VMOS 0.24 Command Reference

Contents

About This Reference 7

A 8

Abs 8
Add 9
and 10
Asc 11
As 11

B 12

Bar 12
Beep 13
Beep Off 13
Bin$ 14
Bob 15
Bob Col 17
Bob Moving 18
Bob Off 19
Bob Path 20
Border 21
Border Dims 21
Border Palette 22
Box 23

C 24

C64 Border 24
C64 Off 25
C64 On 26
C64 Run 27
Chr$ 28
Circle 29
Cls 30
Color 31
Color( ) 32
Color Back 32
Copper Back 33
Copper Border 34
Copper Clear 35
Copper Color 36
Copper Off 37

Copper Offset ... 38
Copper On ... 39
Copper Palette ... 40
Copper Screen ... 41
Copper Sprite ... 42
Copper Tile ... 43
Copper Tile Scroll ... 43
Cos ... 44
Cpu8 6502 Load, Cpu8 Z80 Load ... 45
Cpu8 6502 Run, Cpu8 Z80 Run ... 46
Cpu8 Peek( ) ... 48
Cpu8 Poke ... 49
Cpu8 Speed ... 51
Cpu8 Stop ... 52
Curs ... 53
D ... 53
Data ... 53
Dec ... 54
Degree ... 55
Dim ... 55
Do ... 57
Double Buffer ... 57
Draw ... 58
E ... 59
Else ... 59
End ... 60
End If ... 61
End Proc ... 61
End Type ... 62
Envelope ... 62
Every ... 63
Exit ... 65
Exit If ... 65
F ... 66
False ... 66
Filter ... 67
Fire ... 68
Fm ... 68
Fm Envelope ... 69
Fm Instrument ... 70
Fm Operator ... 70
For ... 71
G ... 72
Get Image ... 72
Global ... 73

Gosub ... 74
Goto ... 75
H ... 75
Hex$ ... 75
Hide ... 76
Home ... 77
I ... 77
If ... 77
Inc ... 78
Ink ... 79
Inkey$ ... 79
Input ... 80
Instr ... 81
Instrument ... 82
J ... 82
Jdown ... 83
Jleft ... 83
Joy ... 84
Jright ... 85
Jup ... 85
K ... 86
Key ... 86
Key State ... 86
L ... 87
Left$ ... 87
Len ... 88
Let ... 88
Load Images ... 89
Load Sample ... 89
Load Tiles ... 90
Locate ... 91
Loop ... 92
Lower$ ... 92
M ... 93
Max ... 93
Mid$ ... 93
Min ... 94
mod ... 94
Mouse Key ... 95
N ... 96
Net Ip$ ... 96
Net Lookup$ ... 96
Next ... 97
not ... 97

O **98**
Off 98
On 98
or 98
P **99**
Palette 99
Paper 100
Param 101
Param$ 101
Param# 102
Param! 103
Paste Bob 103
Path Record 105
Pen 105
Pi# 106
Pi! 107
Ping 107
Play 108
Plot 109
Point 110
Polygon 110
Pop 111
Pop Proc 112
Print 112
Proc 113
Procedure 114
R **115**
Radian 115
Randomize 115
Raster 116
Read 117
Rem 118
Repeat 118
Reserve Bob 119
Restore 120
Return 121
Right$ 121
Rnd 122
Rplay 123
S **123**
Sam Loop 123
Sam Play 124
Sam Stop 125
Scancode 126
Screen 126

Screen Copy 127
Screen Hide 128
Screen Offset 129
Screen Open 130
Screen Palette 130
Screen Show 131
Screen To Front 132
Sgn 133
Shared 134
Show 134
Sid 135
Sid Volume 136
Sin 137
Sound 138
Sound Clr 139
Space$ 139
Sprite 140
Sprite Behind 141
Sprite Col 142
Sprite Flip 142
Sprite Front 143
Sprite Moving 144
Sprite Off 145
Sprite Palette 145
Sprite Path 146
State 147
Step 147
Stop 148
Str$ 148
String$ 149
T 150
Tcp Close 150
Tcp Connected 150
Tcp Open 151
Tcp Recv$ 152
Tcp Send 153
Td Angle 153
Td Camera 154
Td Center 156
Td Clip 157
Td Cls 158
Td Color 159
Td Cull 160
Td Focus 161
Td Kill 162
Td Load 163

