August 1991

This version created for ST FORMAT.

MARCH 1992

Published by: GFA Data Media (UK) Ltd
Box 121
Wokingham
Berkshire
RG11 5XT

Special Offer to ST Format Readers

This document is over 129 pages if printed on A4. We have documented all GFA-BASIC commands. All commands are also contained in the full GFA-BASIC REFERENCE manual which contains much more information than is contained in this 'brief' document. Most commands are fully explained with an example program.

You can obtain the FULL documentation, normally sold for 50.00 pounds for only 25.00 pounds (plus 5.00 postage). All orders must be returned direct to GFA, using the special coupon in the issue of ST Format that this disk was supplied with.

Also you can obtain the GFA-BASIC Compiler 3.5 and manual for only 20.00 (plus postage of 5.00). (normally 30.00). The compiler manual contains 48 pages. You will need the compiler to produce desk top accessories and link C and assembler code into your GFA-BASIC programs.

Or order both the Interpreter and Compiler for only 45.00 (plus 5.00 postage) which would normally retail for 80.00.

Please make Cheques and Postal Orders payable to GFA-DATA MEDIA (UK) LTD. We also can accept VISA and ACCESS credit cards.

TECHNICAL SUPPORT

Technical Support is only available to users who have purchased the full GFA-BASIC 3.5 Interpreter. All technical problems should be sent to GFA in writing, enclosing a stamped addressed envelope, and quoting their registration card number (supplied with each purchased product). Sorry but we are unable to supply technical support to users who are unable to quote their registration

number.

Now to the documentation:

Introduction

Welcome to GFA-BASIC. With the GFA-BASIC interpreter you now have the opportunity to explore the Atari ST, and create professional programs.

We believe that GFA is the best programming language available, not just for the Atari ST, but also for the Amiga, MS-DOS and Windows 3.0/3.1. When you have quickly mastered GFA-BASIC for the Atari ST, you can transport your program to any PC running either MS-DOS or Windows (and soon OS/2 and Unix!). This means that you will only need to learn one language for all these different machines and operating systems.

This manual contains all the GFA-BASIC 3.5 command descriptions and syntax. This manual is an edited version of the full reference manual (which is over 540 pages) and is a reference manual for the GFA-BASIC commands only. The full reference manual contains example programs for most of the commands and reference material for the Atari BIOS, XBIOS and GEMDOS.

This manual is a reference manual for the GFA-BASIC commands. It does not attempt to teach you to program, there are other books available for this purpose. Neither does this edited manual does not document the hardware or operating system features of the Atari.

This is the GFA-BASIC Interpreter. There are many more products in the GFA range to help you develop your programs. Please contact your distributor or GFA for more information and prices of these products.

GFA-BASIC Interpreter Reference Manual (Full version)

This loose leaf manual has over 540 pages, dealing with each command with a full description an example. The Atari system routines, GEMDOS, AES, VDI and GDOS are all described. The Line-A call is also described (although Line-A is now declared as un supported by Atari). Just about every useful table is presented in this manual, including GFA-BASIC error messages, Bomb Error messages, TOS Error messages, Scan code table, ASCII tables, Fill and style patterns, and VT-52 control codes.

GFA-BASIC Compiler

The GFA-BASIC Compiler will compile your interpreter code into fast stand alone programs. Desk top accessories can also be easily created. The GFA-BASIC Compiler also has options for optimising code for either speed or size. External code, either C or assembler can also be linked into your programs using the GFA-BASIC Compiler. The manual (48 pages) fully explains how to create a desk top accessory with examples. The compiler also supports the Atari TT, allowing code to be compiled for either ST or TT ram.

GFA-BASIC Software Development Book

This book covers in depth structured programming, debugging routines, Line A calls, Assembly routines, AES functions and GDOS.

Special attention is paid to dialog boxes and their creation. This book is over 341 pages and is supported by a disk containing the many examples of source code discussed within the book.

GFA GEM Utility Pack

THE GEM GUP is a framework of GFA-BASIC 3 routines that you can merge into your own programs. GUP takes the responsibility of managing the intricate GEM interface, leaving the programmer to concentrate on the main program logic. The example program supplied demonstrates how easy it is to create alternative icons, and use up to 5 different window types. The demonstration also shows how to use all the GEM gadgets, including scroll bars, title bar, info bar, grow box, grab bar etc. GFA GUP is a quick start to programming GEM. GUP is supplied with a 60 page manual and disk, fully documenting all of the GEM routines.

GFA Assembler

An integrated development tool for the Atari ST/STE, comprising editor, macro assembler, linker and debugger. Supports multiple window, file compare, background assembly. Editor checks syntax of machine code on entry, hence ideal for programmers new to assembly language. Also creates files that can be loaded directly into GFA-BASIC.

GFA-Basic and Assembler User Book

This book covers all that you will need to know in interfacing assembler routines into GFA-BASIC on the Atari ST/STE. Includes an introduction into assembly programming, creation of desk top accessories, graphic manipulation. Book (364 pages) plus disk.

GFA G-Shell

Professional development shell for GFA-BASIC Interpreter and Compiler on the Atari ST/STE and TT.
GFA-GShell allows the programmer to skip between Interpreter and Compiler and other programs, such as an assembler and resource editor. Supports make facility and on line help files. If you are using GFA-BASIC Interpreter and Compiler professionally, then GFA-GShell will improve your productivity.

GFA-DRAFT 3.2

Powerful and comprehensive, yet easy to use CAD software for Atari ST/STE and Atari TT. Runs in both high and medium resolutions. Icon and menu driven commands. There are symbol libraries for electrical, electronic users and architects. Libraries of symbols are also easy to create.

A program/macro interface (using GFA BASIC) will enable you to create complex forms and repetive patterns. Exports: HPGL, DXF and GEM files. Will support almost any model of printer or plotter. GFA-Draft Plus 3.2 also supports the Atari laser printer.

These GFA products and others are available from:

GFA Data Media (UK) Ltd
Box 121

Wokingham
Berkshire
UK RG11 5XT

Sales Line Only Tel: 44 (0)734 794941
                 Fax: 44 (0)734 890782

Using GFA-BASIC for the first time

The GFA-BASIC program disk is not copy protected, so before you do anything else, make a copy and store the original disk in a safe place.

GFA-BASIC is started by double clicking on the program file icon GFABASIC.PRG. After a short while the Editor screen will be displayed, which is where you can write and debug your programs. Lets now try entering an example program. Enter the following program lines, pressing the RETURN key at the end of each line.

```
DEFFILL 1,2,8
REPEAT
  WHILE MOUSEK=1
    PBOX MOUSEX,MOUSEY,MOUSEX+30,MOUSEY+30
  WEND
UNTIL MOUSEK=2
```

In the upper right hand corner of the editor screen you will find the RUN menu item. Click the RUN button or press Shift and F10 keys together and the example program will start. A simple program showing the use of the mouse. Move the mouse while holding down the left button. End the program by pressing the right mouse button. Press the Return key to end the program. The program could also have been stopped by pressing the CONTROL, SHIFT and ALTERNATE keys all at the same time. You will now be returned to the editor.

GFA-BASIC does not need line numbers. It is bad programming practice to say GOTO 10. If you must use GOTO, then use:

```
GOTO label
...
label:
```

GFA-BASIC offers you the chance to create structured programs, and indented in such a way to enable easy documentation and understanding of program flow.

How to proceed from here. We suggest you first become familiar with the GFA-BASIC Interpreter's editor. Then load some of the demonstration programs from the disk and run these. Experiment with these programs by changing the values of variables to see the effect. The next step really depends upon the individual, your knowledge of the Atari, your knowledge of programming and your program requirements. What ever your experience or requirements, GFA-BASIC is the best documented programming language available for the Atari ST, and there are several books and development aids to help you.

The Editor

THE GFA-BASIC Editor has been specially designed for program development. Syntax errors are recognised as you type the commands in, and in addition the commands are indented automatically.

Only one instruction per line is allowed, but a comment may be added to the end of a line by beginning the comment with a ! mark.

Program lines may be up to 255 characters long, but when a line exceeds 80 characters, the line will scroll.

Cursor Keypad

Left arrow	moves cursor left one character
Right arrow	moves cursor right one character
Up arrow	moves cursor up one line
Down arrow	moves cursor down one line
Insert	Inserts a blank line
ClrHome	moves cursor to top left corner
Ctrl ClrHome	moves cursor to start of program listing
Help	With the help key procedures and functions can be folded and unfolded. Move the cursor to the start of a procedure or function and then press Help. The procedure will the fold down to a single line

Numeric Keypad

Ctrl 4	moves cursor left one character
Ctrl 6	moves cursor right one character
Ctrl 8	moves cursor up one line
Ctrl 2	moves cursor down one line
Ctrl 7	jumps to start of program
Ctrl 1	jump to end of program
Ctrl 9	move one page up
Ctrl 3	move one page down
Ctrl 0	Inserts a line
Ctrl .	deletes a line

The numeric keypad can be swithed to a mode where it is not required to press the Ctrl key. This mode is toggled on and off by pressing Ctrl and - on the numeric keypad.

Delete	deletes character at cursor position
Backspace	deletes character to left of cursor
Tab	moves cursor right eight characters
Ctrl Tab	moves cursor left eight characters
Return	moves cursor to start of next line
Enter	moves cursor to start of next line
Escape	enter Direct Mode

Further editor commands

Ctrl Delete	deletes line which cursor is on
Ctrl U	un-deletes line
Ctrl Y	deletes line which cursor is on
Ctrl N	inserts blank line

Ctrl Q	call Block menu (as F4)
Ctrl B	mark start of block
Ctrl K	mark end of block
Ctrl R	Page up
Ctrl C	Page down
Ctrl E	replace text
Shift Ctrl E	find replace text
Ctrl F	find text
Ctrl left arrow	cursor to start of line
Ctrl right arrow	cursor to end of line
Ctrl up arrow	page up
Ctrl down arrow	page down
Ctrl Clr Home	cursor to start of program
Ctrl P	delete character right of cursor
Ctrl O	undelete text deleted with Ctrl P
Ctrl Z	cursor to end of program
Ctrl Tab	cursor left eight characters
Ctrl G	move to line number display

Editor marks

Ctrl n	where n= 0 to 6 will mark a position
Alt n	will jump to marked position n, except:
Alt 7	cursor to last cursor position before RUN
Alt 8	to start of program
Alt 9	cursor to start of last search
Alt 0	cursor to position of last change

Menu Bar

Atari Logo

This leads to a GEM menu bar:

Save

A file select box will appear, and a program can be saved by entering a name and selecting OK.

Load

A file select box will appear, and a program can be selected and loaded into the editor.

Deflist

See the DEFLIST command.

New Names

The editor will warn you if you enter a new variable name.

Editor

Returns you to the editor.

Load (F1)

Load a GFA-BASIC 3 .GFA tokenised file.

Save (Shift F1)

Save the program as a tokenised .GFA file.

Merge (F2)

Load an ASCII text file into the editor at the cursor position.
The existing program in the editor is not erased.

Save,A (Shift F2)

Save the program in memory in ASCII format. The extension .LST
will be added to the file name. If the file already exists the old
file will be changed to .BAK.

Llist (F3)

This command controls the output of the print out. These Dot
commands have no effect on the running of the program and only
affect the printout:

```
.ll xx          Max line length
.pl xx          Max page length
.ff xxx         Form feed character for printer
.he text        Text to appear on first line of each page
.fo text        Text to appear on last line of each page
.lr xx          Left margin
.l-             Switch dot commands off
.l+             Switch dot commands on (default)
.n1 to .n9      Switch on line numbers up to 9 characters
.n0             Switch off line numbers
.PA             Force form feed
.P-             Do not list dot commands
.P+             List dot commands
```

In the header and footer text, the following can also be inserted:

```
\xxx            The ASCII character xxx
\d              Date
\t              Time
#               Page number
```

To print the symbols \ and ##, use \\ and \ respectively.

Quit (Shift F3)

Results in Do you really want to quit message. If Yes selected,
thenyou will return to the desktop.

Block (F4)

If no block marked then message Block ??? will appear. If a block
has been selected using the BlkSta and BlkEnd then the block menu
will appear:

Copy

Copies the block to current cursor position.

Move

Moves the block to current cursor position.

Write

Saves the block as an ASCII file (.LST)

Llist

Prints the block out to printer

Start

Moves the cursor to start of block

End

Moves the cursor to end of block

Del

Deletes the block

Hide

Removes the block marker

Clicking the mouse outside the Block menu or pressing a key also removes the block marker.

New (Shift F4)

The program currently in the editor is erased from memory.

Blk End (F5)

The line before the cursor is marked as the end of block. If the start of block marker is located before this line, the block is shown in a different colour (or dotted background on a monochrome screen). Also actioned with Ctrl K.

Blk Sta (Shift F5)

Marks the beginning of a block as above. Also actioned with Ctrl B.

Find (F6)

Enter a text string to be searched for. If the string is found, the search cab be continued with Ctrl F or Ctrl L. Also actioned with Shift Ctrl F or Shift Ctrl L.
Folded procedures and functions are not searched.

Replace (Shift F6)

This function will replace one string of text by another. Input the text to be replaced, then the replacement text. On finding an occurrence, the actual replacement can be effected with Ctrl E. Also activated with Shift Ctrl E.

Pg down (F7)

Scrolls the screen down one page. Also activated with Ctrl C.

Pg Up (Shift F7)

Scrolls the screen up one page. Also activated with Ctrl R.

Insert/Overwr (F8)

Switches between insert and overwrite.

Txt16/Text8 (Shift F8)

Only on monochrome monitor. Either 16 pixel high characters or 8 pixel high characters will provide 23 or 48 lines on the screen respectively.

Flip (F9)

Flip between Edit and Output screen. Press any key to return to the Edit screen

Direct (Shift F9)

Switch to Direct mode, where commands will be actioned immediately. Some commands, such as loop commands are not available in direct mode. Direct mode can also be entered by pressing Esc key. The last 8 commands entered in direct mode are remembered and can be recalled using the up and down arrow keys. The Undo key will recall the last command.

A procedure can be called from direct mode.

Test (F10)

Tests all loops, subroutines and conditional instructions for consistency without running the program.

Run (Shift F10)

The program in the editor memory is started. A running program can be interrupted using Ctrl Shift Alt keys together.

Clock Display

The clock can be set by clicking on the clock and entering a new time.

Line Numbers

Clicking on the line number box (beneath the clock) allows you to enter a line number. On input the cursor will jump to this line number. Also activated with Ctrl G.

COMMANDS AND FUNCTIONS

The following commands and functions have been sorted by alphabetical order. Each entry describes the command syntax and the action of the command. Some of the more complex commands have examples. The full GFA-BASIC Interpreter Reference manual contains examples for most of

the commands. The advanced commands such as the operating system calls BIOS, GEMDOS and XBIOS are not covered here and should be consulted in the GFA-BASIC Interpreter Reference manual.

*
Syntax: *y
Action: returns the address of variables, but for strings or arrays the address of the descritpor is returned. (ARRPTR is the same as *).

+
Syntax: a$+b$
Action: Concatenation operator, to add strings together.

+ Arithmetic operator (a+b adds a and b).
* a*b (a times b).
/ a/b (a divided by b).
^ a^b (a to the power b).
<> a<>b (a not equal b).
<= a<=b (a less than or equal to b).
>= a>=b (a greater than or equal to b).

==
Syntax: a==b
Action: Comparison operator for approximately equal 'a' and 'b' are numeric expressions. The == operator is used in the same way as a comparison with = but only 28 bits of the mantissa are compared i.e. about 8.5 digits.

@
Syntax: @FRED
Action: GOSUB PROCEDURE fred (@ is synonymous with GOSUB).

ABS
Syntax: ABS(X)
Action: Returns the absolute value of a number. (see also SGN).

ABSOLUTE
Syntax: ABSOLUTE x,y
Action: Assigns the address y to the variable x.

ACHAR
Syntax: ACHAR code,x,y,font,style,angle
Action: ASCII characters value 'code' are displayed at the graphics location x,y.
Font can be:
0 = 6x6 (Icon font).
1 = 8x8 (Normal colour font).
2 = 8x16 (Normal monochrome font).
Larger values are taken to be the font header address of a GDOS font.
Text type (bold, faint etc. 0-31) and output angle (0,900,1800,2700) can be specified.

ACLIP
Syntax: ACLIP flag,xmin,ymin,xmax,ymax
Action: To define a 'clipping' rectangle to which LINE-A screen output will be limited.
Flag 0 = clipping off, non zero = on.

ACOS

```
Syntax:    ACOS(x)
Action:    Returns the arc-cosine (in radians) of x.

ADD(x,y)
Syntax:    z%=ADD(x%,y%)
Action:    Integer addition.

ADD
Syntax:    ADD x,y
Action:    Increase value of variable x by y.

ADDRIN
Syntax:    address of the AES Address Input block.
           With an index after this function, the appropriate
           parameter block is accessed directly.
           ADDRIN(2)=x is the same as
           LPOKE ADDRIN+2,x

ADDROUT    address of the AES Address Output block, see above.

AFTER ticks GOSUB proc (see also EVERY).
AFTER CONT
AFTER STOP
           Procedures can be called after the expiry of a set time.
           Time in ticks (200ths of a second).
           To continue the process, use CONT, and to stop
           use STOP.

ALERT
Syntax:    ALERT a,message$,b,button$,var
Action:    Creates an alert box
           'a' chooses type of alert symbol, 0=none, 1=!, 2=?, 3=stop
           'message$' Contains main text.
           Up to 4 lines of 30 characters/line
           lines are separated by the '|' symbol (Shift \).
           'button$'  Contains text for the buttons 'B'.
           'b' is the button to be highlighted (0,1,2,3)
           to be selected by just pressing return.
           'var' This variable is set to the number of the button
           selected.

Example:   ALERT 1,"Pick a|button",1,"Left|Right",a%
           ALERT 0,"You pressed|Button"+STR$(a%),0,"OK",a%

ALINE
Syntax:    ALINE x1,y1,x2,y2,f,ls,m
Action:    Draw a line using LINE A. x1,y1 are the start coordinates
           x2,y2 end of line. f = colour (0-15). ls = line style in 16
           bit information, (solid, dashed, dotted ..).
           m  mode
           0  Replace
           1  Transparent
           2  Inverted
           3  Inverted transparent


AND
Syntax:    x AND y
Action:    Logical operator, performs a logical AND, results in a true
           or false result.
AND()
Syntax:    AND(x,y)
```

Action: See AND above.

APOLY TO
Syntax: APOLY adr_pnt,num_pnt,y0 TO y1,f,m,addr,num_pattern
Action: Similar to POLYFILL, draws a sequence of joined lines, with 'num_pnt' corners, and fills the resulting area with a user defined pattern. 'adr_pnt' is the address of the array holding the alternating x,y corner coordinates. 'num_pnt' is the number of points. y0 and y1 specify the highest and lowest part of the screen where filling can occur. The parameters f,m,addr,num_pattern are the same as for HLINE.

APPL_EXIT
Syntax: APPL_EXIT()
Action: Informs the system prog has finished - this is a dummy function as QUIT or SYSTEM do this automatically.

APPL_FIND
Syntax: APPL_FIND(fname$)
Action: Returns the ID of the sought after application. Either returns the apps ID or -1 if it cant be found. fname$ is an 8 character filename - without extension.

Example: PRINT APPL_FIND("CONTROL")

         prints -1 if it cant find the CONTROL.ACC, or
         returns it's ID [2].

APPL_INIT
Syntax: APPL_INIT()
Action: Announce the program as an application.

APPL_READ
Syntax: APPL_READ(id,len,adr_buffer)
Action: Instruction bytes can be read from the event buffer.
         id - id of the application from whose buffer reading is to be done.
         len - number of bytes to read.
         adr_buffer - address of the buffer.

APPL_TPLAY(mem,num,speed)
APPL_TRECORD(mem,num)
         APPL_TRECORD makes a record of user activities, and TPLAY plays these back at the specified speed (1 - 1000).
         * ONLY VALID on TOS 1.4 and above. *

APPL_WRITE
Syntax: APPL_WRITE(id,len,adr_buffer)
Action: Bytes are written to the event buffer. SEE APPL_READ.

ARECT
Syntax: ARECT x1,y1,x2,y2,f,m,addr,num_pattern
Action: Similar to PBOX, x1,y1 and x2,y2 are opposite corners of a rectangle. The parameters f,m,addr,num_pattern are the same as for HLINE.

ARRAYFILL
Syntax: ARRAYFILL x(),n
Action: Assigns the value 'n' to all elements of a field array x().

