BBC BASIC

REFERENCE
MANUAL

BBC BASIC

ACORN SCIENTIFIC

PART NO 0410, 006
ISSUE NO 1
JULY 1985

The product described in this manual and products for use with it, are subject to continuous developments and improvement. All information of a technical nature and particulars of the product and its use (including the information in this manual) are given by Acorn Computers in good faith.

In case of difficulty please contact your supplier. Deficiencies in software and documentation should be notified in writing, using the Acorn Scientific Fault Report Form to the following address:

Sales Department
Scientific Division
Acorn Computers Ltd
Fulbourn Road
Cherry Hinton
Cambridge
CB1 4JN

All maintenance and service on the product must be carried out by Acorn Computers' authorised agents. Acorn Computers can accept no liability whatsoever for any loss or damage caused by service or maintenance by unauthorised personnel. This manual is intended only to assist the reader in the use of the product, and therefore Acorn Computers shall not be liable for any loss or damage whatsoever arising from the use of any information or particulars in, or any error or omission in, this manual, or any incorrect use of the product.

Published by Acorn Computers Limited,
Fulbourn Road, Cherry Hinton, Cambridge CB1 4JN.

Within this publication the term BBC is used as an abbreviation for the British Broadcasting Corporation.

NOTE: A User Registration Card is supplied with the hardware. It is in your interest to complete and return the card. Please notify Acorn Scientific at the above address if this card is missing.

ISBN 0 907876 37 4 Acorn Scientific

Contents

1 Introduction

This document describes 32000 BBC BASIC in terms of its differences from previous versions of BBC BASIC. Unlike other *BBC Microcomputer System User Guides,* is does not form the basis of a tutorial, nor does it give detailed information about the use of all BASIC functions.

32000 BASIC refers to the implementation of BBC BASIC on Acorn Cambridge Series computers.

1.1 The BASIC language

BASIC is a general purpose programming language which has been implemented on most computers, and especially microcomputers. It is interactive, which means that programs may be typed into the computer, executed and corrected without the user having to use a text editor or a compiler. In addition, the values of a program's variables may be examined and altered to see the effect that this would have on the program.

BASIC is usually interpreted; instead of the BASIC program being converted into machine code instructions (as happens with the Acorn 32000 Pascal and C, for example), the BASIC interpreter examines each BASIC command as it is encountered and performs the required task.

BASIC programs are edited using numbered program lines; a simple line editor is incorporated into 32000 BASIC as well as a more sophisticated screen-based editor. In addition, a text editor may be used to create files which may be inserted using the *EXEC command.

1.2 32000 BBC BASIC

This version of BBC BASIC is an enhanced version of previous versions that operate with the BBC Microcomputer. The version described in the *BBC Microcomputer System User Guide,* BASIC I, will be treated as the standard in this document. Some readers may already be familiar with earlier versions of BBC BASIC, for example BASIC II, as described in the *Acorn Electron User Guide* and later versions of the *BBC Microcomputer System User Guide.* The BASIC II extensions to BASIC I have been incorporated into 32000 BASIC.

Extensions and improvements provided by 32000 BASIC include a built-in screen editor, faster execution (especially in real arithmetic), extensions to statements (e.g. the ON ... PROC statement), and better list formatting when using LISTO, and of course, considerably more memory.

The only feature which is present in the version of BBC BASIC provided for use with the 6502 second processor (6502 BBC BASIC) is the 6502 assembler. A 32000 assembler (Asm32) is provided with Acorn Cambridge Series computers. However, the BBC operating system routine addresses (e.g. OSWRCH at &FFEE) may still be used with the CALL statement to access the useful MOS routines. A 32000 assembler (Asm32) is provided with the 32000 system.

Unlike all other systems software provided with the equipment, 32000 BASIC does not rely on the Panos operating system. Instead it runs directly on the small firmware kernel, Pandora. This means that it operates in the traditional environment of * commands etc. familiar to BBC Microcomputer users. In this way, compatibility with existing BBC BASIC programs is facilitated. For details of how to install 32000 BASIC from the distribution disc, see the User Guide supplied with your system.

2 A description of 32000 BBC BASIC

2.1 Language reference

The language reference followed for 32000 BASIC is BASIC I as described in the *BBC Microcomputer System User Guide*, which is supplied with every BBC Microcomputer.

2.2 Restrictions and variations from BASIC I

This section describes features of BASIC I which are not present in 32000 BASIC, and the differences in the implementation of certain facilities.