Td Move 164
Td Move Rel 165
Td Moving 166
Td Object 167
Td Order 168
Td Path 169
Td Redraw 171
Td Size 172
Td X, Td Y, Td Z 173
Tempo 174
Text 174
Type 175
Then 176
Tile 177
Tile( ) 178
Tile Scroll 178
Tiles 179
Tiles Back 180
Tiles Front 181
Tiles Palette 182
Timer 183
To 184
Track Load 184
Track Loop 185
Track Play 186
Track Playing 187
Track Pos 187
Track Stop 188
True 189
U 189
Udp Close 190
Udp Ip$ 190
Udp Open 190
Udp Port 191
Udp Recv$ 192
Udp Send 192
Until 193
Upper$ 193
V 194
Val 194
Vbl 194
Vic20 Border 195
Vic20 Off 196
Vic20 On 197
Vic20 Ram 198

Vic20 Run 199
Volume 200
W 201
Wait 201
Wait Key 202
Wait Raster 202
Wait Vbl 203
Wend 204
While 204
Window 205
X 206
X Mouse 206
xor 206
Y 207
Y Mouse 207
Appendix A: Operators and Precedence 208
Appendix B: Values and Names 208
Appendix C: Screen Coordinates 209
Appendix D: Key Numbers 209
Appendix E: Run-Time Errors 210
Appendix F: The Copper 211
Appendix G: Fixed-Point Numbers 212
Appendix H: Floating-Point Numbers 213
Appendix I: SID, PCM and FM Music (Play) 213
Appendix J: Tracker Music and YM2151 Songs (Track) 216
Appendix K: 3D Objects (Td) 217
Appendix L: The 8-Bit CPUs, the VIC-20 and the C64 (Cpu8, Vic20, C64) 219
Appendix M: The Network (Net, Ping, Udp, Tcp) 221

About This Reference

VMOS is the programming language of VIXEN-32. Programs are written in the editor on the machine and compiled to native RISC-V code. This reference lists every command, function, operator and keyword of VMOS 0.24, A to Z, one entry each.

Each entry has these fields:

Field	Content
Syntax	the form; `[ ]` marks an optional part
Params	the parameters
Returns	the value a function returns

Field	Content
Action	what it does
Examples	complete programs to type in and run (F1)
See	related entries

Under each name: its kind and the VMOS version that added it.

Kind	Meaning
Command	a statement
Function	returns a number; used in an expression
String function	returns a string
Operator	binary (or unary) operator
Keyword part	a word used inside another statement

Keywords are not case sensitive. Two-word keywords (`X Mouse`, `Bob Off` ...) have exactly one space between the words.

The examples are complete programs. A program returns to the editor as soon as it ends, so the examples end with `Wait Key` to keep their screen until a key is pressed. Examples that load files use the files in the EXAMPLES folder (BALLS.BMP, SHIP.BMP, TILES.BMP, CRATE.BMP, BOXC.OBJ, BOXT.OBJ, SPHERE.OBJ, PLUCK.WAV, BOOM.WAV, MADNESS.MOD, and for the VIC-20 KERNAL.ROM, BASIC.ROM, CHAR.ROM, HELLO.PRG, GAME.CRT, VICSCR.OBJ, VICBOX.OBJ, VICCRATE.BMP; for the C64 HELLO64.PRG, C64SCR.OBJ, C64BOX.OBJ, with its KERNAL.ROM, BASIC.ROM and CHAR.ROM in a C64 folder there): save them there before running them. Number types: `A` is an integer, `A#` fixed-point (16.16), `A!` a float, `A$` a string. The program's screen shows pixels x 32 to 351, y 12 to 211 inside the default border (appendix C).

A

Abs

Function · VMOS 0.1

Syntax

```
Abs(n)
```

Returns

the absolute value of n; fixed-point in, fixed-point out.

