DOCUMENTATION FOR EXTENSIONS TO OS-9 ON THE DRAGON 128

BY VIVAWAY LIMITED

written by Paul Dayan

Dragon 128 OS-9 memory allocation

OS-9 Level 2 normally allocates memory blocks from physical block 0 upwards. On the Dragon 128 the first 128k bytes (32 blocks) can be allocated to the screen. In certain screen modes the same blocks in the two 64k byte pages must be allocated together, (eg blocks 10 to 14 with blocks 26 to 30). Therefore OS-9 has been expanded with the addition of two system calls to manage screen memory. F$GMap reserves screen memory, in the lower or both pages. F$GClr returns the memory. In order to maximise the availability of memory for the screen display, the normal OS-9 memory allocation routines have been modified. First, any memory above the first 32 blocks is used. Then blocks are allocated from the first 32 in an order designed to maximise the availability of screen memory, (which is allocated by F$GMap from the top down). This order is determined by a table in the source file BlkTrans, used in the assembly of OS9P1 and IOMAN.

Dragon 128 OS-9 screen driver

The device driver for the built-in screen and keyboard is called **screen.** It also handles the mouse, lightpen, and beeper. The device descriptor that uses screen is **term.**

1. The keyboard
The keyboard has auto-repeat (except on CAPS LOCK), and as good an approximation to N-key rollover as is possible with a matrix keyboard. Because the serial nature of the hardware makes keyboard readout a lengthy business, typing may interfere with serial port reception at the high baud rates, giving a READ ERROR.

The CAPS LOCK key toggles, (alternates the mode), between alpha upper case only, (light on), and upper and lower case.

The CAPS LOCK key in conjunction with the CODE, (FUNCTION), key toggles between the text and graphics, (if there is one), displays.

The numeric keypad has two modes:

i) numeric. The keys return the normal ASCII codes.
ii) function. The keys, (except ENTER), return the code that would be returned in numeric mode if pressed in conjunction with the FUNCTION key.

There are three modifier keys: SHIFT, CONTROL, and FUNCTION, (in order of precedence). These return no code when pressed, but modify the codes returned by other keys.

The keyboard is scanned on Vertical Sync (VSYNC), ie every 20ms, (16.7ms in the USA). The key codes are put in a 100 character buffer, giving the same type-ahead as the standard serial terminal driver. The mouse buttons and the lightpen also appear as keys on the keyboard, returning their own codes. The mouse buttons do not auto-repeat.

Key code table (main keypad):

Code in hexadecimal

KEY	UNMODIFIED	SHIFT	CONTROL	FUNCTION
SPACE	20	20	20	A0
,	2C	3C (<)	2C	AC
-	2D	3D (=)	2D	AD
.	2E	3E (>)	2E	AE
/	2F	3F (?)	2F	AF
0	30	5F (_)	30	B0
1	31	21 (!)	31	B1
2	32	22 (")	32	B2
3	33	23 (#)	33	B3
4	34	24 ($)	34	B4
5	35	25 (%)	35	B5
6	36	26 (&)	36	B6
7	37	27 (')	37	B7
8	38	28 (()	38	B8
9	39	29 ())	39	B9
:	3A	2A (*)	3A	BA
;	3B	2B (+)	3B	BB
@	40	60	00	80
A to Z	41 to 5A	61 to 7A (a-z)	01 to 1A	C1 to DA
[	5B	7B ({)	5B	DB
\	5C	7C (|)	5C	DC
]	5D	7D (})	5D	DD
^	5E	7E (~)	5E	DE
CLEAR	0C	1C	1E	8C
ESC	1B (ESC)	05 (^-E)	03 (^-C)	8B
ENTER/RETURN	0D	0D	0D	0D
DELETE/RUBOUT	08	1B (^-X)	7F (DEL)	88
TAB	09 (TAB)	1D	1F	89

numeric pad, numeric mode:

#	23	23	03	A3
0 to 9	30 to 39	30 to 39	10 to 19	B0 to B9
*	2A	2A	0A	AA
.	2E	2E	0E	AE

numeric pad, function mode:

0 to 9	B0 to B9	F0 to F9	90 to 99	B0 to B9
#	A3	E3	83	A3
*	AA	EA	8A	AA
.	AE	EE	8E	AE

the mouse buttons return the following two-code sequences:

left button	1F 42
right button	1F 43

the light pen returns the following sequence:

1F 41 column+20 line+20

where column is 0 to 79, line is 0 to 24 (decimal), eg column 23, line 13 (decimal), returns

1F 41 37 2D

The **term** device descriptor has been set up for line editing as follows, (all keys except RUBOUT are on the numeric pad, either in Function mode or in Numeric mode with the FUNCTION key):

KEY	ACTION
RUBOUT	delete char to left of cursor
4	move cursor left (wraps to end of line)
6	move cursor right (wraps to start of line)
7	re-display line up to Carriage Return
9	delete all chars to left of cursor
*	delete char under cursor
£	delete all chars under and to the right of cursor

2. The screen display

Characters written to the screen fall into one of three categories:

i) Printable characters (hex 20 to 7E, and 80 to A0)
ii) Control characters (hex 00 to 1F, and 7F)
iii) Characters within escape sequences (hex 00 to FF)

i) Printable characters
Hex 20 to 7E display characters hex 20 to 7E of the character set ROM. Hex 80 to 9F display characters hex 00 to 1F. Hex A0 displays character hex 7F. The four different character sets in the ROM are selected by escape sequences.

ii) Control characters
The following control characters have effect; the rest are ignored:

CODE hex	ACTION
07	bell (beep)
08	backspace, non-destructive
0A	line feed
0B	home, to 0,0
0C	clear screen and home
0D	carriage return to column 0 (no auto LF)
1B	escape sequence introducer

iii) Escape sequences
These are multi-character sequences introduced by hex 1B (ESC). The next character determines the action type, and is

referred to as the escape code. Depending on the action type, further characters may be required as parameters.

The action routines for the graphics facilities are in separate subroutine module **gfxdrvr**. This module must have been previously loaded into memory for these facilities to be available. Therefore the two sets of escape sequences are listed separately.

a) available in **screen**, for the text display, single digit ASCII numeric parameters (hex 30 to 39) shown as **n**:

ESCAPE SEQUENCE hex	ACTION
1B 41 x+20 y+20	cursor addressing, column then line
1B 42	clear to end of line (cursor unchanged)
1B 43	cursor right one (with wrap and scroll)
1B 44	cursor up one (with scroll)
1B 45	cursor down one (with scroll)
1B 46	set reverse field mode for new chars
1B 47	clear reverse field mode
1B 48	set underline mode for new chars
1B 49	clear underline mode
1B 4A	clear to end of screen (cursor unchanged)
1B 4B	cursor left one (with wrap and scroll)
1B 4C y+20 x+20	cursor addressing, line then column
1B 4D n	select brightness, 0 to 3
1B 4E n	select character set, 0 to 3
1B 4F	set screen to text display
1B 50 n	set cursor type: 0 = block 1 = underline 2 = no cursor
1B 51 n	select foreground colour for new chars, 0 to 7
1B 52 n	select background colour for new chars, 0 to 7
1B 53	set flash mode for new chars
1B 54	clear flash mode
1B 55	go to 40 column display, clear screen
1B 56	go to 80 column display, clear screen
1B 57	set numeric mode for numeric keypad
1B 58	set function mode for numeric keypad
1B 59	select ASCII numeric parameter mode
1B 5A	select binary numeric parameter mode

The settings on power-on or reset are screen clear, 40 column mode; reverse field, underline, and flash all off, brightness 0, foreground colour 2, background colour 0, in text display.

