Laser Basic

SPECTRUM 48K
and
SPECTRUM +

LASER BASIC

by OASIS SOFTWARE

COPYRIGHT NOTICE

THIS MANUAL

Piracy has reached epidemic proportions and it is with regret that we are forced to reproduce this manual in a form which cannot be photocopied. Our apologies for any inconvenience this may cause to our genuine customers. A reward will be paid for information leading to the successful prosecution of parties infringing this Copyright Notice.

NOTE

This manual is essential for the use of Laser BASIC. For this reason we would warn customers to look after it very carefully, as separate manuals will not be issued under any circumstances whatsoever.

ENQUIRIES

If you have any queries on the use of Laser BASIC, please send them to us in a letter, ensuring you enclose the Enquiry Card printed on the last page of this manual. A new card will be returned to you with your reply. Please note that enquiries not accompanied by the card will not be answered.

CONTENTS

LASER BASIC TAPE MAP

TAPE 1 SIDE 1

i)	"LASER"	LASER BASIC PROGRAM. Load using LOAD "LASER" or LOAD ""
ii)	"SPRITE2A"	A file of OPTION2 saved sprites for use in Laser BASIC (see Appendix 2)
iii)	"SPRITE2B"	A file of OPTION2 saved sprites for use in Laser BASIC (see Appendix 3)

TAPE 1 SIDE 2

i)	"SPTGEN"	The Sprite Generator Program, used to create and edit your sprites for use in Laser BASIC. To load type RANDOMISE USR 0 LOAD""
ii)	"SPRITE1A"	A file of OPTION1 saved sprites for use with the Sprite Generator Program (see Appendix 2)
iii)	"SPRITE1B"	A file of OPTION1 saved sprites for use with the Sprite Generator Program (see Appendix 3)

TAPE 2 SIDE 1

i)	"DEMO"	LASER BASIC DEMO Load using LOAD "DEMO" or LOAD ""

TAPE 2 SIDE 2

i)	"GAME"	LASER BASIC GAME. Load using LOAD "GAME" or LOAD ""

LASER EXTENDED BASIC
by Kevin Hambleton

INTRODUCTION

Laser Extended BASIC is an extension to the existing BASIC interpreter in the ZX-Spectrum ROM. Although Sinclair BASIC is a powerful and flexible implementation of the time honoured language, it was necessary at its inception to make its features as general as possible. BASIC has numerous applications but the specific area of our interest is graphics and animation. Laser BASIC was designed to enhance the ease, and particularly the speed, with which complex animated graphics could be produced and over 100 commands and functions are included to do this. A technique akin to semi-compilation is used to further increase the execution speeds of the extra commands.

Those users already familiar with the Lightning series will recognize most of the command set, although for clarity a number of the command names have been changed. At this stage, Laser BASIC does not produce stand-alone programs (you need the extended interpreter to be resident) but a compiler is also being developed which will make your BASIC programs run faster and not require the interpreter to be resident. This will mean that you can market your programs commercially.

Laser BASIC can also be used by the commercial programmer, already familiar with the Lightning series, as a quick and simple to use development tool. The command sets are very similar and so Laser BASIC can be used to get a quick feel for an animated sequence before conversion to White Lightning or Machine Lightning. The interpreted nature of BASIC makes it absolutely ideal for this sort of exercise.

USING LASER BASIC WITH MICRODRIVES

Laser BASIC can be automatically saved to a microdrive cartridge using one of the options in the loader menu. Once Laser BASIC has loaded rewind the tape and select option 5. The various files that make up Laser BASIC will be loaded in and then saved to a microdrive cartridge in microdrive 1, one file at a time. You will need to stop and start the tape recorder as instructed.

Laser BASIC can then be loaded from a microdrive cartridge by typing;

```
LOAD *"M";1;"LASER"
```

Once Laser BASIC has been transferred to microdrive it can be loaded and then RUN in the same way as the tape based program.

There is, however, one unavoidable problem associated with the use of the microdrives. If a microdrive error occurs, such as "File not found" or any other error associated with the microdrive, then control will exit the Laser interpreter and re-enter the Sinclair interpreter. You will know if this has happened because any attempts to type an extended command will result in the flashing "?" symptomatic of a syntax error, or, if you try and RUN a program containing extended commands then "Nonsense in BASIC" will be issued. To return control to the extended interpreter just type a hash ("#") followed by ENTER. If the "#" itself generates a syntax error (the "?") then delete the "#" and type:

```
RANDOMIZE USR 58830             followed by ENTER.
```

We apologise for this unavoidable, annoying inconvenience.

In the extremely unlikely event that the tape based interpreter is exited, use: RANDOMIZE USR 58820.

GLOSSARY OF TERMS USED IN THIS MANUAL

SPRITES

A sprite is a software controllable graphics character. Laser BASIC allows up to 255 sprites to be defined, each with their own user selectable dimensions. The limit on the size and number of sprites available to the user is set by the amount of memory available.

Supplied with Laser BASIC is a program known as the Sprite Generator Program which is used to create software sprites. Once sprites have been created in this program you can save them to tape or microdrive cartridge using one of two 'OPTIONS'.

'OPTION1' sprites are used exclusively with the Sprite Generator Program whilst
'OPTION2' sprites are used exclusively with Laser BASIC.

Two sets of OPTION2 sprites have been provided on tape ready for you to load into Laser BASIC, these being "SPRITE2A" and "SPRITE2B" (see Appendices 2 and 3).

SCREEN WINDOWS

A screen window is a section of the screen defined by four variables COL, ROW, HGT and LEN. COL is in the range 0 to 31, ROW is in the range 0 to 23, HGT is in the range 1 to 24 and LEN is in the range 1 to 32. The unit for each of the above is the character. COL and ROW specify the column and row position on the screen of the top left hand corner of the window, with ROW 0 at the top of the screen and COL 0 on the far left hand side. HGT and LEN define the size of the window.

To see an example of a window on the screen type in the following line and hit ENTER.

```
        .ROW=5:.COL=6:.HGT=4:.LEN=3:.INVV
```

SPRITE WINDOWS

A sprite window is a section of a sprite defined by the variables SPN, SCL, SRW, HGT and LEN. SPN specifies the sprite, SCL and SRW specify the column and row within the sprite and HGT and LEN define the size of the window. If the window defined by these variables lies outside the sprite or overlaps its borders then the command will not execute but no error message will be issued.

SPRITE SPACE

Sprite space is the area of memory containing all previously defined sprites. The top of sprite space is 56575 decimal (DCFF HEX) and the lower end grows downward from this point. Cautious users may wish to find out how far down their sprites have grown by using one of the following routines:

```
        PRINT PEEK (62464)+ 256*PEEK (62465)
    OR
        LET X=?PEK(62464): PRINT X
```

RAMTOP

Note that it is very important that your sprites should never grow down over RAMTOP. To read the value for RAMTOP use:

```
        PRINT PEEK (23730)+ 256*PEEK (23731)
    OR
        LET X=?PEK(23730): PRINT X
```

Every time a sprite is defined it uses 9 x sprite HGT x sprite LEN + 5 bytes.

In most cases the user will not need to worry about sprites moving down over RAMTOP unless sprites are created during runtime using .ISPR or .SPRT. It is possible to check that a sprite created at runtime will fit by performing the above calculations. It is not recommended that newcomers define sprites from within Laser BASIC, instead sprites should be defined from within the sprite generator program and loaded into sprite space using one of the following procedures.

Method i)	Using one of the three options presented by the Laser BASIC loader menu.
Method ii)	Loading the sprites by hand using the following method. Once Laser BASIC has been loaded type; CLEAR (SPRITE.START ADDRESS) - 1 LOAD "(FILENAME)" CODE (SPRITE START ADDRESS) .POKE 62464,(SPRITE START ADDRESS)
Note:	The "FILENAME" is the name given to the sprite file when it is saved by the sprite generator program. The sprite start address is the lowest byte used by the sprite file and is also given by the sprite generator program.

PIXEL DATA

For those not aquainted with the workings of the Spectrum screen display, each character on the screen is produced as follows: each character cell is an array of 64 (8 by 8) pixels, represented by bits in memory. A pixel is a 'dot' which can be INK colour or PAPER colour. The bits which define a particular character or block of characters are referred to as pixel data.

ATTRIBUTE DATA

The colour of the INK and PAPER in each particular cell, together with the brightness and flashing attributes are controlled by a separate byte. The bytes which define the attributes of the block of characters are referred to as attribute data. Pixel data and attribute data are frequently treated as separate entities in Laser BASIC.

SCREEN OPERATIONS

These are operations which are carried out on a particular area of the screen. The area of the screen to be operated on is called the screen window and is defined above. The operations themselves include scrolls, inversions, reflections etc. and all commands in the category are postfixed by a 'V', e.g. .SR1V, .INVV, .MIRV etc. If the window overlaps the edge of the screen then the window will be automatically adjusted to lie "on-screen".

SCREEN/SPRITE OPERATIONS

These are operations between the screen and a sprite. The dimensions of the sprite are used as the dimensions of the screen window and COL and ROW are used to give the co-ordinates of the top left hand corner of the window, thus the operations are defined using the variables SPN, COL and ROW . If the window lies off the screen or the sprite overlaps the border of the screen then only part of the sprite will be 'PUT' or 'GOT'. Commands in this category are prefixed with 'PT' or 'GT', e.g. .GTBL, .PTXR, .PTND etc.

SPRITE OPERATIONS

These cover more or less the same operations as the screen window commands but this time a complete sprite is operated on in memory instead of a section of the screen. The only variable used is SPN and all commands in this category are postfixed with an 'M'.

SCREEN/SPRITE WINDOW OPERATIONS

These are operations between a screen window and a sprite window. As before, ROW, COL, HGT and LEN define the screen window, but this time SCL and SRW are used to define the position of the window within the sprite. SCL and SRW are measured in characters, SCL from the left and SRW from the top. If SRW+HGT is greater than the sprite height, or if SCL+LEN is greater than the sprite width, or if LEN+COL is greater than 32, or if HGT+ROW is greater than 24, then the commands will not execute. Commands in this group are prefixed with 'GW' or 'PW'.

SPRITE/SPRITE WINDOW OPERATIONS

These are operations between a whole sprite and a window within a second sprite. The two sprite numbers are held in SP1 (the sprite not containing the window) and SP2 (the sprite containing the window). The dimensions of the window are the dimensions of the sprite not containing the window and the position of the window in the sprite whose number is held in SP2 is specified by SCL and SRW . If the sprite whose number is held in SP1 overlaps the border of the sprite whose number is held in SP2 then no execution will take place. Commands in this group are prefixed with 'PM' or 'GM'.

SPRITE/SPRITE OPERATIONS

These are commands where a sprite is transformed and the result is left in a second sprite; there are only two commands in this group: .SPNM and .DSPM.

DUMMY SPRITE

A dummy sprite is a sprite which does not contain data for display. It may be used, for instance, to store a machine code subroutine, an array, or maybe used as part of a collision detection routine.

EDITING AND RUNNING PROGRAMS

The Spectrum editor has been extended so that the extra Laser BASIC commands will pass the syntax checking stage of program entry. All Laser Basic commands start with a full stop '.' and this is followed by 4 characters which must be typed in upper case. All Sinclair BASIC commands are entered in the normal way. Assignments such as .COL= or .ROW= must not contain any spaces between the variable name and the '=', e.g. .COL =4 will give a syntax error. All Laser BASIC commands will execute in immediate mode.

The extended functions are made up of a question mark '?' followed by 3 characters, e.g. ?COL, ?ROW, etc. These can only be used to return a value to a variable and cannot be used as part of an expression or PRINT string.

```
LET X=?COL: LET Y=?KBF: LET Z=?SET          would be legal
LET X = ?COL*3+Y                            would be illegal
PRINT ?COL                                  would be illegal
```

If Laser BASIC is being used with interface 1 connected then microdrive errors will cause control to exit from the extended interpreter. When this happens the editor will no longer accept the extended commands and syntax errors will occur each time an attempt is made to enter one of those commands. To re-enter the extended interpreter, delete the line you are trying to enter and simply type a hash '#' followed by ENTER. You should now be able to continue as before. If not, consult the earlier section - "Using Laser BASIC with microdrives".

To execute a BASIC program just type the keyword RUN followed by ENTER. There will be a pause while the additional commands are tokenised and then execution will begin at the first line of the program. If you wish to execute from a specific line of the program then you can do so using RUN followed by the line number; e.g. RUN 1000 will RUN your program from line 1000. There will also be a pause at the end of program execution, or on pressing BREAK, while the additional commands are de-tokenised. Note that programs can also be executed using GOTO, GOSUB or CONTINUE.

GETTING STARTED WITH YOUR FIRST LASER BASIC PROGRAM

At this stage it would be a good idea to load in the Laser BASIC Program and some sprites if you have not already done so. Most of the text covering the Laser BASIC commands include example listings which we hope, if typed in, will enlighten the newcomer to their use. Therefore, we strongly recommend you should slowly work through this manual typing in the examples as you go.

LOADING LASER BASIC

Firstly load in Laser BASIC by typing LOAD "." Stop tape when instructed to. Laser BASIC will be ready to use once the menu has appeared on the screen. You should now load in the "SPRITE2A" file.

LOADING OPTION 2 SPRITES

The OPTION 2 sprites, "SPRITE2A" are loaded by selecting option 3 of the menu and then pressing PLAY on your tape recorder. Once sprites are loaded execute Laser BASIC by selecting option 1 of the menu.

Remember, we are only using the example sprites since the example listings were written to use them. When you are writing your own programs you would load in your own sprites created in the Sprite Generator program. You are now ready to use Laser BASIC.

LASER BASIC

As stated earlier, all the standard Sinclair keywords are entered as normal.

eg. type

```
        10 REM THIS IS YOUR FIRST LASER BASIC PROGRAM.
```

This is typed in, in exactly the same way as when using the normal Sinclair BASIC, the keyword REM being produced by pressing the E key.
Now type in the next line which contains a Laser BASIC command.

```
        20.COL=1
```

Of course there is no keyword .COL= so you have to type . (symbol shift M) COL=1, a character at a time. Remember, there must be no spaces between the any of the characters including the "." or the "=". Do not type .COL =

The extended Laser BASIC Interpreter will accept this line when you hit enter. If you have typed it in wrongly an error will be displayed as with the normal Sinclair editor.

Now type the rest of the program.

```
        30.ROW=1
        40.HGT=20
        50.LEN=30
        60.INVV
```

You can now type RUN or GOTO 10 to execute the program.

We will now go into detail explaining the Laser BASIC commands. The best way to understand their operation is to type in the examples and run them.

TOOLKIT FACILITIES

Four new commands have been added to ease program development, these are .RNUM, .REMK, .TRON and .TROF.

.RNUM

This is a fairly standard renumbering utility and will renumber the program text and any line numbers following GOTO, GOSUB or RESTORE. It executes very slowly when compared to many of its contempories, but does not use any table space whatsoever. It will not, however, renumber computed GOTO's, i.e. GOTO 10*X or GOTO 100*5, since the value of expression is not computed until runtime. The syntax is:

```
        .RNUM FIRST LINE, NEW VALUE FOR FIRST LINE, INCREMENT
```

So typing .RNUM 102,1000,5 would leave lines with numbers less than 102 (and references to them) unaffected, change line 102 to have number 1000, and all subsequent lines to have numbers increasing in increments of 5, e.g.

```
  1Ø.COL=1:.ROW=1:.HGT=9:.LEN=9
  32 DEF FN A#(Y,X): PRINT AT Y,
X;".INVV":.RETN
 1Ø2 GO SUB 5Ø
 118 STOP
7ØØØ .INVV
752Ø.PROCFN A#(11,11)
8ØØØ RETURN
```

would become

```
   10.COL=1:.ROW=1:.HGT=9:.LEN=9
   32 DEF FN A#(Y,X): PRINT AT Y,
X;".INVV":.RETN
1000 GO SUB 50
1005 STOP
1010 .INVV
1015.PROCFN A#(11,11)
1020 RETURN
```

The default line increment is 10, so if you only specify the first line and the new value for the first line then all subsequent lines, and references to subsequent lines, will be renumbered in steps of 10.

So typing .RNUM 102,1000 would produce the following:

```
   10.COL=1:.ROW=1:.HGT=9:.LEN=9
   32 DEF FN A#(Y,X): PRINT AT Y,
X;".INVV":.RETN
1000 GO SUB 50
1010 STOP
1020 .INVV
1030.PROCFN A#(11,11)
1040 RETURN
```

The default 'new line number' for the renumbering to begin at is the 'old line number'.

So typing .RNUM 32 would produce the following:

```
   10.COL=1:.ROW=1:.HGT=9:.LEN=9
   32 DEF FN A#(Y,X): PRINT AT Y,
X;".INVV":.RETN
   42 GO SUB 42
   52 STOP
   62 .INVV
   72.PROCFN A#(11,11)
   82 RETURN
```

The default 'first line number' is the first line of the program
(excluding line 0).

So typing .RNUM with no following parameters would produce the following:

```
   10.COL=1:.ROW=1:.HGT=9:.LEN=9
   20 DEF FN A#(Y,X): PRINT AT Y,
X;".INVV":.RETN
   30 GO SUB 30
   40 STOP
   50 .INVV
   60.PROCFN A#(11,11)
   70 RETURN
```

Errors will be generated if:

a) The new value for the start line is lower than the old value for the previous line. This prevents lines becoming out of sequence.

b) Renumbering would cause line numbers to exceed 10000.

.REMK

This command is used simply to strip a program of all its REM statements and thus save memory when the program is getting large. It does not have any parameters and removes REM statements throughout the whole program.

If you now type the following:

```
10 REM THIS IS AN EXAMPLE
20 PRINT "HELLO" : REM THIS LINE PRINTS HELLO
30 REM THIS LINE DOES NOT PRINT GOODBYE : PRINT "GOODBYE"
```

Now type .REMK, noting what happens to lines 20 and 30; you are left with

```
20 PRINT "HELLO"
```

.TRON and .TROF

One of the difficulties with debugging BASIC programs is knowing just how the program actually flows. The program tracing facility (.TRON) allows you to single step through your BASIC program line by line. Each line is listed before it executes and the interpreter waits until you press a key to execute the line, or CAPS SHIFT BREAK to exit with the option to re-execute using CONTINUE. Executing .TRON will set the trace facility running at the next BASIC line executed and must be included in the program (it is disabled by RUN). The trace is automatically switched off when .TROF is executed or when the program has finished running.

THE GRAPHICS VARIABLES

The way in which parameters are passed to the graphics routines is slightly unusual and is aimed at speeding up program execution. Each graphics command uses a particular subset of the 10 graphics variables. Some commands require up to 5 parameters and in most cases more time would be spent evaluating the 5 expressions than executing the command itself. More often than not only one or two parameters are re-evaluated between successive executions of a command, and so the extended commands require only those parameters which need to be changed, to actually be changed. The other advantage of using dedicated variables is that the routines know exactly where to find the variables and do not need to search the BASIC variables to find the values.

There are actually 16 sets of the 10 variables and these can be individually selected using .SET=<exp> where <exp> is in the range of 0 to 15 and can be any BASIC expression, e.g.

```
        .SET=5*X+PEEK (58471)
```

The 10 variables are:

VARIABLE	GENERAL USE
ROW	Used to hold the row (Y co-ord) in characters, measured from the top of the screen (0-23). The top of the screen has ROW 0.
COL	Used to hold the column (X co-ord) in characters, measured from the left of the screen (0-31). The top left of the screen has COL 0.
HGT	Used to hold the height in characters of the current screen window (1-24).
LEN	Used to hold the length in characters of the current screen window (1-32).
SRW	Used to hold the row (Y co-ord) within a sprite measured from the top (0 to height-1) and in units of characters.
SCL	Used to hold the column (X co-ord) within a sprite measured from the left (0 to length-1) and in units of characters.
NPX	Used to hold the size and direction of vertical scrolls. Positive scrolls are upward and negative scrolls are downward. Units are pixels, not characters, and should be in the range +127 to -128.
SPN	Used to hold the sprite number for those commands which operate on only one sprite. SPN should be in the range 1 to 255.
SP1	Where operations involve a sprite and a sprite window, SP1 contains the number of the sprite which does not contain the window (1-255).
SP2	Where operations involve a sprite and a sprite window, SP2 contains the number of the sprite which does contain the window (1-255).

ASSIGNING VARIABLES

```
        .COL=       .ROW=       .LEN=       .HGT=       .SP1=       .SP2=
        .SPN=       .NPX=       .SCL=       .SRW=       .SET=
```

Using these functions, the graphics variables can be assigned values.

e.g. .COL=5:.ROW=2:.SPN=2

Now type in the following sample program, which XORs a sprite across the screen by changing the value in the variable COL by means of a FOR-NEXT loop.

```
 1 REM EXAMPLE 1
 5 REM USING ONE SET OF GRAPHICS VARIABLES
10.SET=0:.SPN=4:.ROW=0:.COL=-4
20 BORDER 1: BRIGHT 1: INK 6: PAPER 1: CLS :.ATOF
30 FOR I=-4 TO 32
40.PTXR:.COL=I+1:.PTXR
50 PAUSE 4
60 NEXT I
70 STOP
```

To execute type RUN or GOTO 1 then press ENTER.

Line 10	Graphics variable set 0 is selected, sprite number is set to 4, ROW is set to 0 (top row), and COL is set to -4
Line 20	The screens attributes are set, the screen is cleared and the attribute flag set to off.
Line 30	is a simple loop to move sprite 4 (the duck) from column positions -4 to 32.
Line 40	XORs out the old sprite, (see .PTXR) increments the COL variable and then XORs in the new sprite.
Line 50	pauses while the sprite is on screen.

We can now extend this simple routine, to move two sprites (in fact mirror images of the same sprite) in opposite directions again by changing the values in COL, except in this case two variable sets are used. (Note, if you intend to use only one variable set in a program the word .SET= does not have to be used).

```
 1 REM EXAMPLE 2
10 REM USING TWO SETS OF GRAPHICS VARIABLES
20 DEF FN A#(X,Y):.SET=X:.SPN=4:.ROW=Y
30.RETN
35 INK 6: PA?ER 0: BORDER 0: BRIGHT 1: CLS:.ATOF
40.PROCFN A#(0,0):.PROCFN A#(1,1)
50.COL=32:.SET=0:.COL=-4
60 FOR I=-4 TO 32
65.SET=0:.PTXR:.COL=I+1:.PTXR
70.SET=1:.MIRM:.PTXR:.COL=28-I:.PTXR:.MIRM
75 PAUSE 2
80 NEXT I
90 GO TO 40
```

Note the FN part of .PROCFN in line 40 is the keyword FN (symbol shift 2).

This program will move two ducks across each other from either side of the screen. To run this program just type RUN or GOTO 1 and then press ENTER.

Lines 20	and 30 define a procedure which initialises the necessary variables of the respective sets. The first time it is executed SET is 0 and ROW is 0 and the second time SET is 1 and ROW is also 1. This means that the two variable sets are identical except for the value of ROW.
Line 35	sets the attribute values before clearing the screen and switching off the attribute flow.
Line 40	executes the procedure A# twice.
Line 50	initialises the columns for sets 1 and 0 respectively.
Line 60	is a simple loop.
Line 65	moves the sprite from left to right using the variables from set 0.
Line 70	reflects the sprite and moves it from right to left using the variables from set 1. The sprite is re-reflected.
Line 75	is a pause while both sprites are on the screen.

INTERROGATING VARIABLES

As well as being able to assign values to the graphics variables, it is also necessary to be able to interrogate their current values. There are eleven functions provided for doing this.

```
?COL      ?ROW      ?LEN      ?HGT      ?SP1      ?SP
?SPN      ?NPX      ?SCL      ?SRW      ?SET
```

Using these functions, the current values can be assigned to a normal Sinclair variable, for example:

```
LET X=?COL: LET ROW=?ROW: LET ST=?SET
```

Notice that ROW=?ROW is allowed, but don't get confused between the normal variable ROW and the graphics variable ROW.

The above eleven functions cannot be used as part of a normal expression, i.e.:

```
LET X=?COL*3+?ROW/8 would not be legal
```

and they cannot be included as PRINT parameters either, i.e.:

```
PRINT X,Y,?ROW            would also be illegal.
```

However, the same results could be achieved by using:

```
LET COL=?COL: LET Y=?ROW: LET X=COL*3+Y/8        in the first case and
LET R=?ROW: PRINT X,Y,R                           in the second case
```

You can insert the following lines into the above EXAMPLE 2 to print out the two COL values as the sprites are moved.

```
75.SET=0: LET X0=?COL:.SET=1: LET X1=?COL
76 PRINT AT 10,16;" ";AT 11,16;" "
77 PRINT AT 10,12;"COL=";X0;AT 11,12;"COL=";X1
```

Line 75	lets the variable X0 equal the COL value of set 0 and X1 equal the value of set 1 COL.
Line 76	blanks out the old values on the screen.
Line 77	prints the new values.

THE EXTENDED GRAPHICS COMMANDS IN DETAIL

SPRITE UTILITIES

All the sprite utilities described in this section are available at run-time but should be used with caution. We strongly recommend that the unfamiliar user creates all his sprites in the Sprite Generator package.

