fesbasic

COLLABORATORS

	TITLE : fesbasic		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		March 2, 2022	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

fesbasic

1.1 main

```
                 Welcome to FESBasic Guide File.
    Please Click on a subject below


                Copyright& Distribution

                Installation

                Using the Editor

                The BASIC dialect

                Registration & future
```

1.2 c_1

```
     THIS PROGRAM IS COPYRIGHT  (C)  1992/93 FALCON  ENTERPRISE SYSTEMS.
     IT MAY BE DISTRIBUTED PROVIDED THAT:-

   * ALL FILES ARE INCLUDED IN THE DISTRIBUTION. THIS INCLUDES THE
     EDITOR, THE INTERPRETER, THE HELP FILE, ALL DEMONSTRATION FILES AND
     THIS DOCUMENT.

   * NO FILES MAY BE MODIFIED IN ANY WAY. THE ONLY EXCEPTION TO THIS
     IS THE COMPLETE PACKAGE MAY BE "PACKED" USING A READILY OBTAINABLE
     ARCHIVER TO EASE DISTRIBUTION.

   * NO FEE OTHER THAN A REASONABLE COPYING FEE IS CHARGED.

   * THE PACKAGE MAY NOT BE INCLUDED ON A MAGAZINE COVERDISK WITHOUT
     MY PERMISSION.
```

1.3 c_2

1.4 i_1

1.5 i_2

FES Basic requires an amiga with at least 512K of memory and Workbench version 1.3. It is recommended that it is used with 1MB of memory and Workbench 2.0+.

It is recommended that "reqtools.library" is in your LIBS: directory. Without it the program will work but many of the requesters will not appear, this makes editor functions like GOTO line rather useless.

The Menu strip is missing if the program is run using a 1.3 Amiga, this means that you will have to use keys like AMIGA+L to load, rather than having the option to use the mouse. Also on a 1.3 based machine reqtools as mentioned above is neccary.

1.6 i_3

FES Basic can be run from the CLI or from workbench. Under Workbench 1.3 you will need to have reqtools.library in your LIBS: directory. There may be an 'install_library' script enclosed with this distribution to do this for you. Under WB 2.0 reqtools will be used if available, but if not then asl.library will be used instead.

1.7 e_1

1.8 e_2

Text can be entered at the cursor by typing. The Cursor can be moved using the cursor keys. Pressing SHIFT with LEFT or RIGHT moves to the end of the line. SHIFT + UP or DOWN moves up or down by one page. ALT + UP or DOWN moves to either end of the program.

Reserved words (like PRINT, LEN , AND) etc will be made into capital letters automatically for you.

Statements (words like PRINT STOP SCREEN , that always come at the start of a line) can often be abbreviated. The computer will fill in the abbreviation when you try to leave a line. DO NOT PUT A DOT TO INDICATE AN ABBREVIATION as you would on a certain computer. You abbreviate PRINT to P not to P.

At the second and subsequent uses of a variable name the name will be converted to the same case as when it was first used. So if you firsttype 'Count' then later type 'count' or 'COUNT' then they will be changed to 'Count'.

Multi line statements like FOR...NEXT loops or block IFs are automatically indented for you. You cannot change the amount of indentation on a line.

1.9 e_3

Programs can be saved in 3 different formats :- ASCII , FES & Fes_Prot.

ASCII (American Standard Code for Information Interchange) format allows the programs to programs to be easily loaded into word-processors or other versions of basic etc, but tends to be rather slower to load and save than FES format. Use AMIGA+A (or menu) to save a file in ASCII format.

FES format programs cannot be loaded into other programs. But they load faster and can be loaded directly by the stand-alone interpreter. This is the normal format for saving. Use AMIGA+S to save in this format. Use AMIGA+SHIFT+S to save with the same name as the file was loaded.