Examples

Absolute values of integers and fixed-point numbers:

```
Print Abs(-5)              ' 5
Print Abs(12)              ' 12
Print Abs(-2.75)           ' 2.75
Wait Key
```

The distance between two positions, whichever is larger:

```
A=40
B=15
Print "Distance: "; Abs(A-B)     ' 25
Print "Distance: "; Abs(B-A)     ' 25
Wait Key
```

Check whether the mouse is within 10 pixels of a point:

```
Show
Repeat
    Locate 0, 0
    If Abs(X Mouse-192)<10 and Abs(Y Mouse-112)<10
        Print "near the middle"
    Else
        Print "               "
    End If
    Wait Vbl
Until Inkey$<>""
```

See

Sgn

Add

Command · VMOS 0.1

Syntax

```
Add var, n
```

Params

`var`	a numeric variable or array element
`n`	the amount to add (may be negative)

Action

var = var + n. Faster to type and to run than `var = var + n`.

Examples

Add to integer and fixed-point variables:

```
SCORE=100
Add SCORE, 50
X#=1.5
Add X#, 0.25
```

```
Print SCORE, X#            ' 150       1.75
Wait Key
```

Add to an array element; a negative amount subtracts:

```
Dim HP(3)
HP(2)=10
Add HP(2), -3
Print HP(2)                ' 7
Wait Key
```

Move a dot across the screen by a speed:

```
X=40
SPEED=3
Repeat
    Plot X, 100
    Add X, SPEED
    Wait Vbl
Until X>340
Wait Key
```

See

Inc, Dec

and

Operator · VMOS 0.1

Syntax

```
a and b
```

Returns

the bitwise AND of a and b. Comparisons return True (-1) or False (0), so with comparisons it works as a logical AND.

Examples

Two conditions that must both hold:

```
X=5
If X>0 and X<10 Then Print "X is between 1 and 9"
Wait Key
```

Bitwise AND, to keep only some bits:

```
Print 12 and 10            ' 8   (%1100 and %1010 = %1000)
Print 255 and $0F          ' 15  (the low 4 bits)
Wait Key
```

Test a bit of the joystick value (bit 4 = fire):

```
Repeat
    Locate 0, 0
    If Joy(2) and 16 Then Print "FIRE" Else Print "    "
```

```
    Wait Vbl
Until Inkey$<>""
```

See

or, xor, not

Asc

Function · VMOS 0.1

Syntax

```
Asc(s$)
```

Returns

the character code of the first character of s$; 0 for "".

Examples

Character codes:

```
Print Asc("A")              ' 65
Print Asc("abc")            ' 97, the code of "a"
Print Asc("")               ' 0
Wait Key
```

Print each character of a string with its code:

```
S$="VMOS"
For I=1 To Len(S$)
    Print Mid$(S$, I, 1); " = "; Asc(Mid$(S$, I, 1))
Next I
Wait Key
```

See

Chr$

As

Keyword part · VMOS 0.3

Syntax

```
Dim name(a) As TYPE
Dim name As TYPE
```

Examples

An array of records and a single record:

```
Type ENEMY
    X, Y, NAME$
End Type
Dim E(9) As ENEMY
Dim HERO As ENEMY
E(3).X = 100
HERO.NAME$ = "ANN"
```

```
Print E(3).X, HERO.NAME$
Wait Key
```

See

Dim, Type

B

Bar

Command · VMOS 0.3

Syntax

```
Bar x1, y1 To x2, y2
```

Params

`x1,y1`	one corner
`x2,y2`	the opposite corner (both corners are inclusive)

Action

Draws a filled rectangle in the Ink color, using the blitter. It draws into the current screen (both buffers with Double Buffer), in screen pixel coordinates (appendix C), clipped to the 512 x 256 screen. As with Print, anything drawn under a bob is lost when the bob moves.

Examples

A red rectangle:

```
Ink 2
Bar 40, 20 To 120, 80
Wait Key
```

A row of bars in the 16 C64 colors:

```
For C=0 To 15
    Ink C
    Bar 40+C*18, 60 To 55+C*18, 160
Next C
Wait Key
```

A bar graph from data:

```
For I=0 To 5
    Read H
    Ink I+2
    Bar 60+I*40, 200-H To 90+I*40, 200
Next I
Wait Key
Data 50, 120, 90, 170, 30, 140
```

See

Box, Ink, Draw

Beep

Command · VMOS 0.7

Syntax

```
Beep hz
```

Params

`hz`	frequency, 1 to 12288000 Hz; 0 stops it

Action

Plays the hardware square-wave beeper until Beep Off. A new Beep changes the frequency without a gap. It uses no SID or PCM voice and no CPU time.