ARRPTR

```
Syntax:    ARRPTR(x)
Action:    Finds the address of the (6 byte long) descriptor of
           a string or field. (Same as *x)

ASC
Syntax:    ASC(x$)
Action:    Finds the ascii code of the first character of a string.

ASIN
Syntax:    ASIN(x)
Action:    Returns the arc-sine (in radians) of x.

ATEXT
Syntax:    ATEXT x,y,font,s$
Action:    Output text at x,y coordinates using A LINE.

ATN
Syntax:    ATN(x)
Action:    Returns the arc tangent of x.

BASEPAGE
Syntax:    BASEPAGE
Action:    Returns the address of the basepage of GFA-Basic.

BCHG(x,y)
BCLR(x,y)
           Allow setting and resetting of bits.
           BCLR sets the y-th bit of x to zero.

BGET
Syntax:    BGET #i,adr,cnt
Action:    Reads from a data channel into an area of memory

           'i'   \
           'adr'  -- integer expressions.
           'cnt' /            'i'       is the channel number.
           'cnt'     bytes are read in and stored in memory
                     starting at address 'adr'
           Unlike BLOAD, several different areas of memory can be
           read from a file.

BIN$
Syntax:    BIN$(x[,n])
Action:    Changes value of 'x' to a string containing the binary
           value of 'x'. The optional parameter 'n' specifies the
           number of character positions to be used (1 to 32).

BIOS
Syntax:    BIOS(n[,x,y])
Action:    To call the BIOS routine. The optional parameter list can
           be prefixed with W: or L: to denote word or longword
           parameters. (if non given, default is W:)
Example:
           REPEAT
           UNTIL BIOS(11,-1) AND 4

           Waits for the Control key to be pressed.

           Please refer to GFA-BASIC Interpreter Manual for full  list
           of BIOS calls.

BITBLT
```

```
Syntax:    BITBLT s%(),d%(),p%()
Action:    Raster copying command similar to GET and PUT but
           more flexible and faster for some applications.
           's%' the description of the source raster
           'd%' the description of the destination raster
           'p%' co-ordinates of the two equally sized rectangles
                and the copying mode (see PUT).

BLOAD/BSAVE
Syntax:    BLOAD "filename" [,address]
           BSAVE "filename",address,length
Action:    Load and save memory from and to disc drive.

Example:   DEFFILL 1,2,4
           PBOX 100,100,200,200
           BSAVE "RECT.PIC",XBIOS(2),32000
           CLS
           PRINT "IMAGE STORED. Press a key to continue"
           ~INP(2)
           CLS
           BLOAD "RECT.PIC",XBIOS(2)

BMOVE
Syntax:    BMOVE scr,dst,cnt
Action:    Fast movement of memory blocks.
           'scr' is the address at which the block to be moved
           begins.
           'dst' is the address to which the block is to moved
           'cnt' is the length of the block in bytes.

BOUNDARY
Syntax:    BOUNDARY n
Action:    Uses function vsf_perimeter to switch off (or on) borders
           on filled shapes (PBOX, PCIRCLE ..). If n is zero - no
           border, n - non zero = border.
BOX
Syntax:    BOX x,y,xx,yy
Action:    Draws a rectangle with corners at (x,y) and (xx,yy)

BPUT
Syntax:    BPUT #n,adr,cnt
Action:    Reads from an area of memory out to a data channel.
           'n' is a channel number.
           'adr' is start address of memory to read from.
           'cnt' bytes are read from address.

BSAVE      See BLOAD

BSET(x,y)  Allows setting and resetting of bits.
BTST(x,y)  BSET sets the y-th bit of x to 1.
           BTST results in -1 (TRUE) if bit y of x is set.

Example:   x=BSET(0,3)
           PRINT x,BSET(0,5)

BYTE(x)    Returns the lower 8 bits of the numerical expression x.
           (See also CARD(), WORD() ).

BYTE{x}    As  a function eg.  y=BYTE{x} one can read the contents  of
           the address x.  As a command one writes to address  x.  eg.
           BYTE{x}=y. This is similar to PEEK and POKE but is not done
           in supervisor mode.  (See also CARD{},  INT{},  LONG{}, {},
```

FLOAT{}, SINGLE{}, DOUBLE{}, CHAR{}).

C:
Syntax: C:addr([x,y,....L:x, w:y])
Action: Calls a C or assembler program with parameters as in C. The parameters can be sent as 32-bit long words or 16-bit words to the subroutine. eg. a%=C:adr%(L:x,W:y,z) leads to the following situation on the stack:
(sp) ->return address (4bytes)
4(sp) ->x (4 bytes)
8(sp) ->y (2 bytes)
10(sp) ->z (2 bytes)
The value returned by the call is the contents of D0.

CALL
Syntax: CALL addr([x,y,....L:x, w:y])
Action: Calls a machine code or C subroutine at address 'addr'. When the call is made, the return address is on the top of the stack, followed by the number of parameters as a 16-bit word, then the address of the parameter list as a 32-bit word.

CARD(x) Returns the lower 16 bits of the numerical expression x. (See also BYTE, WORD).

CARD{x} Reads/writes a 2-byte unsigned integer (similar to DPEEK/DPOKE). (See also BYTE{}, INT{}, LONG{}, {}, FLOAT{}, SINGLE{}, DOUBLE{}, CHAR{}).

CASE See SELECT.

CFLOAT(x) Changes the integer x into a floating point number. (See also CINT).

CHAIN
Syntax: CHAIN f$
Action: Loads a GFA Basic program file into memory and starts it immediately it is loaded. If no extension is given '.GFA' is assumed.

Example: CHAIN "A:\EXAMPLE.GFA"

CHAR{x} Reads a string of bytes until a null byte is encountered, or writes the specified string of bytes and appends a null byte.

Example: PRINT CHAR{BASEPAGE+129} prints the command line.

CHDIR
Syntax: CHDIR "directory name"
Action: Changes the current directory.

Example: CHDIR "B:\TEST"

CHDRIVE
Syntax: CHDRIVE n or n$
Action: Sets the default disk drive 0=current, 1=A, 2=B etc.

Example:
```
CHDRIVE 1
PRINT DFREE(0)
PRINT DIR$(2)
CHDRIVE "C:\"
```

CHR$(x) Returns the character from a specified ASCII code.

Example: PRINT CHR$(65) !PRINTS A

CINT(x) Changes a floating point number into a rounded integer. (See also CFLOAT).

CIRCLE
HOW: CIRCLE x,y,r[,w1,w2]
Action: Draws a circle with centre coordinates at x,y and a radius r. Additional start and end angles w1 and w2 can be specified to draw a circular arc.

Example: CIRCLE 320,200,100

CLEAR
Syntax: CLEAR
Action: Clears all variables and fields.

CLEARW
Syntax: CLEARW n
Action: Clears the contents of the window numbered 'n'

CLIP x,y,w,h [OFFSET x0,y0]
CLIP x1,y1 TO x2,y2 [OFFSET x0,y0]
CLIP #n [OFFSET x0,y0]
CLIP OFFSET x,y
CLIP OFF

This group of commands provide 'Clipping' functions,ie. the limiting of graphic display within a specified rectangular screen area.
The command CLIP x,y,w,h defines the clipping rectangle starting at upper left coordinates x,y and extends w wide and h high.
The next command CLIP .. TO .. allows input of the diagonally opposite corners of the clipping rectangle.
The third variant makes it possible to dfine the limits of the window 'n'. The optional additional command OFFSET x0,y0 makes it possible to redefine the origin of the graphic display. If used in its own right this command sets the origin for graphic display at x0,y0.
The command CLIP OFF turns clipping off.

CLOSE
Syntax: CLOSE [#n]
Action: Close a data channel or a channel to a previously OPENed device. If the channel number is omitted, all opened channels are closed.

CLOSEW
Syntax: CLOSEW n
Action: Closes the window numbered n.

CLR
Syntax: CLR var [,var]
Action: Deletes and sets specified variables (not arrays) to 0.

CLS
Syntax: CLS [#n]
Action: Clears the screen [numbered n].

```
COLOR
Syntax:    COLOR color
Action:    Sets the colour for drawing/text. (0-15).

COMBIN
Syntax:    z=COMBIN(n,k)
Action:    Calculates the number of combinations of n elements to  the
           kth class without repetitions. Defined as z=n!/((n-k)!*k!).

CONT
Syntax:    CONT
Action:    Resumes execution of a program.
           Continue the execution of a program after interruption.

CONTRL     Address of the VDI control table.
           With  an  index  after  this  function,   the   appropriate
           parameter block is accessed directly.
           CONTRL(2)=x is the same as
           DPOKE CONTRL+4,x

COS
Syntax:    COS(x)
Action:    Returns the cosine of value x (radians)

COSQ(x)    Returns the cosine of value x  from  an internal  table  in
           steps  of 16th of a degree so is 10 times faster than  COS.
           (in degrees).

CRSCOL  CRSLIN
Syntax:    CRSCOL
           CRSLIN
Action:    Returns current cursor line and column. (see also PRINT
           AT).

CURVE
HOW:       CURVE x0,y0,x1,y1,x2,y2,x3,y3
Action:    The  BEZIER-Curve starts at x0,y0,  and ends at  x3,y3.
           The curve  at  x0,y0 is at a tangent with a line from
           x0,y0  to x1,y1;  and  at  x3,y3 is at a tangent with
           a  line  between x3,y3 and x2,y2.

Example:
          x0=10
          y0=10
          x1=50
          y1=110
          x2=150
          y2=200
          x3=350
          y3=300
          LINE x0,y0,x1,y1
          LINE x2,y2,x3,y3
          CURVE x0,y0,x1,y1,x2,y2,x3,y3


CVI   CVL   CVS   CVF   CVD
Syntax:    CVI(x$) .... CVD(x$)
Action:    Changes character strings into numeric variables.
      CVI  Changes a 2-byte string into an integer
      CVL     "    " 4-byte    "    "   "    "
      CVS     "    " 4-byte basic string into a floating  point number
```

```
      CVF     "    " 6-byte    "      "    "  a GFA 1 or 2    "
      CVD     "    " 8-byte    "      "    "  a GFA 3 floating point.

*****************************************************************


DATA
Syntax:    DATA [CONST[,CONST] ...]
Action:    Used as memory variables which can be read by the READ
           command. The constants are separated by commas.

Example:   For i=1 to 3
               READ A
               PRINT A
           Next i
           DATA 1,2,3,4

DATE$
Syntax:    DATE$
Action:    Returns the system date.

           DATE$=date$        Sets the system date.  (either  DD.MM.YY
                              (UK) orMM.DD.YY (US)
                              The format depends on MODE setting.

DEC
Syntax:    DEC var
Action:    Reduces the value of 'var' by 1

DEFAULT    See SELECT

DEFBIT
DEFBYT
DEFWRD
DEFFLT
DEFSTR
           The  instruction  DEFxxx  sets the varaible  type  to  that
           specified.

Example:   DEFBIT "a-z"       defines all variables as boolean.

DEFFILL
Syntax:    DEFFILL [col],[style],[pattern] or  DEFFILLL [col],A$
Action:    Sets fill colour and pattern, or allows user-defined
           patterns.
           'style' - 0=empty, 1=filled, 2=dots, 3=lines, 4=user
           24 dotted patterns and 12 lined can by chosen.
           A user-defined fill pattern is defined in the second
           variation - DEFFILL col,A$ by defining a 16 x 16 bit
           pattern array.

DEFFN
Syntax:    DEFFN func [(x1,x2,..)]=expression
Action:    Allows  the definition of single line functions.  The  term
           'expression' can be any numeric or string expression.

Example:   DEFFN test(y,a$)=x-y+LEN(a$)
           x=2
           PRINT @test(4,"abcdef")
           See also FN

DEFLINE
```

Syntax: DEFLINE [style],[thickness],[begin_s,end_s]
Action: Sets line style, width & type of line start and end.
'style' determines the style of line:
1 Solid line
2 Long dashed line
3 Dotted
4 Dot-dashed
5 Dashed
6 Dash dot dot ..
7 User defined
'thickness' sets width in pixels (odd numbers only).
The start and end symbols are defined by the last parameter, and can be:
0 Square
1 Arrow
2 Round

DEFLIST
Syntax: DEFLIST x
Action: Defines the program listing format.

x	Command	Variable
0	PRINT	abc
1	Print	Abc
2	PRINT	abc#
3	Print	Abc#

DEFMARK
Syntax: DEFMARK [C],[A],[G]
Action: Sets colour,type and size of the corner points to be mark using the command polymark
'C' is the colour register number
'A' defines the type of mark. the following types are possible :-
1=dot
2=plus sign
3=asterisk
4=square
5=cross
6=hash
all other values return the asterisk symbol
'G' sets the size of mark

DEFMOUSE
Syntax: DEFMOUSE n or DEFMOUSE a$
Action: Chooses a pre-defined mouse form or defines a new one
the following mouse forms are available :-

0=arrow	1=expanded (rounded) X
2=bee	3=pointing hand
4=open hand	5=thin crosswire
6=thick crosswire	7=bordered crosswire

A mouse can be defined by the command defmouse a$
16*16 dots are available to create a shape. Also
a 'mask' must be defined so that the cursor remains visible when it is the same colour as the background
one of the 256 dots must be defined as the starting point to which the mouse functions will relate.

Example:

```
DEFMOUSE 2
```

```
            PAUSE 1
            m$=MKI$(0)+MKI$(0)+MKI$(1)+MKI$(0)+MKI$(1)
            FOR i%=1 TO 16
               m$=m$+MKI$(65535)
            NEXT I%
            FOR i%=1 TO 16
               m$=m$+MKI$(1)
            NEXT i%
            PBOX 200,150,400,250
            DEFMOUSE m$
            REPEAT
            UNTIL MOUSEK
```

DEFNUM
Syntax: DEFNUM n
Action: Affects output of numbers by the PRINT command and its variants. All numbers are outputted to occupy n character positions, not counting the decimal point.

DEFTEXT
Syntax: DEFTEXT [colour],[attr],[angle],[height],[fontnr]
Action: Defines the colour,style,rotation and size of text to be printed using the text command.

'colour' colour register number (0-15).
'attr' text style - 0=normal 1=bold 2=light 4=italic 8=underlined 16=outlined (can be combined).
'angle'= rotation only the following are possible :-
0 deg (0), 90 deg (900), 180 deg (1800), 270 deg (2700)
'height' size of text - 4=icon, 6=8*8, 13=std, 32=enlarged.
'fontnr' - the number of a desired character set. This font must have been previously installed (See also GDOS, VST_LOAD_FONT, VQT_NAME).

Example:

```
            FOR i%|=0 TO 5
               DEFTEXT 1,2^i|,0,13
               TEXT 100,i|*16+100,"This is text attribute "+STR$(i|)
            NEXT i|
```

DEG(x) Converts x from radians to degrees. See also RAD.

DELAY x Suspends program operation for x seconds (with a theroetical resolution in milliseconds). See aslo PAUSE.

DELETE
Syntax: DELETE x(i)
Action: Removes the ith element of array x. All elements with a larger index are shifted down one position. See also INSERT.

DFREE
Syntax: DFREE(n)
Action: Locates free space on a disc 'n' = drive number (0-15)

DIM
Syntax: DIM var(indices)[,var(indices),.....]
Action: Sets the dimensions of an array or string array.

DIM?
Syntax: DIM?(field())

```
Action:    Determines the number of elements in an array.
           Note - arrays have an element '0'.

DIR
Syntax:    DIR "filespec" [TO "file"]
Action:    Lists the files on a disc.  The output can be directed to a
           file or other device.

Example:   "LST:" See also FILES.

DIR$
Syntax:    DIR$(n)
Action:    Names the active directory for drive 'n'
           'n' is drive number (1=A:, 2=B: ...).

DIV
Syntax:    DIV var,n
Action:    Divides the value of var by n. As var=var/n but 30% faster.

DIV()
Syntax:    a%=DIV(x,y)
Action:    See DIV above.


DMACONTROL
Syntax:    DMACONTROL ctrlvar
Action:    Controls the DMA sound on the STE.
           cntrlvar = 0 - stop sound
                      1 - Play sound once
                      2 - Play sound in a loop

DMASOUND
Syntax:    DMASOUND beg,end,rate[,ctrl]
Action:    Output of DMA sampled sound on the STE.
           beg  - Sample start address.
           end  - Sample end address.
           rate - Sample rate 0=6.25 kHz,  1=12.5 kHz, 2=25 kHz,
                  3=50kHz.
           ctrl - See DMACONTROL above.

Example:   'Try each of the DMASOUND lines below.
           n%=360*32
           DIM a|(n%)
           'DMASOUND V:a|(0),V:a|(n%),0
           'DMASOUND V:a|(0),V:a|(n%),1
           'DMASOUND V:a|(0),V:a|(n%),2
           DMASOUND V:a|(0),V:a|(n%),3,3
           FOR i%=0 TO n%
            a|(i%)=128+SINQ(i%*i%/7200)*127
           NEXT i%
           REPEAT
           UNTIL MOUSEK
           DMACONTROL 0


DO....LOOP
Syntax:    DO
           (instructions)
           LOOP
Action:    Creates an endless loop, exit only with EXIT IF or GOTO.

DO UNTIL condition
```

DO WHILE condition

The commands DO and LOOP can be extended using UNTIL and WHILE. The loop DO WHILE causes a loop as long as the condition is true. See also LOOP WHILE, LOOP UNTIL.

DOUBLE{x} Reads/writes an 8-byte floating point variable in IEEE double precision format. (See also BYTE{}, CARD{}, INT{}, LONG{}, {}, FLOAT{}, SINGLE{}, CHAR{})

DOWNTO Used within a FOR..NEXT loop as a counter. Instead of using step -1, the command DOWNTO is used, however STEP is not possible with DOWNTO. eg:
FOR c=100 DOWNTO 1
is the same as FOR c=100 TO 1 STEP -1

DPEEK(x) Reads 2 bytes from address x (a word). Works in supervisor mode.

DPOKE x,y Writes y as a 2 byte word to address x. To work in supervisor mode use SDPOKE ..

DRAW
Syntax: DRAW [TO] [x,y]
DRAW [x1,y1][TO x2,y2][TO x3,y3][TO..]
Action: Draws points and connects two or more points with straight lines. DRAW x,y is the same as PLOT x,y. DRAW TO x,y connects the point to the last set point (set by PLOT, LINE or DRAW).

DRAW expression
DRAW(i)
SETDRAW

These instructions give a turtle like approach to drawing. An imaginary pen is moved over the screen and draws relative to the last point. The 'expression' is a LOGO type of convention controlled by graphic commands. The pen leaving a trail over the 'paper' leaves a graphic image. The statement below is an wxample of how these commands can be used.

Example: DRAW "PU FD 40 PD FD 40"

The available commands are:

FD n	ForwarD	Moves the 'pen' n pixels forward
BK n	BacKward	Moves the 'pen' n pixels backwards
SX x	Scale x	Scales the 'pen' movement for FD and BK
SY y	Scale y	by the specified factor. Use SX0 or SY0 to turn off scaling.
LT a	Left turn	Turns the 'pen' left through the specified angle a, given in degrees.
RT a	Right turn	As LT but turns right.

TT a	Turn To	Turns the 'pen' to the absolute angle 'a' in degrees. (see below)

```
                    0
                    |
                    |
          270 -- zero point -- 90
                    |
                    |
                   180
```

MA x,y	Move Absolute	Moves the 'pen' to the absolute coordinates x,y
DA x,y	Draw Absolute	Moves the 'pen' to the absolute coordinates x,y and draws a line in the current colour from the last position to the point x,y
MR xr,yr	Move Relative	Moves the 'pen' position in the x and y directions relative to the last position.
DR xr,yr	Draw Relative	Moves the 'pen' by the specified displacement relative to its last position and draws a line in the current colour from the last position to this point.

The command SETDRAW x,y,w is an abbreviation for the expression DRAW "MA",x,y,"TT",w.

CO c	Colour	Sets 'c' as the character colour (see parameters for COLOR command).
PU	Pen Up	Lifts the 'pen' from the 'paper'.
PD	Pen Down	Lowers the 'pen' onto the 'paper'.

Additionally the following interrogation functions are available:

DRAW(0)	returns the x position.
DRAW(1)	returns the y position.
DRAW(2)	returns angle in degrees.
DRAW(3)	returns the X axis scale factor.
DRAW(4)	returns the Y axis scale factor.
DRAW(5)	returns the pen flag (-1=PD, 0=PU).

All these functions return floating point values.

Example:

```
DRAW "ma 160,200 tt 0"
FOR i&=3 TO 10
  polygon(i&,90)
NEXT i&

PROCEDURE polygon(n&,r&)
LOCAL i&
FOR i&=1 TO n&
  DRAW "fd",r&," rt ",360/n&
```

```
          NEXT i&
          RETURN

EDIT
Syntax:   EDIT
Action:   Returns to the editor.
          When used in direct mode the command allows a return
          to the editor. In a program, is the same as END but
          without the program end alert box.

ELLIPSE
Syntax:   ELLIPSE x,y,rx,ry [,phi0,phi1]
Action:   Draws an ellipse at x,y, having 'rx' as length of the
          horizontal axis and 'ry' as length of the vertical axis
          The optional angles 'phi0' & 'phi1' give start and end
          angles in tenths of a degree, to create an elliptical arc.

ELSE      See command IF.
ELSE IF    "     "    "


END
Syntax:   END
Action:   Closes all files and terminates program execution.


ENDFUNC   See command FN.

ENDIF     See command IF.

ENDSELECT See command SELECT.


EOF
Syntax:   EOF (#n)
Action:   Determines whether the file pointer for the file with
          channel number 'n' is at End Of the File.  Returns -1 if it
          is, otherwise 0.


EQV
Syntax:   x EQV y
Action:   The operator EQV (equivalence) produces a TRUE result  only
          if  the  arguments of both are either TRUE or  both  FALSE.
          (same as NOT(x XOR y)).

EQV(x,y)
Syntax:   EQV(x,y)
Action:   Sets a bit of the result if the appropraite bits in x and y
          are both set, or both reset. eg:
          PRINT BIN$(EQV(15,6),4) prints 0110


ERASE
Syntax:   ERASE field()
Action:   Deletes an array and releases the dimensioned area.


ERR
Syntax:   ERR
Action:   Returns the error code of any error that has occurred.
```

ERR$
Syntax: ERR$(x)
Action: Returns, as a string, the GFA Basic error mesage x.

ERROR
Syntax: ERROR n
Action: Simulates the occurrence of the error with the error code 'n' and the appropriate error message is then displayed.

EVEN
Syntax: EVEN n
Action: Determines if a number is even. Returns TRUE (-1) and FALSE(0). Also see ODD

EVERY
EVERY CONT
EVERY STOP

Syntax: EVERY ticks GOSUB proc
Action: The command EVERY causes the procedure 'proc' to be called every 'ticks' clock units. The clock unit is defined as one two-hundredth of a second. But a branch can only be done every fourth clock unit, resulting in an effective time resolution of one fiftieth of a second.
Using EVERY STOP, the calling of a procedure can be prevented. With EVERY CONT this is again allowed. See also AFTER.