2.2.1 The assembler

The 32000 system is supplied with the Asm32 assembler. This may be used to create object files which can be loaded and called by BBC BASIC programs. The 32000 Assembler is described in the *Acorn 32000 Assembler Reference Manual.*

2.2.2 Calling MOS routines

The CALL statement and USR functions may be used to call operating system routines. Where the address called is in the range &FFCE (OSFIND) to &FFF7 (OSCLI), BASIC emulates the appropriate function. As usual, the integer variables A%, X% and Y% are used to pass parameters to the calls. The difference occurs when X% and Y% contain the address of a parameter block, for example when calling OSWORD. The usual way of achieving this is:

```
1000 DIM parBlk 6
1010 X%=parBlk MOD &100
1020 Y%=parBlk DIV &100
1030 A%=1
1040 CALL &FFF1
```

This is incompatible with 32000 BASIC, which requires X% to hold the complete address of the parameter block, not just the lower eight bits of it. To write the above routine in a way which is compatible with all versions of BASIC use:

```
1000 DIM parBlk 6
1010 X%=parBlk
1020 Y%=parBlk DIV &100 : REM Not used by 32000 BASIC
1030 A%=1
1040 CALL &FFF1
```

As mentioned in the comment, line 1020 is not strictly necessary, but is included to maintain compatibilty with 8-bit versions of BBC BASIC.

CALL and USR with added parameter list

An example is given here to illustrate how 32000 BASIC implements these functions with an added parameter list. The statement

```
CALL locn% A,B$,C%
```

calls a location held at `locn%`. On entering this address, R1 (CPU register 1) holds the address of the parameter block, and R2 holds the number of parameters (in this case 3).

32000 BASIC automatically sets R1 and R2 before executing the code at `locn%`. Each entry in the parameter block consists of a doubleword specifying the address of the parameter followed by one byte specifying its type. The table below would be at the address held in R1.

doubleword	address of parameter 1
byte	type of parameter 1
doubleword	address of parameter 2
byte	type of parameter 2
:	:
:	:
doubleword	address of parameter n
byte	type of parameter n

Parameter types can be in the range 0 to 4. They have the following meanings:

type 0	parameter is a single byte, e.g.?F%
type 1	parameter is an integer, e.g. C%
type 2	parameter is a real, e.g. A

type 3	parameter is a string, e.g. B$
type 4	parameter is a byte string, e.g. $A%

2.2.3 LIST IF

An extra part has been added to the syntax of the LIST command. It may be followed by the keyword IF, which is in turn followed by a string of characters. Only lines in the program which contain this string will be listed. For example

```
>LIST IF DEF     (list all lines with keyword DEF)
>LIST 100, IF =  (list all lines from 100 with the = sign)
```

When pseudo-variables such as LOMEM and TIME appear in the IF part, only lines using them as functions (as opposed to statements) will be found. For example:

```
>LIST IF TIME=100
```

will list lines such as

```
1230 UNTIL TIME=100
```

where TIME is a function, but not

```
3250 TIME=100
```

where TIME= is being used as a statement. This is due to pseudo-variables having two tokens each, only one of which may be found using LIST IF.

2.2.4 LIST formatting

The LISTO command now causes the FOR ... NEXT and REPEAT ... UNTIL structures to be listed in a more obvious manner than in previous versions of BBC BASIC. That is, UNTILs line up vertically with their corresponding REPEATs, and NEXTs are listed directly underneath FORs (assuming that only one loop is opened and closed on any one line). Compare the 'program' below listed firstly in BASIC 1 using LISTO 7:

```
1000 REPEAT
1010   FOR
1020     REM
1030     NEXT
1040   UNTIL
1050 REM
```

and then in 32000 BASIC:

```
1000 REPEAT
1010   FOR
1020     REM
1030   NEXT
1040 UNTIL
1050 REM
```

Trailing spaces in lines (i.e. spaces immediately before the end of the line) are always stripped, and leading spaces (i.e. spaces immediately after the line number) are stripped if a non-zero LISTO is set. The implication of this is that blank lines may not be entered if a non-zero LISTO is set.

2.2.5 COLOR

32000 BASIC accepts the alternative spelling of 'COLOR' for the 'COLOUR' statement. The word is always listed as 'COLOUR' by English versions of 32000 BASIC.

2.2.6 Error handling

When an error occurs while executing a program line, the line is listed in inverse video up until the point where the error was detected, then in normal video until the end of the line. This makes it easier to pin-point the source of the error in long program lines. The line may be edited by typing the statement REPORT, which prints it wholly in normal video.