FES_Prot format is shorter than FES Format. It can ONLY be used with the stand-alone interpreter, a program saved in this format cannot be loaded back into the editor. Use this format when giving copies of your programs to other people when you don't want them to be able to see your listings. Make sure you always keep a copy of the program in a non FES_Prot format so you can make changes. This option is only available from the menus.

Programs can be loaded with AMIGA+L (or menu). The editor automatically detects which format the program is in when loading. Loading a FES_Prot file will crash the machine.

1.10 e_4

To start a program press AMIGA+X (or menu). There may be a burst of disk access while the interpreter is loaded. Then the screen should blank and the program should start executing.

If a program becomes faulty and needs to be stopped press CTRL+C.

Once a program has stopped you can examine the variable contents from the editor. Press AMIGA+V (or you guessed it the meun) and a list of all your int variables and their contents should appear. Press '2' for a list of longs or '3' for strings, any other key to return to the editor.

Pressing ESCape will take you into 'immediate' mode. The screen should switch to the programs screen and a ">" prompt appears. Type BASIC commands here and they will be executed immediately. This mode is usually used to see the contents of arrays or expressions.

Note there is a known bug in the immediate mode. If you type a literal string (eg PRINT "hello") then the string will often become corrupted. Sorry!!!

1.11 e_5

The current program can be cleared using AIMIA+Z (new in the menu)

The line containing the cursor can be cleared using CTRL+Y, the last deleted line can be pasted back using CTRL+U. These two commands can be used to swap the order of two lines,or to duplicate a line.

Any changes made to a line can be removed by pressing CTRL-Z.

The editor can be quitted with AMIGA+Q.

Simple analysis of the program can be performed using AMIGA+T. This checks for multiply defined labels/procedures/functions and mismatched control constructs (eg FOR without NEXT). This ffunction may be extended in later versions of the editor.

Blocks of text can be marked using F1 for start and F2 for end. Then press F3 to delete the block, F4 to copy a block and SHIFT+F4 to move the block.

1.12 e_6

```
      UP    Move up 1 line                  AMIGA A  Save Ascii
SHIFT UP    Move up 1 screen                AMIGA F  Find Text
ALT   UP    Move to top of file             AMIGA G  Goto Line...
    DOWN    Move down 1 line                AMIGA I  Iconify Window
SHIFT DN    Move down 1 screen              AMIGA L  Load File
ALT DOWN    Move to bottom of file          AMIGA N  Find Next
    LEFT    Move left 1 space               AMIGA Q  Quit Editor
SHIFT LT    Move to start of line           AMIGA S  Save File
   RIGHT    Move right 1 space              SH+AM S  Save with old name
SHFT RGT    Move to end of line             AMIGA T  Test program
                                            AMIGA V  View Variables
Esc         Goto Immediate mode             AMIGA X  eXecute Program
                                            AMIGA Z  New Program

CTRL+Y      Delete current line                  F1  Mark Start of Block
CTRL+U      Undelete line                        F2  Mark End of Block
CTRL+Z      Undo changes on line                 F3  Delete Block
                                                 F4  Copy block to Cursor
                                              SH+F4  Move block to Cursor
```

1.13 b_1

```
                General Points

                Expressions

                Functions

                Statments

                Special Notes
                Also note there is
                ON LINE HELP
                 provided within the editor itself.
```

1.14 b_2

```
                FESBasic is a modern version of the BASIC Language. Like most ↩
                   modern
BASICs there are no line numbers , so commands like GOTO are rarely used.

Without line numbers
                control constructs
                 like FOR...NEXT and DO...LOOP take
a much higher importance. As do
                Procedures
                 and user defined functions. FES
BASIC offers  rich choice of these to the programmer, allowing the creation
of well structured programs. In addition the editor automatically formats
```

your programs to give them that 'structured' indented look.

It must be pointed out at this stage that work is still continuing on the Language. See the "Register.doc" file for more information about getting hold of the latest version of the language.

Comments can be placed on any line. A comment is signalled by a ' and continues until the end of the line.