.SPRT Used to set up a new sprite. Sprite space is extended upward so the start of sprites remains fixed but the end of sprites is increased by 9xHGTxLEN+5 bytes. If the sprite number has been previously allocated then the 'old sprite' is first deleted (with the bottom of sprite-space remaining fixed and the top of sprite space being reduced) before the new sprite is allocated. When a sprite is first set up it will probably contain garbage, and data will have to be "GOT" in it using a command such as .GTBL.

Parameter	Use
SPN	Number of the sprite to be set up. (1-255)
LEN	Length of the sprite in characters. (1-255)
HGT	Height of the sprite in characters. (1-255)

Note: Since this command extends sprite space upwards it will very seldom be used and is included for advanced applications only. Most uses require the next command .ISPR. Careless use of .SPRT may cause the system to crash.

.ISPR Used to set up a new sprite. Sprite space is extended downward so the top of sprites remains fixed but the start of sprites moves downward by 9xHGTxLEN+5 bytes. If the sprite has been previously allocated then an error is generated. If executing .ISPR causes the start of sprite space to move below RAMTOP the system will crash. Remember, as with .SPRT the sprite may initially contain garbage and data will have to be "GOT" into it using a command such as .GTBL.

Parameter	Use
SPN	Number of the sprite to be set up. (1-255)
LEN	Length of the sprite in characters. (1-255)
HGTzl	
	Height of the sprite in characters. (1-255)

.WSPR Used to delete a currently existing sprite. Sprite space is contracted so that the start of sprites remains unchanged but the end of sprites is reduced. Again this command is only used in advanced applications and sprites are normaly deleted using .DSPR. An error Q is generated if an attempt is made to delete a non-existant sprite.

e.g. to delete sprite 1 you would type

```
.SPN=1:.WSPR
```

Parameter	Use
SPN	The number of the sprite to be deleted. (1-255)

.DSPR Used to delete a currently existing sprite. Sprite space is contracted so that the top of sprites remains fixed but the start of sprites is moved up in memory. An error Q is generated if an attempt is made to delete a non-existant sprite. This command will normally be used to delete a sprite, and .WSPR will only be used in advanced applications.

e.g. to delete sprite 2 you would type

```
.SPN=2: .DSPR
```

Parameter	Use
SPN	The number of the sprite to be deleted. (1-255)

In example 3 below, memory is checked to see if enough memory is available for .ISPR.

```
 5 REM EXAMPLE 3
10 REM PROCEDURE TO CHECK IF SUFFICIENT SPACE IS AVAILABLE FOR .ISPR
20 REM X=LENGTH,Y=HEIGHT
30 DEF FN T#(X,Y)
40 LET SIZE = 9*X*Y+5
50 LET SPACE = PEEK (62464)-PEEK (23730)+256*(PEEK (23731)-PEEK (62465))
60 IF SIZE > SPACE THEN PRINT "NO ROOM AVAILABLE"
70 IF SIZE <= SPACE THEN PRINT "ROOM AVAILABLE"
80.RETN
```

The procedure is entered with X and Y holding length and height, e.g. type .PROCFN T#(5,5) for a 5 by 5 sprite. The size of the sprite is calculated and compared with the length of free space (SPST-RAMTOP) and an appropriate message generated.

```
 5 REM EXAMPLE 4
10 REM PROCEDURE TO CHECK IF SUFFICIENT SPACE IS
20 REM AVAILABLE AND SET UP SPRITE N IF POSSIBLE
30 DEF FN S#(N,X,Y)
40 LET SIZE = 9*X*Y+5
50 LET SPACE = PEEK (62464)-PEEK (23730)+256*(PEEK (23731)-PEEK (62465))
60 IF SIZE <= SPACE THEN .SPN=N:.LEN=X:.HGT=Y:.ISPR
70 IF SIZE > SPACE THEN PRINT "NO ROOM"
80.RETN
```

The above procedure in example 4 will check if sufficient room is available; if so, execute .ISPR and if not, generate a message, e.g. you could type
.PROCFN S#(1,2,3) to create sprite 1 of 2 by 3 characters.

```
 5 REM EXAMPLE 5
10 REM PROCEDURE TO DELETE A SPRITE IF IT EXISTS
20 REM AND PRINT THE NUMBER OF BYTES RECLAIMED
30 DEF FN D#(N)
40.SPN=N: LET START=?TST: LET X=?HGT: LET Y=?LEN
50.DSPR: PRINT 9*X*Y+5;" BYTES SAVED"
60.RETN
```

The procedure in example 5 will TEST sprite N (see ?TST) to find its dimensions and then delete it if it exists, displaying the number of bytes saved, e.g. you could delete sprite number 1 by typing .PROCFN D#(1).

HORIZONTAL SCREEN SCROLLS

The horizontal screen scrolls are by 1, 4 or 8 pixels, left or right, with or without wrap. The horizontal scroll commands are listed below.

Command	Action
.WL1V	Scroll left 1 pixel with wrap
.WR1V	Scroll right 1 pixel with wrap
.SL1V	Scroll left 1 pixel, no wrap
.SR1V	Scroll right 1 pixel, no wrap
.WL4V	Scroll left 4 pixels with wrap
.WR4V	Scroll right 4 pixels with wrap
.SL4V	Scroll left 4 pixels, no wrap
.SR4V	Scroll right 4 pixels, no wrap
.WL8V	Scroll left 8 pixels with wrap
.WR8V	Scroll right 8 pixels with wrap
.SL8V	Scroll left 8 pixels, no wrap
.SR8V	Scroll right 8 pixels, no wrap

Parameter	Use
COL	Column of left hand edge (0-31)
ROW	Row of the top edge (0-23)
LEN	Length of the window (1-32)
HGT	Height of the window (1-24)

The above commands operate on the screen, scrolling a window whose dimensions are held in the variables LEN and HGT and is positioned at the co-ordinates held in COL and ROW.

EXAMPLE 6 will demonstrate the 1, 4 and 8 pixel scrolls with wrap around.

Line 10	fills the screen with data.
Line 20	defines a window 20 by 10 at a position column 6 and row 2, by storing the values in LEN, HGT, ROW and COL.
Lines 30	to 80 scroll the window (300 times for each example).

```
 5 REM EXAMPLE 6
10 FOR N=1 TO 55: PRINT "LASER BASIC";: NEXT N
20.LEN=20:.HGT=10:.ROW=2:.COL=6
30 REM 1 PIXEL SCROLL
40 FOR N=1 TO 300:.WL1V: NEXT N
50 REM 4 PIXEL SCROLL
60 FOR N=1 TO 300:.WL4V: NEXT N
70 REM 8 PIXEL SCROLL
80 FOR N=1 TO 300:.WL8V: NEXT N
```

Replace the .WLIV scroll with .SL1V (no wrap) and see the result. You must remember that pixel data and attribute data are different 'chunks' of memory and that the above scrolling commands only scroll the pixel data.

EXAMPLE 7 will show the difference between the pixel and attribute data by scrolling data over attributes on the screen.

Line 10	makes sure the attribute flow switch is on.
Line 20	puts sprite 34 (a fly) at position 15,2 (see .PTBL).
Line 30	defines a window the length of the screen around the sprite.
Line 50	scrolls the screen pixel data 256 times to the right with wrap, leaving the attribute data behind.

```
 5 REM EXAMPLE 7
10.ATON
20.SPN=34:.ROW=2:.COL=15:.PTBL
30.LEN=32:.HGT=2:.COL=0:.ROW=2
40 PAUSE 50
50 FOR N=1 TO 256:.WR1V: NEXT N
```

You are not, of course, limited to 1, 4 or 8 pixel scrolls, by combining, say, three 1 pixel scrolls a 3 pixel scroll can be produced, e.g. .WR1V:.WR1V:.WR1V

EXAMPLE 8 demonstrates this. Remember that a 9 pixel scroll, for instance, produced by repeating .WR1V 9 times, might be more elegantly achieved by using .WR8V:.WR1V

Line 10	prints data on the screen.
Line 20	defines the dimensions of the window (length 32 and height 1).
Line 30	is the loop, that once completed, will reassemble the scrolled screen.
Line 40	is the loop that calculates the ROW and number of scrolls on that line.
Line 50	sets the ROW variable for the scroll window.
Line 60	scrolls the window I times.

```
5 REM EXAMPLE 8
10 FOR N=1 TO 20: PRINT AT N,0;N;"PIXEL SCROLL": NEXT N
20.LEN=32:.HGT=1:.COL=0
30 FOR N=1 TO 256
40 FOR I=1 TO 20
50 .ROW=I
60 FOR M=1 TO I:.WR1V: NEXT M
70 NEXT I
80 NEXT N
```

VERTICAL SCREEN SCROLLS

These work in a similar way to the horizontal scrolls, but in addition to setting up the window with the four window parameters COL, ROW, HGT and LEN, a further variable NPX is used to give the size and direction of the scroll in pixels. A positive value for NPX causes upward scrolling and a negative value causes downward scrolling.

e.g. .NPX=-1 is one pixel down.
.NPX=1 is one pixel up.

Command	Action
.WCRV	Vertical scroll with wrap
.SCRV	Vertical scroll, no wrap

Parameter	Use	
COL	Column of left hand edge	(0-31)
ROW	Row of top edge	(0-23)
LEN	Length of the window	(1-32)
HGT	Height of the window	(1-24)
NPX	Size and direction of scroll	(-128 to+127)

Note: All vertical scrolling of pixel data and/or attributes for screen or sprites, requires buffer space. The space required is calculated by multiplying NPX by LEN. This length must not exceed 256 bytes as the printer buffer (23296 to 23551) is used as a temporary store. A line to check this might be:

```
LET X=?NPX: LET Y=?LEN: IF ABS (X)*Y<256 THEN .WCRV
```

Vertical scrolls, as with horizontal scrolls. can be with or without wrap around.

EXAMPLE 9 is similar to EXAMPLE 6 except that this demonstrates a 1 pixel scroll up and a 1 pixel scroll down.

Line 40	produces 300 1 pixel scrolls up.
Line 60	produces 300 1 pixel scrolls down.

```
 5 REM EXAMPLE 9
10 FOR N=1 TO 55: PRINT " LASER BASIC ";: NEXT N
20.LEN=20:.HGT=10:.ROW=2:.COL=6
30 REM 1 PIXEL SCROLL UP
40.NPX=1: FOR N=1 TO 300:.WCRV: NEXT N
50 REM 1 PIXEL SCROLL DOWN
60.NPX=-1: FOR N=1 TO 300:.WCRV: NEXT N
```

In EXAMPLE 10 we scroll vertical columns 1 character wide by 1 pixel. The column is picked at random.

Line 10	prints a row of 'A's on the bottom of the screen.
Line 20	sets up the parameters of the screen window.
Line 30	picks a random column and stores it in the variable COL.
Line 40	scrolls the column up by 1 pixel without wrap.
Line 50	loops around.

```
 5 REM EXAMPLE 10
10 FOR N=0 TO 31: PRINT AT 21,N;"A": NEXT N
20.LEN=1:.HGT=22:.ROW=0:.NPX=1
30 LET X=INT (RND*33)-1:.COL=X
40.WCRV
50 GO TO 20
```

SCREEN ATTRIBUTE SCROLLS

Attribute scrolls are similar to the pixel scrolls but all attribute scrolls are by one character, and with wrap. The buffer size used is equal to LEN and is therefore always less than 33 bytes.

Command	Action	
.ATLV	Scroll attributes left	1 character with wrap
.ATRV	Scroll attributes right	1 character with wrap
.ATUV	Scroll attributes up	1 character with wrap
.ATDV	Scroll attributes down	1 character with wrap

Parameter	Use	
COL	Column of the left hand edge	(0-31)
ROW	Row of the top edge	(0-23)
HGT	Height of the window	(1-32)
LEN	Lenqth of the window	(1-24)

EXAMPLE 11 demonstrates the use of one of the attribute scrolls, .ATRV . In this example, vertical columns of attributes are placed on the screen. Using a series of nested FOR-NEXT loops the top row of attributes is scrolled by 1 character, the second row is scrolled by 2 characters etc. As this sequence is repeated patterns are formed.

```
 5 REM EXAMPLE 11
10 INK 7: PAPER 0: BRIGHT 1: CLS
20 FOR N =0 TO 703: PRINT CHR$(129): NEXT N
30.LEN=1:.HGT=24:.ROW=0: FOR N=0 TO 31:.COL=N: INK INT (RND*7)+1:.SETV:
NEXT N
40.LEN=32:.COL=0:.HGT=1
50 FOR I=0 TO 31
60 FOR Y=1 TO 22
70.ROW=Y-1
80 FOR X=1 TO Y
90.ATRV
100 NEXT X
110 NEXT Y
120 NEXT I
```

Line 10	sets up the attributes.
Line 20	fills the screen with CHR$ 129.
Line 30	fills the screen with random coloured columns (see .SETV).
Line 40	sets the parameters for .ATRV.
Line 50	I is the number of complete scroll operations to reset the columns.
Line 60	Y is the row.
Line 80	is the number of characters scrolled per row.
Line 90	executes the scroll.

The way in which attribute data is scrolled separately from pixel data is shown in EXAMPLE 11, which is similar to EXAMPLE 7. In this case it is the pixel data that is left.

```
 5 REM EXAMPLE 12
10.ATON
20.SPN=34:.ROW=2:.COL=15:.PTBL
30.LEN=32:.HGT=2:.COL=0:.ROW=2
40 PAUSE 50
50 FOR N=1 TO 32:.ATLV: NEXT N
```

Due to the limitations of the Spectrum, attributes can only be scrolled by 8 pixels or one character at a time. If you change line 50 in EXAMPLE 12 to:

```
50 FOR N=1 TO 32:.ATLV: .WL8V: NEXT N
```

both the pixel and attribute data will be scrolled.

HORIZONTAL SPRITE SCROLLS

The horizontal sprite scrolls are by 1, 4 or 8 pixels, left or right, with or without wrap. If the sprite does not exist an error is generated. A list of sprite scrolls are given below.

Command	Action
.WL1M	Scroll left 1 pixel with wrap
.WR1M	Scroll right 1 pixel with wrap
.SL1M	Scroll left 1 pixel, no wrap
.SR1M	Scroll right 1 pixel, no wrap
.WL4M	Scroll left 4 pixels with wrap
.WR4M	Scroll right 4 pixels with wrap
.SL4M	Scroll left 4 pixels, no wrap
.SR4M	Scroll right 4 pixels, no wrap
.WL8M	Scroll left 8 pixels with wrap
.WR8M	Scroll right 8 pixels with wrap
.SL8M	Scroll left 8 pixels, no wrap
.SR8M	Scroll right 8 pixels, no wrap

Parameter	Use
SPN	Number of the sprite to be scrolled (1-255)

In most of the previous examples, Laser BASIC words have ended in the letter "V". This implies "video". Operations ending in "V" effect only the screen, leaving all sprites in memory unaffected.

All the above scroll commands end in "M" which implies "memory", that is to say that if you execute one of these words the sprite in memory is altered. Remember that you will not see the change in the sprite until you have placed it back on the screen.

EXAMPLE 13 is similar to EXAMPLE 6, except a sprite is scrolled in memory and then placed on the screen 300 times. (Note that, as with screen scrolls, only the pixel data is scrolled). Remember that you will need the 'SPRITE2A' file of sprites loaded.

```
 5 REM EXAMPLE 13
10.ROW=10:.COL=14:.SPN=49
20 REM SCROLL SPRITE BY 1 PIXEL
30 FOR N=1 TO 300:.WL1M:.PTBL: NEXT N
40 REM SCROLL SPRITE BY 4 PIXEL
50 FOR N=1 TO 300:.WL4M:.PTBL: NEXT N
60 REM SCROLL SPRITE BY 8 PIXEL
70 FOR N=1 TO 300:.WL8M:.PTBL: NEXT N
```

Line 10	sets the ROW and COL positions for the sprite.
Line 30	scrolls the sprite by 1 pixel and then puts it on the screen 300 times.
Line 50	scrolls the sprite by 4 pixels and then puts it on the screen 300 times.
Line 70	scrolls the sprite by 8 pixels and then puts it on the screen 300 times (this happens so fast that the sprite becomes a blur).

VERTICAL SPRITE SCROLLS

These work in the same way as the vertical screen scrolls where the signed variable NPX is used to determine the size and direction of the scroll. If the sprite does not exist an error is generated.

Command	Action
.WCRM	Vertical scroll with wrap
.SCRM	Vertical scroll, no wrap

Parameter	Use
SPN	Number of the sprite to be scrolled (1-255)
NPX	Number of pixels to be scrolled (-128 to +127)

Note: As with the vertical screen scrolls the amount of buffer space used must not exceed 256 bytes. The length of the sprite multiplied by NPX gives the size of buffer required. The length of the sprite can be obtained by using the ?TST function.

EXAMPLE 14 demonstrates vertical scrolling of sprites, by scrolling and placing sprite 5 (the dancer) such that it fills the screen with scrolled sprites.

```
 5 REM EXAMPLE 14
10 BORDER 0:.SPN =5:.NPX =8
20 FOR Y=0 TO 20 STEP 4
30 FOR X=0 TO 31 STEP 2
40.COL=X:.ROW=Y:.PTBL
50.WCRM
60 NEXT X
70 NEXT Y
80 GO TO 80
```

Line 10	sets the parameters and the border colour.
Lines 20	and 30 set the X and Y co-ordinates for placing the sprite.
Line 40	puts sprite 5 at the X,Y co-ordinates.
Line 50	scrolls the sprite upwards by 8 pixels.

SPRITE ATTRIBUTE SCROLLS

As with the screen attribute scrolls there are 4 commands to scroll the attribute data in the 4 directions by 1 character with wrap.

Command	Action
.ATLM	Scroll attributes left 1 character with wrap
.ATRM	Scroll attributes right 1 character with wrap
.ATUM	Scroll attributes up 1 character with wrap
.ATDM	Scroll attributes down 1 character with wrap

Parameter	Use
SPN	Number of sprite to be scrolled (1-255)

GROUP 1 PUTS AND GETS

PUTs are operations that 'put' a sprite to the screen or another sprite, whilst GETs are the opposite, getting data from the screen or another sprite into a sprite. There are three groups of GETs and PUTs. The first, and the fastest, carry out operations between a full sprite and a previously positioned screen window. All group 1 GETs and PUTs are prefixed with 'GT' or 'PT'. This first group does not have separate commands to move pixel data and attributes but instead uses an attribute switch (see .ATON, .ATOF) to move pixel data with or without attributes. An error message is generated if the sprite does not exist.

Command	Action
.GTBL	Block move screen window into sprite
.GTOR	OR screen window into sprite
.GTXR	XOR screen window into sprite
.GTND	AND screen window into sprite
.PTBL	Block move sprite into screen window
.PTOR	OR sprite into screen window
.PTXR	XOR sprite into screen window
.PTND	AND sprite into screen window

Parameter	Use
COL	Left hand column of target screen position (0-31)
ROW	Top row of target screen position (0-23)
SPN	Number of sprite to be PUT or GOT (1-255)

Note: The dimensions of the screen window are the dimensions of the sprite, if COL+sprite length is greater than 32 or if ROW+sprite height is greater than 24 then the command will partially PUT the sprite.

If you now type:

```
.SPN=39:.ROW=1:.COL=1:.PTBL
```

you in fact place sprite 39 on the screen at ROW and COL position 1,1. This is the fastest and simplest way of putting a sprite on the screen.

Now type .ATOF (see .ATOF) and you have stopped the flow of attributes. So if you typed .ROW=5:.PTBL (remember there is no need to set SPN or COL as they remain the same), sprite 39 would appear on the screen again, but this time without its attributes.

Type .ATON:.PTBL and behold, it has re-appeared with its attributes.

.PTBL removes all data on the screen where the sprite appeared. You are provided with three other operations - .PTOR, .PTXR and .PTND - which logically OR, XOR and AND the sprite to the screen.

If you were to type:

```
.ATOF:.SPN=18:.ROW=10:.COL=10:.PTBL
```

you would have put sprite 18 (a tank) on the screen. Now if you type .SPN=39:.PTXR you would XOR sprite 39 over what was on the screen. Now type .PTXR and hit enter, and the sprite has removed itself, and reset the original data.

Example 15 XORs sprite 18 (a tank) through a field of mice non-destructively.

```
 5 REM EXAMPLE 15
10.ATOF:.SPN=6
20 FOR N=1 TO 50
30 LET X=INT (RND*32): LET Y=INT (RND*20):.COL=X:.ROW=Y:.PTBL
40 NEXT N
50.SPN=16:.ROW=10
60 FOR X=-4 TO 32
70.COL=X:.PTXR: PAUSE 4:.PTXR
80 NEXT X
90.ATON
```

Line 10	switches off the attribute flow and sets SPN to 6 (the mice).
Lines 20	to 40 fill the screen with 50 mice.
Line 50	sets the parameters for sprite 16.
Line 60	is a simple loop to work out the positions for COL.
Line 70	sprite 16 is XORed to the screen, a pause is executed while it is on the screen and then the same sprite is XORed at the same position, thus removing itself and resetting any data previously there.

If you type .SPN=18:.ROW=10:.COL=10:.PTBL sprite 18 appears on the screen as expected. Now see what happens if you type .COL=29:.PTBL (remember the sprite is 6 characters long).

What has happened is that as much of the sprite as possible has been placed on the screen. Now try .COL=-3:.PTBL

The GETs (words beginning in GT), unlike the PUTs, take data from the screen at a position stored in ROW and COL from a window whose dimensions are those of the sprite, storing the data in the sprite.

In EXAMPLE 16 a pixel is plotted in the bottom right hand corner of the screen, the window around this area is GOT into sprite 1 (destroying the vintage car) and the sprite is then placed on the screen. Another pixel is placed in the bottom right hand corner of the screen and sprite 1 is re-GOT and PUT etc.

```
  5 REM EXAMPLE 16
 10.SET=0:.SPN=1:.ROW=20:.COL=28
 20.SET=1:.SPN=1:.ROW=0:.COL=0
100 DEF FN A#()
110 LET PX=INT (RND*32): LET PY=INT (RND*16)
120 PLOT PX+223,PY
130.RETN
200 FOR Y=0 TO 20 STEP 2
220 FOR X=0 TO 32 STEP 4
240.PROCFN A#()
250.SET=0:.GTBL
260.SET=1:.ROW=Y:.COL=X:.PTBL
270 NEXT X
280 NEXT Y
```

Line 10	sets up SET 0 (the position of the bottom right hand corner of the screen.
Line 20	sets up SET 1 (the sprite).
Line 100	defines a procedure A#.
Lines 110	and 120 plot a pixel at a random position in the bottom left of the screen.
Lines	200 and 210 calculate the COL and ROW positions for putting sprite 1.
Line 240	calls the plotting procedure.
Line 250	GETs sprite 1.
Line 260	PUTs sprite 1.

GROUP 2 GETS AND PUTS

These commands allow operations between sprite windows and screen windows. Unlike the group 1 commands, there are separate commands to move pixel data and attributes, although the .ATON and .ATOF commands have the usual effect on their operation. Four new parameters are introduced to specify the column and row in the sprite and the height and length of the window within the sprite. If the window overlaps the boundaries of the sprite or screen the command will not execute and if the sprite does not exist an error is generated. All commands in this group are prefixed with 'GW' or 'PW'.

Command	Action
.GWBL	Block move screen window into sprite window
.GWOR	OR screen window into sprite window
.GWXR	XOR screen window into sprite window
.GWND	AND screen window into sprite window
.PWBL	Block move sprite window into screen window
.PWOR	OR sprite window into screen window
.PWXR	XOR sprite window into screen window
.PWND	AND sprite window into screen window
.GWAT	Block move screen window into sprite window
.PWAT	Block move sprite window into screen window

Parameter	Use
COL	Left hand column of target screen window (0-31)
ROW	Top row of target screen window (0-23)
SCL	Left hand column of sprite window (0 to sprite length -1)
SRW	Top row of sprite window (0 to sprite height -1)
HGT	Height of window (1-24)
LEN	Length of window (1-32)
SPN	Sprite number (1-255)

In EXAMPLE 17, 1 character by 1 character windows are taken at random from sprite 49 (the teddy bear) and placed on the screen.

```
  5 REM EXAMPLE 17
 10.SPN=49:.HGT=1:.LEN=1
200 FOR Y=C TO 20
220 FOR X=0 TO 32
230 LET GX=INT (RND*4)
240 LET GY=INT (RND*5)
250.COL=X:.ROW=Y:.SCL=GX:.SRW=GY
260.PWBL
270.PWAT
280 NEXT X
290 NEXT Y
```

Line 10	sets up SPN with the number of the sprite and sets HGT and LEN with the dimensions of the window.
Lines 200	and 210 calculate the COL and ROW positions for PUTting the 1 by 1 window.
Lines 230	and 240 calculate the random COL and ROW positions in the sprite for GETting the window.
Line 250	sets up these values.
Line 260	GETs the pixel data from the window and PUTs it on the screen.
Line 270	GETs the attribute data.

GROUP 3 GETS AND PUTS

This group, possibly the most useful in the whole set, comprises commands which support operations between a sprite and a window in a second sprite. All group 3 commands are prefixed with 'PM' or 'GM'. If the sprite window overlaps the boundaries of the sprite, the command will not execute, and if either of the sprites does not exist an error message Q is generated.