The error messages printed by 32000 BASIC are different (longer and clearer) than the messages printed by earlier versions of BBC BASIC. The error numbers are the same for equivalent errors.

2.2.7 Array formal parameters

It is no longer possible to use array elements as formal parameters to procedures and functions. For example, the definition

```
1000 DEF PROCfred(n,a(n))
```

would be acceptable under previous incarnations of BBC BASIC. However, with 32000 BASIC, the address of formal parameter a(n) would only be evaluated on the first invocation of the procedure, rather than every time it is called.

2.2.8 The value of PAGE

The pseudo-variable PAGE is no longer confined to a 256 byte boundary. When 32000 BASIC is first called, PAGE is set to just above the interpreter itself (address 47BD approximately). PAGE should not be set lower than this, otherwise the BASIC program being entered will overwrite the interpreter.

LOMEM and TOP have the same meanings as under other versions of BASIC, i.e. they hold the address of the start of variable space and end of the program respectively. The HIMEM

pseudo-variable holds the address of the start of BASIC's stack which grows down in memory as FOR, REPEAT and GOSUB statements are executed, and shrinks again when NEXT, UNTIL and RETURN statements are executed. HIMEM should not be set to above its initial value which is the lowest memory location not used by Pandora.

2.2.9 Precision of printing

As in BASIC II, it is possible to print numbers to 10 digits of precision; the limit under BASIC I was nine digits. For compatibility (i.e. so that by default only nine significant digits are printed), the value of the @% variable is set to &90A when BASIC is first called. To gain the extra digit it should be set to &A0A. When this is done, integers of magnitude up to 2**31-1 may be printed without 32000 BASIC resorting to 'E' notation.

2.3 Extensions to BASIC I

2.3.1 The EXT#= statement

In BASIC I, EXT# is a function returning the length of an open file; in BASIC 32000, EXT# may also be used on the left of an assignment statement in order to set the length of the file. This requires the presence of a suitable filing system, (DFS, ADFS, or NFS). The main use of this statement is to reduce the length of a file, for example:

```
EXT#out=&1000
CLOSE#out
```

will set the length of the file whose channel number is out to &1000, regardless of its previous extent. PTR# may still be used to increase the extent of a file.

2.3.2 The OPENUP function

In BASIC II, and subsequent versions, a function was added for opening files. In BASIC I, OPENOUT opened a file for output only, and OPENIN opened a file for update (i.e. reading and writing). The new function OPENUP performs the same function as OPENIN did (and has the same internal token value), and OPENIN now opens an existing file for input only.

Programs written using BASIC 1 will still work under 32000 BASIC, but will have OPENIN listed as OPENUP. Programs written under 32000 BASIC which use OPENIN will not work under BASIC 1.

The functions OPENIN, OPENOUT and OPENUP correspond directly to calls to the host operating system routine OSFIND with A=&40, &80 and &C0 respectively on entry.

2.3.3 The OSCLI statement

A new statement, OSCLI, was added to BASIC II. It takes a single string parameter and passes it to the operating system command line interpreter. It therefore acts in a very similar way to preceding a line by *, the difference

being that * commands may not contain BASIC variables. For example, if a$="FX" and b$="12,3" then the command *a$+b$ will pass the string "a$+b$" to the operating system, probably resulting in an error, whereas OSCLI a$+b$ will pass the string "FX12,3" to the operating system, which will in turn set the auto-repeat speed.

2.3.4 The '|' delimiter in VDU

If the character '|' appears in the place of a ';' or ',' in a VDU statement, the effect is the same as ,0,0,0,0,0,0,0,0,0, i.e. nine zero bytes are sent to the VDU drivers. This enables long VDU statements to be made shorter, for example VDU 19,1,0,0,0,0 can now be written as VDU 19,1|. The extra zero bytes are ignored by the VDU.

2.3.5 The ON ... PROC statement

An extension has been added to the ON ... GOSUB and ON ... GOTO statement. The new statement has the form:

```
ON choice% PROCleft(d%),PROCright(d%),PROCup(d%),PROCdown(d%)
```

As with ON ... GOTO/GOSUB the statement may be followed by an optional ELSE. The statement following the ELSE is executed if the ON expression is outside of the range 1 to n, where n is the number of PROCs listed.