Programs MUST be written using the supplied editor. Programs written using another editor will not be run by the interpreter.

1.15 b_3

```
                An expression can be used almost anywhere a value is required in ↩
                   FESBasic.

An expression can consist of:

                Constants
                   eg    1    -4     "hello"

                Variables
                   eg    a   hello&  MyString$

                Operators
                   eg    +  -  *  /  =  <  <= AND OR

                Functions
                   eg    RND VAL STR$ INC AND
    Brackets          ( )
    UserFuncs   eg    FNxyz  FNabc$
```

1.16 b_31

There are three types of constant: int , long and string.

An int constant consists of an optional minus sign followed by up to 5 digits (0123456789). An int constant must be in the range -32768 to 32767.

A long constant is like an int but it is followed by a & (eg 65538&), unlike int constants a long can be any size between -4294967296 and 4294967295. If you type a number too big for an int then the editor will automatically add the & to make a long. Arithmetic involving longs is slower than that involving only ints.

A string constant is started by a " and is terminated by the same.

1.17 b_32

```
Like constants a variable can be any of three types, int long or string.

Int variables consist of a letter followed by 0 to 18 alphanumeric symbols,
the underscore _ and the 'at' symbol @ are considered to be letters for this
definition. So valid names are
    x
    abc
    Hello
    Hi_There
    @

Int variables store only whole numbers in the range -32768 to 32767.

Long variables have similar names to ints but they must end with a &. This
is included in the 18 alphanumeric symbols, again the total name cannot be
more than 19 characters. A long can store whole values in the range
-4294967296 to 4294967295.

A string variable name consists of the same restrictions as an int but
must end in a $
A string variable can store up to 100000 characters of text.
```

1.18 b_33

```
                Operators combine the value of two sub-expressions (called  ↩
                   operands)
There are three types of operators that work on numbers:
                Arithmatic

                relational
                  and
                boolean
                 . There are also similar operators for
                strings
                .
```

1.19 b_331

```
Arithmetic operators combine two numbers together to form a third. The
returned value will be a long unless both operands are ints when the return
value will be an int.

There are 4 Arithmetic Operators

    +       Performs Addition of the two operands
    -       Performs Subtraction
    *       Multiplication
    /       Integer Division ( fractions are ignored)

When an expression contains both multiplication/division and
addition/subtraction then the Multiply/Divides are done before the add/subs.
```

```
so  3+5*6  is 33 not 48
and 5+3/4  is 5  NOTE in Integer division 3/4 is zero.

To perform additions first use brackets

eg (3+5)*6
```

1.20 b_332

A relational operator compares the values of its two operands and returns a the int 'one' if some relationship is true, or zero if the relationship is false.

The available relational Operators are

<	Less Than
>	Greater Than
=	Equal
<>	Not Equal
<=	Less than or equal
>=	Greater than or equal

Relational operations are performed after arithmetic ones , so

```
      5+8<7*3     gives a value 1
and   b*a<>b*b    gives 1 unless 'a' and 'b' are equal.
```

1.21 b_333

A boolean operator tests its two operators for "truth" (true being defined as any value except zero) and gives a value based on the "truths" of both.

The three Boolean Operators are

AND	Gives '1' if and only if BOTH are true, else gives '0'
OR	Gives '1' if either or both are true , else 0
XOR	Gives '1' if one but not both are true, else 0

Boolean operators are evaluated after relational ones.

NOTE: Unlike some other BASICs FESBasic's operators are 'logical' not 'bitwise'. ie   5 AND 4 is 1 not 4.

1.22 b_334

There are less operators that can be applied to strings than to numbers.

For 'Arithmetic' operators the only one available is 'concatenation' +

This operator joins two strings together, ie "egg and "+"chips" gives
"egg and chips"

All the relational operators can be applied. For comparison a one string
is less than another if its first letter's ASCII code is less than the
second strings first letter. If the first letters are the same then the
comparison is made on the second letter, then the end of one string is
reached. (
Examples
)

Most basics do not allow operations on strings. FESBasic contains
an extension to the language these.