EVNT_BUTTON
Syntax: EVNT_BUTTON(clicks,mask,state[,mx,my,button,k_state])
Action: Waits for one or more mouse clicks, and returns the number of clicks.
INPUTS
clicks - Maximum allowable clicks.
mask - Mask for the desired mouse key:
Bit 0 = 1 : Left Button
Bit 1 = 1 : Right Button
state - Desired status, in order to terminate the event.
Bit allocation as for mask.
RETURNED VALUES:
mx - x coordinate of mouse pointer on event termination
my - y coordinate of mouse pointer on event termination
button - state of mouse button, bit allocation as for mask.

k_state- Condition of keyboard shift keys:
Bit 0 = Right shift key
Bit 1 = Left shift key
Bit 2 = Control key
Bit 3 = Alternate key

EVNT_DCLICK
Syntax: EVNT_DCLICK(new,get_set)
Action: Sets the speed for double-clicks of a mouse button.
Returns the speed.
new - new speed (0-4)
get_set - determine whether to set or read. 0=read,1=set.

EVNT_KEYBD
Syntax: EVNT_KEYBD()
Action: Waits for a key to be pressed and returns a word-sized value. The low order byte contains the ASCII code, the high byte contains the keyboard scan code.

EVNT_MESAG
Syntax: EVNT_MESAG(adr_buffer)
Action: Waits for the arrival of a message in the event buffer. The returned value is always 1.
adr_buffer is the address of a 16 byte buffer for the message. If 0 is used, then the sytem message buffer is used, ie. MENU(1) to MENU(8).

EVNT_MOUSE
Syntax: EVNT_MOUSE(flag,mx,my,mw,mh,mcur_x,mcur_y,button,k_state)
Action: Waits for the mouse pointer to be located inside (or optionally, outside) a specified rectangular area of the screen.
The returned value is always 1.
INPUTS:
flag - Presence inside(0) or outside(1) the desired area.
mx,my - Coordinates of top left corner of recatngle.
mw - Width of rectangle.
mh - Height of rectangle.
OUTPUTS:
mcur_x x coordinate of mouse when event occurs.
mcur_y y coordinate of mouse when event occurs.
button same as for mask in EVNT_BUTTON.
k_state same as for k_state in EVNT_BUTTON.

EVNT_MULTI
HOW: EVNT_MULTI(flag,clicks,mask,state,m1_flags,m1_x,
m1_y,m1_w,m1_h,m2_flags,m2_x,m2_y,m2_w,m2_h,adr_buffer,
count[,mcur_x,mcur_y,button,k_state,key,num_clicks])
Action: Waits for the occurence of selected events. Returns the event which actually occured, with bit allocation as for 'flag' below:
INPUT:
flag Sets the event(s) to be waited for as follows:

BIT 0 keyboard	MU_KEYBD
BIT 1 mouse button	MU_BUTTON
BIT 2 first mouse event	MU_M1
BIT 3 second mouse event	MU_M2
BIT 4 report event	MU_MESAG
BIT 5 timer	MU_TIMER

OUTPUT:
num_clicks: number of expected mouse clicks

The parameters are already described for EVNT_MESAG, etc..
With ON MENU, which uses this routine internally, the parameters are installed for the instruction ON MENU xxx GOSUB.

MENU(1) to MENU(8)	Message buffer.
MENU(9)	Returned value.
MENU(10)=mcur_x	x mouse position.
MENU(11)=mcur_y	y mouse pos.
MENU(12)=button	Mouse state button.

```
           MENU(13)=k_state    Shift key status.
           MENU(14)=key        ASCII and scan code.
           MENU(15)=num_clicks Number of mouse clicks.


EVNT_TIMER
Syntax:    EVNT_TIMER(count)
Action:    The  function  waits  for a period  of  time  expressed  in
           'count' millisecondes.
           The returned value is always 1.


EXEC
Syntax:    EXEC flg,name,cmd,env
           EXEC (flg,name,cmd,env)
Action:    Loads and executes machine code programs or compiled
           programs from disc.
           flg=0 - load and go.
           flg=3 - load only.
           'name' - the name of the program.
           'cmd'  - command line (see BASEPAGE).
           'env'  - environment string (usually just "").
           The named program is loaded from disc, the absolute
           addresses are relocated, a basepage is created, and if
           required the program run.


EXIST
Syntax:    EXIST ("filespec")
Action:    Determines whether a particular file is present on
           a disc. If present -1 is returned, else 0 is returned.


EXIT
Syntax:    EXIT IF Condition
Action:    Enables the exit from a loop.
           If the EXIT command is met within a loop and the exit
           condition is met, the program continues from the first
           command after the loop.
           This command allows any loop to be left ie: FOR...NEXT
           DO...LOOP,  REPEAT...UNTIL AND  WHILE...WEND.


EXP
Syntax:    EXP(X)
Action:    Calculates the value of an exponent x.


FACT
Syntax:    x=FACT(n)
Action:    Calculates  the factorial (n!) of n and returns the  result
           in x. (0!=1).


FALSE
Syntax:    FALSE
Action:    Constant 0. This is simply another way of expressing
           the value of a condition when it is false and is equal
           to zero (see also TRUE).


FATAL
```

Syntax: FATAL
Action: Returns the value 0 or -1 according to the type of error.
On normal errors the function returns 0. The value -1 is returned on all errors where the address of the last executed command is no longer known.
Normally this is an operating system error which would lead to the 'bomb' errors and the breakdown of the program.

FGETDTA
Syntax: n%=FGETDTA()
Action: Returns the DTA (Disk Transfer Address).

FIELD
Syntax: FIELD #n,num AS svar$ [,num AS svar$,...]
FIELD #n,num AT(x)[,num AT(x),...]
Action: Divides records into fields.
'n' is the channel number of a file previously OPENed.
The integer expression 'num' determines the field length.
'Svar' is a string variable containing data for one field of a data record.
The section 'num AS svar$' can be repeated if the record is to be divided into several fields. The sum of the fields should equal the record length.
By using AT() instead of AS, numeric variables can be read and written. 'num' contains the length of the varaible type. The brackets contain a pointer to the variable.
eg:
FIELD #2,4 AS a$,2 AT(*b&),8 AT(*c#),6 AS d$

FILES
Syntax: FILES p$ [TO name$]
Action: Lists the files on a disk.
Like DIR but more detailed data listing.

FILESELECT
Syntax: FILESELECT [#title$],path$,default$,name$
Action: Creates a fileselect box on the screen.
Title$ can be a string of max 30 characters. Allows a header to be placed in the fileselect box (TOS 1.4 and above).
path$ is the drive and path - if none specified then the default drive is assumed.
default$ contains the name of the file to apppear in the selection line. ("" for no default).
name$ contains the selected file, either an empty string if CANCEL is selected, or the file name selected.
See also FSEL_INPUT

Example:

```
FILESELECT #"LOAD File","A:\*.PRG","GFABASIC.PRG",name$
```

FILL
Syntax: FILL x,y

Action: Fills a bordered area with a pattern commencing at the co-ordinates 'x,y'.
Fill pattern can be chosen with the command DEFFILL.

FIX
Syntax: FIX(x)
Action: Returns the integer of 'x' after it has been rounded. Same as INT(x) for positive numbers but for negative numbers INT(-1.99)=-2 AND FIX(-1.99)=1.
FIX is identical to the function TRUNC and complements FRAC.

FLOAT{x} Reads/writes an 8-byte variable in Basic v3 floating point format. (See also BYTE{}, CARD{}, INT{},LONG{}, {}, SINGLE{}, DOUBLE{}, CHAR{})

FN
Syntax: FN func[(y1,y2...)]
Action: Call to a defined DEFFN function or a FUNCTION. (you can also use @). See also DEFFN, FUNCTION.

FOR...NEXT
Syntax: FOR c=b TO e [STEP s]
instructions
NEXT c
Action: Creates a loop which is executed as many times as specified at the beginning of the loop.

FORM INPUT
Syntax: FORM INPUT n,a$
Action: Enables the insertion of a character string (limited to 255 characters in length) during program execution.
'n' = the maximum length of the character string.
a$ is the name of the string variable.

FORM INPUT AS
Syntax: FORM INPUT n AS a$
Action: Similar to FORM INPUT, except the current value of a$ is displayed, and can be edited.

*** the following 7 commands are part of the AES FORM Library commands, and are similar to C bindings for calling these AES functions ***

FORM_ALERT
Syntax: a%=FORM_ALERT(button,string$)
Action: Creates an alert box.
button = number of the default button (0-3).
string$ = string defining the message in the alert. (in C format) - note that the square brackets are part of the string:
[i][Message][Buttons]
where i = the required alert symbol - see ALERT.
Message is a string max 30 characters.
Buttons = the name of the 3 buttons.
A good use of this command is in trapping errors:

```
Example:   ~FORM_ALERT(1,ERR$(ERR))


FORM_BUTTON
Syntax:    FORM_BUTTON(tree,obj,clicks,new_obj)
Action:    Make inputs in a form possible using the mouse.
           INPUTS:
           tree   -  address of the object tree
           obj    -  current object number
           clicks -  max expected number of mouse clicks
           OUTPUT:
           new_obj-  next object to be edited.
           Returns 0 if the FORM was exited, otherwise >0.

FORM_CENTER
Syntax:    FORM_CENTER(tree,fx,fy,fw,fh)
Action:    Centers the tree, and returns its coordinates.
           INPUT: tree  - address of the object tree.
           OUTPUTS:
           fx,fy    coordinates of top left corner
           fw,fh    form width and height.
           returns a reserved value (always 1).


FORM_DIAL
HOW:       FORM_DIAL(flag,mi_x,mi_y,mi_w,mi_h,ma_x,ma_y,ma_w,ma_h)
Action:    Release (or reserve) a rectangular screen area and draw  an
           expanding/shrinking rectangle.
           Returns 0 if an error occured.
           flag     function
           0        FMD_START   reserve a display area.
           1        FMD_GROW    draw expanding box.
           2        FMD_SHRINK  draw shrinking box.
           3        FMD_FINISH  release reserved display area.
           mi_x,mi_y  top left corner of rectangle at min size
           mi_w,mi_h  width & height  "     "       "  "    "
           ma_x,ma_y  top left corner of rectangle at max size
           ma_w,ma_h  width & height   "     "      "  "    "


FORM_DO
Syntax:    FORM_DO(tree,start_obj)
Action:    Pass  management  of FORM over to the AES until  an  object
           with EXIT or TOUCH EXIT status is clicked on.
           Returns  the number of the object whose clicking or  double
           clicking  caused the function to end.  If it was  a  double
           click, bit 15 will be set.
           tree      = address of the object tree.
           start_obj = Number of the first editable field (if there is
           one).


FORM_ERROR
Syntax:    FORM_ERROR(err)
Action:    Displays the ALERT associated with the error numbered err.

Example:   PRINT FORM_ERROR(10)


FORM_KEYBD
Syntax:    FORM_KEYBD(tree,obj,next_obj,char,new_obj,next_char)
Action:    Allows a form to be edited via the keyboard.
```

Returns 0 if the FORM was exited, otherwise >0.
tree address of the object tree
obj number of the object to be edited
next_obj number of the next EDITable object in the tree
char input character
new_obj object to be EDITed on the next call
returns next_char - next character (derived from keyboard)

This function is a subroutine of FORM_DO.

FRAC
Syntax: FRAC(x)
Action: Returns the digits after the decimal point in a number. 'x' can be any numeric expression. if 'X' is an integer then a zero is returned, therefore FRAC(x)=x-TRUNC(x)

FRE
Syntax: f%=FRE(X) or f%=FRE()
Action: Returns the amount of memory free (in bytes). The parameter 'x' is disregarded, but if present a 'Garbage Collection' is carried out. (non current strings are freed from memory).

FSEL_INPUT
Syntax: n%=FSEL_INPUT(path$,name$,[button])
Action: Calls the AES fileselect library, to provide a fileselector.
The optional parameter 'button':
Returns a 1 or 0 depending whether 'OK' or 'Cancel' was clicked on.
ON ENTRY:
path$ = initial directory path
name$ = Default name
ON EXIT:
path$ = final directory path
name$ = chosen filename.
button = 1 if 'OK'
= 0 if 'Cancel'

FSETDTA
Syntax: ~FSETDTA(addr)
Action: Sets the address of the DTA. (See also FGETDTA).

FSFIRST
Syntax: FSFIRST(p$,attr)
Action: Searches for the first file on a disk to fulfill the criteria specified in p$ (eg: "C:*.GFA"). If found, the filename and attributes are to be found in the DTA.
The parameter 'attr' is the file atributes to search on.

FSNEXT
Syntax: FSNEXT()
Action: Search for the next file which fulfills the conditions of FSFIRST.

Example:
~FSETDTA(BASEPAGE+128)

```
e%=FSFIRST("\*.GFA",-1)           ! all bits set
DO UNTIL e%
  IF BYTE{BASEPAGE+149} AND 16    !if its a folder
    PRINT "*";CHAR{BASEPAGE+158}  ! indicate by a star
  ELSE                            ! otherwise
    PRINT 'CHAR{BASEPAGE+158}     ! a space before
'                                 ! the filename
  ENDIF
  e%=FSNEXT()                     ! continue search
LOOP
```

FULLW

Syntax: FULLW [#]n

Action: Enlarges window 'n' to full screen size. 'n' is the window number. If the window has not yet been opened, this takes place automatically.

FUNCTION

The commands FUNCTION and ENDFUNC form a subroutine, in a similar manner to PROCEDURE. The name of the subroutine and, optionally, the list of varaibles are given after FUNCTION command. Calling the subroutine is done by the use of @ or FN and the function name followed by a list of parameters if necessary.
If the command RETURN is met during program execution, the the value given after it or the value of the named variable is returned. In a function, RETURN can be used several times, with IF or the like. A function cannot be terminated without a RETURN command being before the ENDFUNC command. In a function name ending with the $ character the function returns a string result.

Example:

```
f1%=@fac_loop(15)
PRINT "loop: fac(15) = ";f1%
'
FUNCTION fac_loop(f%)
 w=1
 FOR J%=1 TO f%
   MUL w,j%
 NEXT j%
 RETURN w
ENDFUNC
```

GB — Address of the AES Parameter Block
This (unlike the other AES address blocks) cannot be used with index.

GCONTRL — Address of the AES control block. With index (GCONTRL(2)) the elements can be accessed directly.

GDOS? — Returns TRUE (-1) if GDOS is resident and FALSE (0) otherwise.

GEMDOS

Syntax: GEMDOS(n[,x,y])

Action: To call the GEMDOS routines. The optional parameter list

can be prefixed with W: or L: to denote word or longword parameters. (if non given, default is W:)

Example:

```
DO UNTIL GEMDOS(17)
  ALERT 1,"Printer not ready",1,"retry|break",d%
LOOP UNTIL d%=2
```

GEMSYS
Syntax: GEMSYS n
Action: Calls the AES routine 'n'. The parameters necessary for the operation of the routine must first be placed in the appropriate AES parameter blocks.

GET
Syntax: GET x1,y1,x2,y2,sections$
Action: GET puts a section of the screen into a string variable 'section$' (x1,y1 and x2,y2 are coordinates of diagonally opposite corners). See also PUT.

GET #
Syntax: GET #n[,r]
Action: Reads a record from a random access file.
'n' is the channel number (1 to 99)
'r' is number of the record to be read (1 to 65535)
If 'r' is not given then the next record in the file will be read. (See also PUT #).

GETSIZE
Syntax: bytes%=GETSIZE(x1,y1,x2,y2)
Action: The TT does not have a constant screen memory of 32000 Bytes like the ST. A screen could require much more memory (153600 Bytes). The commands GET and PUT are limited to 32000 Bytes and therefore a function has been introduced to support the larger screen resolutions that require more than 32000 Bytes.
This function will return the number of Bytes required by the screen between the coordinates x1,y1,x2,y2. Several GET or PUT commands could be used to address the entire screen.

GINTIN Address of the AES Integer input block. (Can be used with index GINTIN(0)).

GINTOUT Address of the AES Integer output block. (Can be used with index GINTOUT(0)).

GOSUB
Syntax: GOSUB name [(LIST OF EXPRESSIONS)]
Action: Branches to the procedure called 'name'.
A procedure name can begin with a digit and contain letters, numbers, dots and the underline dash.
'(list of expressions)' contains the values of any local variables to be passed to the procedure.
When the interpreter comes across a GOSUB command,

it branches to the procedure named in the gosub.
It is possible to call further procedures whilst in
a procedure. It is even possible to call the procedure
one is in at the time (recursive call).

GOTO
Syntax: GOTO label
Action: allows an unconditional jump to a label.
'label' must end in a colon and can consist of letters,
numbers, dots, dashes and can begin with a digit.

*** The following 10 functions form the Graphics library calls to the AES.

GRAF_DRAGBOX
Syntax: GRAF_DRAGBOX(iw,ih,ix,iy,rx,,ry,rw,rh[,last_ix,last_iy])
Action: Allows a rectangle to be moved about the screen with the mouse. Its movement is restricted to the interior of a larger specified rectangle. This function should only be called when the left mouse button is held down, as it terminates when the button is released.
Returns 0 if an error occurs.
INPUTS:
iw,ih width & height of the moving rectangle
ix,iy initial coords of top left corner of moving rectangle
rx,ry coords of top left corner of limiting rectangle
rw,rh width & height of limiting rectangle
OUTPUTS:
last_ix coords of top left corner of inside rectangle
last_iy when the function terminated.

GRAF_GROWBOX
Syntax: GRAF_GROWBOX(sx,sy,sw,sh,dx,dy,dw,dh)
Action: Draws an expanding rectangle.
Returns 0 if error occurs.
sx,sy Initial coords of top left corner of rectangle
sw,sh Initial width & height of rectangle
dx,dy Final coords of top left corner
dw,dh Final width & height

GRAF_HANDLE
Syntax: GRAF_HANDLE(char_w,char_h,box_w,box_h)
Action: Returns the ID number of the current VDI workstation and supplies the size of a character from the system set.
OUTPUTS:
char_w width in pixels of a character
char_h height
box_w width of a character cell
box_h height

GRAF_MKSTATE
Syntax: GRAF_MKSTATE(mx,my,m_state,k_state)
Action: This function supplies the current mouse coordinates and status of tht mouse buttons and shift keys.
Returns a reserved value (always 1)
OUTPUTS:
mx,my mouse coordinates

```
           m_state  mouse button status
                    bit 0 left button
                    bit 1 right button
           k_state  see k_state in function EVNT_BUTTON


GRAF_MOUSE
Syntax:    GRAF_MOUSE(m_form,pattern_adr)
Action:    This function allows the mouse shape to be changed.
           (similar command to DEFMOUSE)
           Returns 0 if an error occurs.
           m_form   number of the mouse pointer shape
               0    = Arrow
               1    = Double curly brackets
               2    = Busy bee
               3    = Pointing finger
               4    = Open hand
               5    = Thin cross hairs
               6    = Thick cross hairs
               7    = Outlined cross hairs
               255  = User defined
               256  = Hide mouse
               257  = Show mouse

           pattern_adr = address of bit information defining the mouse
                         pointer. 37 word-sized values as follows:
                    1 =  x coordinate of the action point
                    2 =  y     "      "   "    "      "
                    3 =  number of colour levels, always 1
                    4 =  mask colour, always 0
                    5 =  pointer colour, always 1
                 6 to 21 =  Mask definition (16 words ie.16x16 bits)
                22 to 37 =  Pointer def              "


GRAF_MOVEBOX
Syntax:    GRAF_MOVEBOX(w,h,sx,sy,dx,dy)
Action:    Draws a moving rectangle with constant width & height.
           Returns 0 on error.
           INPUTS:
           w,h      width & height of rectangle
           sx,sy    Initial coords of top left corner of rectangle
           dx,dy    Final coords of top left corner


GRAF_RUBBERBOX
Syntax:    GRAF_RUBBERBOX(tx,ty,min_w,min_h[,last_w,last_h])
Action:    This function draws an outline of a rectangle while the the
           left  mouse  button is held down.  The top left  corner  is
           fixed,  but the width & height of the rectangle change with
           the  position of the mouse.  This function should  only  be
           called  when  the left mouse button is  held  down,  as  it
           terminates when the button is released.
           Returns 0 if an error occurs.
           INPUTS:
           tx,ty         coords of top left corner
           min_w,min_h   minimum width & height of rectangle
           OUTPUTS:
           last_w        width of rectangle when function terminates
           last_h        height "     "       "     "          "
```

GRAF_SHRINKBOX
Syntax: GRAF_SHRINKBOX(sx,sy,sw,sh,dx,dy,dw,dh)
Action: Draws an shrinking rectangle.
Returns 0 if error occurs.
sx,sy Final coords of top left corner
sw,sh Final width & height
dx,dy Initial coords of top left corner of rectangle
dw,dh Initial width & height of rectangle

GRAF_SLIDEBOX
Syntax: GRAF_SLIDEBOX(tree,parent_obj,slider_obj,flag)
Action: This function (really belongs to the OBJECT library) moves one rectangular object within another, in a similar manner to GRAF_DRAGBOX. The object can only be moved vertically or horizontally, and must be a 'child' of the limiting object. This function should only be called when the left mouse button is held down, as it terminates when the button is released. Commonly used in movement of slider bars in windows.
Returns the position of the moving rectangle relative to the limiting one:
Horizontally: 0 = far left 1000 = far right
Vertically: 0 = top 1000 = bottom
INPUTS:
tree address of oobject tree
parent_obj object number of the 'limiting rectangle'
slider_obj " " " " moving rectangle
flag direction (0=horizontal, 1=vertical)

GRAF_WATCHBOX
Syntax: GRAF_WATCHBOX(tree,obj,in_state,out_state)
Action: This function (really belongs to the OBJECT library) monitors an object tree while a mouse button is pressed, checking whether the mouse pointer is inside or outside. When the mouse button is released, the status of the object takes one of two specified values (normal selected/normal), depending on whether the pointer was inside the object or outside.
Returns 1 if the mouse pointer was inside the object when the button was released, or 0 if it was outside.
INPUTS:
tree address of the tree
obj number of the object to be monitored
in_state Status (OB_STATE) to be given to the object if the mouse pointer is found within it.
out_state Status (OB_STATE) to be given to the object if the mouse pointer is found outside it.