2.3.6 'Unlimited' nesting of structures

The control structures GOSUB ... RETURN, REPEAT ... UNTIL, and FOR ... NEXT no longer have fixed limitations on the depth of nesting. The limit of nesting is imposed by the amount of memory available. Since this is usually at least 500 K bytes, there is no practical limit.

The lifting of the nesting limitation is due to the way in which 32000 BASIC organises its stack (the area of memory in which return addresses for loops are stored). 6502 BBC BASIC uses several stacks: one each for GOSUBs, REPEATs, FORs, and FN/PROCs. These are all combined into one structure in 32000 BASIC, which potentially can occupy all of the free memory. This means that the 'Too many' (FORs, REPEATs, GOSUBs) error messages all become 'out of memory' messages.

Because of this new structure, there are some limitations on the way in which the various constructs may be exited. This is reflected by the new error message 'Incorrect nesting of control structures'. An example of a technique which is viable on the version of BASIC which runs with the 6502 second processor, but not the 32000 version is:

```
FOR i=1 TO n:UNTIL TRUE:NEXT
```

which exits 'n' REPEAT loops. Such techniques are very bad programming practice and 32000 BASIC will not tolerate them.

2.3.7 The HELP command

If the user types HELP in immediate mode, 32000 BASIC will print the version number of the current interpreter and other useful information. Because HELP is a command and not a statement, it may not be used from within a program. HELP may not be abbreviated.

2.3.8 The QUIT command

Typing QUIT in immediate mode causes the BASIC interpreter to finish and the Pandora command line interpreter to be re-entered. Again, QUIT may not be used from within a BASIC program as it is a command, not a statement. There is no abbreviation of QUIT.

2.3.9 The EDIT command

A screen editor is provided with 32000 BASIC. This is described in the following chapter.

3 Preparing Programs

BBC BASIC contains a simple line editor, in common with most other dialects of the language. To enter a new line, the line is typed preceded by a line number. For example:

```
>1000 PRINT "Hello, world"
```

will enter a line 1000 with the statement PRINT "Hello, world" in it. Any other line 1000 which existed in the program will be overwritten. To delete a line, its line number is typed without any following text. To delete line 1000, for example, just type:

```
>1000
```

A block of lines may be deleted using the DELETE command:

```
>DELETE 1210,1560
```

will delete all lines between 1210 and 1560 inclusive.
When typing in lines, you may use the normal cursor editing facility, as described in the *BBC BASIC User Guide*. This enables whole or partial lines to be copied, edited, deleted etc.

3.1 The EDIT command

This section describes the facilities offered in the BASIC screen editor. This editor can only be used to edit BASIC programs; it does not cater for text files.

3.1.1 Getting started

If you wish to edit an existing BASIC file, type

```
>LOAD "filename"
>EDIT
```

and the file will be loaded into the editor and displayed on the screen.

If you wish to create a new BASIC program type:

```
>EDIT
```

The editor can be entered at any time from BASIC to edit the currently stored program.

3.1.2 The screen

On entering the editor, you will find the cursor at the centre of the screen, and a status line at the bottom.
The cursor can be moved by using the four cursor keys. The status line at the bottom gives you information on current typing mode, markers and errors. For example:

#I#	insert mode active
#O#	overtype mode active
#I# Marked	marked text involved

Insert mode means that characrers will be inserted into the BASIC text. Characters to the right of the cursor are displaced to make way for the new characters. In contrast, when in overtype mode, characters at the cursor position will be erased when new characters are typed.

3.1.3 Locating text

You can use the ordinary cursor keys in combination with either the [SHIFT] or [CTRL] keys to locate text efficiently.

Key	On its own	SHIFT	CTRL
[↑]	Up a line	Up a screen	Top of file
[↓]	Down a line	Down a screen	Bottom of file
[←]	left a character	Start of screen line	Start of BASIC line
[→]	Right a character	End of screen line	End of BASIC line

3.1.4 Manipulating text

Manipulation of text within the program is achieved by the use of function keys.

Key	On its own	SHIFT	SHIFT + CTRL
f0	Insert/Overtype	Jump to BASIC line	Delete BASIC line
f1	Set marker	Copy to buffer	
f2	Insert from buffer	Delete marked block	
f3	OS star commands	Return to BASIC	

f0 Insert/Overtype

Function key 0 switches between Insert and Overtype mode. The bottom status line of the screen shows the current typing mode. #I# means that the text typed is inserted into the document by shifting everything after the cursor to the right. #O# indicates that overtype mode is active and you can replace characters by overtyping them.