Command	Action
.GMBL	Block move sprite pixel data into sprite window
.GMOR	OR sprite pixel data into sprite window
.GMXR	XOR sprite pixel data into sprite window
.GMND	AND sprite pixel data into sprite window
.PMBL	Block move sprite window pixel data into sprite
.PMOR	OR sprite window pixel data into sprite
.PMXR	XOR sprite window pixel data into sprite
.PMND	AND sprite window pixel data into sprite
.GMAT	Block move sprite attribute data into sprite window
.PMAT	Block move sprite window attribute data into sprite

Parameter	Use
SP1	Number of the first sprite (1-255)
SP2	Number of the second sprite (containing the window) (1-255)
SCL	Left hand column of sprite window (1 - SPRITE LENGTH)
SRW	Top row of sprite window (1 - SPRITE HEIGHT)

.MOVE

Used to provide simple and effective animated or non-animated sprite movement. This command uses the exclusive OR (XOR) operation to provide non destructive sprite movement, so if your sprite starts on screen you will need to .PTXR the sprite onto the screen before you use .MOVE. If your sprite moves 'on screen' from a position 'off screen' then this will be catered for automatically. The exclusive OR (XOR) operation works in the same way as Sinclair's OVER 1 printing.

Parameters	Use	
COL	The COL of the sprite to be moved.	(0-31)
ROW	The ROW of the sprite to be moved.	(0-23)
HGT	The HGT in characters of the movement.	(-24-+24)
LEN	The LEN in characters of the movement.	(-32-+32)
SP1	The number of the sprite to be moved.	(1-255)
SP2	The number of the sprite after movement.	(1-255)

.MOVE XOR's out the previously PUT sprite SP1, that is on the screen at a position held in ROW and COL and places sprite SP2 on the screen at a position COL + LEN, HGT + ROW. ROW and COL are then incremented by the values of HGT and LEN, and SP1 and SP2 are left exchanged.

EXAMPLE 18 moves a sprite non destructively across data from the top left towards the bottom right. It stores in SP1 and SP2 the values of the OLD and NEW sprites (which in this case are both 6 the mouse).

Since the sprite is at the top left of the screen ROW and COL values are initially set to 0.

Now since we wish to move the sprite down and to the right by 1 character, HGT and LEN must be set to 1. .MOVE automatically increments ROW and COL by the values held in HGT and LEN. So in fact all you have to do is type .MOVE to move the sprite.

```
  5 REM EXAMPLE 18
 10 FOR N=1 TO 100
 20 LET X=INT (RND*32)
 30 LET Y=INT (RND*21)
 40 PRINT AT Y,X;"A"
 50 NEXT N
 60.ATOF: .COL=0:.ROW=0:.SPN=6:.PTXR
 70.HGT=1:.LEN=1
 80.SP1=6:.SP2=6
 90 FOR N=1 TO 25
100.MOVE
110 PAUSE 3
120 NEXT N
```

Lines 10 to 50 fill the screen with letter 'A's at random positions.
Line 60 puts sprite 6 (the mouse) in the top left hand corner of the screen.
Line 70 sets the increments in HGT and LEN to both be 1.
Line 80 sets the OLD and NEW sprites to 6.
Line 90 is a simple loop.
Line 100 executes .MOVE.

You can, of course, have negative values in HGT and LEN.

Using suitable values for SP1 and SP2, .MOVE can be used to animate as well as move sprites.

In example 19 negative values are placed in HGT and LEN to give movement in the opposite direction to those of EXAMPLE 18.

```
5 REM EXAMPLE 19
10 FOR N=1 TO 100
20 LET X=INT (RND*32)
30 LET Y=INT (RND*21)
40 PRINT AT Y,X;"A"
50 NEXT N
60.ATOF: .COL=0:.ROW=22:.SPN=6:.PTXR
70.HGT=-1:.LEN=-1
80.SP1=6:.SP2=6
90 FOR N=-1 TO 25
100.MOVE
110 PAUSE 3
120 NEXT N
```

In EXAMPLE 20 sprite 6 is moved around in a circle by constantly changing the values in the increments HGT and LEN.

```
5 REM EXAMPLE 20
10 FOR N=1 TO 150
20 LET X=INT (RND*32)
30 LET Y=INT (RND*22)
40 PRINT AT Y,X;"A"
50 NEXT N:.ATOF
60.ROW=10:.COL=4:.SPN=6:.PTXR
70.SP1=6:.SP2=6
80.HGT=0:.LEN=0
90 LET C=4: LET R=10
100 FOR N=1 TO 30
110.COL=C:.ROW=R
120 LET C=INT (14-10*COS (N/15*PI))
130 LET R=INT (10+10*SIN (N/15*PI))
140 LET OC=?COL: LET OR=?ROW
150.LEN=(OC-C)*-1 :.HGT=(OR-R)*-1
160.MOVE
170 NEXT N
180 GO TO 100
```

Lines 10 to 50 fill the screen with 'A's.
Line 60 puts sprite 6 on the screen.
Lines 70 to 90 set up the parameters.
Lines 100,120 and 130 calculate the COL and ROW position by calculating points on the edge of a circle.
Lines 140 and 150 calculate the offset to be stored in HGT and LEN.
Line 160 executes .MOVE

.ATON/.ATOF

The flow of attributes is controlled by two commands which determine whether or not attributes are to be moved with pixel data by the PUT's and GET's.

Command	Action
.ATON	Enable the flow of attributes between sprites and the screen.
.ATOF	Disable the flow of attributes between sprites and the screen.

Parameter	Use
None	None

TRANSFORMATIONS

To increase the utility of the package, four words have been included to invert (1's complement), spin, reflect and enlarge. The inversion and reflection routines work for screen and sprite data but the rotation and enlargement commands work only for sprites and a second target sprite is required to rotate or enlarge into.

.INVV The screen window defined by COL, ROW, HGT and LEN is inverted (1's complemented), in other words all pixels which were set 'on' become set 'off' and the effect is to exchange the INK and PAPER colours. If the window overlaps the screen boundaries then it will be 'clipped' to lie on screen.

Parameter	Use
COL	Column of the left hand edge of the window (0-31)
ROW	Row of the top edge of the window (0-23)
HGT	Height of the screen window (1-24)
LEN	Length of the screen window (1-32)

If you were to type .ROW=1:.COL=1:.HGT=5:.LEN=5:.INVV then the 5 by 5 window would have all its pixels inverted. Another example of .INVV is demonstrated in Example 21.

```
  5 REM EXAMPLE 21
 10 BORDER 0
 20 FOR N=1 TO 58: PRINT "LASER BASIC ";: NEXT N
 30 FOR N=1 TO 10
 40.HGT=22-(N*2)
 50.LEN=32-(N*2)
 60.ROW=N
 70.COL=N
 80.INVV
 90 PAUSE 10
100 NEXT N
110 GO TO 30
```

Example 21 produces a tunnel effect by changing the position and dimensions of the window.

Command	Action
.INVM	The same operation as .INVV but this time it is carried out on the whole sprite whose number is held in SPN.

Parameter	Use
SPN	Number of the sprite to be inverted (1-255)

To invert sprite 10 you simply type .SPN=10:.INVM

Command	Action
.MIRV	The pixel data in the screen window defined by the variables COL, ROW, HGT and LEN is reflected about a vertical line through its centre. If the window overlaps the screen boundaries the window will be 'clipped'.

Parameter	Use	
COL	Column of the left hand edge of the window	(0-31)
ROW	Row of the top edge of the window	(0-23)
HGT	Height of the screen window	(1-24)
LEN	Length of the screen window	(1-32)

Example 22 demonstrates .MIRV by mirroring the text on one half of the screen.

```
 5 REM EXAMPLE 22
10 LET A$="THIS IS AN EXAMPLE OF THE USE OF THE COMMAND .MIRV IN
LASER BASIC FOR THE SPECTRUM FROM OASIS SOFTWARE ."
20 LET Y=0: LET X=0
30 FOR S=1 TO LEN A$
40 LET C=CODE A$(S TO S)
50 IF C=32 THEN LET Y=Y+1: LET X=0: GO TO 80
60 PRINT AT Y,X;CHR$ (C);AT Y,X+16;CHR$ (C)
70 LET X=X+1
80 NEXT S
90.ROW=0:.COL=0:.LEN=16:.HGT=22:.MIRV
```

Line 10	sets up the string of text.
Lines 20	to 80 print out the string with one word per line.
Line 90	mirrors one half of the screen.

Command	Action
.MIRM	The pixel data in the sprite whose number is held in SPN is reflected about a vertical line through its centre. If the sprite does not exist an error message is generated.

Parameter	Use
SPN	The number of the sprite whose pixel data is to be reflected. (1-255)

Command	Action
.MARV	The attribute data in the screen window defined by the variables COL, ROW, HGT and LEN is reflected about a vertical line through its centre. If the window overlaps the screen boundaries the window will be 'clipped'.

Parameter	Use	
COL	Column of the left hand edge of the window	(0-31)
ROW	Row of the top edge of the window	(0-23)
HGT	Height of the screen window	(1-24)
LEN	Length of the screen window	(1-32)

Command	Action
.MARM	The attribute data in the sprite whose number is held in SPN is reflected about a vertical line through its centre. If the sprite does not exist an error message is generated.

Parameter	Use
SPN	The number of the sprite whose attribute data is to be reflected. (1-255)

Command	Action
.SPNM	Rotate 90 degrees clockwise sprite SP2 into sprite SP1. This command involves an operation between two sprites with transposed dimensions. If, for example, a sprite with dimensions 8 by 3 is to be spun into a second sprite, this second sprite must have dimensions 3 by 8. If the second sprite does not have the transposed dimensions of the first then the command will simply not execute. Pixel and attribute data are both rotated.

Parameter	Use	
SP1	Number of the sprite to be rotated	(1-255)
SP2	Number of the sprite to be rotated into	(1-255)

Note: Sprite SP1 should be cleared using .CLSM before data is rotated into it.

In EXAMPLE 23 sprite 5 (the dancer) is rotated into the cleared sprite 1 and then placed on the screen. Sprite 1 (which is now a 90 degree rotation of sprite 5) is now rotated back into the cleared sprite 5, thus producing a 180 degree rotated sprite 5. This process is continued.

```
 5 REM EXAMPLE 23
10.ATOF:.COL=15:.ROW=10:.SPN=5:.PTBL
20 FOR N=50 TO 1 STEP -1
30.SPN=1:.CLSM:.SP2=1:.SP1=5:.SPNM
40.ROW=10:.COL=15:.HGT=1:.LEN=-1:.SP1=5:.SP2=1
50.MOVE: PAUSE N
60.SPN=5:.CLSM:.SP2=5:.SP1=1:.SPNM
70.ROW=11:.COL=14:.HGT=-1:.LEN=1:.SP1=1:.SP2=5
80.MOVE: PAUSE N
90 NEXT N
```

Line 10	places sprite 5 on the screen.
Line 20	is a FOR-NEXT loop that controls the delay while the sprite is on the screen.
Line 30	clears sprite 1 and rotates sprite 5 into it.
Line 40	sets up parameters for a .MOVE operation.
Line 50	executes .MOVE with a pause.
Line 60	clears sprite 5 and rotates sprite 1 into sprite 5.
Line 70	sets up the parameters for a .MOVE operation.
Line 80	executes the .MOVE with a pause.

Command	Action
.DSPM	Enlarge sprite SP2 into sprite SP1. Sprite SP1 must have exactly double the dimensions of sprite SP2 or the command will not execute. Pixel and attribute data are enlarged together.

Parameter	Use	
SP1	Number of target sprite	(1-255)
SP2	Number of sprite to be enlarged	(1-255)

In EXAMPLE 24 sprite 51, which is a one character space invader, is enlarged into sprite 32, which is two characters high and wide.

```
 5 REM EXAMPLE 24
10.ROW=0:.COL=0:.SPN=51:.PTBL
20.SP1=32:.SP2=51:.DSPM
30.COL=2:.SPN=32:.PTBL
```

Line 10	puts sprite 51 on the screen.
Line 20	enlarges sprite 51 into sprite 32.
Line 30	puts sprite 32 on the screen.

MISCELLANEOUS WORDS

Command	Action
.SETV	Set the attributes to the permanent INK and PAPER colours in the window defined by HGT, LEN, COL and ROW.

Parameter	Use	
COL	Column of the left hand edge of the window	(0-31)
ROW	Row of the top edge of the window	(0-23)
HGT	Height of the screen window	(1-24)
LEN	Length of the screen window	(1-32)

in EXAMPLE 25 a 5 by 5 square is filled with random attributes at random screen positions.

```
 5 REM EXAMPLE 25
10.LEN=5:.HGT=5
20.COL=INT (RND*28)
30.ROW=INT (RND*18)
40 PAPER INT (RND*7)
45 BRIGHT INT (RND*1)
50.SETV
60 GO TO 20
```

Line 10	sets the LEN and HGT.
Lines 20	and 30 set the ROW and COL positions.
Line 40	sets a random PAPER colour.
Line 50	executes a .SETV
Line 60	loops around.

Command	Action
.SETM	Set the attributes to the permanent INK and PAPER colours in the sprite whose number is held in SPN.

Parameter	Use
SPN	Number of the sprite whose attributes are to be set. (1-255)

In EXAMPLE 26 sprite 2 is repeatedly put on the screen, and has its PAPER colour changed using .SETM

```
 5 REM EXAMPLE 26
10.SPN=2: INK 0: BRIGHT 1
20 FOR X=0 TO 32 STEP 4
30 FOR Y=0 TO 21 STEP 2
40.COL=X:.ROW=Y
50 PAPER INT (RND*6)+1
60.SETM
70.PTBL
80 NEXT Y
90 NEXT X
```

Line 10	sets the parameters.
Lines 20	and 30 are the loops for putting the sprite on the screen.
Line 50	picks the paper colour.
Line 60	executes .SETM
Line 70	puts the sprite on the screen.

Command	Action
.CLSV	Clear the screen window defined by variables HGT, LEN, COL and ROW and set the attributes to the permanent INK and PAPER colours.

Parameter	Use	
COL	Column of left hand edge of the window	(0-31)
ROW	Row of the top edge of the window	(0-23)
HGT	Height of the screen window	(1-24)
LEN	Length of the screen window	(1-32)

Command	Action
.CLSM	Clear the pixel data of the sprite whose number is held in SPN. Attribute data is unaffected.

Parameter	Use
SPN	Number of sprite whose pixel data is to be cleared. (1-255)

Command	Action
.ADJM	This command is used to adjust the values in the variables COL, ROW, HGT, LEN, SCL, SRW, SPN so that a particular sprite can be 'partially PUT or GOT' to or from the screen using the group 2 PUTs or GETs. The value in the PUT variables COL, ROW, HGT, LEN, SCL and SRW may all be changed by the execution of this command. Before execution SCL and SRW must be zero, HGT and LEN are ignored and the HGT and LEN of the sprite whose number is held in SPN are used.

Parameter	Use	
SPN	Sprite to be PUT or GOT	(1-255)
COL	Target column	(0-31)
ROW	Target row	(0-23)
SCL	Set to 0 before execution	
SRW	Set to 0 before execution	

Command	Action
.ADJV	Essentially the same idea as ADJM but this time the screen window defined by HGT, LEN, COL and ROW is adjusted to be 'on-screen'.

Parameter	Use	
COL	Target column	(0-31)
ROW	Target row	(0-23)
HGT	Height of window	(1-24)
LEN	Length of window	(1-32)

ASSIGNMENTS

There are 12 assignments in all:

.COL=	.ROW=	.HGT=	.LEN=	.SP1=	.SP2=
.SPN=	.SRW=	.SCL=	.NPX=	.SET=	

The commands assign the expressions (which can be any legal BASIC expression, but none of the extended functions) to the graphics variable. See the section on graphics variables.

THE EXTENDED FUNCTIONS IN DETAIL

There are 16 functions in all:

?COL	?ROW	?HGT	?LEN
?SPN	?SP1	?SP2	?SET
?SCL	?SRW	?NPX	?KBF
?SCV	?SCM	?TST	?PEK

They are all used in the form:

```
LET Var = ?FUN
```

where Var is any normal BASIC variable (including arrays) and ?FUN is any of the above 16 functions.

Note that these functions cannot be used as part of normal BASIC expressions, nor on the left of the above 12 graphics variable assignments, nor as part of a print statement.

e.g.

Function	Description
?COL	Returns the current value for COL
?ROW	Returns the current value for ROW
?HGT	Returns the current value for HGT
?LEN	Returns the current value for LEN
?SPN	Returns the current value for SPN
?SP1	Returns the current value for SP1
?SP2	Returns the current value for SP2
?SCL	Returns the current value for SCL
?SRW	Returns the current value for SRW
?SET	Returns the current value for SET
?NPX	Returns the current value for NPX
?KBF	This function is provided for the quick detection of multiple key presses. All it does in fact is test the specified key and return a non-zero result if the specified key was pressed or a zero if it was not. The variables COL and ROW are used to specify the row and half column of the particular key. For a full description of the columns and rows of the Spectrum keyboard, see page 160 of the Spectrum manual. Below is a summary.

(Spectrum+ keyboard users must note that they should not include the various extra shift keys in the COL and ROW values.)

Row	Keys
1	CAPS SHIFT to V
2	A to G
3	Q to T
4	1 to 5
5	0 to G
6	P to Y
7	ENTER to H
8	SPACE to B

Columns are organised from 1 to 5 and counted from the outside in. This is the order above.

?KBF will return a true value (non zero) whenever a key whose ROW and COL values are stored in the said variables is pressed. Hence if we wished to write a routine to print 'Hello' every time the 'A' key was pressed we would need to first work out its ROW and COL values, these being 2 and 1 respectively, then use a routine such as:

```
10.ROW=2:.COL=1: LET K=?KBF: IF K<>0 THEN PRINT "HELLO";
20 GO TO 10
```

The ability of ?KBF to see if a key is pressed or not means that you can in effect scan all the keys on the keyboard to discover which keys are being pressed at an instance.

EXAMPLE 27, (although rather long-winded), scans all the top row, (keys 1 to 0) and calculates the total value of the keys that are being pressed.

```
  5 REM EXAMPLE 27
 10 LET T=0
 20.ROW=4
 30.COL=1: LET K=?KBF: IF K<>0 THEN LET T=T+1
 40.COL=2: LET K=?KBF: IF K<>0 THEN LET T=T+2
 50.COL=3: LET K=?KBF: IF K<>0 THEN LET T=T+3
 60.COL=4: LET K=?KBF: IF K<>0 THEN LET T=T+4
 70.COL=5: LET K=?KBF: IF K<>0 THEN LET T=T+5
 80.ROW=5
 90.COL=1: LET K=?KBF: IF K<>0 THEN LET T=T+0
100.COL=2: LET K=?KBF: IF K<>0 THEN LET T=T+9
110.COL=3: LET K=?KBF: IF K<>0 THEN LET T=T+8
120.COL=4: LET K=?KBF: IF K<>0 THEN LET T=T+7
130.COL=5: LET K=?KBF: IF K<>0 THEN LET T=T+6
140 PRINT AT 10,10;" ";AT 10,10;T
150 GO TO 10
```

?SCV The character cell of the screen position defined by COL and ROW is scanned for pixel data. If data is found, a non-zero result is returned, otherwise a zero result is returned. If COL or ROW are off-screen a random result is returned.

This is probably one of the most useful functions as it can be used for sprite crash detection. The character square ahead of a moving sprite can be checked to see if it contains data, and if there is then a crash can be reported.

In EXAMPLE 28 you are asked to input the values for ROW and COL and the character square at this position is scanned for data.

```
 5 REM EXAMPLE 28
10 INPUT ;"COL= ";C: IF C<0 OR C>31 THEN GO TO 10
20 INPUT ;"ROW= ";R: IF R<0 OR R>24 THEN GO TO 20
30.ROW=R:.COL=C
40 LET T=?SCV
50 IF T=0 THEN PRINT "NO DATA THERE ": GO TO 10
60 PRINT "THERE IS DATA THERE": GO TO 10
```

Lines 10 and 20 take the values of ROW and COL and check they lie on the screen.
Line 30 sets ROW and COL.
Line 40 scans the character square.
Lines 50 and 60 report the result.

Example 29 uses ?SCV to check for objects in the path of a line of sprites, these objects, the letter 'A', are then avoided by scanning for possible clear paths.

```
  5 REM EXAMPLE 29
 10.SPN=51
 20 FOR N=1 TO 100
 30 LET X=INT (RND*32)
 40 LET Y=INT (RND*22)
 50 PRINT AT Y,X;"A"
 60 NEXT N
 70 LET R=10: LET C=0
 80 LET C=C+1: GO SUB 160: IF G=1 THEN GO TO 80
 90 LET C=C-1
100 LET R=R+1: GO SUB 160: IF G=1 THEN GO TO 80
110 LET R=R-1
120 LET R=R-1: GO SUB 160: IF G=1 THEN GO TO 80
130 LET R=R+1
140 LET C=C-1: GO SUB 160: IF G=1 THEN GO TO 80
150 PRINT AT 0,0;"TRAPPED": STOP
160 LET G=0:.ROW=R:.COL=C: LET S=?SCV
170 IF C=32 THEN PRINT AT 0,0;"SUCCESS": STOP
180 IF S=0 THEN LET G=1:.PTBL
190 RETURN
```

Lines 20 to 60 fill the screen with randomly positioned "A"'s .
Lines 80 to 140 calculate the movement of the sprite.
Lines 160 to 190 are the scanning routines that also report if the sprite has reached the other side.

?SCM The sprite whose number is held in SPN is scanned for pixel data. If data is found, a non-zero result is returned, otherwise a zero result is returned. If SPN does not exist an error is generated.

?TST The sprite whose number is held in SPN is searched for. If it is not found, an error Q is generated, otherwise the address of the sprite in memory is returned and HGT and LEN are set to contain the dimensions of the sprite.

NOTE: Sprites are stored in memory in the following format:

First byte	=	sprite number
Second byte	=	least significant byte of the address of the next sprite
Third byte	=	most significant byte of the address of the next sprite
Fourth byte	=	sprite length
Fifth byte	=	sprite height
8xHGTxLEN bytes	=	pixel data
HGTxLEN bytes	=	attribute data

This means that each sprite actually occupies 9*HGTxLEN+5 bytes.

In EXAMPLE 30 sprites 1 to 50 are interrogated using ?TST and their addresses in memory and dimensions are printed.

```
 5 REM EXAMPLE 30
10 FOR N=1 TO 50
20.SPN=N
30 LET AD=?TST
40 LET H=?HGT
50 LET L=?LEN
60 PRINT AD;" SPRITE ";N;" ";H;" HGT ";L;" LEN"
70 NEXT N
```

Line 10 is the loop for the sprites.
Line 30 lets AD = the address in memory.
Lines 40 and 50 read the HGT and LEN values set by ?TST.
Line 60 prints the data on the screen.

?PEK<exp> This is a 16 bit version of PEEK and unlike the other functions is followed by an expression which represents the address from which the function should read.

LET X = ?PEK(62464) is equivalent to LET X=PEEK (62464)+256*PEEK (62465)

LET X = ?PEK(X+3*Y) is equivalent to LET X=PEEK (2+3*Y)+256*PEEK (2+3*Y+1)

ADDITIONAL COMMANDS

.POKE X,Y

This is a 16 bit version of the normal BASIC POKE. It places the least significant byte of Y into X and the most significant byte of Y into X+1. Y must be in the range 0 to 65535 and X must be in the range 0 to 65534.

.POKE X,Y is equivalent to POKE X,INT (Y/256): POKE X+1, Y-256*(INT (Y/256))

PROCEDURES

One of the most powerful features of structured programming languages such as PASCAL and some advanced BASICs, e.g. BBC BASIC, is the facility to utilise procedures. So what is a procedure?

Quite often a particular piece of code or sequence of instructions is executed at a number of different places within a program. For this reason most languages have some facility for subroutines. The Z80 has a CALL instruction, BASIC has GOSUB, and other languages usually have an equivalent. As often as not the user needs to pass particular values to a subroutine and he also needs to know that the subroutine will not corrupt variables used in other parts of the program. This involves local variables and is a feature not supported by the standard Sinclair BASIC.

Local and Global Variables

When a procedure is defined, the definition contains a list of parameters which can be string or numeric but not array variables, e.g.

```
10 DEF FN A#(X,Y,Z,A$,B$)
```

(Remembering DEF FN is the keyword SYMBOL SHIFT KEY 1).

The variables X, Y, Z, A$ and B$ are now local variables (as far as the parameter is concerned). This means that values they are assigned within the procedure will have no effect on any other variables called X, Y, Z, A$ and B$ that have been used in other parts of the program. This is a very powerful feature as will be seen.

Any other variables encountered in the procedure, however, will have the global value. This means that they will have the value that they would have had anywhere else in the program and if they are assigned values within the procedure, will return with this new value.

Invoking Procedures

To execute the procedure above you could use:

```
.PROCFN A#(3,2,K,"HELLO",B$)
```

(where FN is the keyword SYMBOL SHIFT 2).