GRAPHMODE
Syntax: GRAPHMODE n
Action: Sets the graphic mode 1 to 4.
1=replace 2=transparent
3=xor 4=reverse transparent

HARDCOPY
Syntax: HARDCOPY

Action: Prints the screen (same as pressing <ALT> & <HELP>).

HEX$
Syntax: HEX$(x[,y])
Action: Changes the value of 'x' into a string expression which contains the value of 'x' in hexadecimal form. The optional parameter y specifies the number of character positions (1 to 8) to be used.

HIDEM
Syntax: HIDEM
Action: Switches off the mouse pointer. (see also SHOWM).

HIMEM
Syntax: HIMEM
Action: Returns the address of the area of memory which is not required by GFA Basic. Normally 16384 bytes below the screen.

HLINE
Syntax: HLINE x1,y,x2,f,m,addr,num_pattern
Action: Similar to ALINE, but only horizontal lines can be drawn. x1 and x2 contain the x coordinates of the line start and end points and y the common y coordinate. f is the colour (0-15). m is the graphic mode. addr is the address of a block of memory which contains bit information for several line styles each of 16 bits. Which style is used for a given line depends on both the y coordinate and the parameter num_pattern. They are ANDed together and the resulting number used as an index to the style table.

HTAB Positions the cursor to the specified column. (Counting from 0). See also VTAB

IF
Syntax:
```
IF condition [ THEN ]
  program block
ELSE
  program block
ENDIF
```
Action: Divides a program up into different blocks depending on how it relates to the 'condition'.

ELSE IF condition
Enables nested IF's to be more clearly expressed in a program.

Example:
```
'the following code has no ELSE IF's
DO
  t$=CHR$(INP(2))
  IF t$="l"
    PRINT "Load text"
  ELSE
    IF t$="s"
      PRINT "Save text"
    ELSE
```

```
      IF t$="e"
        PRINT "Enter text"
      ELSE
         PRINT "unknown command"
      ENDIF
    ENDIF
  ENDIF
LOOP
```

The use of ELSE IF produces shorter code:

```
DO
  t$=CHR$(INP(2))
  IF t$="l"
      PRINT "Load text"
  ELSE IF t$="s"
      PRINT "Save text"
  ELSE IF t$="e"
      PRINT "Enter text"
  ELSE
      PRINT "unknown command"
  ENDIF
LOOP
```

IMP
Syntax: x IMP y
Action: The operator IMP (implication) corresponds to a logical consequence. The result is only FALSE if a FALSE expression follows a TRUE one. The sequence of the argument is important.

IMP()
Syntax: IMP(x,y)
Action: This function resets a bit if the appropriate bit in x is set and y is reset, otherwise the bit is set.

INC
Syntax: INC var
Action: Increases the value of 'var' by 1. the same as var=var+1 but executes aprox 3.5 times faster

INFOW
Syntax: INFOW n,"string$"
Action: Allocates the (NEW) information line to the window with the number 'n'.If the string is empty then the line is removed altogether. As the info line cannot be switched off and on when the window is opened, infow has to be used in front of OPENW when an information line is required. If the command INFOW,n,"" is used ("" = null string) before OPENW then the window will have no info line.

INKEY$
Syntax: INKEY$
Action: Reads a character from the keyboard. This function returns a string which is 2, 1 or 0 characters long.

Normal keys, return the ASCII code.
Function keys, HELP, UNDO etc. return two characters:
The ASCII code zero and then the key code.

Example:

```
DO
    t$=INKEY$
    IF t$<>""
        IF LEN(t$)=1
           PRINT "Character: ";t$,"ASCII code:";ASC(t$)
        ELSE
           PRINT "CHR$(0)+Scan code ";CVI(t$)
        ENDIF
    ENDIF
LOOP
```

INLINE
Syntax: INLINE var,length
Action: Reserves an area of memory within a program.
var is any integer variable
length is how much memory to reserve, less than 32700 bytes
The reserved area always starts at an even address and is initially filled with zeros. When implementing INLINE this address is written to the integer variable adr. When the program is loaded or saved, the reserved area is also.

Placing the cursor on the line containing the command INLINE and pressing the HELP key causes a new menu to appear with the entries: LOAD SAVE DUMP CLEAR. Load is used to load a machine code program or data into the reserved area, save saves the reserved area to disk (default filename extension .INL). DUMP printsout the reserved area in hex to the printer and CLEAR clears the araea of memory.

INP INP&(#) INP%(#)
OUT OUT& OUT%
Syntax: INP(#n)
OUT #n,a[,b,c...]
Action: Reads one byte from a file previously opened with OPEN.
Similarly OUT#n sends a byte to a file. The numerical expression n is the channel number under which the channel was opened.
INP and OUT without the # can be used for communicating with the screen, keyboard etc. eg INP(2) takes a character from the keyboard.
These functions cater for 16 and 32 bit input and output.

Example: a%=CVL(INPUT$(4,#1)) is replaced by a%=INP%(#1)

INP(n)
INP?(n)
OUT[#]n,a[,b..]
OUT?(n)

INP reads a byte from a peripheral device. The numerical expression n can accept values 0-5 (see table below), or contains a channel number(#n). The command OUT sends a byte to a peripheral device. You can send several bytes with one OUT command.
INP? and OUT? determine the input or output status of a

device. TRUE(-1) is device is ready ortherwise FALSE(0).

n	Device	
0	LST: (list)	Printer
1	AUX: (Auxiliary)	RS232
2	CON: (Console)	Keyboard/screen
3	MID: (MIDI)	MIDI Interface
4	IKB: (Intelligent kbd)	Keyboard processor
5	VID: (Video)	Screen

INPAUX$
INPMID$

Using these two commands, data can be read from the serial and MIDI interfaces.

Example:

```
DO
  PRINT INPAUX$;
LOOP
```

INPUT

Syntax: INPUT ["text",]x{,y,...]
INPUT ["text";]x[,y,...]

Action: Allows entry of data during program execution. If "text" is given, then a string prompt is displayed.

Example:

```
INPUT a$
INPUT "",b$
INPUT "enter two numbers: ";x,y
```

INPUT$

Syntax: INPUT$(count[,#n]))

Action: Reads 'count' characters from the keyboard and assigns them to a string. Optionally, if the channel number n is specified, the characters are read in from a previously OPENed channel.

INPUT #n,var1[,var2,var3,...]
LINE INPUT #n,a1$[,a2$,a3$,...]

These commands make it possible to take data from a previously OPENed device. Individual variables or variable lists (where the vars are separated by commas) can be input.

INSERT

Syntax: INSERT x(i)=y

Action: Inserts an element into an array. The value of the expression y is inserted into the position x(i). All elements of the array are shifted up by one position, and the last element lost. See also DELETE.

INSTR

Syntax: INSTR([n,]a$,b$) OR INSTR(a$,b$[,n])

Action: Searches to see if b$ is present in a$ and returns its position.

'n' is a numeric expression indicating the position in a$ at which the search is to begin. If 'n' is not given the search begins at the first character of A$.
If b$ is found in a$ the start position is returned.

INT
Syntax: INT(x)
Action: Determines the largest integer that is less than or equal to 'x'.

INTIN Address of the VDI integer input block. Also works with index INTIN(2).

INTOUT Address of the VDI integer output block. Also works with index INTOUT(2).

INT{x} Reads/writes a 2 byte signed integer from/to address x. (See also BYTE{}, CARD{}, LONG{}, {}, FLOAT{}, SINGLE{}, DOUBLE{}, CHAR{}).

KEYDEF
Syntax: KEYDEF n,s$
Action: Assign a string to a Function Key. The number n (1-20) is the function key (for 11 and above use shift + function key). The string is any string. Whilst using the Basic Editor, you must also hold down the Alternate key, otherwise the normal menu commands would not work!

KEYGET n
KEYLOOK n
KEYTEST n

KEYTEST is simialr to INKEY$ and reads a character from the keyboard. If no key was pressed since the last input (apart from Alternate, Control, Shift and Caps Lock) the returned value is zero, otherwise its value corresponds to the key in the fashion shown below for KEYGET.

KEYGET waits for a key to be pressed and then returns a long word value corresponding to the key. This 32 bit word is constructed as follows:
Bits 0-8 the ASCII code
Bits 8-15 Zero
Bits 16-23 the scan code
Bits 24-31 status of Shift, Control, Alternate, Caps lock as follows:

Bit	Key
24	Right shift
25	Left shift
26	Control
27	Alternate
28	Caps Lock

KEYLOOK allows a character to be read from the keyboard buffer, without changing the buffer's contents, as with KEYGET or INKEY$.

KEYPAD n Sets the usage of the numerical keypad. The numerical expression n is evaluated bit by bit and has the following meaning:

Bit	Meaning	0	1
0	NUMLOCK	On	Off
1	NUMLOCK	Not Switchable	Switchable
2	CTRL-KEYPAD	Normal	Cursor
3	ALT_KEYPAD	Normal	ASCII
4	KEYDEF without ALT	Off	On
5	KEYDEF with ALT	Off	On

With bit 0 set the keypad will act as a 'PC' keypad with numlock off ie. it responds with cursor movements.

With bit 1 set the 'PC' numlock mode can be toggled with Alternate and '-', otherwise it can't.

With bit 2 set, numlock is effecively switched off while the Control key is held down. Thus Control-4 (on the keypad) produces cursor movements.

With bit 3 set ASCII values for characters can be typed in with the Alternate key held down. When ALT is released the character appears.

With bit 4 set, the character strings assigned with KEYDEF to the function keys are output when the key is pressed. With bit 5 set, the Alternate key must aslo be held down.

The deafult when the ST is turned on is KEYPAD 0. with GFA Basic in operation it is 46.

KEYPRESS n

This simulates the pressing of a key. The character with the ASCII code contained in the lowest 8 bits of 'n' is added to the keyboard buffer. Additionally the status of the Shift, Control and Alternate keys may be passed in the high order bits as defined in KEYGET. If the ASCII code given is zero, a scan code may be passed in bits 16-23.

Example:

KEYPRESS &H3B0000 presses F1.

Example:

```
FOR i&=65 TO 90                 ! Simulates the pressing
  KEYPRESS i&                   ! of keys A-Z
NEXT i&
KEYPRESS 13                     !followed by Carriage Ret
INPUT a$                        !Characters are taken up
'                               !to  the first CR.
PRINT a$
```

KILL

Syntax: KILL "filespec"

Action: Deletes a file off disk (only one at a time).

L: Enable the passing of numerical expressions to Operating system functions or to machine code routines. L: is a long word.

Example: ~XBIOS(5,L:log_base%,L:phys_base%,-1)

LEFT$
Syntax: LEFT$(a$[,x])
Action: Returns the first [or first 'x'] character[s] of a string.

LEN
Syntax: LEN(x$)
Action: Returns the length of a string.

LET
Syntax: [LET] var=expression
Action: Assigns a variable with the value of an expression.

LINE
Syntax: LINE x,y,xx,yy
Action: Connects two points ('x,y' & 'xx,yy') with a straight line, and is identical to DRAW x,y TO xx,yy.

LINE INPUT
Syntax: LINE INPUT ["text",]var$ [,var$...]
         LINE INPUT ["text";]var$ [,var$...]
Action: Makes it possible to enter a string during program execution.
         This command is the same as INPUT except that a comma is taken as part of the entered string and not as a separator. Only <RETURN> is regarded as a separator.

LINE INPUT# See INPUT#

LIST
Syntax: LIST "filename"
Action: stores the program currently in memory to disk in ASCII format. If the 'filename' is an empty string (eg. "") then the listing is shown on the screen.
         In all other cases this command is the same as the editor menu option SAVE,A.
         Programs to be joined together using the command MERGE must be saved using LIST (or SAVE,A from the menu bar)

LLIST
Syntax: LLIST

```
Action:    Prints out the listing of the current program.  The setting
           for the type of output is controlled by the '.' commands in
           the editor.



LOAD
Syntax:    LOAD "filespec"
Action:    Loads a program into memory.



LOC
Syntax:    LOC(#n)
Action:    Returns the location of the file pointer for the file with
           the channel number 'n'
           The location is given in number of bytes from the start of
           the file.



LOCAL
Syntax:    LOCAL var [ ,var.... ]
Action:    Declares 'var' to be a local variable.



LOCATE
Syntax:    LOCATE row,column
Action:    Positions the cursor to the specified location.



LOF
Syntax:    LOF(#n)
Action:    Returns length of file OPENed for channel number 'n'.



LOG  LOG10
Syntax:    LOG(x)
           LOG10(x)
Action:    Determines the natural logarithm (log) or the logarithm
           base 10 (log10) of 'x'.



LONG{x}    Reads/writes a 4 byte integer from/to address x.
or         (See  also  BYTE{},   CARD{},   INT{},  FLOAT{},  SINGLE{},
{x}        DOUBLE{}, CHAR{} ).


LOOP       See DO

LOOP UNTIL condition
LOOP WHILE condition
           The  commands DO and LOOP can be extended using  UNTIL  and
           WHILE. LOOP WHILE causes the program to jump back to the DO
           command  as  long  as the condition  is  true.  LOOP  UNTIL
           requires the condition to be false to cause the loop back.
```

LPEEK(x) Reads a 4 byte integer from address x. (In supervisor mode)

LPENX For the STE. Returns the x coordinate of a light pen.
LPENY For the STE. Returns the y coordinate of a light pen.

LPOKE n,x Writes a 4 byte ineger 'x' to address n. Not in supervisor mode. (Add an 'S' to do in super mode ie. SLPOKE n,x).

LPOS
Syntax: LPOS(n)
Action: Returns the column in which the printer head (in the printer buffer) is located.

LPRINT
Syntax: LPRINT [expressions [,][;]['']]
Action: prints data on the printer.
'expression' is any number of expressions separated by commas or semicolons or apostrophes. If none of these is given a semicolon is assumed.

LSET
Syntax: LSET var=string
Action: Puts the 'string' in the string variable 'var' justified to the left.

L~A Returns the base address of the LINE-A variables.

MALLOC(x) Allocates an area of memory. (GEMDOS 72) If x is -1, then the function returns the largest contiguous free memory block. If x is positive, then MALLOC reserves that area of memory and returns its base address. If 0 is returned then there was a fault with the allocation.
See also RESERVE, MFREE, MSHRINK.

The following 33 commands are for the handling of MATRIXES

MAT ADD
MAT ADD a(),b()
MAT ADD a(),x
MAT ADD a()=b()+c()
MAT BASE
MAT CLR a()
MAT CPY
MAT CPY a([i,j])=b([k,l])[,h,w]
MAT DET x=a([i,j])[,n]
MAT INPUT #i,a()
MAT INV a()=b()
MAT MUL
MAT MUL a(),x

```
MAT MUL a()=b()*c()
MAT MUL x=a()*b()
MAT MUL x=a()*b()*c()
MAT NORM a(),0
MAT NORM a(),1
MAT ONE a()
MAT PRINT
MAT PRINT [#i]a[,g,n]
MAT QDET x=a([i,j])[,n]
MAT RANG x=a([i,j])[,n]
MAT READ
MAT READ a()
MAT SET a()=x
MAT SUB
MAT SUB a(),b()
MAT SUB a(),x
MAT SUB a()=b()-c()
MAT TRANS a()[=b()]
MAT XCPY
MAT XCPY a([i,j])=b([k,l])[,h,w]
```

Linear operations with vectors and matrices.
All THE MAT functions described relate only to one and/or two-dimensional fields with floating point variables.

System commands

```
MAT BASE 0
MAT BASE 1
```

The MAT BASE command can only sensibly be used when OPTION BASE 0 has been activated. In this case, MAT BASE 1 can be used to set the offset for the start of the row and column indexing of one or two-dimensional fields with floating point variables to 1 for the matrix operations. MAT BASE 0 resets this offset to 0 after a MAT BASE 1.

The setting made with MAT BASE n affects the following commands

```
MAT READ
MAT PRINT
MAT CPY
MAT XCPY
MAT ADD
MAT SUB
MAT MUL
```

The default is MAT BASE 1.

Generating commands

```
MAT CLR a()
MAT SET a()=x
MAT ONE a()
```

a: Name of field with numeric variables
x: aexp

MAT CLR a() corresponds to an ARRAYFILL a(),0, i.e. the command sets all elements in the field (matrix or vector) a() to a value of 0.

MAT SET a()=x corresponds to an ARRAYFILL a(),x, i.e. the command sets all elements in the field a() (matrix or vector) to the value x.

MAT ONE a() generates from a square matrix a() a uniform matrix, i.e. a square matrix in which elements a(1,1),a(2,2),...,a(n.n) are all equally 1 and all other elements equally 0.

Write and Read commands

MAT READ a()
MAT PRINT [#i]a[,g,n]
MAT INPUT #i,a()

i,g,n: iexp
a: Name of field with numerical variables

MAT READ a() reads a previously dimensioned matrix or vector from DATA rows.

MAT PRINT [#i,]a()[,g,n] outputs a matrix or a vector. Vectors are output on one row, the elements being separated by commas. With matrix, each row is followed by a rowfeed.

The output can optionally be redirected with #i, as with PRINT.

If g and n are specified, the numbers are formatted as with STR$(x,g,n).

MAT INPUT #1,a() reads a matrix or vector from a file in ASCII format (the format being the reverse of MAT PRINT, commas and rowfeeds may be varied as with INPUT #).

Copy and Transposition commands

MAT CPY a([i,j])=b([k,l])[,h,w]
MAT XCPY a([i,j])=b([k,l])[,h,w]
MAT TRANS a()[=b()]

a,b: Name of fields with numerical variables
i,j,k,l,h,w: iexp

MAT CPY a([i,j])=b([k,l])[,h,w] copies h rows with w elements each from matrix b to the row and column offset of matrix a defined by i,j, starting from the row and column offset of matrix b defined by l,k.

Special cases

MAT COPY a()=b() copies the complete matrix b into matrix a if the matrix are of the same order.

Only those elements are copied in this process for which identical indices are given in both the source and the destination matrix.

MAT COPY a(i,j)=b() copies matrix b, starting from the row and column offset defined by MAT BASE, to the row and column offset

of matrix a defined by i,j. Only those elements are copied for which identical indices are given in both the source and the destination matrix.

MAT COPY a()=b(i,j) copies matrix b, starting from the row and column offset defined by i,j, to the offset of matrix a defined by MAT BASE. Only those elements are copied for which identical indices are given in both the source and the destination matrix.

MAT COPY a(i,j)=b(k,l) copies matrix b, starting from the row and column offset defined by k,l, to the offset i,j of matrix a. Only those elements are copied for which identical indices are given in both the source and the destination matrix.

MAT COPY a()=b() copies h rows with w elements each from the matrix b, starting from the row and column offset defined by MAT BASE, the row and column offset of matrix a defined by MAT BASE. Only those elements are copied for which identical indices are given in both the source and the destination matrix.

MAT XCPY a([i,j])=b([k,l])[,h,w] works basically in the same manner as MAT CPY a([i,j])=b([k,l])[,h,w], except that matrix b is being transposed while being copied to matrix a, i.e. the rows and columns of matrix b are swapped while it is copied to matrix a. Array b remains unchanged, however. Only those elements are copied for which identical indices are given in both the source and the destination matrix.

Further special cases

As with MAT CPY a(i,j)=b(k,l),w,h.

If MAT CPY or MAT XCPY are applied to vectors, j and l may be ignored. Following a DIM a(n),b(m), a() and b() are interpreted as row vectors, i.e. as matrix of the (1,n) or (1,m) types.

For a and b to be treated as column vectors, they must be dimensioned as matrix of the (n,1) or (m,1) type, ie. DIM a(n,1),b(n,1).

If both vectors are of the same order (both are row or column vectors), MAT CPY must be used. Irrespective of the type of vectors a and b, MAT CPY always treats both vectors syntactically as column vectors, so that the correct syntax to be used for MAT CPY is always

```
MAT CPY a(n,1)=b(m,1)!
MAT CPY a(3,1)=b(1,1) ! interprets a() and b() as column vectors
```

For MAT XCPY, one of the two vectors a and b must be explicitly dimensioned as a row vector, the other as a column vector.

Since MAT XCPY first transposes the second vector before copying it to the first. For this reason, MAT XCPY can only be used for DIM a(1,n),b(m,1): a()=row vector, b()=column vector and DIM a(n,1),b(1,m): a()=column vector, b()=row vector.

Optionally, the parameters h and w can also be used when copying vectors with MAT CPY or MAT XCPY. However, the following applies: with MAT CPY, only the h parameter is used for w=>1. No copying takes place with w=0.

With MAT XCPY, only h is used for w=>1 if b is a column vector to be copied into a row vector after transposition. No copying takes place when w=0. On the other hand, only w is used for h=>1 if b is a row vector which is to be copied to a column vector after transposition. In this case, no copying takes place if h=0.

MAT TRANS a()=b() copies the transposed from matrix b to matrix a if a and b are dimensioned accordingly, i.e. the number of rows from a must correspond to the number of columns in b, and the number of columns from a to the number of rows of n.

In the case of a square matrix, i.e. one with equal numbers of rows and columns, MAT TRANS a() may be used. This command swaps the rows and columns of matrix a and writes the matrix thus changed back to a.

(The original matrix is lost in the process (but can be restored with another MAT TRANS a()).

