLE`
Function	To enable the use of the commands *WIPE and *BUFFER

***FILE**

Syntax	`*FILE <source> <destination> <filename>`
Function	To transfer a file between the RaFS and another filing system,

***HELP**

Syntax	`*HELP`
or	`*HELP RFS`
or	`*HELP RAFS`
or	`*HELP UTILS`
Function	To provide information on commands and their syntax.

***INFO**

Syntax	`*INFO`
Function	To provide information on the RaFS and tiny files therein.

***MEND**

Syntax	`*MEND`
Function	Attempt to restore the RaFS following a Bad ROM error.

***OVERWRITE**

Syntax	`*OVERWRITE`

Function	To reset the RaFS to default whereby saving a file overwrites an old file with the same name.

***PAD**

Syntax	`*PAD`
Function	To pad the free area of the RaFS ready for Eprom blowing.

***RAM**

Syntax	`*RAM`
	`*RAM <socket> <<S>>`
	`*RAM+ <<S>>`
	`*RAM- <<S>>`

Function	To select the area of Ram in which RomIt has control.

***RBASIC**

Syntax	`*RBASIC <filename>`
Function	Renders a BASlC program in a form that may be *RUN

***RBOOT**

Syntax	`*RBOOT<<->>`
Function	To implement or disable a facility that allows a Rom to be auto-booted.

***RELOAD**

Syntax	`*RELOAD <filename> <address>`
Function	To change the load address of a file held in the RaFS

***RENAME**

Syntax	`*RENAME <oldname> <newname>`
Function	To rename a file held in the RaFS.

***RESTRICT**

Syntax	`*RESTRICT <filename> <<R>>`
Function	To prevent any file from being accessed by any command other than *RUN.

***REXEC**

Syntax	`*REXEC <filename> <address>`
Function	To change the run address of a file held in the RaFS

***RHELP**

Syntax	`*RHELP<<->>`
Function	To implement or disable a *HELP facility in your Rom.

***RTITLE**

Syntax	`*RTITLE<<->>`
Function	To add or remove service code to print a title message when BREAK is pressed.

***SNAP**

Syntax	`*SNAP<<R>>`
Function	To save the current screen under the root filename SNAP.

***SPACE**

Syntax	`*SPACE`

Function	To display free space left in the RaFS.

***UNWIPE**

Syntax	`*UNWIPE`
Function	Undo the action of *WIPE.

***WIPE**

Syntax	`*WIPE`
Function	To clear the RaFS of all files. *ENABLE must be issued immediately before this command.