This would make the local variables in the definition take on the values of the parameters in the above invocation, i.e.

```
X=3, Y=2, Z=K, A$="HELLO" and B$=B$
```

A procedure must always end with the word .RETN which is similar to the keyword RETURN in subroutines. The procedure will execute up until the first .RETN and then control returns to the next instruction after the invocation. It is worth noting that procedure execution, like user defined function execution, is speeded up if the definition is put closer to the beginning of the program.

If BASIC comes across a procedure definition it will not execute any of the definition and control will jump to the next command after the corresponding .RETN.

Up to 52 procedures (single letter names or single letters followed by $) can be defined and to distinguish procedures from user defined functions the letter of the procedure name is followed by a #.

```
e.g     DEF FN A#()
        DEF FN C$#()
        DEF FN C#()        etc.
```

NESTING PROCEDURES

A procedure definition can contain a procedure invocation, but it is important to remember that if this is done then the values for the local variables in the latest invocation become the values for both invocations.

This section will be new to most users, and probably rather confusing. So here are a few examples.

If, for example, you wished to write a procedure to print "HELLO" on the screen it would look like EXAMPLE 31. (Note the two empty two brackets, this is because no parameters are going to be passed to the procedure).

```
 5 REM EXAMPLE 31
10 DEF FN A$#()
20 PRINT "HELLO"
30.RETN
```

To call the procedure you would need to execute .PROCFN A$#().(Remember the FN part of .PROCFN is the Sinclair keyword FN under the 2 key).

Now if we wished to specify the X and Y position on the screen where the "HELLO" is printed, you will have to set up these two parameters in the Brackets, see EXAMPLE 32.

```
 5 REM EXAMPLE 32
10 DEF FN A$#(X,Y)
20 PRINT AT X,Y;"HELLO"
30.RETN
```

To print "HELLO" at the X position 1 and the Y position 10, you would type;

.PROCFN A$#(1,10).

If instead of printing "HELLO", we wished to write a procedure that prints any string. The string parameter will have to be included in the brackets, see EXAMPLE 33.

```
 5 REM EXAMPLE 33
10 DEF FN A$#(X,Y,A$)
20 PRINT AT X,Y;A$
30.RETN
```

To print 'GOODBYE' at the X position 5 and Y position 8 you would type;

.PROCFN A$#(5,8,"GOODBYE")

Finally run example 34 to see how local variables are independant of variables with the same name in the main program.

```
  5 REM EXAMPLE 34
 10 DEF FN A$#(X,Y,A$)
 20 PRINT AT X,Y;A$
 30.RETN
 40 LET A$="GOODBYE":  LET X=1:
LET Y=2
 50 PRINT Y,X;A$
 60.PROCFN A$(3,4,"HELLO")
 70 PRINT Y,X,A$
```

WARNING

The inclusion of PROCEDURES in Laser BASIC, interferes with the Spectrum's own stack security and therefore it is important to execute RETURNs from GOSUBs and .RETN's from PROCEDURES in the correct order. Failure to do so may cause the system to crash.

PROGRAMMING TECHNIQUES

The following section is designed to give Laser BASIC newcomers a few hints on programming techniques.

MOVING SOFTWARE SPRITES

The chief problem facing the programmer who wants to move software sprites around the screen, is choosing from the numerous schemes available. We will now consider some of these methods, each with its own merits for speed, simplicity, smoothness and memory.

We will begin with the easiest to implement and then work up to some of the more elaborate techniques.

SCREEN SCROLLING UNDER KEYBOARD CONTROL

This could be used where a sprite on the screen, is moved within a window on the screen, that does not contain any other sprites or data.

Type in the following example, which scrolls a window on the screen by 1 pixel under keyboard control using the ?KBF function.

```
 5 REM EXAMPLE PT1
10 INK 6: PAPER 0: BRIGHT 1: BORDER 0: CLS
20.COL=14:.ROW=10:.SPN=2:.PTBL
30.SET=5:.COL=5:.ROW=4
40.SET=6:.COL=3:.ROW=5
50.SET=7:.ROW=10:.COL=0:.LEN=32:.HGT=2
60.SET=5: LET KB=?KBF: IF KB<>0 THEN .SET=7:.WL1V
70.SET=6: LET KB=?KBF: IF KB<>0 THEN .SET=7:.WR1V
80 GO TO 60
```

Line 10	sets the screen attributes.
Line 20	puts the sprite on the screen.
Line 30	SET 5 points to the ROW and COL position of the 5 key to move left.
Line 40	SET 6 points to the ROW and COL position of the 8 key to move right.
Line 50	sets up the parameters of the window around the screen.
Line 60	if the 5 key is pressed then the window is scrolled left.
Line 70	if the 8 key is pressed then the window is scrolled right.

.SET=5 = the 5 key (move left)
.SET=6 = the 8 key (move right)

SIMPLE PUTTING

Another fairly simple means of moving sprites around the screen is to simply PUT sprites with a blank border around them and simply change the ROW and or COL values.

The sprite required for this type of movement should have a one character border of space around it.

The sprite we will use is sprite 30 which is 3 by 3 characters.

Firstly clear the pixel data in the sprite by typing:

```
.SPN=30:.CLSM
```

Now put sprite 51 into the centre of sprite 30. Since sprite 51 is only 1 character high and wide there will be a one character empty space all around it to rub itself out as it is moved.

So type:

```
        .SP1=51:.SP2=30:.SCL=1:.SRW=1:.GMBL
```

We can now write a routine to move the sprite around the screen under keyboard control using the cursor keys being scanned by ?KBF. The sprite is simply put at a ROW and COL position, the old data being simply removed by the blank part of the sprite.

Type in the following program:

```
  5 REM EXAMPLE PT2
 10.ATOF: INK 5: BRIGHT 1: PAPER 0: BORDER 0: CLS
 20.SET=1:.ROW=4:.COL=5
 30.SET=2:.ROW=5:.COL=5
 40.SET=3:.ROW=5:.COL=4
 50.SET=4:.ROW=5:.COL=3
 60.SET=7:.ROW=10:.COL=13:.SPN=30
 70 LET X=13: LET Y=10
 80.SET=1: LET KB=?KBF: IF KB<>0 THEN LET X=X-1
 90.SET=2: LET KB=?KBF: IF KB<>0 THEN LET Y=Y+1
100.SET=3: LET KB=?KBF: IF KB<>0 THEN LET Y=Y-1
110.SET=4: LET KB=?KBF: IF KB<>0 THEN LET X=X+1
120 IF X>30 THEN LET X=30
130 IF X<-1 THEN LET X=-1
140 IF Y>22 THEN LET Y=22
150 IF Y<-1 THEN LET Y=-1
160.SET=7:.COL=X:.ROW=Y:.PTBL
170 GO TO 80
```

Line 10	sets the attributes of the screen.
Lines 20	to 50 set the ROW and COL values for the keys 5, 6, 7 and 8 (cursor keys).
Line 60	sets the parameters for sprite 30.
Line 70	sets X and Y to the COL and ROW positions of sprite 30.
Lines 80	to 110 scan the keyboard and adjust the values of X and Y depending on which key has been pressed.
Lines 120	to 150 check that X and Y do not put the sprite off the screen.
Line 160	puts sprite 30 on the screen (the empty border around it removes any data on the screen).

.SET=1 = key 5 left
.SET=2 = key 6 down
.SET=3 = key 7 up
.SET=4 = key 8 right
X=COL position of sprite
Y=ROW position of sprite

The great limitation of this routine is, however, that data such as other sprites, already on the screen, will be removed as the sprite is PUT over it.

We will now look at logical operations available to the Laser BASIC user.

LOGICAL OPERATIONS

There are three types of logical operation that are used in Laser BASIC; these are OR, XOR and AND. To get the best out of this package it is important to fully understand these operations.

If a GET or PUT postfixed with "BL" is executed, then data is block moved from the source which may be part of the screen, a sprite, or a sprite window, in such a way that whatever was previously held at the destination, which may also be part of the screen, a sprite, or a sprite window, is obliterated and replaced by whatever was at the source. This may not always be the desired effect and quite often the user will want to merge characters or remove parts of the characters and so on. Hence the need for the 3 logic functions which are commands postfixed with "OR", "XR" and "ND", below is a truth table that explains what the result of these operations are on the individual pixels.

If two sprites are "OR"ed together, the resulting sprite will have pixels set where pixels were set in either or both of the sprites being "OR"ed.

If two sprites are "AND"ed together, the resulting sprite will have pixels set where pixels were set in both of the sprites being "AND"ed.

If two sprites are "XOR"ed together, the resulting sprite will have pixels set where pixels were set in either, but reset where pixels were set or reset in both.

These results are summarised as follows and should make things a little clearer:

SOURCE	DESTINATION	OPERATION	RESULT
PIXEL	PIXEL	OR	PIXEL
PIXEL	NO PIXEL	OR	PIXEL
NO PIXEL	PIXEL	OR	PIXEL
NO PIXEL	NO PIXEL	OR	NO PIXEL
PIXEL	PIXEL	AND	PIXEL
PIXEL	NO PIXEL	AND	NO PIXEL
NO PIXEL	PIXEL	AND	NO PIXEL
NO PIXEL	NO PIXEL	AND	NO PIXEL
PIXEL	PIXEL	XOR	NO PIXEL
PIXEL	NO PIXEL	XOR	PIXEL
NO PIXEL	PIXEL	XOR	PIXEL
NO PIXEL	NO PIXEL	XOR	NO PIXEL

USE OF OR

One way of moving a sprite around the screen without destroying any data is to BLOCK PUT the moving sprite and then "OR" the rest of the screen data after each move.

Example PT3 operates in exactly the same way as example PT2 except that every time sprite 30 is placed on the screen the subroutine at line 300 is called, which "OR"s the data (4 sprites) on the screen using .PTOR .

```
  5 REM EXAMPLE PT3
 10.ATOF: INK 4: BRIGHT 1: PAPER 0: BORDER 0: CLS
 20.SET=1:.ROW=4:.COL=5
 30.SET=2:.ROW=5:.COL=5
 40.SET=3:.ROW=5:.COL=1
 50.SET=4:.ROW=5:.COL=3
 60.SET=7:.ROW=10:.COL=13:.SPN=30: GO SUB 300
 70 LET X=13: LET Y=10
 80.SET=1: LET KB=?KBF: IF KB<>0 THEN LET X=X-1
 90.SET=2: LET KB=?KBF: IF KB<>0 THEN LET Y=Y+1
100.SET=3: LET KB=?KBF: IF KB<>0 THEN LET Y=Y-1
110.SET=4: LET KB=?KBF: IF KB<>0 THEN LET X=X+1
120 IF X>30 THEN LET X=30
130 IF X<-1 THEN LET X=-1
140 IF Y>22 THEN LET Y=22
150 IF Y<-1 THEN LET Y=-1
```

```
160.SET=7:.COL=X:.ROW=Y:.SPN=30:.PTBL: GO SUB 300
170 GO TO 80
300.COL=5:.ROW=10:.SPN=5:.PTOR
301.COL=15:.ROW=7:.SPN=15:.PTOR
302.COL=24:.ROW=13:.SPN=23:.PTOR
304.COL=17:.ROW=19:.SPN=32:.PTOR
305 RETURN
```

The above method is quite acceptable when the position and quantity of data is known.

If, however, the data on the screen is variable a different approach will be required, and XOR should be used.

USE OF "XOR"

In example PT4 the sprite is XORed on the screen, when a key is pressed to move it, the sprite is XORed out and XORed back to the screen in a new position. This means that a sprite can move freely around a screen of variable data without rubbing any of it out.

```
  5 REM EXAMPLE PT4
 10 INK 0: PAPER 5: BORDER 5: CLS:.ATOF
 20.SET=1:.ROW=4:.COL=5
 30.SET=2:.ROW=5:.COL=5
 40.SET=3:.ROW=5:.COL=4
 50.SET=4:.ROW=5:.COL=3
 60 FOR N=1 TO 100
 70 LET X=INT (RND*32)
 80 LET Y=INT (RND*22)
 90 PRINT AT Y,X;"*"
100 NEXT N
110.SET=7:.COL=13:.ROW=10:.SPN=47:.PTXR
120 LET X=13: LET Y=10
130.SET=1: LET KB=?KBF: IF KB<>0 THEN LET X=X-1
140.SET=2: LET KB=?KBF: IF KB<>0 THEN LET Y=Y+1
150.SET=3: LET KB=?KBF: IF KB<>0 THEN LET Y=Y-1
160.SET=4: LET KB=?KBF: IF KB<>0 THEN LET X=X+1
170.SET=7: LET S=?COL: LET T=?ROW: IF S=X AND T=Y THEN GO TO 130
180 IF X>29 THEN LET X=29
190 IF X<0 THEN LET X=0
200 IF Y>19 THEN LET Y=19
210 IF Y<0 THEN LET Y=0
220.PTXR:.COL=X:.ROW=Y:.PTXR
230 GO TO 130
```

Line 10	sets up the attributes.
Lines 20	to 50 set the COL and ROW values for the keyboard.
Lines 60	to 100 print *'s at random positions on the screen.
Line 110	sets up the parameters for a sprite (the lunar lander) and put it on the screen.
Lines 130	to 160 scan the keyboard for keys to be pressed and change the values of X and Y if a key has been pressed.
Line 170	checks to see if the sprite needs to be moved, by comparing its present position with X and Y, and if they are the same, goes back to scan the keyboard again.
Lines 180	to 210 check that the sprite is always on the screen.

Line 220 XORs out the sprite, adjusts ROW and COL and XORs the sprite back on the screen.

.MOVE

.MOVE achieves what line 220 of example PT4 does. Once the values of ROW and COL have been set up, and the increments set in HGT and LEN, every execution of .MOVE will increment ROW and COL. So to move a sprite in a particular direction, XORing over data, you would just have to execute a series of .MOVE commands. Example PT5 demonstrates the use of .MOVE to bounce a ball around the screen, over data.

```
  5 REM EXAMPLE PT5
 10 INK 0: PAPER 6: BRIGHT 1: BORDER 7: CLS :.ATOF
 20 FOR N=1 TO 100
 30 LET X=INT (RND*32): LET Y=INT (RND*22)
 40 PRINT AT Y,X;"*"
 50 NEXT N
 60 LET DR=1:.HGT=DR: LET DC=1:.LEN=DC
 70.SP1=50:.SP2=50: LET R=10:.ROW=R: LET C=13:.COL=C:.SPN=50:.PTXR
 80.MOVE
 90 LET C=?COL: LET R=?ROW
100 IF C=29 OR C=0 THEN LET DC=DC*-1:.LEN=DC: BEEP 0.01,5
110 IF R=19 OR R=0 THEN LET DR=DR*-1:.HGT=DR: BEEP 0.01,5
120 GO TO 80
```

Line 10 sets the attributes.
Lines 20 to 50 fill the screen with stars.
Line 60 sets up the initial incrementation values.
Line 70 sets up the parameters for .MOVE .
Line 80 executes .MOVE .
Line 90 reads the new values of ROW and COL.
Lines 100 and 110 check to see if the ball has hit the side and changes its direction accordingly.

You may have noticed in other examples that ROW and COL can have values that lie outside the dimensions of the screen, e.g. values <0 and >31 for COL and values <0 and >23 for HGT. If .MOVE or group 1 GETs and PUTs are used a sprite can be placed partially "off screen".

Two sprite numbers (stored in SP1 and SP2) are required for .MOVE , if the values in SP1 and SP2 are different, a two sprite animation sequence can be achieved.

HIRESOLUTION PUTTING

Laser BASIC does not include commands to directly PUT a sprite onto the screen with pixel resolution. For those who wish to move a sprite about the screen with a finer resolution, the following methods can be used.

If you wished to move a sprite from left to right by 2 pixels without scrolling the screen, you would first have to use the Sprite Generator Program to create 4 sprites, each one succesively shifted to the right by 2 pixels, such that if the 4 sprites are sequentially placed, the data in the sprite will have moved 8 pixels with a 2 pixel resolution. If these sprites were numbered 1 to 4, the routine to move them could be:

```
10 FOR C=0 TO 31
20.COL=C:.SPN=1:.PTBL:.SPN=2:.PTBL
30.SPN=3:.PTBL:.SPN=4:.PTBL
40 NEXT C
```

Each time all 4 sprites have been placed, COL is incremented and the sequence repeated. (remember, in the above example the sprite must have a trailing blank column to remove the data as it goes along).

In the demo program a yellow bouncing character is shown, hopping across the screen. The animation is obtained by the above method using 4 sprites.

COLLISION DETECTION

Two words are provided to enable collision detection, these are ?SCV and ?SCM .

?SCV is used to scan a particular character position of the screen. If any data is present in the character position specified by COL and ROW, then a non-zero value will be returned (a 0 value is returned if there is no data there).

?SCV is demonstrated in example PT6, which places a random number of stars on the screen, which is then scanned (an X is placed on the screen after the character has been scanned so that you can keep track) and the total number calculated.

```
  5 REM EXAMPLE PT6
 10 LET E=INT (RND*50)
 20 FOR N=0 TO E
 30 LET X=INT (RND*32)
 40 LET Y=INT (RND*22)
 50 PRINT AT Y,X;"*"
 60 NEXT N
 70 LET C=0
 80 FOR Y=0 TO 21
 90.ROW=Y
100 FOR X=0 TO 31
110.COL=X
120 LET S=?SCV
130 IF S<>0 THEN LET C=C+1: PRINT AT Y,X;" ": BEEP 0.01,40: GO TO 160
140 PRINT AT Y,X;"X": BEEP 0.01,10
150 NEXT X
160 NEXT Y
170 PRINT AT 0,0;"THERE WHERE ";C;" STARS."
```

Lines 10	to 60 put a random number of stars on the screen as data.
Lines 80	to 110 calculate the COL and ROW positions of the characters on the screen using FOR-NEXT loops and set COL and ROW.
Line 120	scans the character square pointed to by ROW and COL and sets S with the result.
Line 130	if any data is there, (the *), the count (C) is incremented by 1 and the star rubbed out.
Line 140	now prints an X where the character was scanned.
Line 170	prints the number of squares that contained data.

Often it is insufficient to determine whether a particular character square contains data or not, and for this reason the slower, but more powerful command ?SCM, has been included for the advanced user. This will scan the sprite whose number is held in SPN and return a non-zero result if the sprite contains pixel data, or a zero result if it does not. ?SCM is normally used to perform one of three functions:

1. To see if data will collide.
2. To detect an exact pattern.
3. To detect the presence of a pattern.

Collision detection is most commonly used to detect a collision between a sprite moving across the screen and any data which lies in its path. Often the sprite can pass through an occupied character position without a collision occurring, so the ?SCN command is insufficient. The procedure is basically to load a dummy sprite with the section of screen into which the sprite is about to be PUT, "AND" it with the sprite about to be PUT and then use ?SCM. If a non-zero value is given then the dummy sprite contains data and therefore a collision has occurred. This is all very well, but a problem occurs if the new sprite position overlaps the old sprite position, because this means that the old sprite has to be removed from the screen before beginning the above detection procedure and subsequently PUTting the new sprite. This delay causes flicker. The easiest solution is to work with "XOR"s so that the window can be GOT, "XOR"ed with the old sprite in memory to remove the old sprite data, and then to do the detection followed finally by the blotting and then immediate PUTting.

Once an impending collision is detected it is frequently useful to determine what the sprite has collided with. To begin with, let's assume that the screen window we're examining contains one of a known set of objects and that no other data is present in the window. The method is to load the dummy sprite with the object to be tested and then compare it against the set of sprites with which a match is being sought. To compare the dummy sprite with a known sprite, all you need to do is XOR the sprite being tested into the dummy and do a ?SCM. If the result is zero, an exact match was found, if not, do a second XOR into the dummy to restore it and test the next candidate.

Finally, consider the case where the object being tested contains extraneous data in addition to one of the possible sprites. This time, the dummy sprite is loaded with the contents of the screen window, but the candidates are first "AND"ed into the dummy to remove extraneous data before the XOR and ?SCM. Finally the dummy needs to be reloaded from the screen before the next test. This latter test is limited by the fact that its conclusion is only that the screen contained all the parts of the sprite with which a comparison was made. In the extreme case of the screen window containing all pixels set, then an agreement would be found with all the sprites tested.

SCROLLING LANDSCAPES

Scrolling landscapes are an integral part of so many video games that it is worth a brief description of how they can best be produced using Laser BASIC.

The first and most obvious point is never to scroll more than you have to. If, for instance, you are moving a mountain range where the variation takes place over the top three characters, then only the top three characters need to be stored and moved.

The simplest and most effective method of producing smooth scrolls is to sacrifice a column of the screen for transactions with the sprite being scrolled. Suppose you are scrolling a sprite of 4 or 5 screens width which uses rows 8 to 10 (3 rows). Suppose we require pixel scrolling and there is no horizontal variation in attributes. It doesn't really matter which column we sacrifice, far right (column 31) or far left (column 0), but let's, for this example, use column 0. All that we need to do is set up a window 1 character wide and 3 characters high on the far left of the landscape to have the same INK and PAPER colours. This means that pixel data cannot be seen in this region. Use the .SETV command to do this. To begin with, 31 columns of the sprite are PUT to the active part of the screen using the .PWBL command. If scrolling is to the left, then the dummy column should be loaded with the next column to the right of the sprite now 'on screen'. If scrolling is to the right then the column to the left of the sprite window should be inserted. The full 32 column screen window is now wrapped in the appropriate direction until a total of + or - 8 pixels has been accrued. The dummy column is then loaded from the appropriate sprite column and so on.

REDEFINING CHARACTER SETS AND UDGs

The 21 UDGs available on the Spectrum should not be used as they would corrupt the program. However, as many users know, there is a Sinclair system variable known as CHARS which points to the address in memory 256 bytes less than where the data for the character set is in ROM, starting with character 32 up to character 128. Users can poke new values into CHARS and make it point to new data that will be treated as the character set. The obvious place for Laser BASIC users to store character sets is in sprites.

In EXAMPLE PT7 the data for a little man is poked into sprite 1 and then the system variable CHARS is changed to point to that data and so that every time the space (character 32) is printed, the little man is printed instead.

```
 5 REM EXAMPLE PT7
10.SPN=1
20 LET A=?TST
30 LET A=A+5
40 FOR N=0 TO 7
50 READ D
60 POKE A+N,D
70 NEXT N
```

```
 80.POKE23606,(A-256)
 90 STOP
100 DATA 24,24,0,255,60,60,36,102
110.POKE23605,15360
```

Line 10	sets SPN to 1.
Line 20	finds out the address in memory of sprite 1 using ?TST .
Line 30	REMEMBER THE FIRST 5 BYTES OF SPRITE DATA ARE THE SPRITE NUMBER AND POINTERS, DATA SHOULD BE LOCATED 5 BYTES PAST THE VALUE FOUND BY ?TST. HENCE A=A+5.
Lines 40	to 70 poke the 8 bytes of data from line 100 into the sprite starting 5 bytes after the value of ?TST.
Line 80	pokes CHARS (address 23606) with the address of the data minus 256.
Line 100	is the data for the little man.
Line 110	since all the other characters in the character set have not been redefined they will appear as rubbish. Use POKE 23606,15360 to restore the character set by re-setting the original value of CHARS. (Type GOTO 110).

Using data statements is a slow way of creating character sets in sprites. It is far better to create the character sets in the Sprite Generator Program. If you create a sprite, you must note that data in sprites is stored serially such that the first data is the top line of pixels in the sprite, the next is the second line of pixels etc.

A sprite used for a character set will need to be 1 character wide and the required number of characters high.

THE VARIABLE SETS

Making full use of the 16 variable sets will considerably speed up program execution. If, for instance, you wished to scroll 4 windows on the screen, you could set up the parameters ROW, COL, HGT and LEN of each window in a different variable set.

e.g.
```
.SET=1:.HGT=5:.LEN=5:.ROW=0:.COL=0
.SET=2:.HGT=4:.LEN=6:.ROW=0:.COL=5
.SET=3:.HGT=6:.LEN=4:.ROW=0:.COL=12
.SET=4:.HGT=7:.LEN=3:.ROW=0:.COL=18
```

To execute the scrolls all you would need to type would be:

```
.SET=1:.WR1V:.SET=2:.WL2V:.SET=3:.WL8V:.SET=4:.WR1V
```

Not only is memory saved by not needing to redefine ROW, COL, HGT and LEN every time you wish to scroll a window, but also, execution time is speeded up enormously since less evaluation is done.

LOADING AND SAVING LASER BASIC PROGRAMS

When a Laser BASIC program is running, the extended commands are semi-compiled into what is referred to as a 'tokenised' form. This means that care needs to be exercised when loading and saving from within tokenised programs and a few simple rules need to be observed. Programs can be loaded and saved in command mode (typed in directly) or from within programs, so let's deal with the former case first.

LOADing and SAVEing in Command Mode

To save a 'non-auto-run' Laser BASIC program directly use:

```
SAVE "filename"                                  for tape
SAVE *"m";N;"filename"                           for microdrive
```

NOTE: If you wish, you can put a number of saves and loads into one direct statement, such as:

```
SAVE "filename": LOAD" filename"CODE             for tape
SAVE *"m";N;"filename":LOAD *"m";N;"filename"CODE for microdrive
```

If you wish to execute further standard BASIC commands in the same direct statement line, or if you do not have interface 1, then you should proceed as normal. If, however, you wish to execute further commands in the same statement, and you are using microdrives as opposed to tape, then you will need to execute a RANDOMISE USR 58841 between loads and saves, and the extended commands, e.g.

```
SAVE *"m";N;"filename": LOAD *"m";N;"filename"CODE: RANDOMISE USR
58841:.REMK:.RNUM
```

In point of fact, most people would simply split the statement line into statements, in which case there is no need for the USR call, i.e.

```
SAVE *"m";N;"filename":LOAD *"m";N;"filename"CODE
```

followed by

```
.REMK:.RNUM
```

would be perfectly legal.