Examples

A one-second tone:

```
Beep 440
Wait 50
Beep Off
```

A two-tone alarm until a key is pressed:

```
Repeat
    Beep 1000
    Wait 10
    Beep 750
    Wait 10
Until Inkey$<>""
Beep Off
```

A rising sweep:

```
For F=200 To 2000 Step 20
    Beep F
    Wait 1
Next F
Beep Off
```

See

Beep Off, Sound

Beep Off

Command · VMOS 0.7

Syntax

```
Beep Off
```

Action

Stops the beeper.

Examples

A short click:

```
Beep 2000
Wait 2
Beep Off
```

A warning beep that repeats three times:

```
For I=1 To 3
    Beep 880
    Wait 10
    Beep Off
    Wait 10
Next I
Wait Key
```

See

Beep

Bin$

String function · VMOS 0.1

Syntax

```
Bin$(n [, digits])
```

Returns

n in binary with a "%" prefix, zero-padded to at least digits digits.

Examples

Binary numbers, with and without padding:

```
Print Bin$(5)              ' %101
Print Bin$(5, 8)           ' %00000101
Print Bin$(255)            ' %11111111
Wait Key
```

Show the joystick bits as they change:

```
Repeat
    Locate 0, 0
    Print Bin$(Joy(2), 5)
    Wait Vbl
Until Inkey$<>""
```

See

Hex$, Val

Bob

Command · VMOS 0.3

Syntax

```
Bob n, x, y [, i [, w, h [, a]]]
```

Params

`n`	bob number 0-63 (0 to n-1 after Reserve Bob n)
`x,y`	top-left corner in screen coordinates (appendix C)
`i`	image number 1-1024 (Load Images, Get Image): 1-240 up to 64 x 64, 241-1024 (VMOS 0.22) up to 1024 x 1024; omitted: keeps the bob's current image, size and angle
`w,h`	(VMOS 0.22) size on screen in pixels, larger or smaller than the image; 0 hides the bob; omitted: the image's own size
`a`	(VMOS 0.23) angle, in radians or in degrees after Degree, as for Sin; positive turns clockwise on the screen; omitted: 0 (not turned)

Action

Sets bob n's position (and image). A bob is an image the blitter draws into the displayed screen, with color 0 transparent. All bobs are redrawn at every Wait Vbl: the saved background is restored first, then each bob is drawn at its current position, higher numbers over lower ones. Cls and screen scrolling remove the bobs first.

Big bobs (VMOS 0.22). Images 241-1024 are the bobs' own: any size up to 1024 x 1024, in chip RAM, made by Load Images or Get Image. Sprites cannot show them. With w, h the image is scaled to w x h as Paste Bob scales it (nearest-pixel); Bob Col uses the size on screen. The blitter scales from its 256 x 256 texture cache, so a bigger image is scaled in pieces of 256 x 256 – the same picture, a little slower. Each bob costs about three blitter copies of its size on screen per frame (keep the background, draw, put the background back), so a few big ones are fine, a screenful is not.

Turned bobs (VMOS 0.23). With a, the w x h box is turned by a about its middle, clockwise on the screen (y grows down), so a picture facing right points its nose down with a positive angle. The angle is in radians, or in degrees after Degree, as for Sin and Cos, and may be a whole, # or ! number. x, y stays the top-left corner of the box before it is turned, so changing only the angle never moves the bob. Every image 1-1024 can be turned: the blitter draws it as two textured triangles from the same texture cache (a big one in more), each screen pixel taking the image pixel under its center. Bob n, x, y keeps the image, size and angle; giving i without a sets the angle back to 0. What is kept under a turned bob, and Bob Col, use the box around its four corners – at 45 degrees a square's box is 1.41 times as wide and as tall. Angle 0 draws exactly as a bob without an angle.