Operation commands

```
MAT ADD a()=b()+c()
MAT ADD a(),b()
MAT ADD a(),x

MAT SUB a()=b()-c()
MAT SUB a(),b()
MAT SUB a(),x

MAT MUL a()=b()*c()
MAT MUL x=a()*b()
MAT MUL x=a()*b()*c()
MAT MUL a(),x

MAT NORM a(),0
MAT NORM a(),1

MAT DET x=a([i,j])[,n]
MAT QDET x=a([i,j])[,n]
MAT RANG x=a([i,j])[,n]
MAT INV a()=b()

a,b,c: Names of numerical floating point fields
x:     aexp; scalar value
i.j,n: aexp
```

MAT ADD a()=b()+c() is only defined for matrix (vectors) of the same order, e.g. DIM a(n,m),b(m,m),c(n,m) or DIM a(n),b(n),c(n). Array c is added to matrix b, element by element, and the result is written to matrix a.

MAT ADD a(),b() is only defined for matrix (vectors) of the same order, e.g. DIM a(n,m),b(n.m) or DIM a(n),b(n). Array b is added to matrix a, element by element, and the result is written to matrix a. The original matrix a is lost in the process.

MAT ADD a(),x is defined for all matrix (vectors). Here, the

scalar x is added to matrix a, element by element, and the result is written to matrix a. The original matrix a is lost in the process.

MAT SUB a()=b()+c() is only defined for matrix (vectors) of the same order, e.g. DIM a(n,m),b(n,m),c(n,m) or DIM a(n),b(n),c(n). Array c is subtracted from matrix b, element by element, and the result is written to matrix a.

MAT SUB a(),b() is only defined for matrix (vectors) of the same order, e.g. DIM a(n,m),b(n,m) or DIM a(n),b(n). Array b is subtracted from matrix a, element by element, and the result written to matrix a. The original matrix a is lost in the process.

MAT SUB a(),x is defined for all matrix (vectors). Here, the scalar x is subtracted from matrix x, element by element, and the result is written to matrix a. The original matrix a is lost in the process.

MAT MUL a()=b()*c() is defined for matrix of an "appropriate" order. Arrays b and c are multiplied with each other. The result of this multiplication is written to matrix a. In order for the result to be defined, the matrix on the left (matrix b in this case) must have the same number of columns as the matrix on the right (c in this case) has rows. Array a, in this case, must have as many rows as b and as many columns as c.

Arrays are multiplied as "row by column", i.e. element a(i.j) is obtained by multiplying the elements in the ith row of matrix b with the elements in the jth column of matrix c, element by element, and then adding up the individual products.

With vectors instead of matrix, MAT MUL a()=b()*c() results in the dyadic (or external) product of two vectors.

MAT MUL x=a()*b() is only defined for vectors with an equal number of elements. The result x is the scalar product (the so-called interior product) of vectors a and b. The scalar product of two vectors is defined as the sum of n products a(i)*b(i),i=1,...,n.

MAT MUL x=a()*b()*c() is defined for qualified Vectors a and c as well as qualified Matrix b().

MAT NORM a(),0 or MAT NORM a(),1 are defined for matrix and vectors. MAT NORM a(),0 normalises a matrix (a vector) by rows, MAT NORM a(),1 by columns. This means that after a normalisation by rows (by columns) the sum of the squares of all elements in each row (column) is identical at 1.

MAT DET x=a([i,j])[,n] calculates the determinants of a square matrix of the (n,n) type. The row and column offsets are preset to a(0,0) or a(1,1), depending on MAT BASE 0 or MAT BASE 1, assuming that OPTION BASE 1 is enabled. It is also possible, however, to calculate the determinant of a square part matrix. To do this, the row and column offsets of a() must be specified as i and j, and the number of elements in the part matrix as n. A part matrix of the (n,n) type is then created internally starting from the "position" ith row, jth column.

MAT QDET x=a([i,j])[,n] works in the same manner as MAT DET x = a([i,j])[,n], except that it has been optimised for speed

rather than accuracy. Both will normally produce identical results. With "critical" matrix, whose determinant is close to 0, you should always use MAT DET, though.

MAT RANG x=a([i,j])[,n] outputs the rank of a square matrix. As with MAT DET or MAT QDET, you can select any row and column offset. The number of elements in the part matrix must be specified with n. This creates a part matrix of the (n,n) type internally, starting from the "position ith row, jth column.

MAT INV b()=a() is used to determine the inverses of a square matrix. The inverse of matrix a() is written to matrix b(), hence b() must be of the same type as a().

MAX

Syntax: MAX(x [,y,z,...]) or MAX(a$[,y$,z$....])
Action: Returns the greatest value (or largest string) from a list of expressions.

MENU(x) Returns the information about an event in the variable 'x' (-2 to 15). In the case where an item in a menu is selected, the index of that item is found in MENU(0).
MENU(-2) is the address of the message buffer.
MENU(-1) is the address of the menu object tree.
The Message Buffer lies in the the variables MENU(1) to MENU(8) and the AES Integer Output Block in MENU(9) to MENU(15).
The Identification number of the event is in MENU(1).
The other elements of the message bufffer contain various values, depending on the type of event that occured.

MENU(1)=10	A Menu Item was selected.
MENU(0)	Menu item index in the item list
MENU(4)	Object number of the menu title
MENU(5)	Object number of the chosen menu item
MENU(1)=20	A window redraw is required
MENU(4)	ID number (handle) of the window
MENU(5),(6)	Coordinates of top left corner of the window
MENU(7),(8)	Width & height of the window area
MENU(1)=21	A window was clicked (activated)
MENU(4)	ID number (handle) of the window
MENU(1)=22	The close box of a window was clicked on
MENU(4)	ID number (handle) of the window
MENU(1)=23	The full box was clicked on
MENU(4)	ID number (handle) of the window
MENU(1)=24	One of the four arrow boxes, or a slider bar area was clciked. The movement of a slider is detected as

below, MENU(1)=24 only when the grey area is clicked on.

MENU(4) ID number (handle) of the window

MENU(5) The area of the window that was clicked:
0: Above the vertical slider
1: Below " " "
2: Up arrow
3: Down arrow
4: Left of the horizontal slider
5: Right " " " "
6: Left arrow
7: Right arrow

MENU(1)=25 The horizontal slider was moved

MENU(4) ID number (handle) of the window

MENU(5) Position of the moved slider (1 to 1000)

MENU(1)=26 The vertical slider was moved

MENU(4) ID number (handle) of the window

MENU(5) Position of the moved slider (1 to 1000)

MENU(1)=27 The size of the window was changed (using the size box)

MENU(4) ID number (handle) of the window

MENU(5),(6) New x and y coordinates of top left

MENU(7),(8) New width & height

MENU(1)=28 The window's position was changed

same parameters as above

MENU(1)=29 A new GEM window was activated.

MENU(4) ID number (handle) of the window

MENU(1)=40 An Accessory was selected. This value can only be received by an accessory, which should chech the value in MENU(5) to see if it is that one being referred to.

MENU(5) ID of the accessory

MENU(1)=41 An accessory was closed. This value can only be received from an accessory.

MENU(5) ID of the accessory

The varaible MENU(9) contains bit information on which kind of event has occurred. If the bit for the appropriate event is set, the variables MENU(9) to MENU(15) and GINTOUT(0) to GINTOUT(6) will contain information as follows:

Bit 0 Keyboard
Bit 1 Mouse button
Bit 2 Mouse has entered/left rectangle 1
Bit 3 Mouse has entered/left rectangle 2
Bit 4 A message arrived in the message buffer
Bit 5 Timer event

MENU(10) x position of mouse when event terminated

MENU(11) y position of mouse when event terminated
MENU(12) Mouse buttons pressed:
0 = none
1 = left
2 = right
3 = both buttons
See also ON MENU BUTTON

MENU(13) supplies the status of the keyboard shift keys in a bit pattern:
Bit 0 = right shift
Bit 1 = left shift
Bit 2 = control
Bit 3 = alternate
See also ON MENU KEY

MENU(14) Gives information about a pressed key. The low order byte contains the ASCII code, and the high order byte, the keyboard scan code

MENU(15) Returns the number of mouse clicks that caused the event

MENU
Syntax: MENU m$()
Action: Displays a menu bar. The string array m$() contains the headings, entries and reserved space for accessories for the menu bar. The following format must be adhered to:

m$(0)	Name of the accessory menu heading
m$(1)	Name of the first entry in the first menu
m$(2)	A line of minus signs
m$(3)-m$(8)	Reserved space for accessories. These elements need only be 1 character long.
m$(9)	An empty string, which marks the end of the first menu.

All further menu entries have the following format:
1. Heading of the menu
2. List of menu entries
3. An empty string which marks the end of the menu.

After the last entry, a further empty string signifies the end of the entire pull down menu.

eg:

```
DIM entry$(20)
DATA "  Desk  "," Test  "
DATA ----------,1,2,3,4,5,6,""
DATA "  File  "," Load  "," Save  "
DATA --------," Quit  ",""
DATA "  Titles  "," Entry 1  "," Entry 2  ","",""
DATA End
i%=-1
REPEAT
  INC i%
  READ entry$(i%)
UNTIL entry$(i%)="End"
entry$(i%)=""
MENU entry$()
ON MENU GOSUB evaluate
```

```
'
REPEAT
  ON MENU
UNTIL MOUSEK AND 2
'
PROCEDURE evaluate
  MENU OFF
  ' MENU(0) contains array index of selected item
  m%=MENU(0)
  PRINT entry$(m%)
  '
  ALERT 0,"Tick before item ?",0,"YES|NO",a%
  IF a%=1
    MENU m%,1
  ELSE
    MENU m%,0
  ENDIF
  '
  ALERT 0,"Lightened characters | (Not selectable)",0,"YES|NO",a%
  IF a%=1
    MENU m%,2
  ELSE
    MENU m%,3
  ENDIF
RETURN
```

MENU x,y
Action: The x-th entry of a menu can be given certain (y) attributes:
0 remove tick from in front of menu entry
1 install tick " " " " "
2 make menu entry non selectable (light text)
3 make menu entry selectable (normal text)
See MENU example.

MENU KILL
Action: Deactivates a menu, but does not remove it from the screen. Also turns off the ON MENU GOSUB options.

MENU OFF
Action: Returns a menu title to 'normal' display. (After an item is chosen from a menu, the title is shown in reverse video).

MENU_BAR
Syntax: a%=MENU_BAR(tree%,flag)
Action: Displays/erases a menu bar (from a resource file)
Returns 0 if an error occurred.
tree = address of the menu object tree
flag - 1 display bar
- 2 erase bar
See also MENU x$ and MENU KILL

MENU_ICHECK
Syntax: a%=MENU_ICHECK(tree,item,flag)
Action: Deletes/displays a tick against a menu item.
tree = address of the menu object tree
item = object number of the menu item

flag - 1 delete tick
     - 2 display tick
See also MENU x,0 and MENU x,1

MENU_IENABLE
Syntax: a%=MENU_IENABLE(tree,item,flag)
Action: Enables/disables a menu entry.
tree = address of the menu object tree
item = object number of the menu entry
flag - 1 disable
     - 2 enable
See also MENU x,2 and MENU x,3

MENU_REGISTER
Syntax: a%=MENU_REGISTER(ap_id,m_text$)
Action: Give a desk accessory a name, and insert it into the accessory menu entries. (provided the number of Accs is less than 6).
Returns the object number of the appropriate menu item:
0-5 for a valid result
-1 no more entries possible
ap_id = ID number of the accessory
m_text$ = name for the Accessory

MENU_TEXT
Syntax: a%=MENU_TEXT(tree,item,new_text$)
Action: Changes the text of a menu item.
Returns 0 on error.
tree = address of the menu object tree
item = object number of the menu item
new_text$ the new text for the menu entry (may not exceed the old text length)

MENU_TNORMAL
Syntax: a%=MENU_TNORMAL(tree,title,flag)
Action: Switches the menu title to normal/inverse video.
Returns 0 on error.
tree = address of the menu object tree
item = object number of the menu item
flag - 1 inverse video
     - 2 normal video
See MENU OFF

MFREE
Syntax: a%=MFREE(y)
Action: (GEMDOS 73) Releases the storage location reserved with MALLOC. The parameter 'y' specifies the start of the area of memory to be released. Returns 0 if no error occurred, otherwise negative result.

MID$
Syntax: MID$(a$,x[,y]) (as a function)

Action: Returns 'y' characters in a string from the positon 'x' of the string 'a$'. If x is larger than the length of a$, then a null string is returned. If y is omitted, then the function returns the whole of the string from position x onwards.

MID$
Syntax: MID$(a$,x[,y]) (as a command)
Action: MID$ used as a command, makes it possible to replace part of a string variable a$ with the string expression b$. So with MID$(a$,x,y)=b$, characters from b$ will overwrite those in a$, starting at the x-th postion of a$. The optional parameter y determines how many characters of b$ are used. If y is omitted, then as many characters as possible of a$ are replaced with those from b$. The length of a$ is unchanged, so that no charatcers will be written beyond the end of a$
eg:
a$="GFA SYSTEMTECHNIK"
MID$(a$,5)="BASIC "
would result in a$ being "GFA BASIC TECHNIK"

MIN
Syntax: MIN(expression [,expression...])
Action: Returns the smallest value (or smallest string) from a list of expressions.

MKDIR
Syntax: MKDIR "directory name"
Action: Creates a new directory.
'directory name' is the name of the new directory.

MKI$ MKL$ MKS$ MKF$ MKD$
Syntax: MKI$(N)
MKL$(N)
MKS$(N)
MKF$(N)
MKD$(N)
Action: Transforms a number into a character string.
MKI$ 16-bit number into a 2-byte string.
MKL$ 32-bit number into a 4-byte string.
MKS$ a number into an atari basic 4-byte format.
MKF$ a number into GFA Basics own 6-byte format.
MKD$ a number into a Mbasic compatible 8-byte format.

Every number that is to be stored in a random access file must first be transformed with on of the above functions.
The example above shows that GFA Basic stores numbers internally in the 6-byte format which can also be created using the MKF$ function.
See also CVI,CVL,CVD,CVF

MOD
Syntax: a=x MOD y or a=MOD(x,y)
Action: Produces the remainder of the division of x by y.
The command in brackets operates in integer arithmetic.

MODE
Syntax: MODE n
Action: With MODE the representation of decimal point and the 'thousands comma' are interpreted by PRINT USING (and also by STR$ with 3 parameters).
Also selects the format of date representation used by DATE$, SETTIME, and FILES.

MODE	USING	DATE$
0	#,###.##	16.05.1988
1	#,###.##	05/16/1988
2	#.###,##	16.05.1988
3	#.###,##	05/16/1988

MONITOR
Syntax: MONITOR [x]
Action: Calls a monitor resident in memory. This instruction causes an illegal instruction vector. (address 16). The parameter x is passed via the register D0.

MOUSEX
MOUSEY
MOUSEK
MOUSE mx,my,mk
Syntax: MOUSE x,y,k
x=MOUSEX
y=MOUSEY
k=MOUSEK
Action: Determines the mouse position (x,y) and the status of the mouse buttons:
k=0 no buttons pressed
k=1 left button
k=2 right button
k=3 both buttons

MSHRINK
Syntax: a%=MSHRINK(y,z)
Action: (GEMDOS 74) Reduces the size of a storage area previously allocated with MALLOC. y specifies the address of the area,z gives the required size.
Returns 0 if no error, -40 if incorrect address, or -67 if size wrong.
See also RESERVE MALLOC MFREE

MUL
Syntax: MUL var,n
Action: Multiplies the value 'var' by 'n'.
same as var=var*n but executes 30% faster.

MUL() Same as for MUL. but integers only.

MW_OUT
Syntax: MWOUT mask,data
This command controls the STE-Internal Micro-Wire-Interface, and is currently used for controlling sound.

```
MWOUT &H7FF,x

x=&X10 011 ddd ddd       Set Master Volume
           000 000       -80 dB
           010 100       -40 dB
           101 xxx         0 dB
```

The value of the last 5 Bits is eqivalent to HALF of the volume in dB.

```
x=&X10 101 xdd ddd       Set Right Channel Volume
            00 000       -40 dB
            01 010       -20 dB
            10 1xx         0 dB

x=&X10 100 xdd ddd       Set Right Channel Volume

The last 4 Bits*2 = dB

x=&X10 010 xxd ddd       Set Treble
x=&X10 001 xxd ddd       Set Bass
             0 000       -12dB
             0 110       0 dB (flat)
             1 100       +12 dB

x=&X10 000 xxx xdd       Set Mix
                00       -12dB
                01       Mix GI Sound (normal ST)
                10       Not Mix
                11       Reserved
```

Example: MWOUT &H7FF,&X10000000010 Switches the ST's sound off.

NAME
Syntax: NAME "oldfile" AS "newfile"
Action: Renames an existing file. The contents of the file are not affected.

NEW
Syntax: NEW
Action: Deletes the program currently in memory and clears all variables.

NOT
Syntax: NOT x
Action: Negates a given logical expression.

The following 19 commands belong to the AES Object library.

OBJC_ADD
Syntax: a%=OBJC_ADD(tree,parent,child)
Action: Adds an object to a given tree and pointers between the existing objects and the new object are created.
         Returns 0 on error.
         tree   address of the object tree
         parent   object number of the parent object
         child   object number of the child to be added.

OBJC_CHANGE
Syntax: a%=OBJC_CHANGE(tree,obj,res,cx,cy,cw,ch,new_status,re_draw)
Action: Changes the status of an object.
         Returns 0 on error.
         tree   address of the object tree
         obj   number of the object to be changed
         res   reserved (always 0)
         cx,cy   coordinates of top left corner of clipping rectangle
         cw,ch   width & height of clipping rectangle
         new_status   new object status
         re_draw 1 = redraw object
                  0 = don't redraw

OBJC_DELETE
Syntax: a%=OBJC-DELETE(tree,del_obj)
Action: An object is deleted from an object tree by removing the pointers. The object is still there and can be restored by repairing the pointers.
         Rteurns 0 on error.
         tree   address of the object tree
         del_obj Object number of the object to delete.

OBJC_DRAW
Syntax: a%=OBJC_DRAW(tree,start_obj,depth,cx,cy,cw,ch)
Action: Draws whole objects or part of objects on screen. A clipping rectangle is specified, to which the drawing is limited.
         Returns 0 on error.
         tree   address of the object tree
         start_obj number of the first object to be drawn
         depth   Number of object levels to be drawn
         cx,cy   coordinates of top left corner of clipping rectangle
         cw,ch   width & height of clipping rectangle

OBJC_EDIT
Syntax: a%=OBJC_EDIT(tree,obj,char,old_pos,flag,new_pos)
Action: Allows input and editing in G_TEXT and G_BOXTEXT object types.
         Returns 0 on error.

```
           tree     address of the object tree
           obj      number of the object to be changed
           char     input character (incl. scan code)
           old_pos  current cursor position in input string
           flag     funtion:
                    0 ED_START     -reserved-
                    1 ED_INIT      string is formatted & cursor on
                    2 ED_CHAR      Character processed &  string
                                    redisplayed
                    3 ED_END       Text cursor switched off
           new_pos  returns  new  pos  of  text  cursor  to  this
                    variable.


OBJC_FIND
Syntax:    a%=OBJC_FIND(tree,start_obj,depth,fx,fy)
Action:    Determines  the  object,  if  any,  which  is  at  the
           coordinates specified in fx,fy.
           Returns the object number, or -1 if no object found.
           tree     address of the object tree
           start_obj  number of the object from where to search
           depth    Number of object levels to be searched
           fx       x coordinate (usually MOUSEX)
           fy       y coordinate (usually MOUSEY)


OBJC_OFFSET
Syntax:    a%=OBJC_OFFSET(tree,obj,x_abs,y_abs)
Action:    Calculates  the  absolute screen  coordinates  of  the
           specified object.
           Returns 0 on error.
           tree     address of the object tree
           obj      object number
           x_abs,y_abs  returns  the  x,y  coordinates  to  these
                        variables.



OBJC_ORDER
Syntax:    a%=OBJC_ORDER(tree,obj,new_pos)
Action:    re-positions an object within a tree.
           Returns 0 on error.
           tree     address of the object tree
           obj      object number
           new_pos  new level number



OB_ADR
Syntax:    adr%=OB_ADR(tree,obj)
Action:    Gets the address of an individual object.
           Returns 0 on error.
           tree     address of the object tree
           obj      object number


OB_FLAGS
Syntax:    a%=OB_FLAGS(tree,obj)
Action:    Gets the status of the flags for an object.
           Returns 0 on error.
           tree     address of the object tree
           obj      object number
```

```
           OB_FLAGS     Bit No.
           Normal       -
           Selectable   0
           Default      1
           Exit         2
           Editable     3
           Rbutton      4
           Lastob       5
           Touchexit    6
           Hidetree     7
           Indirect     8


OB_H
Syntax:    h%=OB_H(tree,obj)
Action:    Returns the height of an object
           Returns 0 on error.
           tree     address of the object tree
           obj      object number


OB_HEAD
Syntax:    h%=OB_HEAD(tree,obj)
Action:    Points to the object's first child, or -1 if none.


OB_NEXT
Syntax:    n%=OB_NEXT(tree,obj)
Action:    Points to the following object on the same level,  or,
           if it is the last object on that level,  to the parent
           object, or -1 if none.


OB_SPEC
Syntax:    a%=OB_SPEC(tree,obj)
Action:    Returns the address of the the data structure for  the
           object.


OB_STATE
Syntax:    s%=OB_STATE(tree,obj)
Action:    returns the status of an object:
           OB_STATE     Bit No.
           Normal       -
           Selected     0
           Crossed      1
           Checked      2
           Disabled     3
           Outlined     4
           Shadowed     5


OB_TAIL
Syntax:    t%=OB_TAIL(tree,obj)
Action:    Points to the objects last child, or -1 if none.


OB_TYPE
Syntax:    t&=OB_TYPE(tree,obj)
Action:    Returns the type of object specified.
```

OB_W
Syntax: w%=OB_W(tree,obj)
Action: Returns the width of an object

OB_X
OB_Y
Syntax: x (or y) =OB_X or OB_Y(tree,obj)
Action: Rteurns the relative coordinates of the object relative to its parent (or the screen if it is the parent)

OCT$
Syntax: OCT$(x[,n])
Action: Changes the value 'x' into a string containing the value of 'x' in octal form (prefix &O), the optional parameter n, giving the number of characters to print.