SAVEing an 'Auto-Run' Program in Command Mode

With the above rules in mind, the auto-run facility is used in the normal way, i.e.

SAVE "filename" LINE N	for tape
SAVE *"m";N;"filename" LINE N	for microdrive

The program being saved however, must execute one of the following statements before encountering any of the extended commands.

RUN, GOTO or GOSUB	for tape
RANDOMISE USR 58841 followed by RUN, GOTO or GOSUB	for microdrive

For example, if using tape:

```
10 PRINT "LOADED": GO TO 20
20.COL=4:.ROW=4:.LEN=1:.HGT==5:.INVV
30 STOP
```

would be saved using

```
SAVE "TEST" LINE 10
```

and using microdrive:

```
10 PRINT "LOADED": RANDOMISE USR 58841: GO TO 20
20 .COL=4:.ROW=4:.LEN=1:.HGT=5:.INVV
30 STOP

SAVE *"M";1;"TEST" LINE 10
```

NOTE: Programs saved in direct mode are not tokenised when they are loaded back in. The GOTO, GOSUB and RUN commands check to see if a program is tokenised, and if not, will tokenise it. Hence the procedure. Programs saved in direct mode using this format
can only be loaded back, in direct mode.

SAVEing a 'Non-Auto-Run' Program from within a Program

The program will be saved in a tokenised form and can therefore only be loaded back from within a program. The loaded program will be de-tokenised on return to command mode.

SAVEing an 'Auto-Run' Program from within a Program

The program will be saved in a tokenised form and can therefore only be loaded back from within a program. If the program is loaded from tape, it will simply execute; if it is loaded from microdrive then the loaded program will need to execute a RANDOMISE USR 58841 before continuing with the rest of the program.

To summarise, then, programs saved in direct mode are loaded back in direct mode. Programs saved from within a program are re-loaded from within a program. Before using any of the above schemes, Laser BASIC must be resident and running.

THE SPRITE GENERATOR PROGRAM
by Paul Newnham

INTRODUCTION

The Sprite Generator Program is used for the creation and editing of software sprites that are going to be used in your Laser Extended BASIC programs. In fact sprites are created on the screen, then GOT into memory before being saved to tape.

The program is supplied in tape format, but it can be simply modified for use on microdrives.

LOADING

Clear the Spectrum by typing RANDOMISE USR 0. Insert the tape and type LOAD"SPTGEN" or LOAD"".

If you wish to save the sprite generator program, break into it using the break key in the normal fashion then type GOTO 9999 and the two parts of the program will be saved to tape and then Verified.

You will have to reload the program before you can run it.

To save the sprite generator program to microdrive, first edit line 2:

change LOAD""CODE:
to LOAD *"M";1;"G"CODE:

Using a formatted cartridge in drive 1 type GOTO 9998. This will save and verify the program to a microdrive cartridge.

You can load in the sprite generator program by typing:

```
PRINT USR 0
LOAD *"M";1;"S"
```

GETTING STARTED

First load in the Sprite Generator Program. You will be prompted with COLD or WARM start. This is the first time you are running the program, so type C for COLD start, and Y for yes. The working screen will now be displayed.

GLOSSARY OF TERMS

COLD START

If you enter the sprite generator program via a COLD start, then all sprites previously stored will be cleared and all system variables reset. The program must always be initially entered via a COLD start.

WARM START

If you enter the program via a WARM start then all sprites will be conserved and all system variables left unchanged. It is provided principally for re-entering the program after an accidental BREAK or ERROR. If you do accidentally BREAK; type: GOTO 3 and then enter via the WARM start. You will lose, however, any data on the screen. You could type GOTO 100 which will put you back at command level.

THE CHR$ SQR

CHR$ SQR is the abbreviation used throughout this text for the character square, and refers to the 8 by 8 grid to the left of the sprite screen. This is the area used to create and edit sprites one character at a time.

THE SPRITE SCREEN

This is the area of screen 15 characters by 15 characters on which sprites are created, developed, transformed and generally worked on.

THE CHR$ SQR CURSOR

This is the non-destructive flashing cursor which is used to design and edit the character currently held in the CHR$ SQR.

THE SPRITE SCREEN CURSORS

These are the two flashing cursors, displayed in the row beneath the sprite screen and the column to the right of the sprite screen. They are used to indicate the position of the top left hand corner of the screen window currently being operated upon. The actual cursor positions are measured from the top left hand corner of the sprite screen and are displayed in real time on the screen as X POS (column) and Y POS (row). Top left is X POS 1 Y POS 1. Bottom right is X POS F Y POS F.

SPRITE SCREEN WINDOW

The area of the screen currently being worked on is referred to as the screen window. Its position is defined by X POS and Y POS, which correspond to the positions of the sprite screen cursors, and its dimensions are defined by SPRITE HEIGHT and SPRITE LENGTH. To see the screen window you are currently working on just press F. The window will flash.

SPRITES

Once you have finished creating your sprites they can be saved off to tape or microdrive ready to be loaded into Laser Extended BASIC for use in your programs. When saving your finished sprites you are given two saving options:

OPTION 1:	This is the editing save, which saves off your sprites in a form such that they can be loaded back into the Sprite Generator Program for re-editing etc. Sprites saved via OPTION 1 cannot be used in Laser BASIC programs.
OPTION 2:	This saves off the sprites ready to use in Laser BASIC. You should note down the loading values and SPST and SPND values that are presented to you. Sprites saved in this option cannot be loaded back into the Sprite Generator Program.

For those who do not possess an artistic ability, two sets of sprites have been saved on tape for you in OPTION 1 format ready to be loaded into the Sprite Generator.

SPRITE1A	This is a file of 50 sprites of various arcade characters.
SPRITE2A	This is a file of all the sprites used in the Laser BASIC demo.

The Sprite Development Program allows sprites to be loaded and saved to and from microdrive cartridge. Before a cartridge can be used to store sprites, it has to be specially formatted. This is done from the Sprite Generator Program by hitting Symbol Shift F (TO). This will format the cartridge and set up five dummy files numbered 1 to 5. From now on whenever you save a file of sprites, the old file of that number will be erased to conserve cartridge storage.

THE INFORMATION RECTANGLE

```
MEMORY LEFT 7488                 X POS 1 Y POS 1
SPRITE 60218                  SPRITE HEIGHT - 2
SPST 60218                    SPRITE LENGTH - 4
SPND 65279                    SPRITE NUMBER - 1
     The text line
```

MEMORY LEFT:	This indicates how much memory is available for sprites.
X POS Y POS:	These are the current positions of the SPRITE SCREEN X and Y cursors with reference to the figures on top and to the left of the SPRITE SCREEN.
SPRITE:	This indicates the position, in memory, where your defined sprite is.
SPST:	This indicates the SPrite space STart point, in memory. (Before any sprites are defined this has an initial value of 65218).
SPND:	This indicates the SPrite space eND point, in memory.
SPRITE HEIGHT:	This indicates the height of your defined sprite, in character squares, as indicated by the figures at the top and to the left of the SPRITE SCREEN. (This has an initial value of 1).
SPRITE LENGTH:	This indicates the length of your defined sprite, in character squares, as indicated by the figures at the top and to the left of the SPRITE SCREEN. (This has an initial value of 1).
SPRITE NUMBER:	This indicates the sprite currently defined. (This has an initial value of 1).
The Text Line:	**To show the current function and the available options.**

SPRITE GENERATOR KEY FUNCTION SUMMARY

All the functions of the Sprite Generator are invoked by pressing the appropiate key (or SYMBOL SHIFT key). A list of the functions is given below.

KEY

A

Activates the ATTRIBUTE switch.(same as .ATOF,.ATON)
Press 1 to set switch ON.
Press 0 to set switch OFF.

B

Activates the BRIGHT variable.(same as Sinclair Bright)
Press 1 to set BRIGHT to ON.
Press 0 to set BRIGHT to OFF.

C

Activates the PAPER variable.(same as Sinclair PAPER)
Press any key between 0 and 7 to activate the colour indicated above the key.(paper 8 and 9 not included).

SYMBOL SHIFT C (?)

CREATION OF LARGE SPRITES
Allows the creation of a sprite, whose number is held in the sprite number variable, of user definable dimensions in the range 1-255 characters. The sprite can be said to be empty as no data will have been "GOT" into it.

D

Activates DIRECT DATA INPUT.
Accepts 8 bytes of data, one byte at a time, followed by ENTER, via the keyboard, to the position on the sprite Screen indicated by the cursors. Inputted data must be in the range 0 to 255 Decimal or, H00 to HFF HEX (the character H must precede Hex entry).

NOTE: If Attribute switch = 1, then the four current attributes will be set at the same position as well.

E

Activates the SCREEN FUNCTIONS.
You will be given three options: press 1, 2 or 3.

1 INVERT (same as .INVV)
Option 1, INVERT, sets all OFF pixels to ON and all ON pixels to OFF in a window whose length is held in the "Sprite length" variable and whose height is held in the "Sprite height" variable. The inversion will take place from the position of the sprite screen cursors, i.e. at the intersection of an imaginary line drawn from each cursor.

2 MIRROR (same as .MIRV)
Option 2, MIRROR, 'Flips' a window whose height is held in the "Sprite height" variable and whose length is held in the "Sprite length" variable. The Mirroring will take place about the vertical centre of the screen window.

3 MIRROR ATTRIBUTES (same as MARV)
Option 3, MIRROR ATTRIBUTES, 'Flips' the attributes in a window whose height is held in the "Sprite height" variable and whose length is held in the "Sprite length" variable. The Mirroring of Attributes will take place about the vertical centre of the screen
window.

F

Activates FLASH WINDOW.
Flashes the current screen window whose height is held in the SPRITE HEIGHT variable and whose length is held in the SPRITE LENGTH variable. The Flash will take place at the position of the sprite screen cursors.

Flash is used to check the position of the sprite screen cursors, to check that the height and length parameters are as required or to check that the window is correctly positioned.

SYMBOL SHIFT F (TO)	Format Sprite Cartridge This is used to format a microdrive cartridge ready for saving sprites to. It sets up five dummy files numbered 1 to 5.
G	Activates GET SPRITE function. Gets a sprite of the dimensions held in the "Sprite height" and "Sprite length" variables, using the number held in the "Sprite Number" variable and at the window indicated by the sprite screen cursors - and stores it in memory. NOTE: If the Attribute switch = 1, the sprite and attributes are stored; if the Attribute switch = 0, then any Attributes will be ignored. If a sprite is defined with the Attribute switch = 0, then the attribute data will probably be garbage.
H	Activates the SPRITE HEIGHT Variable.(same as .HGT=) Permits the input of the height of a sprite window in the range of 1-15 characters.
I	Activates the ATTRIBUTE DUMP function. This fills the window of dimensions held in the "sprite height" and "sprite length" variables with the current attribute values.
SYMBOL SHIFT I (AT)	Activates the SCROLL WINDOW RIGHT BY 1 PIXEL function. (same as .WRIV) This scrolls the window whose dimensions are held in the SPRITE HEIGHT and SPRITE LENGTH variable by 1 pixel to the right with wrap around.
J	Activates the move CHR$ SQR TO SPRITE SCREEN function. Dumps the bit pattern set in the CHR$ SQR to a character square in the sprite screen, indicated by the sprite screen 6cursors. NOTE: If the Attribute Switch = 0, no Attributes will move with the pattern. If the Attribute switch = 1, then the Attributes held in the Attribute Variables will move with the pattern.
SYMBOL SHIFT J (-)	Activates the LOAD SPRITES facility. Sprites can be loaded in from tape or microdrive. Three groups of data will be loaded. Once loaded the text line will clear. NOTE: Any sprites in memory will be destroyed when this command is executed.
K	Activates the MOVE SPRITE SCREEN CHARACTER TO CHR$ SQR function. Picks up the Character Square indicated by the Sprite Screen Cursors, into the CHR$ SQR. NOTE: ATTR = 0 ignores Character Attributes. ATTR = 1 takes the Attributes of the character and loads them into the Attribute Variables.
L	Activates the SPRITE LENGTH variable.(same as .LEN) Permits the input of the length of a Sprite Window in the range of 1-15 characters.
M	Activates the Sprite Functions. You will be given three options which act in the same way as the 'SCREEN FUNCTIONS E', except that these functions operate on the sprite in memory only and have no effect directly on the screen.
N	Activates the No, negative response to (Y/N) questions.
O	Activates the Sprite Logic functions. You will be given three options. Each option GETS an area of the sprite screen, the dimensions of which are specified as those of the defined sprite, having a top left-hand corner at the sprite screen cursor positions and logically GETs the data into the defined sprite — whose number is in the Sprite Number Variable.

	NOTE: ATTR = 0 leaves the sprite attributes as they are. ATTR = 1 takes the attributes from the screen and places them into the sprite. 1 GETORS, ORs the screen data with the pre-defined sprite, and leaves the result in the sprite (screen display unaffected). 2 GETXRS, XORs the screen data with the data of a pre-defined sprite, and leaves the result in the sprite, (screen display unaffected). 3 GETNDS, ANDs the screen data with the data of a pre-defined sprite, and leaves the result in the sprite (screen display unaffected).
P	Activates the PUT SPRITE function.(same as .PTBL) This PUTs the sprite whose number is held in the "SPRITE NUMBER" variable onto the sprite screen at the position indicated by the sprite screen cursors. NOTE: You will get an error message if the sprite does not exist or will not fit on the screen.
Q	Activates the CLEAR CHR$ SQR function. Sets all CHR$ SQR bits to OFF.
SYMBOL SHIFT Q (<=)	Activates the CLEAR SPRITE SCREEN function. Clears the sprite screen of all data and attributes.
R	Activates the ROTATE SPRITE function.(same as .SPNM) Rotates a sprite, in memory, by 90 degrees, leaving the original sprite unaffected. The new Rotated sprite must be given a new sprite number, as asked for. Attributes are automatically Rotated with the pixel data.
S	Activates the SPRITE NUMBER variable.(same as .SPN=) Permits the defining of sprites and asks for a sprite number in the range 1 to 255 NOTE: If a sprite to be defined is given an existing sprite number, a warning is displayed, advising you of this fact. The existing sprite, or the new sprite, are in no way corrupted.
SYMBOL SHIFT S (NOT)	Activates the SAVE SPRITES facility. All files will be verified. Once the programs have verified, the sprite development program will return to command level with the text line cleared. NOTE: If the program breaks because of failure to verify, type GOTO 100 and your data will not be lost.
T	Activates the TEST SPRITE function.(same as ?TST) Performs a test on the sprite whose number is held in the "Sprite Number" variable, and does the following: 1. Places the sprite height into the "Sprite height" variable. 2. Places the sprite length into the "Sprite length" variable. 3. Places the address in memory of where the sprite data starts, into the "Sprite" variable. 4. Places the address of the start of sprite space into the variable "SPST". 5. Places the address of the end of sprite space into the variable "SPND". 6. Calculates the remaining memory available for sprite storage and places it into the "Memory Left" variable. NOTE: The screen display of these variables will be updated if necessary.
SYMBOL SHIFT T (>)	Activates the SCROLL WINDOW LEFT BY 1 PIXEL function. (same as .WL1V) This scrolls the window whose dimensions are held in the SPRITE HEIGHT and SPRITE LENGTH variables by 1 pixel to the left with wrap around.

U	Activates the PICK UP ATTRIBUTES function. Picks up the attributes of the character from the sprite screen, indicated by the position of the sprite screen cursors and Loads them into the four Attribute variables.
SYMBOL SHIFT U (OR)	Activate the SCROLL WINDOW UP BY 1 PIXEL function (same as .WCRV). This scrolls the window whose dimensions are held in the SPRITE HEIGHT and SPRITE LENGTH variables by 1 pixel up with wrap around.
V	Activates the FLASH variable. This is one of the four attributes. (same as Sinclair Flash) Press 1 to put switch ON. Press 0 to put switch OFF.
W	Activates the WIPE SPRITE function.(same as .DSPR) Wipes the sprite indicated by the "Sprite number" variable totally from memory. All other sprites stored in memory below that sprite are moved up to fill the space previously occupied by the Wiped sprite.
X	Activate the INK variable which is one of the four attributes. (same as Sinclair INK) Press any key between 0 and 7 to set the colour indicated above the key.
Y	Activates the YES, positive response to (Y/N) questions.
SYMBOL SHIFT Y (AND)	Activates the SCROLL WINDOW DOWN BY 1 PIXEL function. (same as .WCRV) This scrolls the window whose dimensions are held in the SPRITE HEIGHT and SPRITE LENGTH variables by 1 pixel down with wrap around.
BREAK and SPACE	Activates the PLACE SPRITE INTO SPRITE WINDOW facility. This allows you to place a sprite of smaller dimensions into a second sprite of greater dimensions, at a position of ROW, COL in the greater sprite in memory - the smaller sprite is left unaltered. NOTE: ATTR = 0, Attributes of smaller sprite ignored. ATTR = 1, Attributes of smaller sprite taken and placed with sprite. Four options are given: 1 GETBLS: GETs the smaller sprite directly into the window of the larger sprite. 2 GETORS: GETs the smaller sprite and ORs it into the window of the larger sprite. 3 GETXRS: GETs the smaller sprite and XORs it into the window of the larger sprite. 4 GETNDS: GETs the smaller sprite and ANDs it into the window of the larger sprite.
5	Activates the MOVE CHR$ CURSOR 1 place to the left - non-destructive.
6	Activates the MOVE CHR$ SQR CURSOR 1 place down - non-destructive.
7	Activates the MOVE CHR$ SQR CURSOR 1 place up - non-destructive.
8	Activates the MOVE CHR$ SQR CURSOR 1 place to the right — non-destructive.

9	Activates the SET CHR$ ON at current position.
0	Activates the SET CHR$ OFF at current position.
(%) SYMBOL SHIFT 5)	Activates the MOVE SPRITE SCREEN CURSOR 1 place to the left.
(&) SYMBOL SHIFT 6)	Activates the MOVE SPRITE SCREEN CURSOR 1 place down.
(') SYMBOL SHIFT 7)	Activates the MOVE SPRITE SCREEN CURSOR 1 place up.
(() SYMBOL SHIFT 8)	Activates the MOVE SPRITE SCREEN CURSOR 1 place to the right.

THE LASER BASIC SPRITE GENERATOR EXAMPLE SESSION

This chapter is written to enable the user to gain experience and understanding of the use of the Sprite Generator Program supplied with the package.

You will first need some sprites to work with. Position the tape so as to be ready to load the SPRITE1A file. Type SYMBOL SHIFT J (LOAD SPRITES) and type Y for yes, press PLAY on the tape recorder and the file of OPTION 1 saved sprites will be loaded into the memory of the Sprite Generator Program. (Please note that Sprites are loaded and Saved in 3 parts)

SG1. Let's firstly familiarise ourselves with the use of the two screens.

THE CHR$ SQR

This is the grid square on which you create and edit sprites a character at a time. To move the cursor:

1. Press the 5 key for each movement to the left.

2. Press the 6 key for each movement downward.

3. Press the 7 key for each movement upward.

4. Press the 8 key for each movement to the right.

Now that you know how to move the cursor, let's fill in a few squares:

1. Move the cursor to any square that you like and release the keys.

2. Press the 9 key to set the square.

3. Now move the direction keys and fill in a few more squares.

Now that we have set some squares, what about deleting a few of them? This is simple:

1. Move the cursor to a square that you have set and release the keys.

2. Press the 0 key to clear the square.

Now have a go at setting and clearing some squares, just to get used to it.

THE SPRITE SCREEN CURSORS

Moving the sprite screen cursors:

1. Move the X cursor by pressing SYMBOL SHIFT and the 5 or 8 key to move left or right respectively.

2. Move the Y cursor by pressing SYMBOL SHIFT and the 7 or 6 key to move up or down respectively.

CLEARING THE SCREENS

1. Press the Q key and respond to the prompt in the text line by pressing Y and the CHR$ SQR will clear.

   Just to get you used to a similar function, let's clear the Sprite Screen as well, even though it's clear:

1. As you can see, to clear the CHR$ SQR press Q, to clear the sprite screen press SYMBOL SHIFT Q.

SG2. EDITING A SPRITE BY A CHARACTER AT A TIME USING THE CHR$ SQR

1. Move the X and Y cursors to 1 and 1 respectively.
2. Press the S key to select a sprite number.
3. Input the number 51 and hit ENTER.
4. Now type P (PUT sprite) and hit Y for yes.

(You have now PUT sprite 51 on the screen without any attributes.)

5. Press the K key and hit Y for yes to load the CHR$ SQR with the data of the character pointed to by the sprite screen cursors.

(You can now edit the character held in the CHR$ SQR using the 5,6,7,8 keys to move the non-destructive cursor and the 9 or 0 keys to set or unset the pixels.)

6. Once you have edited the character, hit the F key to flash the cursor in the sprite to check that it is in the right position.
7. Press the J key to move the data on the CHR$ SQR to the sprite screen, pressing Y for yes.

SG3. INPUTTING DATA VIA THE DIRECT DATA INPUT FUNCTION

1. Press SYMBOL SHIFT Q to clear the sprite screen.
2. Press the D key, answer Y to the question, and enter the following, very carefully, pressing ENTER after each entry:
   a) H24 126 HDB 255 HFF 153 129 102
3. You should have a space invader type character.

This is the DIRECT DATA INPUT. Direct Data characters are built up from 8 bytes of data, one byte at a time.

NOTE: Data can only be entered using values in the range 0 to 255 Decimal or H00 to HFF HEX. The character H must precede a HEX entry.

SG4. SETTING THE ATTRIBUTE VALUES

1. Clear the sprite screen (SYMBOL SHIFT Q)
2. Press X to activate the INK variable and then set it to 2.
3. Press C to activate the PAPER variable and then set it to 7.
4. Press B to activate the BRIGHT switch and then press 1 to switch it ON.
5. Press V to activate the FLASH variable and then press 0 to switch it OFF.
6. Press A to activate the ATTRIBUTE switch and then press 1 to switch it ON.

(You will have noticed, that both PAPER and FLASH were already set to 7 and 0 respectively from the COLD start; we only run through them all for completeness and to get used to using them.)

7. Now if you repeat example SG3, because the ATTRIBUTE switch is set to 1, attributes will be used, hence you have defined a RED invader.

SG5. GETTING A SPRITE INTO MEMORY

1. Let's imagine we have designed a sprite on the screen, so type S for input sprite number and input 6.
2. Type P for PUT SPRITE and hit Y for yes.

(You now have a mouse of 2 characters by 2 characters which we can say is the data we wish to get.)

3. Type W and Y for yes to wipe sprite 6 from memory.
4. The mouse data is 2 characters wide so type L for length and input 2 for the length of the window.
5. The mouse data is 2 characters high so type H for height and input 2 for the length of the window.
6. Type F to flash the current window to check that its dimensions and position cover the data you want to get as the sprite.
7. Type S and input 6 as the number of the sprite.
8. Now type G and Y to GET the sprite into memory.
9. You can check that the sprite was correctly got into memory by moving the sprite screen cursors and putting the new sprite somewhere else on the screen.

SG6. SPRITE SCREEN FUNCTIONS

1. Firstly clear the sprite screen using SYMBOL SHIFT Q.
2. Set the sprite number variable to 19 using the S key.
3. PUT the sprite on the screen using the P key.

(You will see a ship of 7 characters long by 2 high.)

4. If you type E and Y for Sprite Screen functions and then type 1 for INVERT, all pixels that were ON are now OFF and all the pixels that were OFF are now ON.
5. If in this function you type 2 for MIRROR, the data (not the attributes) will be mirrored about a central vertical axis, reversing the direction of the ship.
6. Function 3 in this mode will do the same as function 2, except that the attributes, not the pixel data, will be mirrored.

(All the above three operations are local operations, that is to say sprite 19 has not been affected, only the data on the screen.)

SG7. SPRITE FUNCTIONS

The same operations as SG6 are available using the M key, however these operations operate in memory, permanently changing the sprite being operated on.

SG8. SPRITE ROTATION

This enables a sprite to be rotated through 90 degrees (clockwise).

1. PUT sprite 9 on the screen using the S and P keys, and you should have a bi-plane.
2. To rotate this sprite hit the R key and Y for ROTATE.
3. You will have to input a new number of a sprite that sprite 9 is going to be rotated into, so type 100. Sprite 9 is left unaffected, but sprite 100 contains the rotated sprite 9.
4. Use the S and P keys to PUT sprite 100 on the screen.