The text screen display is 40 or 80 columns, 25 lines, with each character cell 8 dots wide, 10 rows high.

b) available in **gfxdrvr**, for the graphics display. Graphics

escape sequences take numeric and string parameters. Strings are terminated by hex OD (carriage return). Numeric parameters are either two-byte binary (BASIC09 integer), or variable-length ASCII decimal if the ASCII numeric parameter mode is set, (default = clear). ASCII numeric parameters are separated by hex 2C (comma), with the last one terminated by hex OD (carriage return). The escape sequences are all of the form:

ESC code number-of-parameters parameters

where ESC is hex 1B
code is a single character, giving the action type
number-of-parameters is a numeric parameter giving
the number of parameters to follow (may be 0)
parameters are numeric or string, as required.

For example, in binary numeric parameter mode
GMODE(1,3) would be (in hex):

1B 61 00 02 00 01 00 03

in ASCII decimal numeric parameter mode it would be (in hex):

1B 61 32 2C 31 2C 33 OD

equivalent in BASIC09 to
PRINT CHR$($1B);"a2,1,3"

Note that in GGET and GPUT the array parameter is the address of the array, not the contents of the array.

The graphics subroutine package **graphics** translates BASIC09 procedure calls such as GMODE(1,3) into escape sequences for the screen driver. It also provides some additional commands. See the section on the graphics subroutine package.

The following table relates the appropriate escape code to the graphics subroutine package keyword, as used in BASIC09. The details of the action taken are given in the section on the graphics subroutine package.

ESCAPE CODE hex	KEYWORD
61	GMODE
62	GPAGE
63	GMAP
64	CSET
65	GCLEAR
66	COLOUR
67	GRAPHICS
68	TRANSLATE
69	SCALE
6A	ROTATE
6B	MOVE
6C	PLOT
6D	DRAW
6E	GGET
6F	GPUT
70	PMODE
73	PAINT
74	CIRCLE
75	BORDER
76	GPRINT

NOTE: while the graphics is active at least 20k of memory is being used, and **gfxdrvr** is linked into the system address space. The latter consideration means that there is insufficient room in the system address space for **format.** To return the graphics memory and unlink gfxdrvr from the system address space, use either

```
GMODE(0, 0)
```

inside a BASIC09 program, or

```
OS9: display 1B 61 00 02 00 00 00 00
```

from OS-9 **shell.** (This latter could conveniently be assigned to a macro key).

3. GetStt and SetStt
The I$GetStt and I$SetStt codes recognized by **screen** are:

i) GetStt

mnemonic	hex	ACTION
SS.Ready	01	returns carry set and B=E$NotRdy if no keyboard chars are waiting, else carry clear
SS.EOF	06	always returns carry clear
SS.Mouse	20	returns the current mouse x coordinate in X, the current y coordinate in Y (range -$8000 to +$7FFF).

SS.Size	02	returns the current graphics mode: Y = MSB = 0 lower 64k page only = 1 both pages LSB = 0 graphics not active = 1 320x256 mode = 2 640x512 mode X = number of first block in lower page

ii) SetStt

mnemonic	hex	ACTION
SS.SSig	1A	input is signal code in X. Sends the signal code to the task (F$Send) when data is available
SS.Relea	1B	clears the SS.SSig request
SS.Mouse	20	resets the mouse x and y coordinates to 0

Additional OS-9 service requests on the Dragon 128

F$GMap

hex 54

Input B = number of 4k blocks required in each 64k page
       A = 0 required in lower page only
           1 required in both pages

Output X = first block number in lower page
       or carry set, B has error code, if memory not available

Graphics memory comes from the first 128k bytes. Some modes require memory from the first 64k page only, others require memory from both pages, in which case the memory must be in the same position in the two pages. OS-9 allocates graphics memory from the top down (ie blocks 15 and 31 first).

F$GMap only reserves the memory - it does NOT map the memory into the user's address space. This must be done using F$MapBlk and F$ClrBlk. Similarly, the memory is NOT automatically deallocated when the process ends. F$GClr must be used. Therefore F$GMap must be used with caution.