SHIFT + f0 Jump to line

Pressing SHIFT + f0 and supplying a line number will cause a jump to that BASIC line. You will be prompted for the line number again if an illegal line number has been given.

SHIFT + CTRL + f0 Delete a BASIC line

Move the cursor to the beginning of the BASIC line (i.e. beginning of the line number) and press SHIFT + CTRL + f0. The entire BASIC line including any wrap-over lines will be deleted.

f1 Set marker

This is necessary for block-text operations. The marker is set at the current cursor position; subsequent block-text operations are performed on the block of text defined by the marker and the subsequent cursor position, including the character at the cursor. BASIC line numbers are not included in the block, since moving or deleting a block will cause line numbers to be altered.

SHIFT + f1 Copy to buffer

This copies a block of markered text into a temporary buffer so that it can be recalled and inserted elsewhere later on. See under f1 for a definition of a block of markered text.

f2 Insert from buffer

This recalls the buffer contents and inserts it to the left of the cursor. This may be used to duplicate or move text.

SHIFT + f2 Delete block

This deletes the block of text between the marker and the cursor position, including the character at the cursor.

f3 OS star commands

When (f3) is pressed, the editor will respond with a * prompt. This gives access to the BBC OS command line interpreter, so that commands such as *cat and *net can be performed. This mode will persist, allowing repeated * commands, until a blank line is supplied.

SHIFT + f3 Exit the Editor

This returns to BASIC. To save a program in a file use the BASIC 'SAVE' command.

4 Errors and debugging

BBC BASIC reports errors by listing the program line with the error and printing a message describing the error underneath. The program line is listed in inverse video up to the point BASIC reached when the error occurred. The remainder of the line is listed normally.

When an error is detected during the execution of a program, the usual action is to report the error as described above and print the line number. However, errors may be 'trapped' using the BASIC ON ERROR statement. This has the form:

```
ON ERROR <statement>
```

or

```
ON ERROR OFF
```

After a statement of the first form, errors cause program execution to jump to the <statement>. In addition the BASIC interpreter terminates any GOSUB, REPEAT and FOR statements it has executed, and exits from any user-defined functions or procedures which are active.

The <statement> is usually a call to an error handling procedure or subroutine. For example:

```
1000 ON ERROR PROCerror
```

The body of PROCerror would take the appropriate action depending on the type of error. Two functions ERR and ERL return the latest error number and the line number on which the error occurred. Error numbers are given in the descriptions below, and also in the table in section 8.2. There is a class of errors called 'fatal' errors which may not be trapped. They cause program execution to cease, even if an ON ERROR is active. All fatal errors have an error number of zero.

The second form of the ON ERROR statement disables error trapping so that errors cause the program to stop.

4.1 Errors in alphabetical order

This section lists all the reported errors along with a brief explanation.

```
Argument of EXP too large (Error 24)
```

The function EXP raises the constant e (2.7182818...) to the power given in its argument. The largest number which can be represented by BBC BASIC is 1.7E38, thus the largest argument which EXP can take is approximately LN(1E38), or 88.02. Thus numbers greater than this will give error 24.

```
Argument of LOG or LN must be positive (Error 22)
```

The arguments to LOG and LN functions must be greater than zero.

```
Argument of SIN, COS, or TAN too large  (Error 23)
```

The argument to the SIN, COS and TAN functions should lie in the range -8388608 to +8388608 radians.

```
Argument of SQR cannot be negative  (Error 21)
```

Without resorting to imaginary notation, it is not possible to obtain the square root of negative numbers. BBC BASIC will generate this error if an attempt is made to evaluate a function such as SQR(-1). The error may also occur when taking the ASN or ACS of a number outside of the range -1 to +1.

```
Array already defined  (Error 10)
```

This error is caused by attempting to define an array with the same name and dimension as one which has already been declared. Note that the arrays a(10) and a$(10) are different (as is a%(10)) and may co-exist in the same program, but a(10,10) and a(10) may not both exist, even though they have a different number of parameters.

```
Array subscript out of range (Error 15)
```

This occurs when a subscript of an array is out of the range allowed by the array's DIM statement. For example, if an array were defined with the statement DIM A(100), subscripts of the array would have to lie in the range 0 to 100.

```
Bad program   (No error number)
```

This error cannot be trapped and has no corresponding error number. It is given when the interpreter detects that the BASIC program has become corrupt in some way, e.g. through careless use of the indirection operators, or by setting LOMEM to below PAGE. The only way around the problem is to type NEW and start again.

```
Bad use of RENUMBER or AUTO  (Error 0)
```