SG9. SETTING ATTRIBUTES INTO WINDOWS

Now let's look at attribute handling in more detail - position the X and Y cursors to X POS 1 Y POS 1. The following two examples will show how to download and pick-up attributes between the attribute variables and the sprite screen:

1. Press X (INK) and set to 3 (magenta).
2. Press C (PAPER) and set to 2 (red).
3. Press V (FLASH) and set to 1 (ON).
4. Press B (BRIGHT) and set to 0 (OFF).
5. Press A (ATTR) and set to 0.
6. Set the window length to 5 using the L key.
7. Set the window height to 5 using the H key.
8. Press I (ATTRIBUTE DUMP) - the attributes will appear on the sprite screen in the 5 by 5 character window. Any data in that window will remain, but its attributes will have changed.
9. Now set all the attributes, X, C, V, B, and A to 0.
10. Press U (PICK UP ATTRIBUTES) and the attributes on the screen will be loaded into the attribute variables.

SG10. SCROLLING SPRITE WINDOW DATA

1. First put a sprite on the screen, at X pos 1, Y pos 1, using the S key for 'Input Sprite Number' and inputting the number 10.
2. Type P for PUT sprite, you will see a helicopter appear on the screen.
3. Now set both the window height and length values to 10 using the H and L keys.
4. You can now scroll, with wrap, the helicopter within the window by 1 pixel using the SYMBOL SHIFT key in conjunction with the T, Y, U or I keys.

One way of achieving fast, smooth, hi-resolution animation of sprites, is to define a series of sprites in different positions, offset by a few pixels. Then by sequentially placing these sprites hi-resolution animation can be obtained.

SG11. SAVING SPRITES

You may now wish to save off all the sprites that you have just created in a form that they can be loaded back into the Sprite Generator Program at a future date. You will have to save the sprites in the OPTION1 format.

1. Type SYMBOL SHIFT S for save sprites.
2. Type Y for yes.
3. Type 1 for OPTION1 sprites.
4. Now save sprites to tape or pre-formatted microdrive, inputting the file name.
5. Sprites are saved off in 3 parts, which will be verified, this means once saved, the tape will have to be rewound.

Once all your sprites are finished you can save them off in OPTION2 format ready to load into LASER BASIC.

1. Type SYMBOL SHIFT S for save sprites.
2. Type Y for yes.
3. Type 2 for Option2.
4. Input the file name.
5. Note down the CODE values that are displayed on the screen.
6. The code will be verified next, so rewind the tape after it has been saved.

CREATING SPRITES IN THE SPRITE GENERATOR PROGRAM

A SUMMARY

Step 1	Load in the Sprite Generator Program and execute a COLD start.
Step 2	Set the Attribute flag to 1 using the A key.
Step 3	Set the INK and PAPER colours.
Step 4	Create your sprite, a character at a time, using either the CHR$ SQR or the direct data entry method.
Step 5	Enlarge the flashable window so that it takes up the dimensions of your sprite, using the H key to set the height and the L key to set the length.
Step 6	Position the sprite screen cursors to the top left of your sprite.
Step 7	Flash the window using the F key to make sure all the sprite data will be "GOT" into memory.
Step 8	GET the sprite into memory, using the G key.
Step 9	Test to see that the sprite is OK by moving the sprite screen cursors to a free part of the screen and "PUT" the sprite, using the P key.
Step 10	Carry out any other operations or create more sprites.
Step 11	Save off the sprites in OPTION1 format so that they can be loaded in the sprite generator program for editing etc. at a later stage.
Step 12	Save off the sprites in OPTION2 format for use with Laser BASIC, noting down the values of the "Sprite start address".
Step 13	Clear the machine by typing RANDOMIZE USR 0.
Step 14	Load in Laser BASIC by typing LOAD""
Step 15	Load the sprites using option 2 of the loader menu. The loader will prompt you for the "Sprite start address" and you should type in the value you noted down at step 12.

APPENDIX 1 LASER BASIC COMMANDS(IN ALPHABETICAL ORDER)

WORD	PARAMETERS	ACTION
.ADJM	SPN, COL, ROW	Adjust COL, ROW, HGT, LEN, SCOL, SROW such that GETS and PUTS lie on the screen.
.ADJV	HGT, LEN, COL, ROW	Adjust the screen window to lie on the screen.
.ATDV	HGT, LEN, COL, ROW	Scroll the window attributes 1 character down with wrap.
.ATDM	SPN	Scroll the sprite attributes 1 character down with wrap.
.ATLM	SPN	Scroll the sprite attributes 1 character left with wrap.
.ATLV	HGT, LEN, COL, ROW	Scroll the window attributes 1 character left with wrap.
.ATOF		Disable attribute switch.
.ATON		Enable attribute switch.
.ATRM	SPN	Scroll the sprite attributes 1 character right with wrap.
.ATRV	HGT, LEN, COL, ROW	Scroll the window attributes 1 character right with wrap.
.ATUM	SPN	Scroll the sprite attributes 1 character up with wrap.
.ATUV	HGT, LEN, COL, ROW	Scroll the window attributes 1 character up with wrap.
.CLSM	SPN	Clear the sprite.
.CLSV	HGT, LEN, COL, ROW	Clear the screen window and fill with the current attributes.
?COL		Assign the value in the Graphics variable COL to a BASIC variable.
.COL=	BASIC EXPRESSION	Assign the value of the BASIC expression to the Graphics variable COL.
DEF FN	N#()	Define a procedure N.
.DSPM	SP1, SP2	Enlarge sprite SP2 into sprite SP1.
.DSPR	SPN	Delete sprite and recover bytes from below.
.GMAT	SP1, SP2, SCOL, SROW	Block move attributes of sprite SP1 into sprite SP2 at SCOL,SROW.
.GMBL	SP1, SP2, SCOL, SROW	Block move sprite SP1 into sprite SP2 at SCOL,SROW.
.GMND	SP1, SP2, SCOL, SDROW	Logically AND sprite SP1 into sprite SP2 at SCOL,SROW.
.GMOR	SP1, SP2, SCOL, SROW	Logically OR sprite SP1 into sprite SP2 at SCOL,SROW.
.GMXR	SP1, SP2, SCOL, SROW	Logically XOR sprite SP1 into sprite SP2 at SCOL,SROW.
.GTBL	SPN, COL, ROW	Block move screen data from screen to sprite.

.GTND	SPN, COL, ROW	Logically AND screen data into sprite data.
.GTOR	SPN, COL, ROW	Logically OR screen data into sprite data.
.GTXR	SPN, COL, ROW	Logically XOR screen data into sprite data.
.GWAT	SPN, COL, ROW, SCOL, SROW, HGT, LEN	Block move attributes from screen window into sprite window.
.GWBL	SPN, COL, ROW, SCOL, SROW, HGT, LEN	Block move screen data from screen window into sprite window.
.GWND	SPN, COL, ROW, SCOL, SROW, HGT, LEN.	Logically AND screen data from screen window into sprite window.
.GWOR	SPN, COL, ROW, SCOL, SROW, HGT, LEN	Logically OR screen data from screen window into sprite window.
.GWXR	SPN, COL, ROW, SCOL, SROW, HGT, LEN	Logically XOR screen data from screen window into sprite window.
?HGT		Assign the value in the Graphics variable HGT to a BASIC variable.
.HGT=	BASIC EXPRESSION	Assign the value of the BASIC expression to the Graphics variable HGT.
.INVM	SPN	Invert sprite data.
.INVV	HGT, LEN, COL, ROW	Invert screen window.
.ISPR	SPN, HGT, LEN	Create sprite and move current sprites down to accommodate.
?KBF	COL, ROW	Detect multi key presses.
?LEN		Assign the value in the Graphics variable LEN to a BASIC variable.
.LEN=	BASIC EXPRESSION	Assign the value of the BASIC expression to the Graphics variable LEN.
.MARM	SPN	Mirror sprite attributes about centre.
.MARV	HGT, LEN, COL, ROW	Mirror screen window attributes about centre.
.MIRM	SPN	Mirror sprite about its centre.
.MIRV	HGT, LEN, COL, ROW	Mirror screen window about its centre.
.MOVE	SP1, SP2, HGT, LEN, COL, ROW	Move and animate.
?NPX		Assign the value in the Graphics variable NPX to a BASIC variable.
.NPX=	BASIC EXPRESSION	Assign the value of the BASIC expression to the Graphics variable NPX.
?PEK		PEEK a 16 bit number.
.PMAT	SP1, SP2, SCOL, SROW	Block move attributes of window at SCOL, SROW of sprite SP2 into sprite SP1.

.PMBL	SP1, SP2, SCOL, SROW	Block move window at SCOL,SROW of sprite SP2 into sprite SP1.
.PMND	SP1, SP2, SCOL SROW	Logically AND window at SCOL,SROW of sprite SP2 into sprite SP1.
.PMOR	SP1, SP2, SCOL, SROW	Logically OR window at SCOL,SROW of sprite SP2 into sprite SP1.
.PMXR	SP1, SP2, SCOL, SROW	Logically XOR window at SCOL,SROW of sprite SP2 into sprite SP1.
.POKE	N,M	Poke a 16 bit number M at N.
.PROCFN N#()		Call the procedure N.
.PTBL	SPN, COL, ROW	Block move sprite data from sprite to screen.
.PTND	SPN, COL, ROW	Logically AND sprite data into screen data.
.PTOR	SPN, COL, ROW	Logically OR sprite data into screen data.
.PTXR	SPN, COL, ROW	Logically XOR sprite data into screen data.
.PWAT	SPN, COL, ROW, SCOL, SROW, HGT, LEN	Block move sprite window attributes into screen window.
.PWBL	SPN, COL, ROW, SCOL, SROW, HGT, LEN	Block move sprite data from sprite window into screen window.
.PWND	SPN, COL, ROW, SCOL, SROW, HGT, LEN	Logically AND sprite window data into screen window.
.PWOR	SPN, COL, ROW, SCOL, SROW, HGT, LEN	Logically OR sprite window data into screen window.
.PWXR	SPN, COL, ROW, SCOL, SROW, HGT, LEN	Logically XOR sprite window into screen window.
.REMK		Remove all REM statements in a program.
.RETN		Return from procedure.
?ROW		Assign the value in the Graphics variable ROW to a BASIC variable.
.ROW=	BASIC EXPRESSION	Assign the value of the BASIC expression to the Graphics variable ROW.
.RNUM		Renumber lines.
?SCL		Assigns a Sinclair variable with the current value of SCL.
.SCL=		Store a value in the variable SCL.
?SCM	SPN	Scan the sprite for data. 0 = no data, 1 = data.
?SCV	ROW, COL	Scan a character square at ROW, COL for data. 0 = no data, 1 = data.
.SCRM	SPN	Scroll the sprite vertically without wrap by NPX pixels.
.SCRV	HGT, LEN, COL, ROW, NPX	Scroll the window vertically without wrap by NPX pixels.

?SET		Assigns the value in the Graphic variable SET to a BASIC variable.
.SET=	BASIC EXPRESSION	Assign the value of the BASIC expression to the Graphics variable SET.
.SETM	SPN	Fill the sprite with the current attributes.
.SETV	HGT, LEN, COL, ROW	Fill the screen window with the current attributes.
.SL1M	SPN	Scroll the sprite 1 pixel left without wrap.
.SL4M	SPN	Scroll the sprite 4 pixels left without wrap.
.SL8M	SPN	Scroll the sprite 8 pixels left without wrap.
.SL1V	HGT, LEN, COL, ROW	Scroll the window 1 pixel left without wrap.
.SL4V	HGT, LEN, COL, ROW	Scroll the window 4 pixels left without wrap.
.SL8V	HGT, LEN, COL, ROW	Scroll the window 8 pixels left without wrap.
.SPNM	SP1, SP2	Rotate sprite SP2 90 degrees clockwise into sprite SP1.
?SPN		Assign the value in the Graphics variable SPN to a BASIC variable.
.SPN=	BASIC EXPRESSION	Assign the value of the BASIC expression to the Graphics variable SPN.
?SP1		Assign the value in the Graphics variable SP1 to a BASIC variable.
.SP1=	BASIC EXPRESSION	Assign the value of the BASIC expression to the Graphics variable SP1.
?SP2		Assign the value in the Graphics variable SP2 to a BASIC variable.
.SP2=	BASIC EXPRESSION	Assign the value of the BASIC expression to the Graphics variable SP2.
.SR1M	SPN	Scroll the sprite 1 pixel right without wrap.
.SR4M	SPN	Scroll the sprite 4 pixels right without wrap.
.SR8M	SPN	Scroll the sprite 8 pixels right without wrap.
.SR1V	HGT, LEN, COL, ROW	Scroll the window 1 pixel right without wrap.
.SR4V	HGT, LEN, COL, ROW	Scroll the window 4 pixels right without wrap.
.SR8V	HGT, LEN, COL, ROW	Scroll the window 8 pixels right without wrap.
.SPRT	SPN, HGT, LEN	Create sprite at free space after last sprite.

.TROF		Switch off trace function.
.TRON		Switch on trace function.
?TST	SPN	Test sprite.
.WCRM	SPN	Scroll the sprite vertically with wrap by NPX pixels.
.WCRV	HGT, LEN, COL, ROW, NPX	Scroll the window vertically with wrap by NPX pixels.
.WL1M	SPN	Scroll the sprite 1 pixel left with wrap.
.WL4M	SPN	Scroll the sprite 4 pixels left with wrap.
.WL8M	SPN	Scroll the sprite 8 pixels left with wrap.
.WL1V	HGT, LEN, COL, ROW	Scroll the window 1 pixel left with wrap.
.WL4V	HGT, LEN, COL, ROW	Scroll the window 4 pixels left with wrap.
.WL8V	HGT, LEN, COL, ROW	Scroll the window 8 pixels left with wrap.
.WSPR	SPN	Delete sprite and recover bytes from above.
.WR1M	SPN	Scroll the sprite 1 pixel right with wrap.
.WR4M	SPN	Scroll the sprite 4 pixels right with wrap.
.WR8M	SPN	Scroll the sprite 8 pixels right with wrap.
.WR1V	HGT, LEN, COL, ROW	Scroll the window 1 pixel right with wrap.
.WR4V	HGT, LEN, COL,	Scroll the window 4 pixels right with wrap. ROW
.WR8V	HGT, LEN, COL, ROW	Scroll the window 8 pixels right with wrap.

APPENDIX 2 'SPRITE1A' AND 'SPRITE2A' SPRITES

Example sprites for use in users own Laser BASIC games or the Sprite Generator Program.

SPRITE	DESCRIPTION	LEN	HGT	INK	PAPER	FLASH	BRIGHT
1	Vintage car	4	2	5	0	0	1
2	Van	4	2	6	0	0	1
3	Dragster	4	2	3	0	0	1
4	Duck	3	3	6	0	0	1
5	Dancer	2	4	7	0	0	1
6	Mouse	2	2	7	0	0	1
7	Spaceship #1	4	2	5	0	0	1
8	Tank #1	4	2	0	4	0	1
9	Bi-plane	4	2	1	5	0	0
10	Helicopter #1	4	2	0	5	0	0
11	Spaceship #2	4	2	5	0	0	1
12	Spacetank	4	2	0	6	0	1
13	Rocket	4	2	7	1	0	1
14	Jet fighter #1	5	2	1	5	0	1
15	Spaceship #3	5	2	5	0	0	1
16	Spaceship #4	4	2	6	0	0	1
17	Jet fighter #2	4	2	2	5	0	1
18	Tank #2	6	3	0	4	0	0
19	Liner	7	2	0	5	0	1
20	Jet fighter #2	5	2	3	5	0	1
21	Alien	4	2	7	2	0	1
22	Spaceship #5	4	2	4	0	0	1
23	Spaceship #6	6	2	5	0	0	1
24	Spaceship #7	6	2	7	0	0	1
25	Tank #3	6	3	4	7	0	0
26	Helicopter	7	2	0	5	0	1
27	Tri-plane	4	2	2	5	0	1
28	Bulldozer	5	2	0	3	0	1
29	Spaceship #8	5	2	7	0	0	1
30	Frog	3	3	4	0	0	1
31	Rabbit	2	3	7	3	0	1
32	Ghost	2	2	6	0	0	1
33	Pac-men	2	2	5	0	0	1
34	Fly	3	2	7	0	0	1
35	Jet fighter #3	5	3	0	5	0	1
36	Crocodile	6	3	4	1	0	1
37	Hovercraft	5	3	1	6	0	1
38	Submarine	8	2	5	1	0	1
39	Tank destroyer	6	3	0	4	0	1
40	Jet fighter #4	5	2	1	7	0	1
41	Space buggy	5	3	0	6	0	1
42	Cannon	5	3	4	0	0	1
43	Soldier	3	6	0	4	0	1
44	Diamond	3	3	7	1	0	1
45	Sword	6	2	6	0	0	1
46	Truck	8	2	5	0	0	1
47	Lunar lander	3	3	6	0	0	1
48	Jet fighter #5	5	2	0	4	0	1
49	Teddy	4	5	6	0	0	1
50	Ball	3	3	0	5	0	1

These sprites can be loaded by the loader program or by hand. To load these sprites in by hand type:

```
CLEAR 51512: LOAD"SPRITE2A"CODE 51513: .POKE 62464, 51513.
```

APPENDIX 3 'SPRITE1B' AND 'SPRITE2B' SPRITES

The sprites that were used in the Laser BASIC demo, available to be used in your Laser BASIC Programs or the Sprite Generator Program.

SPRITE	DESCRIPTION	LEN	HGT	INK	PAPER	FLASH	BRIGHT
1	Tortoise	4	2	5	0	0	0
2	Mouse	4	2	7	0	0	1
3	Hare	4	2	6	0	0	1
4	Flower	2	4	4	0	0	1
5	Car	4	2	3	0	0	0
6	Road section	3	2	—	—	—	—
7	Bouncing man #1	5	4	6	0	0	1
8	Bouncing man #2	5	4	6	0	0	1
9	Bouncing man #3	5	4	6	0	0	1
10	Bouncing man #4	5	4	6	0	0	1
11	Bouncing man #5	2	4	6	0	0	1
12	Girder section	2	2	4	0	0	0
13	Ground	15	1	4	0	0	1
14	Invader	3	3	0	6	0	1
15	Landscape #1	15	3	6	0	0	1
16	Landscape #2	15	3	6	0	0	1
17	Landscape #3	2	3	6	0	0	1
18	Spaceship	3	3	7	1	0	1
19	Planet with ring	3	3	5	0	0	1
20	Quill	3	3	7	0	0	1
21	Top of space vehicle	8	4	5	0	0	1
22	Lantern	4	3	—	—	—	—
23	Spiders web	4	3	7	0	0	1
24	Planet with moon	3	3	4	0	0	1
25	Clock	3	3	6	0	0	1
26	Track of space vehicle	8	2	5	0	0	1
27	Turned track of space vehicle	8	2	5	0	0	1
28	Spinner cap	3	3	5	0	0	1
29	Bell	3	3	7	0	0	1
30	Screw jack	3	3	4	0	0	1
31	Lever	3	3	4	0	0	1
32	Chess piece	2	4	0	7	0	1
33	Oasis logo	12	4	—	—	—	—
34	Top of train	11	2	5	0	0	1
35	Train wheels #1	11	1	7	0	0	0
36	Train wheels #2	11	1	7	0	0	0
37	Train wheels #3	11	1	7	0	0	0
38	Train wheels #4	11 1	7 0	0	0		
39	Dot	1	1	7	0	0	0
40	Radar dish #1	1	2	5	0	0	1
41	Radar dish #2	1	2	5	0	0	1
42	Radar dish #3	1	2	5	0	0	1
43	Radar dish #4	1	2	5	0	0	1
44	Radar dish #5	1	2	5	0	0	1
45	Radar dish #6	1	2	5	0	0	1
46	Radar dish #7	1	2	5	0	0	1
47	Radar dish #8	1	2	5	0	0	1
48	Top of coach	10	2	—	—	—	—
49	Wheels of coach #1	10	1	7	0	0	0
50	Wheels of coach #2	10	1	7	0	0	0

51	Track	8	1	7	0	0	1
52	Rotating face 0 degrees	2	2	4	0	0	1
53	Rotating face 90 degrees	2	2	4	0	0	1
54	Rotating face 180 degrees	2	2	4	0	0	1
55	Rotating face 270 degrees	2	2	4	0	0	1
56	Helicopter #1	3	3	5	0	0	1
57	Helicopter #2	3	3	5	0	0	1
58	Key #1	3	3	7	0	0	1
59	Key #2	3	3	7	0	0	1

These sprites can be loaded using the loader program or by hand. To load these sprites by hand type:

```
        CLEAR 50303: LOAD"SPRITE2B"CODE 50304: .POKE 62464, 50304.
```

APPENDIX 4 THE LASER BASIC DEMO EXPLAINED

This appendix is a brief outline of the Laser BASIC Demo. It's aim is to help users to familiarise themselves with the operation of some of the commands.

The Demo was written as a series of subroutines which run independantly of each other, so they can be simply executed by typing GOSUB (line number) without having to run the whole program.

THE INTERMEDIATE OASIS LOGO SCREEN

Execution	variables to be set on entry.
GOSUB 1000	none.

```
1000  INK 4: PAPER 0: BRIGHT 1:
BORDER 0: CLS
1001.COL=9:.ROW=6:.SPN=60:.ATON:
.PTBL:
1002 PRINT AT 13,11; INK 5;"BASI
C DEMO";AT 11,10; INK 5;"    LAS
ER   "
1003 PLOT 73,90: DRAW 108,0: DRA
W 0,-14: DRAW -108,0: DRAW 0,14
1004 PLOT 73,74: DRAW 108,0: DRA
W 0,-14: DRAW -108,0: DRAW 0,14
1005.NPX=1:.COL=10:.HGT=1:.LEN=1
2
1006 FOR N=0 TO 192:.ROW=11:.WL1
V:.ROW=13:.WCRV: NEXT N: PAUSE 5
0: RETURN
```

In this screen the OASIS logo sprite is placed on the screen. Text is placed under it with boxes drawn around the text using the Sinclair DRAW commands.

Line 1005 defines a window which will fit over the text but inside the boxes.
Line 1006 scrolls the top text horizontally and the bottom text vertically
246 times by 1 pixel.

THE SPRITES MOVING THROUGH THE PILLARS SCREEN

Execution	variables to be set on entry.
GOSUB 770	none.

```
 770.ATOF:  INK 4: PAPER 0: BRIG
HT 1: BORDER 0:.HGT=24:.LEN=32:.
ROW=0:.COL=0: CLS :.SETV
 771  INK 5: PAPER 5:.LEN=3:.HGT
=22:.ROW=0: FOR N=8 TO 26 STEP 8
:.COL=N:.SETV: NEXT N
 772 INK 1: PAPER 1:.LEN=2:.HGT=
22:.ROW=0: FOR N=10 TO 29 STEP 8
:.COL=N:.SETV: NEXT N
 774  INK 4: PAPER 1:.LEN=1:.HGT
=12:.ROW=8: FOR N=6 TO 26 STEP 8
:.COL=N:.SETV: NEXT N
 775 INK 4: PAPER 5:.LEN=1:.HGT=
12:.ROW=8: FOR N=5 TO 26 STEP 8:
.COL=N:.SETV: NEXT N
 776.SPN=33:.COL=0:.ROW=15:.PTBL
: PAUSE 50:.HGT=6:.LEN=32: FOR N
=1 TO 300:.SR1V: NEXT N
 780.SP1=14:.SP2=14:.LEN=1:.HGT=
0: FOR N=1 TO 20::.ROW=INT (RND*
10)+8:.COL=-4: FOR X=-4 TO 32:.M
OVE: NEXT X: NEXT N:.ATON: RETUR
N
```

In this screen, the invader sprites and the Oasis logo appear to move in front of, and behind blue and cyan pillars. The way this is achieved, is in fact, quite simple. The pillars that the data moves in front of, are in fact, just columns of attributes with INK set to green. You cannot see the green INK until some pixel data (the sprites or logo) pass over it. In order to give the impression of the sprites moving behind the larger pillars, the INK colours of these pillars are set to the same colour as the PAPER colour. Data still passes over the attributes as with the small pillars, but you cannot see it as there is no difference between the INK and PAPER colours.

The attribute flag must be set to off (.ATOF) for the above method to work (See line 770).

Lines 771	and 772 create the foreground pillars.
Lines 774	and 775 create the background pillars.
Line 776	places sprite 33 (the Oasis logo) on the screen and a window, the length of the screen, is defined around it. The data is then simply scrolled by 1 pixel 300 times without wrap.
Line 780	moves sprite 14 across the screen from left to right using .MOVE.

THE FIRST JUMPING MAN SCREEN

Execution	variables to be set on entry.
GOSUB 160	none.

```
 160.ATON: INK 5: PAPER 0: BORDE
R 0: CLS :.SET=4:.HGT=3:.LEN=3:.
COL=12:.ROW=11:.SPN=25:.PTBL:.SE
T=5:.COL=7:.ROW=21:.SPN=30:.PTBL
:.COL=4:.PTBL:.COL=1:.PTBL:.HGT=
3:.LEN=9:.SET=6:.ATOF:.SPN=19:.C
OL=0:.ROW=3:.PTBL:.ATON:.LEN=32:
.HGT=3:.SET=1
 161.COL=13:.ROW=20:.SPN=60:.PTB
L:.COL=28:.ROW=20:.SPN=23:.PTBL:
.ROW=6: FOR N=2 TO 32 STEP 8:.CO
L=N:.SPN=22:.PTBL: NEXT N
 162.ROW=18:.SPN=12: FOR N=-30 T
O 32 STEP 2:.COL=N:.PTBL: NEXT N
 163.ROW=14: FOR I=-30 TO 40 STE
P 2:.COL=I
 164.SPN=7:.PTBL: GO SUB 165:.SP
N=8:.PTBL: GO SUB 165:.SPN=9:.PT
BL: GO SUB 165:.SPN=10:.PTBL: GO
 SUB 165: NEXT I: RETURN
 165.SET=4:.MIRV:.SET=5:.MIRV:.S
ET=6:.WL1V::.SET=1: PAUSE 2: RET
URN
```

In this screen a little man hops across the screen from left to right, while a planet scrolls above. A clock and some objects that look like fly presses are also animated.