ODD
Syntax: ODD(n)
Action: Determines whether a number is odd. (see also even)

ON BREAK Syntax: ON BREAK
ON BREAK CONT
ON BREAK GOSUB name
Action: ON BREAK CONT makes it impossible to stop a program by pressing break (<ALT><SHIFT><CNTRL>).
ON BREAK reactivates it.
ON BREAK GOSUB makes it possible to jump to the procedure 'name' by the above key combination.

ON ERROR
Syntax: ON ERROR
ON ERROR GOSUB name
Action: Performs the procedure 'name' when an error occurs. The program is not interrupted and no error message is given.
See also RESUME

ON...GOSUB
Syntax: ON x GOSUB proc1,proc2.....
Action: Depending on the result of 'x' one of several given procedures is processed.
'proc1' .. is a list of procedure names separated by commas. The result of 'x' denotes which procedure is carried out.
Eg: If result = 1 then the first procedure in the procedure list is processed.
If result = 2 then the second procedure in the procedure list is processed.
If result = 3 then the third procedure in the procedure list is processed and so on.
If the value is not in the range then no procedure will be executed.

ON MENU

Syntax: ON MENU[t]

Action: This command handles EVENTs. Prior to use, the required action should be specified with an ON MENU xxx GOSUB command. For constant supervision of events, ON MENU is usually found in a loop.
The parameter t is the time in thousandths of a second to elapse before the ON MENU command is terminated.

ON MENU xxx GOSUB

Syntax: ON MENU BUTTON clicks,but,state GOSUB proc

Action: Sets up the action to be taken when one or more mouse clicks are received. With a subsequent ON MENU command, the named procedure will be branched to if the condition imposed by the parameters are met.

- clicks - sets the maximum number of clicks that will generate a response.
- button - The expected button combination:
  0 - any
  1 - left
  2 - right
  3 - both
- state - Specifies which button state (up or down) will cause the event. 0 = up, 1 = down
- proc - The procedure to branch to.

Syntax: ON MENU GOSUB proc

Action: The procedure to which control will be passed on selection of a menu entry is determined. If an accessory is currently open, the procedure will not be called.
See also MENU(0)

Syntax: ON MENU IBOX n,x,y,b,h GOSUB proc

Action: Monitors the mouse coordinates, and branches to the named procedure if the mouse enters Y
k=MOUSEK

Action: Determines the mouse position (x,y) and the status of the mouse buttons:
k=0 no buttons pressed
k=1 left button
k=2 right button
k=3 both buttons

Syntax: ON MENU KEY GOSUB proc

Action: Monitors the keyboard, and branches to proc if a key was pressed during an ON MENU loop.
See MENU(13) & MENU(14) for the keys.

Syntax: ON MENU MESSAGE GOSUB proc

Action: Brances to proc if a message arrives in the message

buffer during an ON MENU loop.
See MENU(x) for the messages.

Syntax: ON MENU OBOX n,x,y,w,h GOSUB proc
Action: Monitors the mouse coordinates, and branches to the named procedure if the mouse leaves a rectangular screen area. It is possible to wait for two such boxes to be left (n can be 1 or 2). x and y are the top left coordinates of the rectangle, w & h being its width and height. Continuous monitoring is done with ON MENU.

OPEN
Syntax: OPEN mode$,#n,name$[,len]
Action: Opens a data channel to a file or a peripheral device.
'mode' must always be written in quotes and is one of the following :-
'O' (output) opens a write file creating a new file if needed.
'I' (input) opens a read file.
'A' (append) enables data to be annexed to an existing file.
'U' (update) read/write, but file must be opened by 'o' first.
'R' stands for random access file.
Instead of a filename, a periphral device can ne specified. The expression 'len' is used only with Random Access mode.
the following can be used instead of filenames :-
'CON:' for the console.
'LST:' or 'prn:' for the printer.
'AUX:' for the serial interface.
'MID:' for midi.
'VID:' for the console in transparent mode (commands are produced but not executed).
'IKB:' for direct access to the keyboard controller.
'STD:'. (This is the same as 'Stdin','Stdout' resp. in C-programs.) So you can use a shell to redirect the output of a GFA-BASIC program.

GFABASIC TEST >DUMMY

This line starts GFA BASIC and the program TEST.PRG
Any output via 'STD:' is redirected to the file DUMMY.
IMPORTANT: CONTROL-C will cause a hang- up when given while reading/writing DUMMY. the default for input/output is the keyboard/console.
The numerical expression 'n' contains the channel number (0-99), and the variable name$, the access path and filename.

OPENW
Syntax: OPENW nr[,x a rectangular
screen area. It is possible to wait for two such boxes to be entered (n can be 1 or 2). x and y are the top left coordinates of the rectangle, w & h being its width and height. Continuous monitoring is done with ON MENU.

OPTION BASE
Syntax: OPTION BASE 0 (default)
OPTION BASE 1
Action: This command can determine whether an array is to contain a zero element or not. ie. with OPTION BASE 0, doing a DIM a%(10) will allow a%(0) to exist.

OR
Syntax: x OR y
Action: The command OR (disjunction) checks whether at least one of two logical expressions x and y is TRUE. Only if x and y are both FALSE will the result FALSE be produced.

OR()
Syntax: OR(x,y)
Action: The result of OR contains bits set in the places in which bits are set in either x or y or both.

OTHERWISE See SELECT

OUT
Syntax: OUT [#]n,a[,b..]
Action: Transfers a byte[s] with the value 'a' to a peripheral device or file 'n'.
See OPEN for valid peripherals.
See also INP

OUT# See INP#

OUT?
Syntax: OUT?(n)
Action: Determines the output status of a periphery.
This function returns 0 if a character can be output.
(see also INP?)

PADT(i)
Syntax: a=PADT(i)
Action: Reads the paddle buttons on the STE

PADX(i)
PADY(i)
Syntax: a=PADX(i) or PADY(i)
Action: Reads the x or y position of the paddles on the STE. i can be 0 or 1.

PAUSE
Syntax: PAUSE x

Action: Interrupts a program for exactly x/50 seconds.
See also DELAY.

PBOX
Syntax: PBOX x,y,x1,y1
Action: Draws a filled rectangle with the coordinates of the two opposite corners specified by x,y and x1,y1.
See also BOX,PRBOX, RBOX.

PCIRCLE
Syntax: PCIRCLE x,y,r[,w1,w2]
Action: Draws a filled circle with centre coordinates at x,y and a radius r. Additional start and end angles w1 and w2 can be specified to draw a circular arc.

PEEK DPEEK LPEEK
Syntax: PEEK(x)
DPEEK(x)
LPEEK(x)
Action: Returns the contents of the memory at address 'x'
PEEK returns a 1 byte at address x
DPEEK returns a 2 byte number from x and x+1
LPEEK returns a 4 byte number from x, x+1, x+2 & x+3
for DPEEK and LPEEK, 'x' must be an even number.

PELLIPSE
Syntax: PELLIPSE x,y,rx,ry [,phi0,phi1]
Action: Draws a filled ellipse at x,y, having 'rx' as length of the horizontal axis and 'ry' as length of the vertical axis
The optional angles 'phi0' & 'phi1' give start and end angles in tenths of a degree, to create an elliptical arc.

PI
Syntax: PI
Action: Returns the value of PI. The value of PI is 3.141592653.....etc.

PLOT
Syntax: PLOT x,y
Action: Plots a point on the screen coordinates 'x,y'.
This command is the same as draw x,y.

POINT
Syntax: POINT x,y
Action: Checks if a graphic dot (at 'x,y') has been set and returns its colour value.

POKE DPOKE LPOKE
Syntax: POKE x,n
         DPOKE x,n
         LPOKE x,n
Action: Writes 1, 2 or 4 bytes into memory at an address which starts at 'x'.
         The value of 'x' must be an even number for DPOKE and LPOKE.

POLYLINE POLYFILL POLYMARK
Syntax: POLYLINE n,x(),y()[OFFSETx_off,y_off]
         POLYFILL n,x(),y()[OFFSETx_off,y_off]
         POLYMARK n,x(),y()[OFFSETx_off,y_off]
Action: POLYLINE draws a polygon with n corners. The x,y coordinates for the corner pointa are given in arrays x() and y(). The first and last points are automatically connected. The optional parameter OFFSET can be added to these coordinates.
         POLYFILL fills the polygon with the pattern previously defined by DEFFILL.
         POLYMARK marks the corner points with the shape defined by DEFMARK.

POS
Syntax: POS(n)
Action: Returns the column in which the cursor is positioned.
         'n', a hypothetical argument, is optional.
         See also CRSCOL, CRSLIN, TAB, HTAB, VTAB.

PRBOX
Syntax: PRBOX x,y,x1,y1
Action: Draws a filled rectangle with rounded corners.
         See also BOX, PBOX, RBOX.

PRED
Syntax: a$=PRED(b$)
Action: Supplies the character with the ASCII code one less than that of the first character of the specified string.
         See also SUCC.

PRED()
Syntax: i%=PRED(n%)
Action: Returns the next lower number of the integer argument.
         See also SUCC().

PRINT
Syntax: PRINT
         PRINT expression
Action: Displays information on the screen.
         'expr' can be any number of expressions which must be

separated by commas, semicolons or apostrophes.
; -items are printed one after an other in one line.
, -items are printed at intervals of 16 columns.
' -each apostrophe causes a space to be printed.

PRINT AT
Syntax: PRINT AT(column,row);expression
Action: Prints 'expression' at a specified row and column.
NB. These start at 1, not 0.

PRINT USING
Syntax: PRINT USING format$,expression[;]
PRINT AT(column,row);USING format$,expression[;]
Action: Prints formatted digits and character strings.
format$ is a string expression which sets the printing format using a list of expressions separated by commas.

#	reserves space for a digit.
.	position of the decimal point.
+	executes a plus sign.
-	reserves space for a minus sign.
*	zeros before the comma are replaced by * otherwise the same as #.
$	prefix $.
,	insertion of a comma.
^	execution in exponent form E+
!	indicates that the first character of a string is issued.
&	the whole string is issued.
\..\	as many characters as the length of \..\ is issued (including back-slashes).
-	prints the proceeding character.

PRINT TAB
Syntax: PRINT TAB(n)
Action: Prints spaces until POS(0) reaches n. If POS(0) already exceeds n then a Line Feed/Carriage Return is executed first.

PRINT#
Syntax: PRINT #n,expression
PRINT #n,USING format$,expression
Action: Outputs data to a specified channel n (0-99). PRINT# USING allows formatted data to be output.

PROCEDURE
Syntax: PROCEDURE proc[(var1,var2,...)]
Action: Marks the beginning of a procedure.
Basic will only process a procedure when it is called by the command GOSUB (or by simply naming the procedure, or using @proc. If it comes across the command

procedure during 'normal' running of the program, it considers it to be the end of the program.
Not only the values of variable, but also the variable's address can be passed to procedures using the VAR command in the Procedure's header.

PSAVE
Syntax: PSAVE f$
Action: Saves the current program to disk with the name f$, it is saved with protection, and cannot be subsequently listed on re-loading; PSAVEd programs RUN automatically on loading.
See also SAVE

PTSIN Address of the VDI point input table
PTSOUT Address of the VDI point output table
These two commands can be used with index, to address the array directly. eg. PTSIN(0).

PTST()
Syntax: a=PTST(x,y)
Action: Corresponds to the POINT command. Returns the colour of the pixel at x,y.

PUT
Syntax: PUT x,y,section$[,mode]
Action: Places a graphics block on the screen at x,y which has been previously grabbed by GET, and stored in section$.
'mode' (optional) sets the way the image is placed.

0 -	All points are cleared
1 - s AND d	Only points set in both remain set.
2 - s AND (NOT d)	Sets only points which are set in the source and clear in the destination.
3 - s	Overwrite (default GRAPHMODE 1)
4 - (NOT s)AND d	
5 - d	
6 - s XOR d	
7 - s OR d	
8 - NOT(s OR d)	
9 - NOT(s XOR d)	
10 NOT d	
11 s OR(NOT d)	
12 NOT s	
13 (NOT s)OR d	
14 NOT(s AND d)	
15 1	All points set.

The important ones are:
3 Repalce
4 XOR
7 Transparent
13 Inverse Transparent.

PUT #

Syntax: PUT #n[,r]

Action: Writes a record to a random access file.
'n' data channel number (0 to 99).
'r' is an integer expression between 1 and the number of records in the file (max 65535) and denotes the record number of the record to be written.
See also GET #, RECORD #

QSORT

Syntax: QSORT a(s) [OFFSET o] [WITH i()] [,n[,j%()]]
QSORT x$(s) WITH i() [,n[,j%()]]

Action: Sorts the elements of an array. 's' can be a minus sign or a plus sign, indicating an ascending sort(+) or a descending sort(-), the default being ascending. The parameter 'n' specifies that only the first 'n' elements are to be sorted. (Depends on OPTION BASE) whether 0 or 1. If n=-1, then all elements are sorted. When a further array is specified, then that array will be sorted along with the first array.
OFFSET determines how many characters off the beginning shall not be considered.
During sorting of string arrays a sorting criteria can be specified in an array of at least 256 elements by using WITH. Without using this option, a normal ASCII sort is used.
eg:

```
DIM a$(256)
FILES "*.*" TO "liste"
OPEN "i",#1,"liste"
RECALL #1,a$(),-1,x%
CLOSE #1
QSORT a$() OFFSET 13,x%
OPEN "o",#1,"con:"
STORE #1,a$(),x%
CLOSE
```

Saves the directory as 'LISTE', then reloads the file, sorts the array, not on name but on file length.

```
DIM x%(20)
PRINT "Unsorted:        ";
FOR i%=0 TO 10
  x%(i%)=RAND(9)+1
  PRINT x%(i%);"  ";
NEXT i%
PRINT
'
QSORT x%(),11
PRINT "Ascending sort:  ";
FOR i%=0 TO 10
  PRINT x%(i%);"  ";
NEXT i%
PRINT
'
QSORT x%(-),11
PRINT "Descending sort: ";
FOR i%=0 TO 10
  PRINT x%(i%);"  ";
NEXT i%
```

PRINT

QUIT
Syntax: QUIT[n]
Action: Terminate the program and leave GFA Basic.
Returns a two byte integer to the calling routine (normally the desktop).

RAD
Syntax: RAD(degrees)
Action: Converts from degrees to radians. (equivalent to x*PI/180).
See also DEG

RAND
Syntax: RAND(y)
Action: Produces a 16 bit random integer in the range 0 to y-1. Where y is an integer max value &HFFFF.

RANDOM
Syntax: RANDOM(x)
Action: Returns a random integer between 0 (inclusive) and 'x' (exclusive).

RANDOMIZE
Syntax: RANDOMIZE [y]
Action: Initialises the random number generator [with the value y].

RBOX
Syntax: RBOX x,y,x1,y1
Action: Draws a rectangle with rounded corners from the two diagonally opposite corner points 'x,y' and 'x1,y1'
See also BOX, PBOX, PRBOX.

RCALL
Syntax: RCALL addr,reg%()
Action: Calls an assembler routine (similar to C: or CALL) with pre-allocated values in the registers.
The integer array reg% must have 16 elements and holds the values. At the end of the routine, the values are also returned in the array.
Data registers d0 to d7 --->reg%(0) to reg%(7)
Address registers a0 to a6 --->reg%(8) to reg%(14)
User Stack Pointer (a7) --->reg%(15)

RC_COPY
Syntax: RC_COPY s_adr,sx,sy,sw,sh TO d_adr,dx,dy[,m]
Action: Copies rectangular screen sections between areas of memory.
s_adr source address
sx,sy top left corner of source rectangle
sw,sh width & height " " "
d_adr destination address
dx,dy destination x and y coordinates
m optional mode (see PUT for modes).

RC_INTERSECT
Syntax: y%=RC_INTERSECT(x1,y1,w1,h1,x2,y2,w2,h2)
Action: Detects whether two rectangles overlap. The rectangles being specified by the coordinates of the top left corner(x,y) and their width & height (w,h).
Returns TRUE (-1) if they do overlap and the variables x2,y2,w2,h2 contain the size of the common rectangle.

READ
Syntax: READ var[,var1, ...]
Action: Reads values from a DATA command and assigns them to a variable 'var'. Reading is taken from the last point a RESTORE was done (if any).

RECALL
Syntax: RECALL #i,x$(),n[TO m],x
Action: Inputs n lines from a text file to the array x$(). If n=-1 all available lines are read. x contains the number of lines read.

RECORD

RELSEEK
Syntax: RELSEEK [#]N,X
Action: Moves tINT
'
QSORT x%(-),11
PRINT "Descending sort: ";
FOR i%=0 TO 10
PRINT x%(i%);" ";
NEXT i%
PRINT

QUIT
Syntax: QUIT[n]
Action: Terminate the program and leave GFA Basic.
Returns a two byte integer to the calling routine (normally the desktop).

RAD
Syntax: RAD(degrees)
Action: Converts from degrees to radians. (equivalent to x*PI/180).
See also DEG

RAND
Syntax: RAND(y)
Action: Produces a 16 bit random integer in the range 0 to y-1. Where y is an integer max value &HFFFF.

RANDOM
Syntax: RANDOM(x)
Action: Returns a random integer between 0 (inclusive) and 'x' (exclusive).

RANDOMIZE
Syntax: RANDOMIZE [y]
Action: Initialises the random number generator [with the value y].

RBOX
Syntax: RBOX x,y,x1,y1
Action: Draws a rectangle with rounded corners from the two diagonally opposite corner points 'x,y' and 'x1,y1'
See also BOX, PBOX, PRBOX.

RCALL
Syntax: RCALL addr,reg%()
Action: Calls an assembler routine (similar to C: or CALL) with pre-allocated values in the registers.
The integer array reg% must have 16 elements and holds the values. At the end of the routine, the values are also returned in the array.
Data registers d0 to d7 --->reg%(0) to reg%(7)
Address registers a0 to a6 --->reg%(8) to reg%(14)
User Stack Pointer (a7) --->reg%(15)

RC_COPY
Syntax: RC_COPY s_adr,sx,sy,sw,sh TO d_adr,dx,dy[,m]
Action: Copies rectangular screen sections between areas of memory.
s_adr source address
sx,sy top left corner of source rectangle
sw,sh width & height " " "
d_adr destination address
dx,dy destination x and y coordinates
m optional mode (see PUT for modes).

RC_INTERSECT
Syntax: y%=RC_INTERSECT(x1,y1,w1,h1,x2,y2,w2,h2)
Action: Detects whether two rectangles overlap. The rectangles being specified by the coordinates of the top left corner(x,y) and their width & height (w,h).
Returns TRUE (-1) if they do overlap and the variables x2,y2,w2,h2 contain the size of the common rectangle.

READ
Syntax: READ var[,var1, ...]
Action: Reads values from a DATA command and assigns them to a variable 'var'. Reading is taken from the last point a RESTORE was done (if any).

RECALL
Syntax: RECALL #i,x$(),n[TO m],x
Action: Inputs n lines from a text file to the array x$(). If n=-1 all available lines are read. x contains the number of lines read.
The optional parameter TO will read in the start of the file to the named elements of the array.
eg.

```
DIM a$(20)
FOR n=0 TO 19
  a$(n)="Line # "+STR$(n)
NEXT n
OPEN "o",#1,"test"
STORE #1,a$()
CLOSE
DIM b$(20)
OPEN "i",#1,"test"
RECALL #1,b$(),12 TO 15,x
'or RECALL #1,b$(),-1,x
CLOSE
PRINT x
FOR n=0 TO 20
  PRINT b$(n)
NEXT n
```

See Also: STORE

RECORD
Syntax: RECORD #n,r
Action: Sets the number of the next record to br read or stored with GET or PUT.

Example: RECORD #1,15

See Also: FIELD,GET#, PUT#, SEEK

RELSEEK
Syntax: RELSEEK [#]n,x

```
Action:    Moves the random access file pointer forward (+X) or
           backwards (-X) 'X' number of bytes.


REM
Syntax:    REM remark
Action:    Whatever follows a REM coomand on a particular line
           is ignored by Basic. ' is synonoymous with REM.

Example:   REM This is a comment



RENAME
Syntax:    RENAME old$ AS new$
Action:    Renames a file.



REPEAT...UNTIL
Syntax:    REPEAT
           UNTIL end
Action:    Creates a pre-defined loop. The section of the program
           between repeat and until is repeated continuously until
           the condition is fulfilled.

Example:   REPEAT
           UNTIL MOUSEK       'Waits for mouse key to be pressed.



RESERVE
Syntax:    RESERVE n
Action:    Increases or decreases the memory used by basic
           'n' is a numeric expression which determines how big
           FRE(0) should be after this command. (see HIMEM, EXEC)

Example:   RESERVE 2560
           EXEC 0,"\PROGRAM.PRG","",""
           RESERVE

           2560  bytes  are  reserved and PROGRAM.PRG  is  loaded  and
           started. After running the reserved space is restored.

           Memory can be reserved in blocks of 256 bytes.
           If  n  is  negative then the whole of the  free  memory  is
           reserved.


RESTORE
Syntax:    RESTORE [label]
Action:    Positions the data pointer for READ.
           Places the data pointer at the beginning, or behind the
           label names 'label'
           'label' can be any list of characters and can contain
           digits, letters, underscore and full stops.  Unlike
           other variable names it can begin with a digit.


RESUME
Syntax:    RESUME    RESUME NEXT    RESUME label
Action:    The  RESUME command is only meaningful with  error  capture
```

(ON ERROR GOSUB) where it allows a reaction to an error.