1.23 b_334a

So the following relationships are all true:-

```
    "chips" < "egg"         c comes before e
    "Egg" < "chips"         Capitals come before lower-case
    "2Chips" < "egg"        Numbers come before letters
    " Egg" < "chips"        Spaces come before letters.
    "chips"<>"CHIPS"        The case's are different so not equal
    "chips" = "chips"       Identical.
```

1.24 b_334b

FESBasic contains the ability to perform AND and OR operators on strings.
These are defined as:-

```
    string AND number   gives   string if number is 'true'
                                 ""    if number is 'false'

    string1 OR string2  gives   string1 if string1<>""
                                string2 if string1=""
```

Some examples:

```
OTHER BASIC:    c$=a$
                IF a$="" then c$=b$
FES BASIC:       c$=a$ OR b$

OTHER BASIC:    PRINT "There are ";n;" cat";
                IF n<>1 THEN PRINT "s";
                PRINT " on the roof"
FES BASIC:      PRINT "There are ";n;"s" AND n<>1;" on the roof"
```

1.25 b_4

```
                A function takes zero or more values (called parameters) and  ↩
                   produces a
single value as a result (known as the return value).

For a function with just one parameter the brackets are optional, so it is
acceptable to write ABS(x) or ABS x

If a function takes two or more parameters then the function name must be
followed by a '(' and each of the parameters must be separated by commas
so for example BTST(a,3).

For functions with no parameters there MUST NOT be brackets, eg TIMER

Functions are evaluated BEFORE operators so     ABS a+b
means (ABS a)+b   not    ABS(a+b).

For a complete list of the functions available in FESBasic refer to the
"FESBasic_HelpFile", where they are all listed alphabetically with descriptions.


                String Manipulation

                Bit Manipulation

                Arithmatic

                Input/Output

                Conversions
```

1.26 b_41

```
If we have a string ( for example h$ or "hello") we may need to split it up
into smaller pieces. The most function to do this is MID$. This takes as
parameters the string to be sliced, a position to start and a length. It
then returns a string made up of a part of the initial string.

so  MID$("Hello",2,3)   gives "ell"     e being the 2nd letter and continuing
                                        for 3 characters.

If the second number is left out then the string continues to the end of the
first string, ie  MID$("Hi There",4)  is "There"

FESBasic also allows the second number to be specified as a position, so you
can write MID$("Egg and Chips",5 TO 7)  to get "and".

Other available functions are LEFT$(a$,n) which gives the first 'n' characters
of a$, and RIGHT$(a$,n) which gives the last 'n' characters.

To convert the case of a string you can use UCASE$(a$) which converts all
small letters in string to capitals, and LCASE$ which is the opposite.

Finally the function TRIM$ removes any spaces from the beginning and end of a
string, so TRIM$("  hello  ") gives "hello"
```

1.27 b_42

```
These functions are only needed by more experienced programmers.

You may have noticed that the AND,OR and XOR operators in FESBasic
are 'logical' not bitwise. The bitwise operations are available but
as functions. So to mask out bits except 0 and 1 of 'a' use
                a=AND(a,3)

Also direct bit manipulation functions are available:
    BSET(a,n)   sets bit n of a
    BCLR(a,n)   clears bit n of a
    BCHG(a,n)   toggles bit n of a
    BTST(a,n)   tests bit n of a

Note these are functions not statements, so write a=BCLR(a,0)
instead of BCLR a,0

Also note that NOT is a logical function. To do a bitwise negation use
something like XOR(n,-1)
```