Lines 160	to 162 place all the scenario data on the screen.
Line 163	is a loop to set the value of COL in the range -30 to 40. A sequence of 4 sprites (7,8,9 and 10) are used to animate the little man.
Line 165	mirrors the clock and the sprites in the bottom left of the screen, as well as scrolling the planet and taking a PAUSE of 2 to slow the movement down.

Once the sequence of 4 sprites has been placed the COL value is incremented by 2. The trailing blank edge of the sprite removes any data left by the old sprite.

THE ARRAY OF INVADERS ON THE CYAN BACKGROUND

Execution	variables to be set on entry.
GOSUB 680	none.

```
 68Ø BRIGHT Ø:  INK Ø: PAPER 5:
BORDER 5:.HGT=24:.LEN=32:.ROW=Ø:
.COL=Ø: CLS :.SETV:.ATOF:.COL=8:
.ROW=15:.SPN=33:.PTBL
 681.ATOF:.SPN=15:.ROW=21:.COL=Ø
:.PTBL:.COL=15:.SPN=16:.PTBL:.CO
L=3Ø:.SPN=17:.PTBL:.SPN=14:.ROW=
Ø: GO SUB 72Ø:.ROW=3: GO SUB 72Ø
:.ROW=6: GO SUB 72Ø:.ROW=9: GO S
UB 72Ø
 682.ROW=Ø:.COL=Ø:.HGT=12:.LEN=3
2: FOR N=1 TO 16Ø:.WR1V: NEXT N
 683.ROW=Ø:.COL=2Ø:.HGT=15:.LEN=
12:.NPX=-1: FOR N=1 TO 24:.WCRV:
 NEXT N
 684.ROW=3:.COL=Ø:.HGT=12:.LEN=3
2: FOR N=1 TO 88:.WL1V: NEXT N
 69Ø .SP1=14:.SP2=14:.HGT=1
 691 FOR M=1 TO 16
 695 LET X=INT (RND*4): LET Y=IN
T (RND*4):.COL=(X*3)+9:.ROW=(Y*3
)+3
 696 LET S=?SCV: IF S=Ø THEN  GO
 TO 695
 698.LEN=INT (RND*5)-2: FOR N=1
TO 3Ø:.MOVE: PAUSE 1: NEXT N: NE
XT M:.ATON: BRIGHT 1: RETURN
 72Ø.COL=Ø:.PTBL:.COL=3:.PTBL:.C
OL=6:.PTBL:.COL=9:.PTBL: RETURN
```

In this screen an array of 16 invaders scroll across the screen and then break off one at a time.

Line 681,	which uses line 720 as a subroutine, places the 16 invaders (sprite 15) on the screen.
Lines 682,	683 and 684 scroll the array of invaders by creating a window around them and then scrolling them by 1 pixel horizontally, vertically and then horizontally again.
Lines 690	to 698 scan the area of screen now occupied by the invaders for data using ?SCV (line 696). If any data (the invader) exists there, then the sprite is moved off screen using .MOVE. This process is repeated until all the sprites have been removed.

THE CIRCLE OF COLOURED SQUARES

Execution	variables to be set on entry.
GOSUB 670	none.

```
 67Ø.HGT=4:.LEN=4: GO SUB 1ØØ: F
OR M=1 TO 3: FOR N=1 TO 4Ø:.COL=
14-1Ø*COS (N/2Ø*PI):.ROW=1Ø+1Ø*S
IN (N/2Ø*PI)
 671 INK Ø:  PAPER INT (RND*7)+1
:.SETV: NEXT N: NEXT M: RETURN
```

In this screen a subroutine starting at line 100 is called which puts some text and the Oasis logo on the screen.

```
 100.ATOF: INK 7: PAPER 0: BORDE
R 0: BRIGHT 1: CLS
 101 PRINT AT 4,8;"THE LASER BAS
IC";AT 6,4;"EXTENDED INTERPRETER
 FROM"
 102.COL=9:.ROW=11:.SPN=33:.PTBL
:.ATON: PAUSE 100: RETURN
```

Line 670	is a simple circle calculating routine.
Line 671	sets windows of 3 by 3 characters with a random PAPER colour and a constant INK colour using .SETV.

FALLING INVADERS OVER A STAR FILLED BACK DROP

Execution	variables to be set on entry.
GOSUB 380 none.	

```
 380.ATOF: INK 7: BORDER 1: PAPE
R 1: BRIGHT 1: CLS
 390 FOR N=1 TO 300: PLOT INT (R
ND*255),INT (RND*175): NEXT N:.C
OL=0:.ROW=0:.SPN=33:.PTBL:.COL=2
0:.ROW=5:.SPN=19:.PTBL:.ROW=20:.
COL=0:.SPN=16:.PTBL:.SPN=17:.COL
=15:.PTBL:.SPN=15:.COL=17:.PTBL
 399 FOR V=1 TO 20
 400.SPN=14:.ROW=-3:.COL=INT (RN
D*5)+13:.HGT=1:.LEN=INT (RND*3)-
1:.SP1=14:.SP2=14
 410 FOR N=-4 TO 24:.ROW=N:.MOVE
: PAUSE 1: NEXT N
 420 NEXT V:.ATON: RETURN
```

In this screen .MOVE is used to non-destructively move sprite 14 (the invader) down from a position above the top of the screen over data below.

Line 390	plots 300 pixels on the screen. Places the ringed planet (sprite 19), the Oasis logo (sprite 33), and the landscape sprites (sprites 15,16 and 17) on the screen
Lines 399,	400, 410 and 420 move twenty invaders down the screen from a random COL position. A random value is set in LEN (-1 to 1). .MOVE is used to move the sprites.

THE SECOND JUMPING MAN SCREEN (THE MOVING FLOOR)

Execution	variables to be set on entry.
GOSUB 210	none.

```
 210 BORDER 0: INK 4: PAPER 0: B
RIGHT 1: CLS :.COL=9:.ROW=0:.SPN
=60:.PTBL
 211.ROW=5:.SPN=22: FOR N=2 TO 3
2 STEP 8:.COL=N:.PTBL: NEXT N
 212.SPN=24:.ROW=8:.COL=3:.PTBL:
.COL=23:.PTBL:.ROW=11:.COL=9:.PT
BL:.COL=29:.PTBL:.SET=5:.HGT=3:.
LEN=32:.COL=0:.SET=1
 220.ROW=20:.SPN=12: FOR N=0 TO
20 STEP 2:.COL=N:.PTBL: NEXT N:.
COL=30:.PTBL:.COL=28:.PTBL
 230.SET=1:.COL=-4:.ROW=16
 231.SET=2:.COL=20:.ROW=20:.LEN=
8:.HGT=2
 240 FOR I=-28 TO 32 STEP 2:.SET
=1:.COL=I:.SPN=7:.PTBL:.SET=2:.W
R4V: GO SUB 244
 241.SPN=8:.PTBL:.SET=2:.WR4V: G
O SUB 244
 242.SPN=9:.PTBL:.SET=2:.WR4V: G
O SUB 244
 243.SPN=10:.PTBL:.SET=2:.WR4V:
GO SUB 244: NEXT I: RETURN
 244.SET=5:.ROW=8:.WL1V:.ROW=11:
.WR1V:.SET=1: PAUSE 4: RETURN
```

In this screen the jumping man moves on screen and syncronises his jump with a part of the floor which is moving.

Lines 211,	212 and 220 place all the data on the screen.
Line 230	sets up SET 1 for the man.
Line 231	sets up SET 2 for moving the floor.
Lines 240	to 243 move the man, scrolling the planets and the floor after each movement.

The little man is animated using four sprites (7,8,9 and 10), which, when sequentially put, move the data 2 characters to the right, hence the need for the STEP2 in the controlling FOR-NEXT loop (line 240).

THE RAT RACE SCREEN

Execution	variables to be set on entry.
GOSUB 350	none.

```
 350 INK 7: PAPER 0: BRIGHT 1: B
ORDER 0: CLS :.ATOF
 360 LET P1=5: LET P2=2: LET L=1
6: LET H=3: LET C=8: LET R=1: GO
 SUB 2000: PRINT AT 2,10;"THE RA
T RACE"
 365.SPN=2:.LEN=32:.COL=0:.HGT=2
: INK 0: FOR N=1 TO 7:.ROW=5+(N*
2): PAPER N:.SETV:.PTBL: NEXT N
 370 FOR N=1 TO 1000: LET R=INT
(RND*7)+1:.ROW=5+(R*2):.WR1V: NE
XT N:.ATON: RETURN
```

In this screen the 7 colours from Blue to White are used to form 7 tracks 32 characters long by 2 high. Seven rats, 1 in each track, are placed on the screen.

Line 376	picks one of the tracks at random and scrolls it by 1 pixel to the right (.WR1V). This is repeated 1000 times giving a race with a random outcome.

The window and shadow that contains the text 'THE RAT RACE' is created in a subroutine at line 2000 (see later).

THE THIRD JUMPING MAN SCREEN (THROUGH THE PILLARS)

Execution	variables to be set on entry.
GOSUB 450	none.

```
 450 PAPER 0: INK 6: BRIGHT 1: B
ORDER 0: CLS
 451 INK 6: PAPER 5:.LEN=3:.HGT=
24:.ROW=0:.COL=4:.SETV:.COL=14:.
SETV:.COL=24:.SETV
 453 INK 1: PAPER 1:.LEN=2:.HGT=
24:.ROW=0:.COL=8:.SETV:.COL=18:.
SETV:.COL=28:.SETV
 460.SPN=12:.ROW=20: FOR N=0 TO
6 STEP 2:.COL=N:.PTBL:.COL=N+10:
.PTBL:.COL=N+20:.PTBL:.COL=N+30:
.PTBL: NEXT N
 461 INK 5: PAPER 5:.LEN=1:.HGT=
24:.ROW=0:.COL=7:.SETV:.COL=17:.
SETV:.COL=27:.SETV
 462.LEN=2:.HGT=5:.ROW=15: INK 6
: PAPER 1:.COL=5:.SETV:.COL=15:.
SETV:.COL=25:.SETV
 470.ATOF:.ROW=16: FOR I=-4 TO 3
2 STEP 2:.COL=I
 480 .SPN=7:.PTBL: PAUSE 8:.SPN=
8:.PTBL: PAUSE 8:.SPN=9:.PTBL: P
AUSE 8:.SPN=10:.PTBL: PAUSE 8: N
EXT I:.ATON: RETURN
```

In this screen the same technique as in 'The Sprites moving through the pillars screen', was used to build up purely attribute based pillars. Their INK and PAPER colours are set such that data would appear to go either behind or in front of them.

Lines 470 to 480 sets the attribute switch off (.ATOF). The sprites 7,8,9 and 10 are animated in the previously described manner across the screen from a COL value of -4 to a value of 32 in steps of 2.

(Note the PAUSE 8 after each movement in line 480)

THE ANIMATION SCREEN

Execution	variables to be set on entry
GOSUB 870	none.

```
 870.ATON: BORDER Ø:  INK 7: PAP
ER Ø: BRIGHT 1: CLS :.COL=18:.RO
W=20:.SPN=60:.PTBL:.ATOF: LET C=
20: LET R=Ø: LET L=11: LET H=3:
LET P1=6: LET P2=4: GO SUB 2000:
 PRINT AT 1,21;"ANIMATION"
 871.LEN=3:.HGT=3:.ROW=5:.COL=12
: GO SUB 860:.ROW=8:.COL=8: GO S
UB 860:.COL=16: GO SUB 860:.ROW=
12:.COL=5: GO SUB 860:.COL=19: G
O SUB 860:.ROW=16:.COL=8: GO SUB
 860:.COL=16: GO SUB 860:.ROW=19
:.COL=12: GO SUB 860
 872 FOR N=1 TO 40:.SPN=58: GO S
UB 850: PAUSE 3:.SPN=59: GO SUB
850:.SPN=58: PAUSE 3: NEXT N:.AT
ON: RETURN
```

The subroutine at line 2000 is used to draw the window and shadow in which the text "ANIMATION" is printed.

Line 871	fills windows, the size of the sprites (3x3), with random INK. Using a subroutine at line 860.

```
860 PAPER Ø: INK INT (RND*6)+2:
.SETV: RETURN
```

Line 872	places sprite 58 on the screen and then, after a pause, sprite 59 is placed, (both use the subroutine at line 850). After a second pause, line 872 is repeated giving two frame animation.

```
 850.ROW=5:.COL=12:.PTBL:.ROW=8:
.COL=8:.PTBL:.COL=16:.PTBL:.ROW=
12:.COL=5:.PTBL:.COL=19:.PTBL:.R
OW=16:.COL=8:.PTBL:.COL=16:.PTBL
:.ROW=19:.COL=12:.PTBL: RETURN
```

THE TORTOISE AND THE HARE SCREEN

Execution	variables to be set on entry.
GOSUB 600	none

```
 600 INK 6: PAPER Ø: BORDER Ø: C
LS :.SPN=60:.ROW=15:.COL=1Ø:.PTB
L
 601 LET C=3: LET R=19: LET H=3:
 LET L=28: LET P1=7: LET P2=5: G
O SUB 2000: PRINT AT 20,4;"THE T
ORTOISE AND THE HARE"
 605.SPN=39: FOR N=Ø TO 31:.COL=
N:.ROW=1: GO SUB 630:.ROW=6: GO
SUB 630:.ROW=7: GO SUB 630:.ROW=
12: GO SUB 630: NEXT N
 606.LEN=32:.COL=Ø:.ROW=3:.HGT=2
: INK 5:.SETV
 607.SPN=1:.PTBL:.SPN=3:.ROW=9:.
PTBL
 610.LEN=32:.HGT=2:.COL=Ø: FOR M
=1 TO 256:.ROW=9:.SR1V: GO SUB 6
27:.SR8V: NEXT M
 611 RETURN
 627 IF M<180 THEN .ROW=13: RETU
RN
 628.ROW=3: RETURN


 630 INK INT (RND*7)+1: PAPER 0:
.SETM:.PTBL: RETURN
```

In this screen the two race tracks for the tortoise and the hare are outlined by sprite 39 which is put on the screen in the Subroutine at line 630. Since the characters are to be scrolled, the attributes in their respective screen windows need to be set before the scrolling can begin.

The hare is scrolled by 1 pixel (.SR1V) and the tortoise is scrolled by 8 pixels (.SR8V).

THE RANDOM COLOUR FLOWERS SCREEN

Execution	variables to be set on entry.
GOSUB 550	none

```
 550.ATON:.SPN=4:.HGT=1:.LEN=2:
INK 0: PAPER 0: BORDER 0: CLS
 555 FOR N=0 TO 200:.COL=INT (RN
D*30):.ROW=INT (RND*22)
 560.PTBL: INK INT (RND*7)+1:.SE
TV: NEXT N: PAUSE 50: RETURN
```

In this screen sprite 4 is "PUT" on the screen at random COL and ROW positions. A random INK colour is set and the flower's petals coloured using .SETV.

THE FOURTH JUMPING MAN SCREEN (RIDE ON THE MOVING PLATFORM)

Execution	variable to be set on entry.
GOSUB 510	none.

```
 51Ø INK 6: PAPER Ø: BORDER Ø: B
RIGHT 1: CLS :.SPN=6Ø:.ROW=16:.C
OL=7:.PTBL
 511.LEN=2:.HGT=24:.ROW=Ø:.COL=Ø
: INK 1: PAPER 1:.SETV:.COL=3Ø:.
SETV
 512.LEN=3:.COL=2: INK 5: PAPER
5:.SETV:.COL=27:.SETV
 513.LEN=1:.COL=5: INK 6:.SETV:.
COL=26:.SETV
 514.LEN=2:.HGT=4:.ROW=1:.COL=3:
 INK 6: PAPER 1:.SETV:.COL=27:.S
ETV
 515.ROW=9:.COL=3:.SETV:.COL=27:
.SETV
 516.ROW=17:.COL=3:.SETV:.COL=27
:.SETV
 517.HGT=3:.ROW=5:.COL=3: INK 4:
.SETV:.COL=27:.SETV
 518.ROW=13:.COL=3:.SETV:.COL=27
:.SETV
 519.ROW=21:.COL=3:.SETV:.COL=27
:.SETV
 52Ø.HGT=3:.LEN=1:.ROW=5:.COL=5:
 INK 4: PAPER 5:.SETV:.COL=26:.S
ETV
 521.ROW=13:.COL=5:.SETV:.COL=26
:.SETV
 522.ROW=21:.COL=5:.SETV:.COL=26
:.SETV
 523.LEN=2Ø:.HGT=3:.ROW=5:.COL=6
: INK 4: PAPER Ø:.SETV:.ROW=13:.
SETV:.ROW=21:.SETV
 524 .SPN=12:.ATOF:.ROW=5:.COL=3
:.PTBL:.COL=16:.PTBL:.ROW=13:.PT
BL:.COL=3:.PTBL:.ROW=21:.PTBL:.C
OL=16:.PTBL:.SPN=31:.COL=15:.ROW
=1:.PTBL:.SPN=25:.COL=21:.ROW=18
:.PTBL
 53Ø.LEN=26:.HGT=3:.ROW=9: FOR I
=-36 TO 32 STEP 2:.COL=I
 531.SPN=7:.PTBL: GO SUB 54Ø:.SP
N=8:.PTBL: GO SUB 54Ø:.SPN=9:.PT
BL: GO SUB 54Ø:.SPN=1Ø:.PTBL: GO
 SUB 54Ø: NEXT I
 532.ATON: RETURN
```

In this highly animated screen, the little man is seen to hop in time with a moving platform as it scrolls across the screen.

Lines 511 to 522 create the pillars from which the man and platforms appear to emerge. The pillars are created from attributes with no pixel data present. The technique is described in a previous section.

Line 523 sets attributes for the parts of the screen that are going to be occupied by the platforms.
Line 524 places the various sprites on the screen. The animation sequence, used in previous example, is used to animate the man, calling the subroutine at line 540 after each sprite has been placed.

```
 540.COL=15:.ROW=1:.LEN=3:.MIRV:
.ROW=18:.COL=21:.MIRV:.LEN=26:.R
OW=5:.COL=3:.WR1V:.ROW=21:.WR8V:
.ROW=13:.WR4V:.ROW=9:.COL=I: RET
URN
```

This subroutine produces all the other animation that is seen on the screen by mirroring two sprites and scrolling the 3 platforms by 1,4 and 8 pixels with wrap around.

THE DIFFERENT COLOURED COLUMNS MOVING ACROSS THE SCREEN

Execution	variables to be set on entry.
GOSUB 250	none.

```
 250 GO SUB 100:.ROW=0:.COL=0:.L
EN=2:.HGT=8
 260 INK 0: FOR M=1 TO 500:.INVV
 270 PAPER INT (RND*7)+1:.SETV
 280  LET X=?COL: LET Y=?LEN: IF
 X+Y>=32 THEN  LET X=-1:.COL=0
 290.COL=X+1
 300  LET X=?ROW: LET Y=?HGT: IF
 X+Y>=25 THEN .ROW=1: LET X=-1
 310.ROW=X+1
 320 NEXT M: RETURN
```

This screen starts by inverting a window of length 2 and height 8, using the .INVV command. It's attributes are then set with a random paper colour using .SETV.

Lines 280 to 320 calculate the ROW and COL position of the window.

THE TRAIN SCREEN

Execution	variables to be set on entry.
GOSUB 130	none.

```
 130 INK 6: PAPER 0: BORDER 0: C
LS :.COL=9:.ROW=20:.SPN=60:.PTBL
 131 LET P1=7: LET P2=6: LET R=4
: LET C=2: LET H=3: LET L=28: GO
 SUB 2000: PRINT AT 5,3;"UP TO 2
55 SOFTWARE SPRITES"
 132.COL=0:.ROW=13:.SPN=48:.PTBL
:.COL=10:.PTBL:.COL=20:.SPN=34:.
PTBL
 133.ROW=15:.SPN=49:.COL=0:.PTBL
:.COL=10:.PTBL:.COL=20:.SPN=35:.
PTBL
 134.ROW=16:.SPN=51: FOR N=0 TO
32 STEP 8:.COL=N:.PTBL: NEXT N
 135.SET=1:.COL=0:.ROW=16:.HGT=1
:.LEN=32
 136.SET=2:.COL=20:.ROW=15
 137.SET=3:.ROW=15:.COL=0
 139 FOR I=1 TO 200
 140.SET=2:.SPN=35:.PTBL:.SET=3:
.SPN=49: GO SUB 150
 141.SET=2:.SPN=36:.PTBL:.SET=3:
.SPN=50: GO SUB 150
 142.SET=2:.SPN=37:.PTBL:.SET=3:
.SPN=49: GO SUB 150
 143.SET=2:.SPN=38:.PTBL:.SET=3:
.SPN=50: GO SUB 150
 144 NEXT I: RETURN
```

In this screen, 4 sprites (35 to 38) are successively placed to animate the wheels of the locomotive. The coach is animated using 2 sprites that make up the wheels. These are sprites 49 and 50. The track is scrolled with wrap around.

Lines 131	to 134 place the sprites on the screen.
Lines 139	to 143 call up a subroutine at line 150 which animates the wheels of the coach.

```
150.COL=0: .PTBL:.COL=10:.PTBL:
.SET=1:.WL1V: RETURN
```

THE HORIZONTAL SCROLLING DEMO

Execution	variables to be set on entry.
GOSUB 330	None

```
 330.ATOF: INK 1: PAPER 1: BORDE
R 1: BRIGHT 0: CLS
 332.LEN=32:.HGT=16:.COL=0:.ROW=
5: INK 0: PAPER 6:.SETV
 333 LET P1=0: LET P2=2: LET L=2
2: LET R=2: LET H=3: LET C=5: GO
 SUB 2000: PRINT AT 3,6;"HORIZON
TAL SCROLLING"
 334.SPN=15:.COL=0:.ROW=7:.PTBL:
.SPN=16:.COL=15:.PTBL:.SPN=17:.C
OL=30:.PTBL
 335 PLOT 0,20: DRAW 255,0:.COL=
0:.ROW=16:.SPN=18:.PTBL:.ROW=10:
.COL=1:.PTBL:.COL=4:.ROW=13:.PTB
L
 340.COL=0:.LEN=32
 341 FOR N=1 TO 500:.HGT=3:.ROW=
7:.WL1V:.HGT=3:.ROW=10:.WR4V:.RO
W=13:.WR1V:.ROW=16:.WR8V: NEXT N
:.ATON: RETURN
```

In this screen the landscape is seen to scroll left by 1 pixel, with wrap while the 3 spaceships are seen to scroll in the opposite direction by 4,1 and 8 pixels respectively. The subroutine at line 2000 is used to define a window and it's shadow, in which the text, 'HORIZONTAL SCROLLING' is printed.

Lines 344	and 335 PUT all the scenario data on the screen. The landscape is built up of 3 sprites, numbers 15,16 and 17, which are 15,15 and 2 characters long.
Line 341	scrolls the landscape and the 3 spaceships in the window whose length is 32 columns.

THE FIFTH JUMPING MAN SCREEN (THE EIGHT PLANETS)

Execution	variables to be set on entry.
GOSUB 640	none.

```
 640 BORDER 0: INK 6: PAPER 0: C
LS :.HGT=24:.LEN=32:.ROW=0:.COL=
0:.SETV
 641.COL=10:.ROW=15:.SPN=60:.PTB
L:.ROW=5:.SPN=12: FOR N=0 TO 30
STEP 2::.COL=N:.PTBL: NEXT N
 642.ATOF:.SPN=19:.HGT=3:.LEN=3:
.ROW=8:.COL=1: INK 7:.SETV:.PTBL
:.COL=7:.ROW=14: INK 2:.SETV:.PT
BL:.COL=11:.ROW=9: INK 3:.SETV:.
PTBL
 643.COL=9:.ROW=21: INK 4:.SETV:
.PTBL:.COL=18:.ROW=11: INK 5:.SE
TV:.PTBL:.COL=29:.ROW=10: INK 6:
.SETV:.PTBL:.COL=24:.ROW=14: INK
 7:.SETV:.PTBL:.COL=21:.ROW=21:.
SETV:.PTBL
 645.ROW=1: FOR I=-30 TO 36 STEP
 2:.COL=I
 646.SPN=7:.PTBL: GO SUB 660:.SP
N=8:.PTBL: GO SUB 660:.SPN=9:.PT
BL: GO SUB 660:.SPN=10:.PTBL: GO
 SUB 660: NEXT I:.ATON: RETURN
```

In this screen the now familiar man moves across a platform at the top of the screen whilst 8 planets of various colours are being animated.