RESUME repeats the erroneous command.
RESUME NEXT resumes program execution after an incorrect command.
RESUME 'label' branches to the 'label'.
If a fatal error occurs only RESUME 'label' is possible

Example:
```
ON ERROR GOSUB error_trap
ERROR 5
PRINT "and again..."
ERROR 5
PRINT "is not reached."
'
PROCEDURE error_trap
    PRINT "OK, error intercepted"
    RESUME NEXT
RETURN
```

RETURN
Syntax: RETURN
Action: Terminates a sub-routine

Syntax: RETURN x
Action: If the command RETURN is reached during program ececution and is within a FUNCTION...ENDFUNC execution, then the value given after it is returned.

RIGHT$
Syntax: RIGHT$(string[,n])
Action: Returns the last characters or 'n' number of characters (from the right) of a character string 'string'
Example:
```
PRINT RIGHT$"Hello GFA",3)   'PRINTS GFA
```

RINSTR
Syntax:
```
RINSTR(a$,b$)
RINSTR(a$,b$,[x])
RINSTR([x],a$,b$)
```
Action: Operates in same way as INSTR except that search begins at the right end of a$.

RMDIR
Syntax: RMDIR "directory name"
Action: Deletes empty directories

RND
Syntax: RND [(x)]
Action: Returns a random number between 0 and 1

The optional parameter (x) is disregarded, and returns a random number between 0 (inclusive) and 1 (exclusive)

ROL
Syntax: ROL(x,y)
ROL&(x,y)
ROL|(x,y)
Action: Rotates a bit pattern left.

ROR
Syntax: ROR(x,y)
ROR&(x,y)
ROR|(x,y)
Action: Rotates a bit pattern right.

ROUND
Syntax: ROUND(x[,n])
Action: Rounds off the numeric expression x.

Example: y=ROUND(-1.2)
PRINT y,ROUND(1.7)

RSET
Syntax: RSET a$=b$
Action: Moves a string expression, right justified to a string.
See Also: LSET,MID$

The following commands are part of the Resource Library. These routines provide the creation of a graphical user interface. The full descriptions of these functions are beyond the scope of these abreviated manual. A full description is contained within the full GFA-BASIC Reference manual and also the GFA-BASIC Software Development Book.

RSRC_FREE
Syntax: ~RSRC_FREE(0)
Action: This function releases the memory space reserved by RSRC_LOAD.
Returns 0 if an error.

RSRC_GADDR
Syntax: ~RSRC_GADDR(type,index,addr)
Action: This function determines the address of a resource structure after it has been loaded with RSRC_LOAD. Depending on the version of GEM, this function may only work for Object trees and Alert boxes.
Returns 0 if an error.

Type:0 OBJECT TREE
1 OBJECT
2 TEDINFO
3 ICONBLK
4 BITBLK
5 STRING
6 image data
7 obspec
8 te_ptext
9 te_ptmplt
10 te_pvalid

```
                11   ib_pmask
                12   ib_pdata
                13   pb_ptext
                14   bi_pdata
                15   ad_frstr
                16   ad_frimg

            Index:   The  number  of  the  object  whose  address  is
                     required, counting objects of that type one by one
                     from the beginning of the resource file.

            addr:    The required address.

Example:    ~RSRC_GADDR(0,0,TREE%)



RSRC_LOAD
Syntax:     RSRC_LOAD(name$)
Action:     This  function reserves memory and loads a  resource  file.
            Then  internal  pointers are set and  the  co-ordinates  of
            characters converted into pixel format.

Example:    ~RSRC_LOAD("TEST.RSC")



RSRC_OBFIX
Syntax:     RSRC_OBFIX(tree,obj)
Action:     This function converts the coordinates of an object  within
            a  tree,  from character coordinates to pixel  coordinates,
            taking  into account the current screen resolution.  It  is
            automatically called by RSRC_LOAD,  but must be used if the
            object is created direct in memory by POKE.

            tree:    address of the object tree
            obj:     object number



RSRC_SADDR
Syntax:     RSRC_SADDR(type,index,addr)
Action:     This function sets the address of an object.
            Returns 0 if an error.

            type:    type of structure
            index:   the number of the object
            addr     address



RUN
Syntax:     RUN(a$)
Action:     Runs the program in memory,  or if a file name is  supplied
            will load and then run the appropriate program.

Example:    RUN "A:\PROGRAM.GFA"



SAVE
PSAVE
```

Syntax: SAVE a$
PSAVE a$
Action: Saves a program file (psave is with list protection)
'file name' is the name of the program.
Programs which are saved with psave are not listed but
run straight after the command 'load' is given.

SEEK
Syntax: SEEK [#]n,x
Action: Sets the file pointer on the byte number 'x' of file #n
'n' is an integer expression between 0 and 99 which
refers to the channel number. 'x' has a value (total)
either greater or smaller than the length of the file
addressed.

SCRP_READ
Syntax: SCRP_READ(path$)
Action: This function reads data, left there by another program,
from a small internal buffer, thus allowing communication
between GEM programs. Returns 0 if an error.
Example: SCRP_READ(a$)
See Also: SCRP_WRITE

SCRP_WRITE
Syntax: SCRP_WRITE(path$)
Action: This function writes data, into a small internal buffer,
thus allowing communication between GEM programs.
Example: SCRP_WRITE("A:\PROGRAM.TXT")
See Also: SCRP_READ

SDPOKE
Syntax: SDPOKE x,y
Action: Allows DPOKE to operate in supervisor mode, so that
protected address (0 to 2047) can be modified.

SEEK
Syntax: SEEK #n,pos
Action: Absolute positioning of data pointer within file. This
allows the realisation of indexed sequential file access.
The numerical expression n contains the channel number of
the file.

SELECT
Syntax: SELECT x
CASE y [TO z] or CASE y [,z,...]
CASE TO y
CASE y TO
DEFAULT
ENDSELECT
CONT
Action: A conditional command which enables execution of specified
program segments depending on an integer.

The maximum of a CASE is 4 characters (eg CASE "A,B,C,D"

The CONT command provides a method of jumping over a CASE or DEFAULT command.

Example:
```
REPEAT
    a%=ASC(INKEY$)
    SELECT a%
    CASE 65 TO 90
         PRINT "CAPITAL LETTER"
    CASE 97 TO 122
         PRINT "LOWER CASE LETTER"
    DEFAULT
         PRINT "NOT CAPITAL OR LOWER CASE"
    ENDSELECT
UNTIL a%=27
```

SETCOLOR

Syntax: SETCOLOR i,r,g,b
SETCOLOR i,n

Action: Defines the colours red, green and blue for the colour register 'i'.
'r,g,b' are the levels of the three primary colours from 0 to 7.
Another way of defining colours is to use the value 'n' where n=r*256+g*16+b

See Also: COLOR,VSETCOLOR

SETDRAW See DRAW command.

SETMOUSE

Syntax: SETMOUSE mx,my,[,mk]

Action: The SETMOUSE command permits the positioning of the mouse cursor under program control. Tje optional parameter mk can simulate the mouse button being pressed or released.

Example:
```
FOR i%=0 TO 300
    HIDEM
    SETMOUSE i%,i%
    PLOT MOUSEX,MOUSEY
    SHOWM
    PAUSE 2
NEXT i%
```

SETTIME

Syntax: SETTIME time$,date$

Action: Sets the time and the date.

time$ is a string expression which contains the time. hours, minutes and second can be displayed. The colons are optional as two digits have to be entered. The seconds can also be left out.

date$ is a character string expression for the date. It must always contain: day, month and year, each separated by a full stop.

```
Example:   PRINT DATE$,TIME$
           SETTIME "17:30:30","27.10.1952"
           PRINT DATE$,TIME$
```

```
SGET
Syntax:    SGET screen$
Action:    Fast reading of the entire screen area into a string
           variable.
Example:   PCIRCLE 100,100,50
           SGET b$
           ~INP(2)
           CLS
           ~INP(2)
           SPUT b$
See Also:  SPUT, GET, PUT and BMOVE
```

```
SGN
Syntax:    SGN(x)
Action:    Ascertains whether 'x' is positive, negative or 0
           'x' can be any numeric expression.  SGN(x) is the
           mathematic sign function.
```

The following commands are part of the Shell Library and enable an application to call another, preserving both the original application and its environment.
The full descriptions of these functions are beyond the scope of these abreviated manual. A full description is contained within the full GFA-BASIC Reference manual.

```
SHEL_ENVRN
Syntax:    SHEL_ENVRN(addr,search$)
Action:    This function determines the values of variables in the GEM
           environment.
           Returns 1.

           search$: The string to be sought
           addr:    address of the byte following the string

Example:   PRINT SHEL_ENVRN(a%,"PATH")
           PRINT CHAR{a%-4}

           ' Displays: PATH=A:\
```

```
SHEL_FIND
Syntax:    SHEL_FIND(paths$)
Action:    This function searches for a file and supplies the full
           file specification.  First the path on the current drive is
           searched, then the root directory of drive A:.

           Returns 0 if file not found, or 1 if found.

           On entry:
           path$:   String contains sought after filename.
```

On exit:
path$: Contains the full file specification if the file was found, otherwise it is unchanged.

SHEL_GET
Syntax: SHEL_GET(num,x$)
Action: This function reads data from the GEMDOS environmental string buffer (into which the file DESKTOP.INF is read on start up).

Returns 0 if an error.

num: number of bytes to be read
x$: string to contain data

Example:
```
SHEL_GET(500,x$)
PRINT x$
```

SHEL_PUT
Syntax: SHEL_PUT(len,x$)
Action: This function writes data into the GEMDOS environmental string buffer.

Returns 0 if an error.

x$: String containing the data to be written
len: number of bytes to be written

Example:
```
'Register GFA-BASIC
~SHEL_GET(2000,a$)
q%=INSTR(a$,CHR$(26))
IF q%
    a$=LEFT$(a$,q%-1)
    IF INSTR(a$,"GFABASIC.PRG")=0
         a$=s$+"#G 03 04 A:\GFABASIC.PRG@*.GFA
                   +MKI$(&HDOA)+CHR$(26)
         ~SHEL_PUT(LEN(a$),a$)
    ENDIF
ENDIF
```

' Registers that all .GFA files cause GFABASIC.PRG to be loaded when clicked on.

SHEL_READ
Syntax: SHEL_READ(cmd_string$,tail_string$)
Action: This function allows the program to identify the command by which it was invoked and supplies the name, eg GFABASIC.PRG, and the command line if any.

cmd_string$ string variable to contain the command line.

tail_string$ string variable to contain name.

SHEL_WRITE
Syntax: SHEL_WRITE(prg,grf.gem.cmd$,nam$)
Action: This function informs the AES that another application is to be started after the current one has terminated. In contrast to p_exec (GEMDOS 75), however the current program does not remain in memory.

```
prg:    0 Back to desktop
        1 Load new program
grf:    0 TOS program
        1 Graphic application
gem:    0 not a GEM application
        1 GEM application
cmd$    command line string
nam$    name of next application
```

Example: ~SHEL_WRITE(1,1,1,"","GFABASIC.PRG")

SHL
Syntax: SHL(x,y)
SHL&(x,y)
SHL|(x,y)
Action: Shifts a bit pattern left

SHOWM
Syntax: SHOWM
Action: Makes the mouse pointer appear.
See Also: HIDEM

SHR
Syntax: SHR(x,y)
SHR&(x,y)
SHR|(x,y)
Action: Shifts a bit pattern right

SIN
Syntax: SIN(x)
Action: Returns the sine value of 'x'

SINGLE{}
Syntax: SINGLE{x}
Action: Reads/writes a 4 byte floating point variable in IEEE single precision format.

SINQ
Syntax: SINQ(degrees)
Action: Returns the extrapolated sine of a numeric expression.

SLPOKE
Syntax: SLPOKE x,y
Action: Allows LPOKE to operate in supervisor mode, so that protected address (0 to 2047) can be modified.

SOUND
Syntax: SOUND chn,vol,note,octave[,dur]
SOUND chn,vol,note,#period[,dur]
Action: GENERATES MUSICAL NOTES
'chn' is a 1, 2, or 3 and selects the sound channel.
'vol' selects the volume.
'note' is a value of 1 to 12 and selects notes:
1=C,
2=C#
3=D
4=D#
5=E
6=F
7=F#
8=G
9=G#
10=A
11=A#
12=B
'octave' is between 1 and 8, and determines octave.
'dur' is the time in 1/50ths of a second that GFA Basic has to wait before execution of the next command.
A further possibility to choose the pitch is to enter 'period' prefixed by '#' instead of 'note' and 'octave'.
The period can be calculated from the frequency with:

Period = TRUNC(125000/frequency +0.5)

SPACE$
Syntax: SPACE$(x)
Action: Creates a character string containing 'x' spaces.

SPC
Syntax: SPC(n)
Action: Produces 'n' spaces in a print command

SPOKE SDPOKE SLPOKE
Syntax: SPOKE x,n SDPOKE x, SLPOKE x,n
Action: Writes 1, 2 or 4 bytes into an area of memory which begins with the address 'x'

SPRITE
Syntax: SPRITE A$[,x,y]
Action: Puts the sprite defined in a$ at (X,Y) or, if no coordinates are given, deletes it.
A$ = MKI$(X POSITION)
+ MKI$(Y POSITION)
+ MKI$(0=NORMAL OR 1=XOR MODE)

```
              + MKI$(SCREEN COLOUR MOSTLY 0)
              + MKI$(SPRITE COLOUR MOSTLY 1)
              + BIT PATTERN OF SCREEN AND SPRITE

           Unlike defmouse, the bit patterns for screen and sprite
           are not stored in separate blocks but in alternate
           words (16 bits).
           If the same sprite is put onto the screen in another
           position then the first sprite is deleted.


SPUT
Syntax:    SPUT var
Action:    Fast copying of a 32000 byte string into the screen
           area.
See Also:  SGET, PUT, GET and BMOVE



SQR
Syntax:    SQR(X)
Action:    Calculates the square root of 'X'.



SSORT
Syntax:    SSORT a(s) {OFFSET o][WITH i()][,n[,j%()]]
           SSORT x$(s) WITH i() [,n[,j%{}]]
Action:    Sorts the elements in an array by its size using the Shell-
           Metzner method.

           a()      array or string array
           i()      integer array
           j%       4byte integer array
           x$()     string array
           s        + or - or no sign



STE?
Syntax:    STE?
Action:    Returns -1 for STE otherwise 0



STICK
Syntax:    STICK m
           STICK(p)
Action:    The function STICK(p) returns the position of a joystick.
           STICK 0 causes port 0 to supply mouse information.
           STICK 1 causes port 1 to read the joystick.

Example:   STICK 1
           REPEAT
               direction%=STICK(0)
               fire%=STRIG(0)
               SELECT direction%
               CASE 4
                    PRINT "LEFT"
               CASE 8
                    PRINT "RIGHT"
               CASE 2
```

```
                PRINT "DOWN"
            CASE 1
                PRINT "UP"
            ENDSELECT
        UNTIL fire!
        WHILE STRIG(0)
        WEND
```

```
STOP
Syntax:    STOP
Action:    Stops execution of a program.
           Unlike the END command it does not close any files and
           by typing CONT the program will resume from the line
           following the STOP command.
```

```
STORE
Syntax:    STORE #i,x$()[,n[TO m]]
Action:    Fast save of a string array as a text file. The instruction
           STORE  is  used for sending the contents of an array  to  a
           file or data channel (elements seperated by CR/LF).

See Also:  RECALL
```

```
STR $
Syntax:    STR$(X)
Action:    Transforms the value 'X' into a character string.
```

```
STRING$
Syntax:    STRING$(N,string)  OR  STRING$(N,C)
Action:    Produces a string formed by repeating 'string' or
           CHR$(C) 'N' times. 'N' is a number from 0 to 32767.
```

```
SUB
Syntax:    SUB VAR,N
Action:    Deducts 'N' from 'VAR'. Same as VAR=VAR-N but executes
           almost twice as fast.
Example:   x=57
           SUB x,3*5
           PRINT x  'PRINTS 42
```

```
SUB()
Syntax:    SUB(x,y)
Action:    Corresponds to x-y

Example:   PRINT SUB(5^3,4*20+3)   'PRINTS 42
```

```
SUCC()
Syntax:    SUCC(n)
Action:    Determines the next higher number.
```

See Also: PRED()

SWAP
Syntax: SWAP var1,var2
Action: Exchanges the values of 'var1' and 'var2'.
The variables must be of the same type.
When swapping array fields the dimensions are also swap
ped.

SWAP()
Syntax: SWAP(n)
Action: Swaps the high and low words of a varaible.

SYSTEM
Syntax: SYSTEM
Action: Causes a return to the desktop, same as quit.

TAB
Syntax: TAB(n)
Action: Sets the tabulator to the nth column.
Tab can only be used in conjunction with the print
command.
If the current position is already past 'N' then the
tab function is set for the next line.

TAN
Syntax: TAN(X)
Action: Returns the tangent of 'X' (X is the angle in radians).

TEXT
Syntax: TEXT X,Y, [L,]string
Action: Puts a text onto the screen at graphics coordinates
'X,Y'. The graphics can first be defined by using
the command DEFTEXT.

TIME$
Syntax: TIME$
Action: Returns the system time as a string.
Format: hh:mm:ss and is updated every two seconds.

Example: PRINT TIME$

TIME$=
Syntax: TIME$=a$
Action: The time can be set.

```
Example:   TIME$="20:15:30"
```

```
TIMER
Syntax:    t%=TIMER
Action:    TIMER suuplies the elapsed time in 1/200 seconds since  the
           system was started.

Example:   t%=TIMER
           FOR i%=1 TO 2500
           NEXT i%
           PRINT (TIMER-t%)/200;" Seconds"
```

```
TITLEW
Syntax:    TITLEW n,"title"
Action:    Gives the window number 'n', the new title 'title'.
```

```
TOPW
Syntax:    TOPW #1
Action:    Activates the windows number n.
```

```
TOUCH
Syntax:    TOUCH[#]n
Action:    Updates the date and time stamps od a file,  giving it  the
           current system time and date.

Example    OPEN "u",#1,"TEST.TXT"
           TOUCH #1
           CLOSE #1
```

```
TRACE$
Syntax:    TRACE$
Action:    The  variable TRACE$ contains the command which is next  to
           be processed.

See Also:  TRON,TROFF
```

```
TRIM$
Syntax:    TRIM$(a$)
Action:    Removes spaces at the beginning of a string expression.
```

```
TROFF
Syntax:    TROFF
Action:    Switches the trace function off.
```

TRON
Syntax: TRON
Action: Switches the trace function on. This causes each command to be listed on the screen.

TRON#
Syntax: TRON #1
Action: Switches the trace function on. This causes each command to be listed to the relevant channel number.

TRONproc
Syntax: TRON tr_proc
Action: A procedure can be specified which is called before the execution of each command.

TRUE
Syntax: TRUE
Action: Constant 0. This is simply another way of expressing the value of a condition when it is true and is equal to zero. (see also FALSE).

TRUNC
Syntax: TRUNC(X)
Action: Returns the integer portion of 'X'.

TT?
Syntax: TT?
Action: Returns -1 for 68020 or 68030 processeor, otherwise 0.

TYPE
Syntax: TYPE(ptr)
Action: Determines the type of the variable at which a pointer is set.

'ptr' is an integer expression (usually *var).
TYPE(ptr) returns a code according to the type of variable to which 'ptr' is pointing.

0=var
1=var$
2=var%
3=var!
4=var()
5=var$()
6=var%()
7=var!().

On errors -1 is returned.

UPPER$
Syntax: A$="basic"
PRINT UPPER$(A$)
PRINT UPPER$("1a")
Action: Transforms all lower case letters of a string to upper case. Any non letter characters are left unchanged.

V:
Syntax: V:x
Action: Returns the address of a variable or strings or elements of an array.

VAL
Syntax: VAL(X$)
Action: Transforms 'X$' into a number, as far as possible. In the case of a purely alphabetical string the value 0 is returned.

VAL?
Syntax: VAL?(X$)
Action: Determines the number of characters starting at the beginning of a string that can be converted into a numerical value with VAL.

VAR
Syntax: name([a,b,...] VAR x,y..a(),b(),...)
Action: Declaration part of the parameter list for a PROCEDURE or FUNCTION.

Example:
```
sum(13,12,a)
sum(7,9,b)
PRINT a,b
'
PROCEDURE sum(x,y,VAR z)
    z=x+y
RETURN
```

VARIAT()
Syntax: VARIAT(n,k)
Action: Returns the number of permutations of n elements to the kth order without repitition.

Example:
```
PRINT VARIAT(6,2)       'prints 30
```

See Also: FACT(), COMBIN()

VARPTR
Syntax: VAPTR(var)
Action: Determines the address or starting address of a variable 'var'.

VDIBASE
Syntax: VDIBASE
Action: Dangerous pokes!
Determines the address above the area used by basic and the required tables and variables.
This is the point from which this version of gem keeps parameters for the vdi (text style, clipping etc.).
By use of peek and poke in this area, various effects (and nasty crashes!) can be obtained.

VDISYS
Syntax: VDISYS[opcode [,c_int,c_pts[,subopc]]]
Action: The VDI function with function code opcode is called. If opcode is not specified, then the function code must, like other parameters, be placed in the control block with DPOKE.

The depth of this command is betond the scope of this abbreviated manual.

VOID
Syntax: VOID exp
Action: This command performs a calculation and forgets the result. Sounds silly but there are occasions when this command is required, eg. forced garbage collection (fre(0)), waiting for a keystroke (inp(2)), or calling various bios, xbios, gemdos or c: routines which have no parameters.

VQT_EXTENT
Syntax: VQT_EXTENT(text$[,x1,y1,x2,y2,x3,y3,x4,y4])
Action: Returns the corner coordinates of a rectangle which will surround the text in text$. The coordinates can either be found in the variables x1,y1 to x4,y4, or in PTSOUT(7). The corner pointers are numbered in a clockwise direction.