1.28 b_43

```
Various mathematical operations are available:-
    ABS n       the "absolute value" of n, ie n made positive
    MOD(a,b)    the remainder when 'a' is divided by 'b'
    -           negation
    INC n       n+1
    DEC n       n-1
    MULT(a,b)   a*b
    DIV(a,b)    a/b

The top three of these are standard BASIC, the last 4 are new to FES.

MULT and DIV are subtly different to the * and / operators. All work is
performed in 16 bits, with no error checking performed. Division by
zero gives int machine infinity, (ie 32767). Multiplication results are
modulo 65536, sign adjusted. If this does not mean much to you then you
will probably never need to use these functions.

INC and DEC can be useful in avoiding brackets sometimes, 3*INC a instead
of 3*(a+1). It is a matter of personal choice really. There is little
performance difference between either form.
```

1.29 b_44

```
A range of functions are available to perform input and output operations:-

    INKEY         Key Press as ASCII code
    INKEY$        Key Press as string
    SHIFTKEY      Which Qualifier keys are pressed
    INPUT$(a)     Read 'a' characters from keyboard
    INPUT$(#n,a)  Read 'a' characters from a file.
    MOUSEB        Which Mouse Buttons are pressed
    MOUSEX        Mouse X position
    MOUSEY        Mouse Y position
    STICKB(n)     Joystick 'n's Fire Button
    STICKX(n)     Joystick 'n's Left/Right Position
    STICKY(n)     Joystick 'n's Up/Down Position
```

1.30 b_45

To convert an int to a string of digits use STR$. To perform the
reverse operation, a string of digits to a long, use VAL.

To convert an int/long to an ASCII character use CHR$, the reverse can be
done with ASC.

Ints and Longs can be converted to 2 or 4 character strings respectively
by using MKI$ or MKL$. These are different from STR$ in that STR$
produces a string that makes sense to a human, eg STR$(84) is "84".
MKI$(84) is "_T". To convert back from the MK?$ format use CVI or CVL.

CVI and CVL can also be used to convert ints to longs or vice versa.
Although this is normally done automatically it is sometimes necessary to
do it explicitly. For example

```
  PRINT 3000*9000       gives an overflow error, as it multiplies two ints
                        to give an int, but the answer is too big for an
                        int to cope with, but
  PRINT 3000*CVL 9000   does work, as 9000 is now a long, so the result of
                        the multiplication is a long, which can cope with
                        the value 27000000. Note if, as in this case, you
                        are working with constants, you could use
  PRINT 3000*9000&      which is more succinct.
```

1.31 b_5

A 'statement' is the word at the beginning of each line which ↩
describes what
that line is going to do. So PRINT,GOTO,IF,LINE,CLS,COLOR are all statements.

FESBasic currently contains about 65 statements (There is some confusion about
lines containing a comment only, or about assignment lines, but this is
all besides the point!).

For a full list of all the statements please refer to the FESBasic_HelpFile.
It consists off all the statements with descriptions.

Control Constructs

Input/Output

Graphics

1.32 b_51

Probably the most important parts of computer programs are the statements which control the flow of the programs. Performing calculations is fine but if that is all that is required then the end-user is probably better off with a pocket calculator. The main advantage of a computer is its ability to perform operations many times, this looping constructs.

There are several Constructs that can be used in FES Basic to perform branching and looping.

Labels:...GOTO

IF...[THEN]

FOR...NEXT

REPEAT...UNTIL

WHILE...WEND

DO...LOOP

SELECT...CASE

PROC
There is also the
EXIT
command which can be used to make an early exit from a FOR...NEXT, DO...LOOP ,WHILE...WEND or REPEAT...UNTIL.

1.33 b_511

A Label is a method of marking a point in a program. In FESBasic this is done by placing an Alphanumeric word at the beginning of a line followed by a :.

The computer can be made to jump to a label at any point by the use of the GOTO command. The word GOTO is followed by the label of the point to jump to. There must not be a : after the label at the GOTO.

For example, the standard program that everyone writes:-

```
        Label:
        PRINT "FOZZ is ace!"
        GOTO Label
```

Care must be taken with GOTO's not to jump into or out of other control constructs. If you do the behaviour is not defined. There may be an error message, the program may work correctly on some computers but not on others, etc. Sod's law states that in this last case then the only computer in the cosmos that the program works on will be the programmers own when no-one else is in the room.