F$GClr

hex 55

Input B = number of blocks to deallocate in each 64k page
       A = 0 deallocate in lower page only
           1 deallocate in both pages
       X = first block number in lower page

Output none

This call deallocates screen memory reserved by F$GMap. Note that NO CHECK is made on the validity of the call. Therefore F$GClr must be used with extreme caution.

F$Timer

hex 53

Input X = address of timer service routine
       U = address of service routine's static storage

Output carry set with error code in B is table is full

As there are no hardware timers in the Dragon 128, this facility has been added to give device drivers a form of interrupt-driven time keeping. The facility is part of the clock module **clock**. The service request causes the address of the service routine and the address of its static storage to

be stored in a table, (max 64 entries). From then on, that routine is called, (with its U register set to the address of its static storage), every 20ms (16.7ms in the USA). The routine may be removed from the table by a further F$Timer call, with U as before, but X set to 0.

THIS IS A PRIVILEGED SYSTEM MODE REQUEST

F$InsErr

hex 21

Input X = address of error messages file pathlist, or name of error messages module

Output carry set with error code in B if path not found.

Error reporting under OS-9 has been extended on the Dragon 128. (Standard OS-9 Level 2 just prints the error number). The process descriptor for each process has a 32 byte area for a module name or file pathlist. This string is inherited from the parent process, (the initial string is **errmsg**), but may be changed by the F$Instrr service request.

When F$PrtErr is called to report an error, OS-9 first tries to link to a module of the given name. If this is unsuccessful OS-9 attempts to open a file with the given name. If one of these succeeds, the module or file is scanned for a message for that error number. If it is found, it is printed. If neither module not file is found, or the message is not found, the process is repeated using the string in the system process descriptor. (This is the initial string mentioned above, usually **errmsg**, and is taken from the **Init** data module). The module and file have the same structure - the module is simply the file loaded into memory. Thus the module may be loaded into memory for speed, or left on disk to save memory.

The secondary attempt with the system string means that a process can indicate a file or module containing a subset of error messages, such as compiler errors, leaving the remainder to the default, (system), path. Because of the structure of the error messages module, the messages can actually be within the program. The module structure extends the standard OS-9 module header with a two byte offset from the start of the module to the error messages table. Each error message has the error number in ASCII decimal, followed by a space, followed by the message terminated by carriage return. For example:

```
Type        set Prgrm+Objct
Revs        set Reent+1

            mod ProgEnd,ProgNam,Type,Revs,ProgEnt,Progmem

            fdb ErrTab

ProgNam     fcs 'program'
Edition     fcb 1

ErrTab      fcc '12 syntax error'
            fcb $0D
            fcc '13 mismatched parentheses'
            fcb $0D
            fcb 0
```

The table is terminated by a byte of 0.

NOTE if F$InsErr is used to indicate a file rather than a module, the full pathlist must be given, eg

```
            /d0/sys/errmsg
```

Extension to Shell on the Dragon 128

OS-9 Shell on the Dragon 128 has been extended to allow the changing of the error messages file or module name, via the **seterr** command built into Shell.

```
OS9: seterr /d0/sys/errmsg
```

indicates a file, and

```
OS9: seterr errmsg
```

indicates a module. Where a file is intended, the full pathlist must be given.

The floppy disk device driver for OS-9 on the Dragon 128

The floppy disk driver module is **wd2797**. It can read/write the following disk formats, (standard OS-9):

i) single and double sided disks
ii) single density disks
iii) double density disks with track 0, side 0 in single density
iv) 48 tpi and 96 tpi disks
v) 48 tpi disks in 96 tpi drives

The following formats have been added:

i) double density on all tracks, the first sector on each track is sector 1, (Dragon 64 compatible).
ii) 128 bytes per sector, first sector on each track is sector 1.

The additional formats are indicated in the device descriptor. Double density on all tracks is indicated if bit 2 of PD.DNS is set. In this case 'sectors per track' and 'sectors per track on track 0' must be the same value, usually 18. 128 bytes per sector is indicated if bits 5 and 6 of PD.TYP are set. In this case each track is regarded as being composed of one 256 byte sector for each two 128 byte sectors, with a phantom sector zero, (which the calling software must skip). Thus 'sectors per track' and 'sectors per track on track 0' must be half the actual number of sectors, plus one. This format is really only useful for single sided, single density disks.