This is a fatal error, indicating that RENUMBER or AUTO functions have been used incorrectly.

```
Cannot divide by zero  (Error 18)
```

This occurs when an attempt is made to evaluate an expression such as (a + c)/c where c is equal to zero. The / operator might also be a DIV or MOD for the error to occur.

```
Escape       (Error 17)
```

This error occurs when the user presses the ESCAPE key (assuming the key is enabled). ESCAPE can be used to interrupt a program which has been RUN (or CHAINed), an immediate mode statement, or a program listing.

```
Function or procedure arguments do not match definition  (Error 31)
```

When a user-defined function or procedure is called, the actual parameters supplied must match in number and type the formal parameters in the DEF line. If they differ in number, error 31 will be given; if they differ in type, error 6 will be given.

```
Functions cannot be called in function or procedure definition  (Error 47)
```

This error is very rare. It only occurs when a call is made to a user-defined function in the formal parameter of another user-defined function or procedure. For this to happen, the formal parameter must be an array element. An example of a line which will cause the error, when the procedure is called, is as follows:

```
1000 DEF PROCrec(a%(FNfunc))
```

The call to FNfunc cause the error. It should be noted that using array elements as formal parameters in 32000 BASIC is not recommended anyway.

```
Illegal line number  (Error 41)
```

This is given when an attempt is made to GOTO, GOSUB or RESTORE to a line number which is not present in the program. Common causes are deleting the line accidentally, and using the wrong expression in statements such as (sline% + 100).

```
Illegal name for function or procedure (Error 30)
```

This error may be caused by attempting to call a user defined function or procedure with an illegal name, e.g.

```
1230 PROC%%%(1,2)
1560 LET S=FN@+1
```

Procedure and function names may consist of one or more letters, digits and the pound and at (@) characters. Note that the PROC or FN at the start acts as the initial letter of the name, so names such as FN123 and PROCOSCLI are allowed.

```
Illegal start of statement (Error 16)
```

This error indicates that a character has been encountered at the start of a line which is meaningless in that context. For example, if you mistype a * command by pressing @ instead of *, BASIC will produce this error.

```
Illegally terminated statement  (Error 16)
```

This is given when BASIC comes across a character at the end of a statement which should not be there. The newline character, : and ELSE are the only items allowed at the end of a statement. The statement: A=12" causes error 16 to be given.

```
Incorrect DIM statement (Error 10)
```

An attempt to dimension an array with a subscript of greater than 65534 will cause the error, as will putting an unacceptable character in a subscript, or missing out the right bracket e.g.

```
DIM A(!0
```

```
Incorrect range for ON (Error 40)
```

The expression after ON in an ON...GOTO/GOSUB/PROC should be in the range 1 to N , where N is the number of line numbers/PROCs after the

expression. For example, ON a PROCa, PROBb, PROCc should have a in the range 1 to 3. If this is not true, error 40 will be given.

```
Incorrect use of ON  (Error 39)
```

This error is given when BASIC cannot find a valid expression after an ON keyword, or the token ERROR. For example 'ON WIDTH' would cause the error to be given.

```
Incorrectly nested control structure  (Error 46)
```

This error occurs when control structures are nested in an incompatible way, i.e. when an attempt is made to execute a NEXT or UNTIL statement when subroutine, procedure or function has been called since the corresponding FOR or REPEAT. The program fragment below illustrates this:

```
1000 FOR i=1 TO 100
1010 GOSUB 2000
1020 ...
1030 ...
2000 REM Subroutine
2010 NEXT i
2020 RETURN
```

The error will occur at line 2010.

```
Invalid argument in function or procedure definition  (Error 48)
```

This occurs when a formal parameter in a user procedure or function definition is not a valid variable name. For example, an attempt to call the procedure defined by

```
1230 DEF PROCtest(a+b,c)
```

would cause the error to be generated. Formal parameters may be any type of variables, except array elements, but including indirection variables such as !&70.

```
Line too long to be added  (Error 0)
```

Before a new line is added to the program, it is 'tokenised', a process which usually makes the line smaller than before. However, in some extreme (and very unusual) cases, the line may actually grow after being tokenised, It is

possible that a line which filled the 238 character input buffer grows to more than 255 characters, which is the upper limit of BASIC lines when stored in the program. This error will be produced in the advent of such a rare event.

```
LOCAL can only occur in a function or procedure (Error 12)
```