1.34 b_512

This is the main decision making construct. It can take two forms:-

```
    IF  THEN statement

or  IF
      statement
      statement
      :
    ENDIF
```

The first form is quicker to type, requires less lines, but can only cope with one statement. The second form can cope with any number of statements, even with more IF statements.

If we are using the longer IF construct then we can have several conditions, by using or .
This allows the use of multi-way decisions:-

1.35 b_512b

```
    IF a=1
        PRINT "MONO-"
    ELSEIF a=2
        PRINT "BI-"
    ELSE
        PRINT "MULTI-"
    ENDIF
```

There can be any number of ELSEIF clauses, and each block of statements can contain any number of statements, but in all cases only one block will get executed.

1.36 b_512a

What forms a "condition"?

It can in fact be any expression that gives a numeric result. If the result is not zero then it is regarded as "true",if zero then "false".

```
So we can say:-
    IF a<b THEN PRINT "a is less than b"
or  IF a=2 OR a=3 THEN PRINT "2 or 3"

{NOTE it is not possible to write
    IF a=2 OR 3 THEN PRINT "2 or 3"
 See section 2.3.3 for what this means! }
```

1.37 b_513

The FOR...NEXT construct is used to repeat a section of the program a given number of times.

A variable is used as a counter, It is set to a value at the start of the loop, then increased by a given value on each loop until it reaches another given value.

```
    ' eg count from 1 to 10
    FOR k=1 TO 10
      PRINT k
    NEXT
```

If it is required to count down then either use STEP with a negative value or FOR val=value DOWNTO value2.

1.38 b_514

REPEAT...UNTIL does exactly that. The block of program between the REPEAT and the UNTIL is executed repeatedly until a condition is satisfied.

```
    PRINT "Enter 0 to exit"
    REPEAT
      INPUT "Give me a number ";a
      PRINT a;" Squared is ";a*a
    UNTIL a=0
```

Note that when using REPEAT...UNTIL the block is executed at least once.

1.39 b_515

This is the mirror of REPEAT...UNTIL. Unlike REPEAT the condition is placed at the beginning of the block. The block is terminated with a WEND instruction.

This allows a block to be executed zero times if the condition fails the first time round.

```
    PRINT "Exponential Tables :"
    INPUT "Give me a number ";a
    WHILE a<10000
      PRINT a
      a=a*a
    WEND
```

1.40 b_516

This construct is a sort of combined . The loop repeats continuously until an command is executed.

```
    PRINT "Enter 0 to stop"
    DO
      INPUT "Give me a number ",a
      EXIT a=0
      PRINT a;" squared is ";a*a
    LOOP
    PRINT "Finished"
```

1.41 b_517

This allows a multi-way decision.

After the SELECT comes an expression. The computer then searches through the list of CASE's for a match. If one is found then the block after that case is executed. The word REMAINDER can be used to catch any cases not matched before.

```
    PRINT "Press:-"
    PRINT |"1.  Load"||"2. Save"||"3. Exit"
    INPUT ">",action

    SELECT action
    CASE 1
      INPUT "Load which File? ",FName$
      PROC LoadFile FName$
    CASE 2
      INPUT "Save which file? ",FName$
      PROC SaveFile FName$
    CASE 3
      STOP
    CASE REMAINDER
      PRINT "Invalid Option"
    END SELECT
```

1.42 b_518

Procedures are the recommended way of splitting a program up into manageable chunks. They allow blocks of program to be named, each block can have its own 'local' variables separate from all other blocks, and generally makes your programs far better structured.

Unfortunately the syntax for PROCedures in FES Basic is rather non-standard. I'm not sure how the syntax came about, but I'm stuck with it now. Its not to bad once you get used to it.