Examples

A ball moving across the screen (run from EXAMPLES):

```
Load Images "BALLS.BMP", 1, 16, 16
For X=32 To 336
    Bob 0, X, 100, 1
    Wait Vbl
Next X
Wait Key
```

Four bobs bouncing, each with its own speed:

```
Load Images "BALLS.BMP", 1, 16, 16
Dim X(3), Y(3), DX(3), DY(3)
For I=0 To 3
    X(I)=40+I*60
    Y(I)=30+I*30
    DX(I)=1+I
    DY(I)=2
Next I
Repeat
    For I=0 To 3
        Add X(I), DX(I)
        Add Y(I), DY(I)
        If X(I)<32 or X(I)>335 Then DX(I)=-DX(I)
        If Y(I)<12 or Y(I)>195 Then DY(I)=-DY(I)
        Bob I, X(I), Y(I), I+1
    Next I
    Wait Vbl
Until Inkey$<>""
```

A bob over a drawn background; the background comes back as it moves:

```
Load Images "BALLS.BMP", 1, 16, 16
Ink 6
Bar 32, 90 To 351, 130
For X=32 To 336 Step 2
    Bob 0, X, 102, 2
    Wait Vbl
Next X
Wait Key
```

A ball growing from 16 x 16 to 128 x 128 (run from EXAMPLES):

```
Load Images "BALLS.BMP", 1, 16, 16
For S=16 To 128
    Bob 0, 192-S/2, 112-S/2, 1, S, S
    Wait Vbl
Next S
Wait Key
```

A big bob: a picture of 100 x 60 drawn, grabbed into image 241 and zoomed to 300 x 180:

```
Ink 2
Bar 32, 12 To 131, 71
Ink 7
Circle 81, 41, 25
Get Image 241, 32, 12 To 131, 71
Cls
For S=10 To 300 Step 2
    Bob 0, 192-S/2, 112-S*3/10, 241, S, S*6/10
    Wait Vbl
Next S
Wait Key
```

A ship turning round once (run from EXAMPLES):

```
Degree
Load Images "SHIP.BMP", 1
For A=0 To 360 Step 3
    Bob 0, 176, 100, 1, 32, 32, A
    Wait Vbl
Next A
Wait Key
```

A ship flying up and down, its nose up as it climbs and down as it falls:

```
Degree
Load Images "SHIP.BMP", 1
For F=0 To 720 Step 3
    Bob 0, 60, 100+Sin(F)*40, 1, 24, 24, 90+Cos(F)*35
    Wait Vbl
Next F
Wait Key
```

See

Bob Off, Bob Col, Reserve Bob, Double Buffer, Wait Vbl, Load Images, Paste Bob

Bob Col

Function · VMOS 0.3

Syntax

```
Bob Col(n)
```

Returns

True if bob n's image rectangle overlaps another bob's.

Examples

Two balls moving toward each other; the program stops when they touch:

```
Load Images "BALLS.BMP", 1, 16, 16
X=40
Repeat
    Bob 0, X, 100, 1
    Bob 1, 330-X, 100, 2
```

```
    Wait Vbl
    Inc X
Until Bob Col(0)
Print "hit at X = "; X
Wait Key
```

A ball you steer with the joystick; touching the other ball turns the border red:

```
Load Images "BALLS.BMP", 1, 16, 16
X=60
Y=100
Bob 1, 250, 100, 2
Repeat
    If Jleft(2) Then Add X, -2
    If Jright(2) Then Add X, 2
    If Jup(2) Then Add Y, -2
    If Jdown(2) Then Add Y, 2
    Bob 0, X, Y, 1
    Wait Vbl
    If Bob Col(0) Then Border 2 Else Border 0
Until Inkey$<>""
```

See

Bob, Sprite Col

Bob Moving

Function · VMOS 0.15

Syntax

```
Bob Moving(n)
```

Returns

True (-1) while bob n's path (Bob Path) is running; False (0) when it has ended, was stopped, or none was set.