An additional 'boot' entry point has been added to the standard branch table. This entry point is used by the **boot** program.

The **format** program for OS-9 on the Dragon 128

The standard **format** program has been extended to format disks that are double density on all tracks, with the first sector on each track being sector 1. This selection is made from the device descriptor - it is not an option when **format** is run.

The following formats are therefore available:

i) single density on all tracks, 10 sectors per track
ii) single density on track 0, side 0, 10 sectors; double density on all other tracks, 16 sectors per track.
iii) double density on all tracks, 18 sectors per track.
iv) single and double sided disks.
v) 48 tpi and 96 tpi disks
vi) 48 tpi disks on 96 tpi drives.

The sequential character file manager (SCF) has been greatly extended to provide additional line editing facilities and a macro facility. The keys and codes for the additional line editing facilities are taken from an extended version of the device descriptor. Because of the limited size of the path descriptor, they are not transferred to the path descriptor, and so cannot be changed by **tmode**, or be accessed by the SS.Opt option in I$GetStt and I$SetStt.

The input (key) codes and output codes may be one or two bytes. The device descriptor contains a lead-in code for input, and a lead-in code for output. If an input character received is found to match the input lead-in character, the next character is taken in and its bit 7 is set. It is then treated in the same way as characters that do not match the input lead-in character - it is checked for a match with the special input codes; if found it is acted on, otherwise it is put in the input line at the current cursor position. (Any characters under or to the right of the cursor are first moved right one space).

In implementing the extended features, certain special output commands are required, which may be one or two bytes. If the code given for the function in the device descriptor has bit 7 set, and the output lead-in character is non-zero, the lead-in character is sent, followed by the code with bit 7 cleared. Otherwise the code alone is sent, (unmodified). The additional bytes follow the XON and XOFF bytes in the standard device descriptor, as follows:

Number of bytes	Function
1	Number of columns per line
1	0=has extended editing facilities
1	lead-in character for input
1	lead-in character for output
1	'move left' output code
1	'move right' output code
1	'move left' key code
1	'move right' key code
1	'delete char under cursor' key code
1	'delete to end of line' key code

The extended line editing features are available on the standard I$ReadLn service request. In addition, an 'edit line' facility is available using I$GetStt:

Mnemonic: SS.Edit
hex: 1C

Input X = address of line to edit
U = initial position of cursor in line (base 0)

Output X = unchanged
        carry set if error, with error code in B

Macros

An input macros facility has been added to SCF. Any key code, (or two byte sequence, as previously described), not already allocated to a special function may be defined as a macro key. The macro may be any string of characters, but it must be borne in mind that characters within a macro are not checked for special functions. For example, a Carriage Return, (hex OD), in a macro will not be seen by SCF as an End-of-Line. This allows macros to contain unrestricted sequences of characters. A Read (I$Read) and a Read Line (I$ReadLn) request process macros slightly differently. Successive Read requests for amounts of data less than the macro size, (eg one byte at a time), will return successive chunks of the macro until it is used up. On a Read Line request, if the macro will not fit into the caller's line buffer, none of the macro characters are put in the buffer. This prevents Read Line requests unintentionally receiving parts of macros.

SCF keeps a macro buffer for each path. 253 bytes are available for macros in the buffer. Each macro takes two bytes plus the length of the string.

Macros are inserted and deleted using I$SetStt.

Mnemonic: SS.SMac define a macro
hex: 1D

Input X = address of macro string
       Y = MSB = macro key code
           LSB = macro length

Output carry set if macro buffer full

If the macro key code is already defined it is overwritten.

Mnemonic: SS.CMac clear a macro
hex: 1E

Input Y = MSB = macro key code

Output none

If the macro key code is 0, all macros are cleared. Otherwise just the specified macro is cleared.