The definition of a procedure is started with

To run the procedure use the command PROC, followed by the name then the values to give each parameter. So to display 5 spaces then 3 stars in the above example use

```
        PROC stars 5,3
```

Either value could have been a variable or a complete expression.

See the example file "procs.fes" for a complete example.

1.43 b_518a

How to define a procedure. First comes the statement DEFPROC then the procedure name (alpha numeric just like a variable) then a space then any parameters it takes, all separated by commas.

```
eg      DEFPROC myproc a&,b,c$
```

Note that the variables used in a DEFPROC line are independent of any variables that may have the same name in the rest of the program.

After the DEFPROC comes the program lines that make up the procedure, and the whole thing is finished of with an ENDPROC.

```
        DEFPROC stars n_spaces,n_stars
           k       'the variable 'k' is independent of any other 'k's

          FOR k=1 to n_spaces
            PRINT " ";
          NEXT

          FOR k=1 to n
            PRINT "*";
          NEXT
          PRINT
        ENDPROC
```

1.44 b_518aa

FESBasic supports LOCAL variables. These are variables that only exist within a procedure. A local variable can have the same name as a normal (global) variable and yet be completely independant. Different procedures can have locals with the same name, but these are all mutualy independant.

```
so          k=3             'Outside PROC k=3
            PROC xyz
            PRINT k         'k is still 3, despite proc
            STOP

            DEFPROC xyz
            LOCAL k         'k in PROC is independant of other k's
            k=5             'k in proc is 5
            ENDPROC
```

1.45 b_519

```
The Exit command can be used to leave a loop part of the way through. It is
the only way to leave a DO...LOOP, and can be used to exit from any of the
other three loops.

The simplest form of the command is

    EXIT type_of_loop

eg  EXIT
    EXIT FOR
    EXIT REPEAT
    EXIT WHILE

Exit on its own is to exit from a DO...LOOP.

In addition a condition may be placed after the "type", in this case the loop
will only exit if the condition is true.
NOTE this looks weird when used on a WHILE loop, eg
    EXIT WHILE a<3      means exit the loop if a is less than 3.
```

1.46 b_52

```
PRINT           Displays the values of zero or more expressions on the screen.
                Expressions can be separated by semi-colons in which case they
                will be displayed side by side on the screen, by commas when
                they will be displayed in separate columns or by bars '|'
                when they will appear on separate lines.

INPUT           Displays a prompt then accepts information from the user and
                places it in a variable.

OPEN            Opens a channel to a (disk) file. Information can be sent to
                the channel using PRINT # or WRITE #. Information can be read
                using INPUT #,  INPUT$(#n) or READ #.
CLOSE           Closes a channel opened by OPEN.

PRINT #n,       Similar to PRINT but writes the output to channel 'n'

WRITE #n,       Writes values to channel 'n' in a computer readable format.
                This format is understood by READ #

INPUT #n,       Similar to INPUT but reads from a file not the keyboard.

READ #n,        Retrieves information from a file written to by WRITE #n.

BWRITE          Writes a block of memory to a channel opened with OPEN.
BREAD           Reads a block of memory from a channel.
```

1.47 b_53

```
SCREEN 1,w,h,n  Determines the screen mode that Basic will use. Note the
                slightly strange syntax with a 1, at the beginning!

COLOR f,b,m     Chooses the colour to draw in.

PALETTE c,r,g,b Changes the selection of colours available

LINE x,y TO x,y Draws straight lines on the screen.
BOX x,y TO x,y  Draws hollow or filled boxes.
CIRCLE x,y,r    Draws filled/hollow circles/ellipses.

AREA x,y        Fills an arbitrary shape with up to 20 vertices.
AREAFILL

FLOOD x,y,n     Fills a shape already drawn on screen.

PSET x,y        Sets a pixel on the screen.

GET             Copy a section of screen to a buffer
PUT             Copy a buffer onto the screen.

SCROLL          Move a part of the screen around.
```

1.48 b_7

Help is availible on all FES Basic Commands/Functions in the Editor.