Lines 612	and 613 set attributes in 3 by 3 windows on the screen using .SETV. The planet sprite is now placed in these windows with the attribute switch set to off (.ATOF).
Line 660	is used as a subroutine which animates the sprites by mirroring the planet sprite data in the 3 by 3 windows.

```
 660.COL=1:.ROW=8:.MIRV:.COL=7:.
ROW=14:.MIRV:.COL=24:.MIRV:.COL=
11:.ROW=9:.MIRV:.COL=9:.ROW=21:.
MIRV:.COL=18:.ROW=11:.MIRV:.COL=
29:.ROW=10:.MIRV:.COL=21:.ROW=21
:.MIRV:.COL=I:.ROW=1: RETURN
```

THE BOUNCING PLANET SCREEN

Execution	variables to be set on entry.
GOSUB 110	none

```
 110 GO SUB 100: INK 0: LET DR=1
:.HGT=DR: LET DC=1:.LEN=DC:.SP1=
19:.SP2=19: LET R=0:.ROW=R: LET
C=0:.COL=C:.SPN=19:.PTXR
 115 FOR N=1 TO 1000:.MOVE
 116  LET C=?COL: LET R=?ROW
 117 IF C=29 OR C=0 THEN  LET DC
=DC*-1:.LEN=DC: GO SUB 120
 118 IF R=20 OR R=0 THEN  LET DR
=DR*-1:.HGT=DR: GO SUB 120
 119 NEXT N: RETURN
```

The original data on the screen is set up by a subroutine at line 100.

Lines 115 to 118 move the planet (sprite 19) about the screen using .MOVE. Every time the edge of the screen is hit its paper colour is changed by the subroutine at line 120.

```
 120 PAPER INT (RND*7)+1:.SETM:
RETURN
```

THE SMOOTH SCROLLING SCREEN

Execution	variable to be set on entry.
GOSUB 500	none.

```
 500 BORDER 7:  INK 1: PAPER 7:
BRIGHT 0: CLS :.ATOF:.ROW=16:.CO
L=1:.SPN=33:.PTBL
 501 LET P1=6: LET P2=0: LET L=1
1: LET H=5: LET R=7: LET C=4: GO
 SUB 2000: PRINT AT 8,5;"SMOOTH"
;AT 10,5;"SCROLLING"
 502.ATOF:.COL=25:.ROW=19:.SPN=2
0:.PTBL
 503.COL=25:.ROW=0:.LEN=3:.HGT=2
2:.NPX=1: FOR N=1 TO 134: PLOT 2
12,1:.WCRV: NEXT N
 504.COL=0:.ROW=2:.LEN=32:.HGT=3
: FOR N=1 TO 170: PLOT 212,136:.
WL1V: NEXT N:.ATON: PAUSE 50: RE
TURN
```

In this screen a quill appears to draw a blue line vertically and then horizontally.

This effect is achieved very simply. A window is defined around the quill sprite and that data is scrolled. A pixel is placed right at the edge of the window. After every scroll is executed, a pixel is plotted, drawing the line.

Line 503	is the vertical scroll (.WCRV).
Line 504	is the horizontal scroll (.WL1V).

THE ATTRIBUTE SCROLLING SCREEN

Execution	variable to be set on entry.
GOSUB 760	none.

```
 760 GO SUB 100: INK 0:.LEN=1:.H
GT=13:.ROW=3: FOR N=3 TO 29:.COL
=N: PAPER INT (RND*6)+2:.SETV: N
EXT N
 761 LET L=19: LET H=3: LET R=19
: LET C=7: LET P1=7: LET P2=4: G
O SUB 2000: PRINT AT 20,8;"ATTRI
BUTE SCROLLS":.LEN=27:.COL=3:.HG
T=1: FOR I=3 TO 29: FOR Y=3 TO 1
6:.ROW=Y-1
 762 FOR X=1 TO Y:.ATRV: NEXT X:
 NEXT Y: NEXT I: PAUSE 50: RETUR
N
```

The screen is set up using the subroutine at line 100.

Line 760	sets up 26 widows with INK 0 and a random PAPER colour. These windows are 1 character wide by 13 high.
Line 761	creates the window that contains the text "ATTRIBUTE SCROLLS", via the subroutine at line 2000.

The top line of attributes is now scrolled by 1 character to the right, the second line is scrolled by 2 characters, the third by 3 etc., until the bottom is reached. This process is repeated until the attributes have reformed in their original columns.

THE SPACE TANKS WITH ROTATING RADAR DISHES SCREEN

Execution	variable to be set on entry.
GOSUB 800	none.

```
 800 INK 4: PAPER 0: BORDER 0: C
LS
 801 .SPN=15:.ROW=6:.COL=0:.PTBL
:.COL=15:.SPN=16:.PTBL:.COL=30:.
SPN=17:.PTBL
 802.SPN=21:.ROW=9:.COL=13:.PTBL
:.COL=5:.PTBL:.SPN=27:.ROW=13:.C
OL=13:.PTBL:.SPN=26:.COL=5:.PTBL
 803.ATOF:.SPN=13:.COL=0:.ROW=15
:.PTBL:.COL=13:.PTBL:.COL=27:.PT
BL:.ATON: PAPER 4:.LEN=32:.HGT=6
:.ROW=16:.COL=0:.SETV
 805 PAPER 0:.HGT=3:.LEN=32:.ROW
=0:.COL=0: INK 5:.SETV:.ROW=3: I
NK 7:.SETV:.ATOF
 806.SPN=18:.ROW=0:.COL=6:.PTBL:
.COL=23:.PTBL:.ROW=3:.COL=15:.PT
BL
 807 LET L=18: LET H=3: LET P1=6
: LET P2=1: LET R=17: LET C=12:
GO SUB 2000: PRINT AT 18,13;"SPR
ITE ANIMATION"
 810 FOR M=1 TO 50: FOR N=40 TO
47:.SPN=N: GO SUB 820: NEXT N: N
EXT M:.ATON: RETURN
```

In this screen space vehicles are displayed with rotating radar dishes. To obtain the animation the 8 sprites are sequentially PUT, like frames from a cartoon. Sprites 40 to 47 are used.

Lines 801,	802,803 and 806 build up the screen scenario.
Line 807	uses the subroutine at line 2000 to set up the window and shadow that the text is printed in.
Line 810	animates the radar dishes by sequentially putting a sequence of 8 sprites.
Line 820	is used as a subroutine to scroll the spaceships overhead.

```
 820.COL=15:.ROW=9:.PTBL:.COL=7:
.PTBL:.HGT=6:.ROW=0:.COL=0:.LEN=
32:.WR8V: RETURN
```

THE ROTATING SCREEN

Execution	variables to be set on entry.
GOSUB 830	none.

```
 830 BORDER 0:  INK 6: PAPER 0:
BRIGHT 1: CLS :.COL=18:.ROW=20:.
SPN=60:.PTBL:.ATOF: LET C=21: LE
T R=0: LET L=10: LET H=3: LET P1
=7: LET P2=4: GO SUB 2000: PRINT
 AT 1,22;"ROTATION"
 840.LEN=2:.HGT=2:.ROW=5:.COL=12
: GO SUB 860:.ROW=8:.COL=8: GO S
UB 860:.COL=16: GO SUB 860:.ROW=
12:.COL=5: GO SUB 860:.COL=19: G
O SUB 860:.ROW=16:.COL=8: GO SUB
 860:.COL=16: GO SUB 860:.ROW=19
:.COL=12: GO SUB 860
 841 FOR N=1 TO 19:.SPN=52: GO S
UB 850: PAUSE 20-N:.SPN=53: GO S
UB 850: PAUSE 20-N:.SPN=54: GO S
UB 850: PAUSE 20-N:.SPN=55: GO S
UB 850: PAUSE 20-N: NEXT N
 842.ATON: RETURN
```

In this screen 8 faces are rotated. These sprites are in fact, rotated copies of sprite 52. Sprites 53,54 and 55 were created in the sprite generator program. The 4 sprites are sequentially "PUT" to achieve a rotation effect on the screen.

Line 840	sets up the attributes of the rotating characters. Using a subroutine at line 860.

```
 860 PAPER 0: INK INT (RND*6)+2:
.SETV: RETURN
```

Line 841	places the sprites on the screen with a decrementing pause after each PUT operation. The sprites are placed by calling a subroutine at line 850.

```
 850.ROW=5:.COL=12:.PTBL:.ROW=8:
.COL=8:.PTBL:.COL=16:.PTBL:.ROW=
12:.COL=5:.PTBL:.COL=19:.PTBL:.R
OW=16:.COL=8:.PTBL:.COL=16:.PTBL
:.ROW=19:.COL=12:.PTBL: RETURN
```

THE ATTRIBUTE TUNNEL SCREEN

Execution	variables to be set on entry.
GOSUB 570	none.

```
 570 GO SUB 100: INK 0: FOR M=1
TO 10
 573 BORDER INT (RND*7)+1:  FOR
N=16 TO 1 STEP -1
 575 .COL=(16-N):.ROW=(16-N):.HG
T=N+8:.LEN=(N*2): PAPER INT (RND
*7)+1:.SETV: NEXT N
 580 BORDER INT (RND*7)+1:  FOR
N=1 TO 16
 585 .COL=(16-N):.ROW=(16-N):.HG
T=N+8:.LEN=(N*2): PAPER INT (RND
*7)+1:.SETV: NEXT N
 590 NEXT M: RETURN
```

In this screen the data is placed on the screen from the subroutine at line 100, then using a simple routine, the dimensions and position of the window are changed and a random PAPER colour is set in it using .SETV.

THE SIXTH JUMPING MAN SCREEN (THE STAIRCASE)

Execution	variables to be set on entry.
GOSUB 730	none.

```
 730 INK 6: PAPER 0: BORDER 0: B
RIGHT 1: CLS :.COL=0:.ROW=16:.SP
N=60:.PTBL
 731.ROW=0:.COL=28:.SPN=23:.PTBL
:.ROW=4:.COL=20:.SPN=22:.PTBL:.R
OW=12:.COL=28:.PTBL:.SPN=25:.COL
=3:.ROW=12:.PTBL
 740.SPN=12:.COL=0:.ROW=4:.PTBL:
.COL=2:.PTBL: FOR N=4 TO 24 STEP
 2:.ROW=N:.COL=N:.PTBL:.COL=N+2:
.PTBL: NEXT N:.ROW=22: FOR N=0 T
O 32 STEP 2:.COL=N:.PTBL: NEXT N
 750.NPX=-1:.HGT=6:.LEN=3:.ROW=0
: FOR I=-10 TO 6 STEP 2:.COL=I:
 751.SPN=7:.PTBL: PAUSE 8:.SPN=8
:.PTBL: PAUSE 8:.SPN=9:.PTBL: PA
USE 8:.SPN=10:.PTBL: PAUSE 8: NE
XT I
 752 LET R=0: FOR I=8 TO 24 STEP
 2:.COL=I: LET R=R+2
 753 FOR Y=1 TO 16:.WCRV: NEXT Y
:.ROW=R:.SPN=7:.PTBL: PAUSE 8:.S
PN=8:.PTBL: PAUSE 8:.SPN=9:.PTBL
: PAUSE 8:.SPN=10:.PTBL: PAUSE 8
: NEXT I
 754.ROW=18: FOR I=26 TO 34 STEP
 2:.COL=I:
 755.SPN=7:.PTBL: PAUSE 8:.SPN=8
:.PTBL: PAUSE 8:.SPN=9:.PTBL: PA
USE 8:.SPN=10:.PTBL: PAUSE 8: NE
XT I:.ATON: RETURN
```

In this screen the now familiar Jumping Man is seen to jump down a staircase.

Lines 730	to 740 place all the scenario data on the screen.
Lines 750	to 751 animate the man until he reaches a COL value of 6.
Lines 752	and 753 sequentially PUT the man to give the impression of leaping. He is then scrolled down in a window defined around him onto the step below. The ROW and COL values for the new position of the man are set and the process repeated.
Lines 754	to 755 animate the man once he has reached the bottom of the staircase. He is moved 'off screen' from COL position 26.

THE HELICOPTER ANIMATION SCREEN

Execution	variables to be set on entry.
GOSUB 880	none.

```
 880.ATOF: INK 5: PAPER 0: BRIGH
T 1: BORDER 0: CLS :.HGT=24:.LEN
=32:.ROW=0:.COL=0:.SETV: FOR N=0
 TO 300: PLOT INT (RND*256),INT
(RND*143): NEXT N:.HGT=3:.LEN=3:
 INK 6:.COL=5:.ROW=10:.SPN=19:.P
TBL:.SETV:.COL=28:.ROW=7:.PTBL:
INK 4:.SETV
 881 LET L=11: LET R=0: LET H=3:
 LET C=9: LET P1=7: LET P2=3: GO
 SUB 2000: PRINT AT 1,10;"ANIMAT
ION": .ATOF:.ROW=21:.SPN=57: FOR
 N=1 TO 28 STEP 3:.COL=N:.PTBL:
NEXT N
 882 FOR C=1 TO 5
 883.ROW=22:.COL=1+INT (RND*9)*3
: LET S=?SCV
 884 IF S=0 THEN  GO TO 883
 885.ROW=21: FOR T=30 TO 1 STEP
-4:.SPN=56:.PTBL: PAUSE T:.SPN=5
7:.PTBL: PAUSE T: NEXT T
 886 FOR T=1 TO 10:.SPN=56:.PTBL
: PAUSE 3:.SPN=57:.PTBL: PAUSE 3
: NEXT T
 887.LEN=INT (RND*3)-1:.HGT=-1:
FOR T=1 TO 22:.SP1=57:.SP2=56:.M
OVE: PAUSE 3:.SP1=56:.SP2=57:.MO
VE: PAUSE 3: NEXT T: NEXT C:.ATO
N: RETURN
```

In this screen helicopters with rotating rotor blades are seen to take off and fly across a starry background.

Line 80	plots the stars on the screen and puts the planet sprite.

The text 'ANIMATION' is placed in a window created in the subroutine at line 2000. The helicopters are animated and moved over the stars using the .MOVE command. The sprite numbers of the two sprites are stored in SP1 and SP2, and swapped after each execution by .MOVE.

Line 883	picks the helicopter to be flown at random using ?SCV (to see if the helicopter has yet to be flown). The helicopter is animated over the stars using .MOVE.

THE .MOVE SCREEN

Execution	variables to be set on entry.
GOSUB 900	none.

```
 900 BRIGHT 0:  INK 0: PAPER 6:
BORDER 6:.HGT=24:.LEN=32:.ROW=0:
.COL=0: CLS :.SETV:.ATOF:.COL=18
:.ROW=20:.SPN=33:.PTBL: LET L=8:
 LET H=3: LET C=0: LET R=19: LET
 P1=5: LET P2=0: GO SUB 2000: PR
INT AT 20,1;".MOVE"
 901.ATOF:.SPN=14:.ROW=0: GO SUB
 720:.ROW=3: GO SUB 720:.ROW=6:
GO SUB 720:.ROW=9: GO SUB 720
 902.ROW=0:.COL=0:.HGT=12:.LEN=3
2: FOR N=1 TO 80:.WR1V: NEXT N
 905.ROW=0:.COL=10:.HGT=18:.LEN=
12:.NPX=-1: FOR N=1 TO 40:.WCRV:
 NEXT N
 906 FOR M=0 TO 15:.SET=M
 907 LET X=INT (RND*4): LET Y=IN
T (RND*4):.ROW=(Y*3)+5:.COL=(X*3
)+10
 908 LET S=?SCV: IF S=0 THEN  GO
 TO 907
 909.SP1=14:.SP2=14: GO SUB 920:
 FOR N=1 TO 30:.MOVE: NEXT N: NE
XT M
 910 FOR M=0 TO 15:.SET=M: LET H
=?HGT: LET H=H*-1:.HGT=H: LET L=
?LEN: LET L=L*-1:.LEN=L
 911 FOR N=1 TO 30:.MOVE: NEXT N
: NEXT M
 912.ROW=0:.COL=10:.HGT=18:.LEN=
12:.NPX=1: FOR N=1 TO 40:.WCRV:
NEXT N
 913.ROW=0:.COL=0:.HGT=12:.LEN=3
2: FOR N=1 TO 80:.WL1V: NEXT N
 919 RETURN
```

In this screen an array of invaders are scrolled into the middle of the screen and then fly off, one at a time in different directions. Once all the invaders have flown they reassemble the array and scroll off again. This screen not only demonstrates the full use of .MOVE, but also the use of .SET' to store and recall the positions of the sprites stored in the various variable sets.

Line 901	places the array of sprites on the screen by calling the subroutine at line 720 which was also used by the other invader array screen.
Lines 902	and 905 scroll the invaders, using .WR1V and .WCRV.

The invaders are then moved off at random after using ?SCV to find them on the screen. A random number between -2 and 2 is stored in both HGT and LEN. These values are checked to make sure they are not both zero which would mean the sprite would not move.

```
 920.LEN=INT (RND*5)-2:.HGT=INT
(RND*5)-2: LET L=?LEN: LET H=?HG
T: IF L=0 AND H=0 THEN  GO TO 92
0
 921 RETURN
```

Since a different variable set is used for each invader, the direction of the invaders can be reversed by negating HGT and LEN, reassembling the array.

Lines 912	to 913 scroll the array off the screen.

THE SCROLLING OASIS LOGO ACROSS THE ATTRIBUTES SCREEN

Execution	variables to be set on entry.
GOSUB 930	none.

```
 930.SET=2:.HGT=4:.LEN=32:.ROW=1
0:.COL=0:.SPN=33: PAPER 0: BORDE
R 0: INK 7: CLS :.ATOF:.PTBL:.SE
T=1: FOR M=1 TO 9
 931  FOR N=1 TO 16 STEP 2
 932.COL=N:.ROW=3+N/2:.HGT=(17-N
):.LEN=2*(17-N): PAPER INT (RND*
6):.SETV: GO SUB 940: NEXT N
 933  FOR N=15 TO 1 STEP -2
 934 .COL=N:.ROW=3+N/2:.HGT=(17-
N):.LEN=2*(17-N): PAPER INT (RND
*6):.SETV: GO SUB 940: NEXT N
 935 NEXT M: RETURN
```

In this screen a white Oasis logo is seen to scroll from left to right across the screen over flashing attributes.

Line 930	places the logo (sprite 33) on the screen.
Lines 931	to 934 create a window of decreasing and then increasing dimensions. The PAPER colour is set to a random value while the INK colour remains white. A subroutine at line 940 is called each time, to scroll the logo using .WR1V.

THE HORIZONTAL SCROLLING GRAPH SCREEN

Execution	variables to be set on entry.
GOSUB 950	none.

```
 95Ø INK 7: PAPER 1: BORDER 1: B
RIGHT 1: CLS :.ATOF:.ROW=3:.COL=
9:.SPN=33:.PTBL
 951 LET P1=7: LET P2=4: LET L=3
Ø: LET H=7: LET R=7: LET C=Ø: GO
 SUB 2ØØØ
 952 LET P1=6: LET P2=5: LET L=2
2: LET H=3: LET R=19: LET C=4: G
O SUB 2ØØØ: PRINT AT 2Ø,5;"HORIZ
ONTAL SCROLLING"
 955.LEN=28:.HGT=5:.ROW=8:.COL=1
: LET X=92
 956 FOR M=1 TO 5ØØ: LET I=INT (
RND*3)-1
 957 LET X=X+I
 958 IF X=112 THEN  LET X=111
 959 IF X=71 THEN  LET X=72
 96Ø  PLOT 8,X:.SR1V: NEXT M: RE
TURN
```

In this screen a graph appears to emerge from the left and scroll to the right where it disappears. This is achieved by plotting on the left of the scrolling window after every scroll operation.

Line 955	defines the window of length 28 and height 8. The original Y position for the pixel plotting is 92.
Lines 956	and 957 select a random number in the range -1 to 1 and this is added to the Y position for the plot to produce the random wavey line.
Lines 958	and 959 check that the pixel is never plotted outside the window that is being scrolled.
Line 960	plots the pixel and then scrolls the window using .SR1V. This sequence is repeated 500 times.

THE LAST JUMPING MAN SCREEN (THE CHESS BOARD)

Execution	variables to be set on execution.
GOSUB 970	none.

```
 970  INK 6: PAPER Ø: BRIGHT 1:
BORDER Ø: CLS :.SET=7:.LEN=32:.H
GT=4:.ROW=9:.COL=Ø:.SET=Ø:.ATOF
 971 FOR Y=Ø TO 16 STEP 4:  FOR
X=Ø TO 28 STEP 4
 972.HGT=2:.LEN=2:.ROW=Y:.COL=X:
 PAPER 4:.SETV:.COL=X+2:.ROW=Y+2
:.SETV: PAPER 1:.HGT=1:.COL=X:.R
OW=Y+2:.SETV:.COL=X+2:.ROW=Y+4:.
SETV: NEXT X: NEXT Y
 973.SET=2:.HGT=4:.LEN=32:.SPN=3
2:.COL=5:.ROW=Ø:.PTBL:.COL=Ø:.SE
T=3:.ROW=4:.COL=2Ø:.SPN=32:.PTBL
:.HGT=4:.LEN=32:.COL=Ø:.SET=4:.S
PN=32:.ROW=16:.COL=28:.PTBL:.HGT
=4:.LEN=32:.COL=Ø:.SET=Ø
 975.ROW=2Ø:.COL=Ø:.SPN=6Ø:.ATON
:.PTBL:.ATOF:.SET=Ø:.ROW=9: FOR
I=-33 TO 32 STEP 4:.COL=I
 976.SPN=7:.PTBL: GO SUB 98Ø:.SP
N=8:.PTBL: GO SUB 98Ø:.SPN=9:.PT
BL: GO SUB 98Ø:.SET=7:.SRBV: GO
SUB 98Ø:.SET=Ø:.SPN=1Ø:.COL=I+2:
.PTBL: GO SUB 98Ø:: NEXT I
 977.ATON: RETURN
```

In this screen the Jumping Man moves across the screen which is being patrolled by 3 scrolling chess pieces.

Lines 971	and 972 build the chequered pattern out of attributes, with their INK colours set to yellow and the paper colours set to CYAN, BLUE and BLACK.
Line 973	places the sprites on the screen and define the windows around them.
Lines 975	and 976 animate the little man calling a subroutine at line 980 which will scroll the 3 chess pieces by swapping between the variable sets.

```
 98Ø.SET=2:.WL1V:.SET=3:.WR1V:.S
ET=4:.WL1V:.SET=Ø: PAUSE 2: RETU
RN
```

THE TORTOISE EATING THE PLANTS SCREEN

Execution	variables to be set on entry.
GOSUB 990	none.

```
 990  INK 4: PAPER Ø: BORDER Ø:
CLS :.COL=9:.ROW=3:.SPN=6Ø:.PTBL
:.ATOF:
 991 PAPER 4:.LEN=32:.HGT=6:.ROW
=12:.COL=Ø:.SETV
 992.SPN=4:.ROW=9: FOR N=4 TO 31
 STEP 2:.COL=N:.PTBL: NEXT N
 993.ATON:.ROW=1Ø:.SPN=1:.COL=Ø:
.PTBL
 994 FOR N=4 TO 31 STEP 2:.ROW=9
:.HGT=3:.LEN=2:.COL=N:.NPX=-1: F
OR G=1 TO 24:.SCRV: NEXT G
 995 BRIGHT Ø: INK 5: PAPER Ø:.S
ETV:.COL=N-4:.LEN=6: FOR G=1 TO
16:.WR1V: NEXT G: NEXT N
 996 PRINT AT 8,28;"BURP": PAUSE
 5Ø: PRINT AT 8,28,"    ":.MIRV
 997.LEN=32:.ROW=9:.HGT=4:.COL=Ø
: FOR N=1 TO 256:.SL1V: NEXT N
 998.ATON: RETURN
```

In this screen the tortoise is seen to walk from left to right eating the plants as it goes along.

Line 992	places the plants (sprite 4) on the screen.
Line 993	places the tortoise (sprite 1) on the screen.

The plant directly in front of the tortoise is scrolled down. The attributes of the tortoise are set in front of it and the tortoise is scrolled forward on to the next plant. This process is repeated until the tortoise has reached the end of the screen where it is mirrored and then scrolled all the way back.

THE TEXT WINDOW SUBROUTINE

Execution	variables to set on entry.
GOSUB 2000	L = the length of the window. H = the height of the window. R = ROW position. C = COL position. P2 = the paper colour of the shadow. P1 = the paper colour of the window.

```
2ØØØ .LEN=L:.HGT=H:.ROW=R+1:.COL
=C+1: PAPER P2: INK 9:.SETV
2ØØ1 .ROW=R:.COL=C: PAPER P1:.SE
TV
2ØØ2 LET C=C*8: LET R=175-(R*8):
 LET H=H*8-1: LET L=L*8-1
2ØØ3 PLOT C,R: DRAW L,Ø: DRAW Ø,
-H: DRAW -L,Ø: DRAW Ø,H
2ØØ4 RETURN
```

This subroutine is used to create the coloured windows in which the titles of the various screens are placed. The window is in fact cleared twice, the first time with an offset of one character to the right and down to produce the shadow effect. The colour of the shadow is defined by the variable P2. The main window colour is defined in the variable P1.

Laser Basic