The LOCAL statement may only be executed in a function or procedure definition. It is meaningless to attempt to make variables local in the 'main program', and if you attempt to do so, this error will be given.

```
Memory full (Error 0)
```

This is a fatal error (which therefore cannot be trapped) which is given when BASIC's stack is full. Causes are too many function or procedure calls without ENDPROCs or = <expression>s, too many nested GOSUBs, REPEATs and FORs, or very complex expressions when the program is very large.

```
Missing channel '#' (Error 45)
```

Statements such as BPUT which are followed by a file channel number required a # symbol immediately after the keyword. If one is not present, as in the statement BPUT2,128, this error will be given.

```
Missing closing bracket ')' (Error 27)
```

If an expression starts with an open bracket (, it should end with a matching closing bracket). Failure to do this will give error 27. It also applies to DIM statements and functions such as RIGHT$ where a) is required to terminate the function.

```
Missing closing quote '"'  (Error 9)
```

Strings in programs are started with " and terminated by another ". To embed a double quotation mark in the string, two of them are used: " ". This error will be produced when BASIC encounters the opening " but cannot find the matching closing " before the end of the line.

```
Missing comma ',' (Error 5)
```

Some functions such as LEFT$ required two arguments separated by a comma, e.g. a$ = LEFT$(b$,i). If the comma is omitted, e.g. a$ = LEFT$(b$), the error message given above will be produced by BASIC.

```
Missing FOR in TO statement (Error 36)
```

After the first expression of a FOR loop, i.e. the 1 in FOR I=1 the word TO must appear. If the expression is followed by any other character, the current error message will be produced.

```
Missing hex digit after '&'  (Error 28)
```

A hexadecimal number is & followed by 1 or more hexadecimal digits. A hexadecimal digit is 0-9 and A-F. If & is followed by anything other than one of these, error 28 is produced.

```
Missing numeric variable in FOR statement (Error 34)
```

Following a FOR statement must be a numeric variable, e.g. a, a%, !a etc. An attempt to use a string variable, or some other unrecognisable object will cause this error to be produced.

```
Mis-spelt keyword or missing equals sign "=" (Error 4)
```

In an assignment such as

```
LET A=123
```

or

```
A$="123ABC"
```

the variable must be followed immediately by an equals sign =. If BASIC can't find one, it will give the message indicated above.

```
No FOR corresponding to this NEXT  (Error 32)
```

This error is given when an attempt is made to execute a NEXT statement when there is no FOR loop currently active.

```
No GOSUB corresponding to this RETURN  (Error 38)
```

This error is given when an attempt is made to execute a RETURN statement when there is no GOSUB currently active. The most common cause of the error is when execution 'falls through' into a subroutine because the main program has not been terminated by a STOP or END statement.

```
No more DATA to read  (Error 42)
```

There should be the same number of items in DATA statements as read by READ statements. If an attempt is made to READ, say, 10 items, and only

9 are present as DATA, this error will be given. Possible causes are incorrect RESTORE statements and missing commas in DATA statements.

```
No REPEAT corresponding to this UNTIL  (Error 43)
```

When an UNTIL statement is encountered and no REPEAT loop is active, this error is produced. Its most likely cause is jumping into the body of a REPEAT loop using a GOTO statement.

```
Not enough room for this array  (Error 11)
```

This error occurs when an attempt is made to dimension an array when there is not enough memory left to hold that variable. Each element of an integer array occupies four bytes; real array elements take five bytes each, and strings require four bytes plus the number of characters in the string.

```
Not enough space for RENUMBER  (Error 0)
```

This occurs after a RENUMBER command. The command needs some workspace in order to operate (two bytes for each line in the program) . If the space is not available, the error is given. Note that RENUMBER uses the memory just above the program for this purpose, and therefore performs an automatic CLEAR.

```
Not in a function  (Error 7)
```

This occurs when an = <expression> statement is executed without a corresponding FN call. A common cause is omitting the END or STOP from a program and 'falling through' into a function definition.

```
Not in a procedure  (Error 7)
```

This error message appears if an ENDPROC statement is encountered when no user-defined procedure is active. It can be caused by omitting an END or STOP immediately before a procedure definition.

```
Number too large  (Error 20)
```

Integers in BBC BASIC must lie in the range -2**31 to +2**31-1. If a statement or function which requires an integer expression inside this range is given a value outside of the range, this error will be given. For example, `PRINT INT(2!34)` will cause error 20. Similarly, real numbers must lie in the range approximately +/-2E39 to +/-1.7E38. An attempt at calculating a value greater in magnitude than the permitted limit will cause the error.

```
Numbers and strings are incompatible  (Error 6)
```