Place the cursor on the command that you want to check the syntax of
and press Help.

Place the cursor on a blank line and press help for a list of all implemented
commands and functions.

1.49 b_6

```
                INTEGERS ONLY

                PROCS and FNs

                Logical operators

                PRINT USING

                Auto-run Programs
```

1.50 b_61

The freely distributable version of FESBasic is limited to working only with Integer Numbers (ie numbers with no decimal points). To obtain a version that supports real-numbers you need to register for the finished version.

This limitation means that for example 5/3 is, as far as the computer is concerned, equal to 1, as 1 is the largest number of times that 3 can go into 5.

1.51 b_62

FESBasic does NOT support the PRINT USING syntax found on many other versions of BASIC. Instead the C-Style function FORMAT$ is used.

FORMAT$ can be used in any context unlike USING which could only be used in conjunction with PRINT.

```
eg      'print a persons name and age
        PRINT FROMAT$("Name: %15s  Age %2d",name$,age)

        would give something like:-

        Name:     Simon Forey  Age 20
```

1.52 b_63

If you have written a program using FESBasic it is not neccary to load it into the editor every time you want to run it.

If you create a program with an Icon then the program can be run by double-clicking its icon. Just make sure that the program "basic" is on your disk in the same place as the "Tool Type" of the Icon says it is.

Running programs from the CLI is equally easy. Just make sure that "basic" is in your command path (eg put it in the 'c' directory of your disk) then type   basic file_name.fes

NOTE: For a program to be executable like this it must have been saved using SAVE or SAVE PROT from the editor. Programs saved using SAVE ASCII cannot be run without loading them back into the editor.

1.53 r_1

The version of FESBasic included with this document (V1.0) is the ↩
UNREGISTERED
freely distributable version. If you find this program useful then you may wish to obtain a copy of a later version of the program.

Later versions of the program are NOT freely distributable. They are only

obtainable from me. However If you write a program using a later version
you are permitted to distribute the interpreter file "basic" with your
program(s).

Registration

Future Plans

1.54 r_3

To obtain the latest version of this program write to me at this
Please include £10 Sterling and a letter stating that you want the latest
version of FESBasic. Make Cheques / Postal Orders payable to S.D.Forey.
PLEASE MAKE SURE YOU INCLUDE YOUR ADDRESS, and telephone number and/or
email address if possible.

I have tried to make the package as inexpensive as possible, whilst still
giving me a reasonable incentive to continue working on the program. Software
like this takes a lot of time and effort to write.

I can only accept cheques in POUNDS STERLING, drawn on a British Bank.
If you cannot obtain Sterling then I am prepared to accept the equivalent
to £15 in French Francs / German DM / Danish DM. (Saves me needing to change
money when I go on holiday!). Obviously I cannot reccomend sending Cash
through the post however since £10/£15 is not a massive sum the risks are
probably not too great...

1.55 r_4

By the time you read this I should have completed
version 1.1
.

By the summer of 1993 I hope to have
version 1.2
completed.

Whether I continue to work on the language after 1.2 depends very much on
how big a response I get.
Plans...

1.56 r_41

```
    * Floating point Maths: including   '!' variables and arrays.
                                        All the normal f/p functions.
                                        Support in FORMAT$()
```

* Static Variables in procedures/functions.

* Passing Arrays as parameters to Procedures and Functions.

* Calling O/S functions.

1.57 r_42

* Better Intuition Support: Windows, Multiple Screens, Menus

* Sprite and BOB commands.

* The ability to use Fonts and Styles of text in programs.

1.58 r_43

I would like to add:-

* Trace Function; possibly a debugger?

* Event Driven Code (commands like ON ERROR, ON MOUSE etc)

* A Compiler???

* Any more suggestions???

1.59 r_5

Please send any bug reports / suggestions / money to

```
                =SDF=
                14 Stretton Close
                Mickleover
                Derby
                DE3 5NW

                [England]
```