1. Extensions to the Edit mode

Three features have been added to the Edit mode:

i) Targeting on line numbers. If a number in an edit command is preceded by a #, it is taken to be a line number. Examples:

```
E:#100          Move to line 100
E:#100L5        List 5 lines from line 100
E:#100L#200     List from line 100 to line 200
E:#100D5        Delete 5 lines starting with line 100
```

ii) Line editing. The E command displays the current line and allows it to be edited using the SCF line editing facilities. For example:

```
E:#100          Move to line 100
E:E             Edit it
```

iii) Automatic line numbering. The A command followed optionally by a start line number and increment012enters the automatic line numbering mode. ENTER in response to a line number exits the mode. The default start line number is 100; the default increment is 10. A space is required between the **A** and the start line number. For example:

```
E:A 200,15      Start line 200, increment 15
200

E:A 300         Start line 300, increment 10
```

2. Graphics extension

BASIC09 has been extended to link into the graphics subroutine package. If the compiler does not recognize a keyword, it tries to link to the subroutine package **graphics.** If successful, it scans the **graphics** keyword table for the keyword. If found, the compiler compiles a RUN call to the subroutine package, inserting the routine number as the first parameter. The decompilation routine does the reverse, restoring the original keyword when a RUN GRAPHICS command is seen.

See the section on the graphics subroutine package for the structure of the keyword table.

Graphics subroutine package for the Dragon 128

The graphics subroutine package **graphics** comprises a set of subroutines and a keyword table. The keyword table is used by BASIC09 keywords to include the set of subroutines. Most of the subroutines are used to perform calls to the graphics facilities of the screen driver subroutine package **gfxdrvr.** For this reason, and to conserve memory, the two modules are merged into the file **graphics** in the CMDS directory of the system disk, and may be loaded into memory by

```
OS9: load graphics
```

Both **graphics** and **gfxdrvr** must be unlinked to free the memory they occupy. THIS MUST NOT BE DONE WITH THE GRAPHICS ACTIVE.

The following table details the subroutines available. The number given for each routine is the routine number, which is given as the first parameter, (two byte), in a call to **graphics.** BASIC09 automatically inserts this parameter. Thus

```
GCLEAR(1)
```

is equivalent to

```
RUN graphics(5,1)
```

With a knowledge of these numbers, and provided the BASIC09 parameter passing convention is adhered to, the subroutine package may be called by other languages. A * beside the routine name indicates that the routine makes a call to **gfxdrvr.**

The DRAW routine expects to be passed a string terminated by hex FF, (standard BASIC09), or the whole string, (whose length is given in the calling structure), is used. When the string is sent to **gfxdrvr, graphics** terminates it with a hex OD.

The Graphics routines

1.*GMODE (p1,p2)
Select resolution, number of colours, number of pages, (p2). p2 defaults to 1.

p1	p2	effect
0	0	exit graphics, return graphics memory
1	1-3	320x256, 4 colours
2	1-2	320x256, 16 colours
3	1	640x512, 2 colours
4	1	640x512, 4 colours

An error is generated if the memory is not available.

2.*GPAGE (p1)
Select current graphics page for drawing in, (p1). If p1 is not given, it defaults to 1.

3.*GMAP (p1,p2)
Request that some or all of the graphics memory be mapped into this task's address space. p1 gives the number of bytes to be mapped in, starting with the 4k block that contains the cursor. p2 is used to return the address at which the memory has been mapped in. This facility is primarily for games writers, who will want to manipulate the graphics memory directly.

An error is generated if there is insufficient room in the user's memory map.

4.*CSET (p1, p2, p3, p4)
For modes with less than 16 colours, this command selects the colours used to make up the colour sets.

The 4 colour modes require all 4 parameters. The 2 colour mode requires only p1 and p2.

If all parameters are omitted the default colours used are colour 0 and colour 1 if in the 2 colour mode, or colours 0, 1, 2 and 3 if in a 4 colour mode.

5.*GCLEAR (p1)
Clear the current graphics page to colour p1, and set the background to p1. If p1 is not given, it defaults to colour 0.