Examples

Wait until a bob has finished its path:

```
Load Images "BALLS.BMP", 1, 16, 16
Bob 0, 40, 100, 1
Bob Path 0, "M 280,0,100"
Repeat
    Wait Vbl
Until Not Bob Moving(0)
Print "arrived"
Wait Key
```

Several bobs on paths of different lengths; count how many are still moving:

```
Load Images "BALLS.BMP", 1, 16, 16
For I=0 To 3
```

```
    Bob I, 40, 40+I*40, I+1
    Bob Path I, "M 280,0,"+Str$(50+I*50)
Next I
Repeat
    N=0
    For I=0 To 3
        If Bob Moving(I) Then Inc N
    Next I
    Home
    Print "moving: "; N
    Wait Vbl
Until N=0
Wait Key
```

See

Bob Path

Bob Off

Command · VMOS 0.3

Syntax

```
Bob Off [n]
```

Action

Removes bob n, or all bobs if n is omitted. They disappear at the next Wait Vbl.

Examples

Show two bobs, remove one, then the other:

```
Load Images "BALLS.BMP", 1, 16, 16
Bob 0, 100, 100, 1
Bob 1, 200, 100, 2
Wait Vbl
Wait 50
Bob Off 0
Wait Vbl
Wait 50
Bob Off
Wait Vbl
Wait Key
```

Shoot a bob away when fire is pressed:

```
Load Images "BALLS.BMP", 1, 16, 16
Bob 0, 184, 100, 3
Print "press fire to remove the bob"
Repeat
    Wait Vbl
Until Fire(2)
Bob Off 0
```

```
Wait Vbl
Wait Key
```

See

Bob

Bob Path

Command · VMOS 0.15

Syntax

```
Bob Path n, "steps"
Bob Path n, Play "file" [, scale [, times]]
Bob Path n Off
```

Params

`n`	the bob number (the bob must exist: use Bob first)
`steps`	a script as for Td Path, in x and y only: M x,y,n, W n, L
`file`	a recorded path (Path Record); its z movement is ignored
`scale`	scale factor for the recording (default 1; 0.5 = half size)
`times`	number of plays (default 1; 0 = loop forever)

Action

Moves bob n automatically, one step per frame, like Td Path does for 3D objects. Moves are relative to the bob's current position; Bob n, x, y can still reposition it. Play first moves the bob to where the recording started, so it follows the path where it was drawn. Sub-pixel fractions accumulate between frames. Off stops the path and leaves the bob where it is. The path advances at Wait Vbl, so the program loop must contain one.

Examples

A ball moving right and back, forever:

```
Load Images "BALLS.BMP", 1, 16, 16
Bob 0, 60, 100, 1
Bob Path 0, "M 250,0,100 M -250,0,100 L"
Repeat
    Wait Vbl
Until Inkey$<>""
```

A square path with pauses at the corners:

```
Load Images "BALLS.BMP", 1, 16, 16
Bob 0, 100, 50, 2
Bob Path 0, "M 150,0,50 W 10 M 0,100,50 W 10 M -150,0,50 W 10 M 0,-100,50 W
10 L"
Repeat
    Wait Vbl
Until Inkey$<>""
```

Play a recorded path at half size, twice (record SPIN.PTH with Path Record first):

```
Load Images "BALLS.BMP", 1, 16, 16
Bob 0, 192, 112, 3
Bob Path 0, Play "SPIN.PTH", 0.5, 2
Repeat
    Wait Vbl
Until Not Bob Moving(0)
Wait Key
```

See

Td Path, Bob Moving, Path Record

Border

Command · VMOS 0.3

Syntax

```
Border c
```

Params

`c`	color 0-255

Action

Sets the border color. The editor restores its own border color when the program ends.

Examples

A red border:

```
Border 2
Wait Key
```

Flash the border through all 16 C64 colors:

```
For C=0 To 15
    Border C
    Wait 10
Next C
Wait Key
```

See

Border Dims, Paper

Border Dims

Command · VMOS 0.3

Syntax

```
Border Dims l, r, t, b
```

Params

`l, r` left and right border width in pixels, 0-255

`t, b` top and bottom border height in pixels, 0-255

Action

Sets the border size on each side of the 384 x 224 display (default 32, 32, 12, 12). The visible area is then x = l to 383 - r. Nothing moves when it changes: sprites, bobs, tiles and the mouse keep the same coordinates (appendix C); the border just covers more or less of them. The mouse pointer can reach the whole visible area. The hardware keeps at least 1 border line at the top and bottom: 0, 0, 0, 0 gives 0, 0, 1, 1. The editor restores its own border when the program ends.