In BBC BASIC, you may use a real number where an integer is required (the real is rounded to the nearest integer), and vice versa. However, you may not use a number where a string is required, or use a string where a number is required. An attempt to do so will cause this error to be generated. For example, the statement `PRINT STRING$("ABC",3)` would yield the error, as the string and numeric arguments of the STRING$ function have been put in the wrong order (it should be `STRING$(3,"ABC")`).

```
Operand of LOG and LN must be greater than zero  (Error 22)
```

It is mathematically impossible to obtain real logarithms of numbers less than or equal to zero, and attempting to obtain one in BBC BASIC will cause this error to be generated.

```
PROCs cannot be exited unless one is active  (Error 13)
```

This is very similar to the error above, but is caused by an ENDPROC statement being encountered when no user-defined procedure is active. The cause is also similar: omitting an END or STOP immediately before a procedure definition.

```
Stopped  (Error 0)
```

This fatal error is printed when a BASIC STOP statement is executed. Usually STOP is used to indicate that the program is ending because of some error condition, whereas END implies that the program terminated properly.

```
String too long (max 255)  (Error 19)
```

Strings in BBC BASIC may be between 0 and 255 characters long. An attempt to make a string with more than 255 characters will cause this error. Potential sources of error are the STRING$ function and the + operator.

```
There is no room to insert the line  (Error 0)
```

This only occurs after an attempt is made to enter a new program line. If there is not enough memory to accept the line, the error will be produced. It is very unlikely that the error will be encountered, not only because of the

very large amount of memory available to 32000 BASIC programmers, but also because a 'Memory full' error would usually be given first.

```
Undefined arrays (no DIM)  (Error 14)
```

Before an array element can be accessed, the array must exist, i.e. it must have been dimensioned using a DIM statement. A common cause of this error is the mis-spelling of a function name, e.g. A$ = LFT$(B$,I) will cause the error, unless by some coincidence LFT$ has been DIMed. If it has, a type error (number 6) will be caused.

```
Unknown function or procedure (Error 29)
```

When a call is made to a user defined function or procedure, and there exists no corresponding DEF line in the program, error 29 is given. The usual cause of the error is a mis-spelling in either the call to the function or procedure, or in its definition.

```
Unknown or mis-spelt variable (Error 26)
```

In many situations, BASIC requires an expression to be present. Examples are the COLOUR statement and the OSCLI statement. If the expression is omitted, or contains a variable whose value has not be defined, or includes binary operator such a / with an operand missing, then error 26 is produced.

4.2 Errors in numerical order

4.2.1 Fatal errors and command errors

Line too long to be added
Bad use of RENUMBER or AUTO
Memory full
Not enough space for RENUMBER
Stopped
There is no room to insert the line

4.2.2 Non-fatal errors

4 Mis-spelt keyword or missing equals sign '='
5 Missing comma ','
6 Numbers and strings are incompatible
7 Not in a function
7 Not in a procedure
9 Missing closing quote '"'
10 Incorrect DIM statement
10 Array already defined
11 Not enough room for this array
12 LOCAL can only occur in a function or procedure
14 Undefined arrays (no DIM)
15 Array subscript out of range
16 Illegal start of statement
16 Illegally terminated statement
17 Escape
18 Cannot divide by zero
19 String too long (max is 255)
20 Number too large
21 Argument of SQR cannot be negative
22 Argument of LOG or LN must be positive
23 Argument of SIN, COS or TAN too large
24 Argument of EXP too large
26 Unknown or mis-spelt variable
27 Missing closing bracket ')'
28 Missing hex digit after '&'
29 Unknown function or procedure
30 Illegal name for function or procedure
31 Function or procedure arguments do not match definition
32 No FOR corresponding to this NEXT
34 Missing numeric variable in FOR statement
36 Missing TO in FOR statement
38 No GOSUB corresponding to this RETURN
39 Incorrect use of ON
40 Incorrect range for ON
41 Illegal line number
42 No more DATA to read

43 No REPEAT corresponding to this UNTIL
45 Missing channel '#'
46 Incorrectly nested control structure
47 Functions cannot be called in function or procedure definition
48 Invalid argument in function or procedure definition

Index

Acorn Computers Limited
Scientific Division
Fulbourn Road
Cherry Hinton
Cambridge CB1 4JN
Telephone 0223 245200