6.*COLOUR (p1)
Select the drawing colour, (p1). If p1 is not given it defaults to colour 1.

7.*GRAPHICS (p1)
Switches the display to graphics page p1, defaults to 1.

8. TEXT
Switches the display to text.

NOTE: The next three functions affect all plotting. A transformation matrix is produced that is applied to each point before it is plotted. This includes the GGET and GPUT commands, so that objects can easily be moved, scaled, and rotated. The order of transformation is scale-rotate-translate. Subsequent transformations of the same type add rather than replace - ie transformations are always relative to the transformed axes.

9.*TRANSLATE (p1,p2)
All subsequent points have p1 added to their x coordinate, and p2 added to their y coordinate. If there are no parameters, the translation is reset to the physical origin.

10.*SCALE (p1,p2)
All subsequent points are scaled by p1 in the x direction, and p2 in the y direction. The two parameters are in units of 1/256. Thus a parameter of 256 results in a scale factor of 1. A negative scale factor results in a reflection of the object about the orthogonal axis. Subsequent calls to SCALE multiply. For example
SCALE(512,512)
SCALE(512,512)
SCALE(64,64)
causes a scale factor of 2, then 4, then 1. If there are no parameters, the scale factors are reset to 1.

11.*ROTATE (p1)
All subsequent points are rotated through an angle of p1 degrees, about the translated coordinate origin, effectively a rotation of the axes. Rotation is anticlockwise. If there is no parameter, the x axis is returned to the horizontal.

12.*MOVE (x1,y1)
Sets the drawing cursor position to x1,y1. If there are no parameters, the cursor is returned to the coordinate origin.

13.*PLOT (x1,y1,x2,y2,x3,y3,.....)
Point and line drawing with x-y addresses. If there are no parameters, a point is plotted at the current cursor position. Otherwise, a line is drawn to the first point, then from there to the next point, and so on, until all parameters are exhausted. The cursor is moved to the end of each line plotted, whether or not the end of line appears on the screen.

14.*DRAW (p1)
Provides for relative movement and line drawing. Drawing starts from the current cursor position with the current heading. The parameter p1 is a string, with numbers separated by command letters. Spaces are optional around the numbers, which can have a leading minus sign. The commands are

Command		Action
D	n	draw n points in the current direction, updating the cursor position
M	n	move n points in the current direction without drawing
T	n	turn anticlockwise by n degrees, (n can be negative). The heading at the start of each DRAW command is along the positive x axis. A value of 0 resets the heading to along the positive x axis.

An example, drawing one box within another

```
DRAW("D10T90D10T90D10T90D10M5T-90M5T180
      D20T90D20T90D20T90D20")
```

15.*GGET (P1,P2,P3)
Saves a rectangle of the screen into an array. The bottom left-hand corner of the rectangle is the current cursor position. p1 is the width of the rectangle, p2 is the height, and p3 is the array in which to store the rectangle. Two points are stored in one byte of the array, bits 7-4 as the colour number of the first point and bits 3-0 as the colour number of the second point. The bottom row is stored first, from left to right, then the next row up, and so on. An error is returned if all of the area to save does not lie on the screen.

An example, saving a rectangle of 16 by 20, at (100,73)

```
DIM ARRAY(179):BYTE
DIM X,Y,WIDTH,HEIGHT:INTEGER

X=100
Y=73
WIDTH=17
HEIGHT=21
MOVE(X,Y)
GGET(WIDTH,HEIGHT,ARRAY)
```

Note: array size must always be rounded up to the nearest whole number.

16.*GPUT (P1,P2,P3,P4)
The converse of GGET. p1, p2, p3 as for GGET. The array p3 could have been set up with a GGET, or by the user program writing the individual points into the array. Parameter P4 is the background colour used when GGET was done. Any points of this colour are not plotted when GPUT is done, leaving them as the current background colour. If p4 is omitted it defaults to the current background colour.