Example:
```
INPUT text$
CLS
ATEXT 100,25,2,text$
~VQT_EXTENT(text$,x1,y1,x2,y2,x3,y3,x4,y4)
BOX x4+100,y4+25,x2+00,y2+25
```

VQT_NAME
Syntax: VQT_NAME(i,font_name$)
Action: Supplies the handle of the font with the indentification number i and places the name of the loaded character set into the string variable font_name$.

VSETCOLOR
Syntax: VSETCOLOR colour,red,green,blue
VSETCOLOR colour,composite
Action: Due to an error in TOS, SETCOLOR does not correspond to the registers used by COLOR. VSETCOLOR is used to overcome this problem.

```
Low Resolution
SETCOLOR     0 1 2 3 4 5 6 7 8  9 10 11 12 13 14 15
VSETCOLOR    0 2 3 6 4 7 5 8 9 10 11 14 12 15 13  1

Medium Resolution
SETCOLOR     0 1 2 3
VSETCOLOR    0 2 3 1

High Resolution
SETCOLOR 0,even = VSETCOLOR 0,0
SETCOLOR 0,odd  = VSETCOLOR 0.&H777
```

The term composite is calculated the same way as with SETCOLOR: ie rgb=(r*256)+(g*16)+b

VST_LOAD_FONTS
Syntax: VST_LOAD_FONTS(x)
Action: Loads the additional character sets specified in ASSIGN.SYS, and the number of loaded fonts is returned.

Example:
```
RESERVE 25600
num_fonts%=VST_LOAD_FONTS(0)
face%=VQT_NAME(num_fonts%,fonts$)
FOR i%=1 to num_fonts%
    DEFTEXT,,,,face%
    TEXT 80,80,"This is the "+font$+" font."
    ~INP(2)
NEXT i%
~VST_UNLOAD_FONTS(0)
RESERVE
```

VST_UNLOAD_FONTS
Syntax: VST_UNLOAD_FONTS(x)
Action: Removes the character set previously loaded with VST_LOAD_FONTS from memory.

VSYNC
Syntax: VSYNC
Action: Enables synchronization with the screen by waiting until the next vertical sync pulse is received - helps avoid flickering of the screen during animation when using GET and PUT.

Example:
```
t%=TIMER
FOR i%=1 TO 100
    VSYNC
```

```
NEXT i%
PRINT SUB(TIMER,t%)/200
' PRINTS the time for 100 scans of screen.
```

VTAB
Syntax: VTAB line
Action: VTAB positions the cursor to the specified column or line number . Note that the cursor columns and lines are counted from 1, not 0.

See Also: HTAB, PRINT AT, TAB

The following VDI Workstation routines and functions are only available if GDOS has been booted and a valid ASSIGN.SYS file is available. In depth documentation on the VDI routines are beyond the scope of this abbreviated manual. Further information can be found in the full GFA-BASIC Interpreter Manual or Software Development Book.

V_CLRWK
Syntax: V_CLRWRK()
Action: This function clears the output buffer. For example the screen or the printer buffer is cleared.

V_CLSVWK
Syntax: V_CLSVWK(id)
Action: Closes a virtual workstation opened with V_OPNVWK.

V_CLSWK
Syntax: V_CLSWK()
Action: Closes the current workstation opened with V_OPNWK().

V_OPNVWK
Syntax: V_OPNVWK(id,1,1,1,1,1,1,1,1,1,2)
Action: Opens a virtual screen driver and supplies the handle for the specified device id.

V_OPNWK
Syntax: V_OPNWK(id)
Action: Supplies the handle for the specified device id.

V_UPDWK
Syntax: V_UPDWK()
Action: Sends buffered graphic instructions to the attached device.

V~H
Syntax: V~H

```
Action:    Returns the internal VDI handle of GFA-BASIC.

           V~H=x    'Sets internal VDI handle

           V~H=-1   'Sets VDI handle to value from V_OPNVWK()


W:
Syntax:    W:x
Action:    Allows  passing of numerical expressions to  the  operating
           system and C routines as a word (2-byte).

See Also:  L:



WAVE
Syntax:    WAVE voice,env,form,per,del
Action:    Produces noises from the three sound channels.
           WAVE 0,0 switches off all sound channels.

           voice:   Any  channel  or combination of  channels  may  be
                    activated  simultaneously.  The value of voice  is
                    256 * by the period.

                    1 Channel 1
                    2 Channel 2
                    4 Channel 3
                    8 Noise (Channel 1)
                   16 Noise (Channel 2)
                   32 Noise (Channel 3)

           env:     Specifies  the channels for which the wnvelope  is
                    to be active.

                    1 Channel 1
                    2 Channel 2
                    3 Channel 3

           form:    Envelope shape

                    0 - 3     As 9
                    4 - 7     As 15
                    8         Falling sawtooth
                    9         Falling linear
                    10        Triangle falling
                    11        Falling linear, then to max
                    12        Rising sawtooth
                    13        Rising linear and holding
                    14        Triangle, then rising
                    15        Linear rising, then to zero

           per:     Period of the waveform multiplied by 125000.

           del:     Delay in 1/59ths second before the next  GFA-BASIC
                    command is executed.

Example:   SOUND 1,15,1,4,20
           SOUND 2,15,4,4,20
           SOUND 3,15,8,4,20
           WAVE 7,7,0,65535,300
           '  A tone is generated from each channel then modulated  by
```

```
            '  WAVE.

See Also:   SOUND
```

```
WHILE....WEND
Syntax:     WHILE condition
            WEND
Action:     Creates a conditional loop between while and wend until
            the 'condition' is fulfilled. This is checked at the
            beginning of the loop and so it is possible that the
            loop is never executed.
```

The following functions are all functions of the Window Library

```
WINDTAB
Syntax:     WINTAB
            WINTAB(i,j)
Action:     Gives the address of the Window Parameter Table. This table
            contains  the  data  that determines the  appearance  of  a
            window.

            Window Parameter Table:

            Offset
            0         Handle of Window 1
            2         Attributes for Window 1
            4         x coordinates of Window 1
            6         y coordinates of Window 1
            8         Width of Window 1
            10        Height of Window 1
            12-22     Parameters for Window 2
            24-34     Parameters for Window 3
            36-46     Parameters for Window 4
            48        -1
            50        0
            52-58     Coordinates and size of Desktop
            60-63     Coordinates of the joint of the four windows
            64-65     Origin for graphic instructions (CLIP OFFSET)

            Window Attibute element:

            Bit
            0         Window Title
            1         Close box
            2         Full box
            3         Move box
            4         Information line
            5         Sizing box
            6         Up arrow
            7         Down arrow
            8         Vertical slider
            9         Left arrow
            10        Right arrow
            11        Horizontal slider

Example:    OPEN #1,100,120,200,70,&HFFF
```

```
            ' corresponds to:
            DPOKE WINTAB+2,&HFFF
            DPOKE WINTAB+4,100
            DPOKE WINTAB+6,120
            DPOKE WINTAB+8,200
            DPOKE WINTAB+10,70
            OPENW 1
            ' or
            WINTAB(1,1)=&HFFF
            WINTAB(1,2)=100
            WINTAB(1,3)=120
            WINTAB(1,4)=200
            WINTAB(1,5)=70
            OPENw 1

WIND_CALC
Syntax:     WIND_CALC(w_type,attr,ix,iy,iw,ih,ox,oy,ow,oh)
Action:     This function computes the total size of the work area from
            the size of the window.
            Returns 0 if an error.

            w_type:             0 Compute total size
                                1 Compute work area size

            attr:               Bit
                                0   Title bar with name
                                1   Close box
                                2   Full size box
                                3   Move bar
                                4   Info line
                                5   Size box
                                6   Up arrow
                                7   Down arrow
                                8   Vertical slider
                                9   Left arrow
                                10  Right arrow
                                11  Horizontal slider

            ix,iy               top left coorinates
            iw,ih               width and height

            ox,oy               Calculated top left coordinates
            ow,oh               Calculated width and height



WIND_CLOSE
Syntax:     WIND_CLOSE(handle)
Action:     Closes the specified window.



WIND_CREATE
Syntax:     WIND_CREATE(attr,wx,wy.ww.wh)
Action:     Allocates  a  new window,  specifying  the  attributes  and
            maximum size. The handle of the window is returned.

            attr:               Bit
                                0   Title bar with name
                                1   Close box
                                2   Full size box
                                3   Move bar
```

```
                              4   Info line
                              5   Size box
                              6   Up arrow
                              7   Down arrow
                              8   Vertical slider
                              9   Left arrow
                              10  Right arrow
                              11  Horizontal slider

            wx                Max x position of left edge
            wy                Max y position of top edge
            ww                Max width of window
            wh                Max height of window


WIND_DELETE
Syntax:     WIND_DELETE(handle)
Action:     Deletes a window allocation and frees reserved memory.



WIND_FIND
Syntax:     WIND_FIND(fx,fy)
Action:     Determines  the  id  number of a window  within  which  the
            specified coordinates lie.

            fx:               x coordinates
            fy:               y coordinates



WIND_GET
Syntax:     WIND_GET(handle,code,w1,w2,w3,w4)
Action:     Supplies information about a window determined by the code.

            handle:           Id number of the window

            code:             depending upon the code,  information is
                              supplied in w1,w2,w3,w4.


            code: 4           supplies size of window work area
                     w1:      x coordinates
                     w2:      y coordinates
                     w3:      width
                     w4:      height

            code: 5           supplies  total  size of  entire  window
                              including borders

                     w1:      x coordinates
                     w2:      y coordinates
                     w3:      width
                     w4:      height

            code: 6           supplies total size of previous window

                     w1:      x coordinates
                     w2:      y coordinates
                     w3:      width
                     w4:      height
```

code: 7 supplies the total max size of the window.

w1: x coordinates
w2: y coordinates
w3: width
w4: height

code: 8 supplies the position of the horizontal slider

w1: 1=far left 1000=far right

code: 9 supplies the position of the vertical slider

w1: 1=top 1000=bottom

code: 10 supplies the id number of the top (active) window

w1: id number of active window

code: 11 supplies the coordinates of the first rectangle in the specified rectangle list.

w1: x coordinates
w2: y coordinates
w3: width
w4: height

code: 12 supplies the coordinates of the next rectangle in the specified windows rectangles list

w1: x coordinates
w2: y coordinates
w3: width
w4: height

code: 13 reserved

code: 15 supplies the size of the horizontal slide bar compared to its max possible

w1: -1 = minimum size
1 = small
1000 = full width

code: 16 supplies the size of the vertical slide bar compared to its max possible

w1: -1 = minimum size
1 = small
1000 = full height

WIND_OPEN
Syntax: WIND_OPEN(handle,wx,wy,ww,wh)
Action: Draws on the screen a window previously created with WIND_CREATE.

WIND_SET

Syntax: WIND_SET(handle,code,w1,w2,w3,w4)

Action: Changes the parts of a window according to the specified function code.

code: 1 — Sets new windows components as with WIND_CREATE.

- w1: new window element

code: 2 — Gives a window a new title

- w1: Hi word
- w2: Low word of address of title string

code: 3 — Specifies a new information line

- w1: Hi word
- w2: Low word of address of information string

code: 5 — Sets the window size

- w1: x coordinates
- w2: y coordinates
- w3: width
- w4: height

code: 8 — Positions the horizontal slider

- w1: 1=far left 1000=far right

code: 9 — Positions the vertical slider

- w1: 1=top 1000=bottom

code: 10 — Sets top (active) window

- w1: id number

code: 14 — Sets a new desk top Menu tree

- w1: Low word
- w2: High word of address of new tree
- w3: id number of the first object to be drawn

code: 15 — Sets the size of the vertical slide bar

- w1: -1=minimum size
  1= small
  1000 max size

code: 16 — Sets the size of the horizontal slider bar

- w1: -1=minimum size
  1= small
  1000 max size

WIND_UPDATE

```
Syntax:    WIND_UPDATE(flag)
Action:    Coordinates  all functions concerned with  screen  redraws,
           in particular with drop down menus.

           flag: 0            screen redraw completed
                 1            screen redraw starting
                 2            application loses mouse control
                 3            application takes on mouse control


WORD()
Syntax:    WORD(x)
Action:    Extends  a  word to long word length (32 bits)  by  copying
           bits  15  to bit positions 16 to 31,  thus  preserving  the
           sign.


WORK_OUT()
Syntax:    WORK_OUT(x)
Action:    Determines  the  values found in INTOUT(0)  to  INTOUT(44),
           PTSOUT(0)  and PTSOUT(1) after returning from the  function
           OPEN_WORKSTATION.



WRITE
Syntax:    WRITE [ expressions ][ ; ]
           WRITE #n [ expressions ][ ; ]
Action:    Stores data in a sequential file to be read with input.
           Unlike the PRINT command the numbers are separated by
           commas and the strings are enclosed in quotes.



WRITE#
Syntax:    WRITE#n,expression
Action:    Saves  data  to sequential file,  for  later  reading  with
           INPUT#.

Example:   OPEN "o",#1,"TEST.DAT"
           WRITE #1,"Version ",3,".6"
           CLOSE #1
           OPEN "i",#1,"TEST.DAT"
           INPUT #1,v1$,v2$,v3$
           CLOSE #1
           PRINT v1$+V2$+v3$



W_HAND
Syntax:    W_HAND(#n)
Action:    Returns  the GEM handle of the window whose channel  number
           is n.

Example:   OPENW 2
           PRINT W_HAND(#2)
           ~INP(2)
           CLOSE #2


W_INDEX
Syntax:    W_INDEX(#hd)
```

Action: Returns the window number of the specified GEM handle.
         Reverse of W_HAND().

XBIOS

The XBIOS function is used to call XBIOS system routines.

~XBIOS(0,t%,l:p%.l:v%)

         Initialises the mouse handling routine but not compatible
         with GEM.

         t% 0   Switches mouse off
             1   Switches mouse into relative mode
             2   Switches mouse into absolute mode
             4   Mouse in keyboard mode
         p%   Address of information structure
         v%   Address of the mouse handling routine

r%=XBIOS(2)

         Returns the base address of the physical screen memory.

         r%   Address of the physical screen memory

r%=XBIOS(3)

         Returns the address of the logical screen memory when
         writing to the screen.

         r%   Address of the logical screen memory

r%=XBIOS(4)

         Returns the current screen resolution

         r% 0   320 x 200
             1   640 x 200
             2   640 x 400

~XBIOS(5,l:l%,l:p%,r%)

         Enables the resolution to be changed from low res and high
         res when using a colour monitor. Can not be used with GEM.

         l%   New address of logical screen memory
         p%   New address of the physical screen memory
         r%   New screen resolution (see XBIOS(4))

~XBIOS(6,L:adr%)

         Allows all colour registers to be reset at one time.

         adr%   Address of a table of 16 words, which contains new
                 pallete data.

r%=XBIOS(7,n%,c%)

         Sets or gets a colour register.

```
            r%       For  c%=-1 the previous specified colour  register
                     is returned.
            n%       Colour register
            c%       New colour, at c%=-1 see r%

r%=XBIOS(8,L:b%,L:f%,d%,sec%,t%,side%,n%)

            Reads sectors of a disk

            r%       0 if no error
            b%       address of the area from which sectors are read
            f%       unused
            d%       drive number (0=A, 1=B etc)
            sec%     sector number
            t%       track number
            side%    disk side (0 or 1)
            n%       number of sectors to be read

r%=XBIOS(9,L:b%,L:f%,d%,sec%,t%,side%,n%)

            Writes sectors to a disk

            r%       0 if no error
            b%       address of the area to which sectors are written
            f%       unused
            d%       drive number (0=A, 1=B etc)
            sec%     sector number
            t%       track number
            side%    disk side (0 or 1)
            n%       number of sectors to be written

r%=XBIOS(10,L:b%,L:f%,d%,sec%,t%,side%,i%,L:m%,v%)

            A trace of the disk formats

            r%       0 if no error
            b%       address of an area for intermediate memory
            f%       unused
            d%       drive number (0=A, 1=B etc)
            sec%     sectors per track
            t%       track number to be formatted
            side%    disk side (0 or 1)
            i%       Interleave factor (normaaly 1)
            m%       Magic number &H87654321
            v%       value in sectors of format (normally &HE5E5

~XBIOS(12,n%,L:adr%)

            Outputs the contents of a block of memory to MIDI.

            n%       number of bytes -1
            adr%     address of the source storage area

~XBIOS(13,n%,L:adr%)

            Sets the MFP interrupt vector on the ST.  This can only  be
            used from assembly language or C and is not available  from
            GFA-BASIC.

            n%       Interrupt number
            adr%     new address of the interrupt
```

r%=XBIOS(14,d%)

Returns the address of the I/O table used by the serial interface.

r%	Address of the data buffer for the serial I/O table
d%	0: RS232 1: IKBD 2: MIDI

~XBIOS(15,b%,h%,ucr%,rsr%,tsr%,scr%)

Configures the serial interface. The parameters remain unchanged with a value of -1.

b%	Baud rate
h%	hand shake mode 0: no handshake 1: XON/XOFF 2: RTS/CTS 3: both
ucr%	USART control register of MFP
rsr%	receiver status register of MFP
tsr%	transmitter status register of MFP
scr%	synchronous character register of MFP

r%=XBIOS(16,L:us%,L:sh%,L:cl%)

Changes the keyboard translation tables.

r%	address of the KEYTAB structure
us%	address of the table for keys without shift
sh%	address of the table for keys with shift
cl%	address of the table for keys with Cap-lock

r%=XBIOS(17)

Returns a random number

r%	number with 24 bit accuracy (0 to 16777215)

~XBIOS(18,L:b%,L:s%,d%,f%)

Creates a boot sector for the disk in memory

b%	address of a 512 byte buffer for producing the boot sector
s%	serial number that forms part of the boot sector -1: previous serial retained >24 bits: random number returned
d%	disk type (tracks/sides) 0:40 tracks,single sided (180K) 1:40 tracks,double sided (360K, IBM) 2:80 tracks, single side (360K) 3:80 tracks, double sided (720K)
f%	0:non executable boot sector 1:executable -1:leave unchanged

r%=XBIOS(19,L:b%,L:f%,d%,sec%,t%,side%,n%)

Verifies the disk contents

b% address of the memory area with which a comparison is made.
f% not used
d% disk drive number
sec% start sector
t% track number
side% disk side
n% number of sectors

~XBIOS(20)

Calls the hardcopy routine and thus dumps the screen to printer.

r%=XBIOS(21,c%,s%)

Configure cursor.

r% when c%=5 returns the cursor blink rate
c% 0: Hide cursor
    1: Show cursor
    2: blink cursor
    3: solid cursor
    4: blink rate set in s%
    5: see r%
s% when c%=4, blink rate set to s%

~XBIOS(22,L:t%)

Sets date and time

t% Bits 0-4: seconds
         5-19: minutes
         11-15: hours
         16-20: day
         21-24: month
         25-31: year - 1980

r%=XBIOS(23)

returns date and time

r% see XBIOS(22) for bit settings

~XBIOS(24)

re installs the original keyboard allocation (see XBIOS(16))

XBIOS(25,n%,L:adr%)

writes bytes from memory to the keyboard processor (IKBD)

n% number bytes-1 to be sent
adr% address where the data to be sent is stored

~XBIOS(26,i%)

Disables an MFP interrupt.

i% interrupt number (0-15) to be disabled

~XBIOS(27,i%)

Enables an MFP interrupt.

i% interrupt number (0-15) to be enabled

~XBIOS(28,d%,reg%)

Reads and writes from and to the sound chip register

r% returns register value when reading
d% value to be writen (8 bits)
r% register number (0-15), bit 7 defines write mode when set.

~XBIOS(29,b%)

Sets the bit of port A on the register of the sound chip to zero

b% bit pattern wicj is OR'ed with existing contents.

~XBIOS(30,b%)

Sets the port A bit of the sound chip register to 1

b% bit pattern which is ANDed with the existing contents.

XBIOS(31,t%,c%,d%,L:adr%)

Sets the MFP timers

t% number of the timer (0 to 3)
c% control register
d% data register
adr% address of the timer interrupt routine

~XBIOS(32,L:adr%)

Starts a sound sequence, whic is processed in the interrupt

adr% address of the staorage area

r%=XBIOS(33,c%)

Sets or reads the printer parameters

r% current configuration when c%=1
c%

Bit	set	reset
0	Dot Matrix	Daisy Wheel
1	Monochrome	Colour
2	Atari	Epson
3	Parallel	RS-232
4	Continuous	Single sheet

r%=XBIOS(34)

```
            returns  address of table with vectors to the keyboard  and
            MIDI processor.

            r%       returned address

r%=XBIOS(35,a%,w%)

            sets and reads keyboard repeat rate

            r%       current data
                     bits      0-7  repeat rate
                               8-15  time of repeat delay
            a%       repeat delay
            w%       repeat rate

~XBIOS(36,L:adr%)

            Hardcopy routine returns parameter block address

            adr%     address  of  a parameter block  for  the  hardcopy
                     routine.

~XBIOS(37)

            waits for next vertical blank interrupt.

~XBIOS(38,L:adr%)

            calls an assembler routine in supervisor mode

            adr%     address of assembler routine

~XBIOS(39)

            turns off AES if not in ROM

r%=XBIOS(64,b%)

            contols and interrogates the blitter

            r%       b%=-1          current blitter status
                                    bit 1 : blitter there
XOR
Syntax:     x XOR y
Action:     Logical exclusive OR operator


XOR()
Syntax:     XOR(x,y)
Action:     Sets those bits in x that are different in x and y.


_DATA
Syntax:     _DATA
            _DATA=
Action:     Specifies the position of the DATA pointer.  _DATA is 0  if
            the next READ would result in an out of data message.


~
Syntax:     ~function
Action:     Similar to VOID. Forget the returned value
```

End of File