ROTATE, TRANSLATE, and SCALE operate on GGET and GPUT, allowing very easy animation effects, with objects moving around, rotating, and changing size.

17.*PMODE (p1)
Selects plotting mode. Two modes are available. The default p1 is 1.

p1	Plot action
1	Points plotted replace the existing display
2	Points plotted are exclusive-ORed with data already in the display.

Plot mode 2 allows objects to be drawn over other objects non-destructively. Simply drawing the object again removes it from the display, leaving the original picture. In fact, any number of objects may be written over each other, and then 'unwritten' in any order. The disadvantage is that the colours of the objects will change, depending on the colour they are covering.

18. GSAVE (p1)
Saves the current graphics display page to path number p1. A byte by byte copy of the screen memory is done.

19. GLOAD (p1)
Loads the current graphics page from path number p1, honouring the saved graphics settings. The current mode must be the same as when the picture was GSAVEd, otherwise an error is generated or the display is garbage.

20.*PAINT (p1)
Fills in the area around the cursor with the current drawing colour. If p1 is omitted, the boundary is taken to be any colour other than the one found at the cursor, otherwise the boundary is taken to be colour p1.

21.*CIRCLE (r,p1,p2)
Draws a circle or an arc of a circle. The centre is the cursor position. The radius is r. p1 is the angle at which drawing starts, p2 is the angle at which drawing finishes, allowing any arc to be drawn. p1 and p2 default to 0 and 360, ie a complete circle. Using SCALE before CIRCLE allows ellipses to be drawn.

22.*BORDER (P1)
Selects the colour used for the screen border. If p1 is omitted, the default value used is the current background colour.

23.*GPRINT (s1)
Prints a string s1 starting at the current graphics cursor, writing in the positive x direction. Characters are 8 pixels wide and 10 high, and the current scale, rotate and translate settings are honoured. The characters come from a module **chrset**, which must be already loaded into memory. The entry point offset in the module header points to the character set table. There are 96 characters in the set, (hex codes 20 to

7F), each has 10 bytes, one per character row, from the bottom of the character cell upwards. A set bit causes a point to be plotted in the current foreground colour. A clear bit causes no plotting. The cursor position is updated in the x direction by 8 x number of characters.

GENERAL NOTES:
i) Unless specified, all parameters are 16-bit, ie they must be defined as type INTEGER, or be entered as constants with no decimal point.

ii) x-y coordinate parameters may take any value from -32767 to +32767. Line portions outside the physical display area are clipped. In this way pictures can be defined that are 100 times larger than the screen. TRANSLATE can then be used to display portions of the picture, or SCALE can be used to display the picture in miniature.

iii) The physical screen is considered to be 640 points wide by 512 high in the high resolution mode, and 320 by 256 in the lower resolution mode.

iv) The physical coordinate origin is the bottom left hand corner of the screen.

v) The GMODE setting applies to all pages. There cannot be pages of different resolutions/numbers of colours in memory at the same time.

24. INKEY (s1,p1)
s1 is a string, p1 is an optional path number, (defaults to standard input). If data is available from the path, one character is read into the string. Otherwise an empty string is returned.

25. CHKRDY (p1,p2)
p1 is a Boolean, p2 is an optional path number, (defaults to standard input). If data is available from the path, p1 is set TRUE, else p1 is set FALSE. Note that no characters are read from the path.

26. MOUSE (p1,p2)
Returns the current mouse X and Y coordinates in p1 and p2 respectively, (range is -32768 to 32767). If p1 and p2 are omitted, the mouse coordinates are reset to zero.

The ACIA device driver for OS-9 on the Dragon 128

The ACIA device driver for the on-board 6850 has been expanded to include software selectable baud rate. This is done by setting a code in PD.BAU in the device descriptor, following the standard OS-9 codes, except that 19.2k baud is not available.

CODE	BAUD RATE
0	150
1	300
2	600
3	1200
4	2400
5	4800
6	9600

R.HARDING 2017
ORIGINAL DOC: DUNCAN SMEED
DragonData.co.uk