Examples

Use the whole display; text at the very edges:

```
Border Dims 0, 0, 0, 0
Text 0, 1, "TOP LEFT"
Text 320, 214, "BOTTOM"
Wait Key
```

A wide-screen letterbox: thick borders top and bottom:

```
Border Dims 32, 32, 40, 40
Border 0
Ink 6
Bar 0, 0 To 511, 255
Wait Key
```

See

Border, appendix C

Border Palette

Command · VMOS 0.18

Syntax

```
Border Palette n
```

Params

`n` a palette, 0-7 (default 0)

Action

The border shows its color from palette n, at once.

Examples

The border's red is blue in palette 1:

```
Palette 1
Color 2, 0, 0, 255
Border 2
```

```
Print "A key: Border Palette 1"
Wait Key
Border Palette 1
Wait Key
```

The screen and the border in palettes of their own: palette 1 dims only the screen:

```
For C=0 To 15
    V=Color(C)
    Palette 1
    Color C, (V/65536)/2, ((V/256) mod 256)/2, (V mod 256)/2
    Palette 0
Next C
Border 7
Ink 7
Bar 40, 40 To 340, 180
Screen Palette 1
Wait Key
```

See

Border, Palette

Box

Command · VMOS 0.3

Syntax

```
Box x1, y1 To x2, y2
```

Params

`x1,y1`	one corner
`x2,y2`	the opposite corner

Action

Draws a rectangle outline, 1 pixel wide, in the Ink color. It draws into the current screen (both buffers with Double Buffer), in screen pixel coordinates (appendix C), clipped to the 512 x 256 screen. As with Print, anything drawn under a bob is lost when the bob moves.

Examples

A yellow frame:

```
Ink 7
Box 40, 20 To 120, 80
Wait Key
```

Nested frames shrinking toward the middle:

```
For I=0 To 9
    Ink I+1
    Box 32+I*8, 12+I*8 To 351-I*8, 211-I*8
```

```
Next I
Wait Key
```

A filled panel with a border: Bar, then Box over it:

```
Ink 11
Bar 100, 60 To 280, 160
Ink 1
Box 100, 60 To 280, 160
Text 150, 106, "PANEL"
Wait Key
```

See

Bar, Ink

C

C64 Border

Command · VMOS 0.24

Syntax

```
C64 Border l, r, t, b
```

Params

`l, r`	pixels of the C64's border shown left and right of its screen, 0-64 (default 0)
`t, b`	pixels shown above and below it, together 0-56 (default 0)

Action

Sets how much of the C64's border the Td texture C64 (map_Kd C64) shows round its 320 x 200 screen, on each side, from the next Td Redraw on; it can change at any time. A side shows at most what the C64's picture has (64 pixels left and right); more is cut to that. Top and bottom together are at most 56 pixels, so the picture still fits the texture cache as its two halves ("Top + bottom is 56 pixels at most"), but they need not be equal: 18, 38 shows less above than below. A face keeps its texture coordinates 0..1, which then cover the screen and the border asked for; 0, 0, 0, 0 shows the screen alone. Vic20 Border does the same for the VIC-20.

Examples

A frame of border all round, the screen's shape kept: 384 x 240 on the 320 x 200 face (run from EXAMPLES, the ROMs in its C64 folder):

```
Cpu8 6502 Load "C64/KERNAL.ROM", $E000
Cpu8 6502 Load "C64/BASIC.ROM", $A000
Cpu8 6502 Load "C64/CHAR.ROM", $D000
C64 On
C64 Border 32, 32, 20, 20
```

```
Td Load "C64SCR.OBJ"
Td Object 1, "C64SCR", 192, 112, 0, 0, 0, 0
Repeat
    Td Redraw
    Wait Vbl
Until Inkey$=Chr$(27)
```

The border above and below grows and shrinks while it runs:

```
Cpu8 6502 Load "C64/KERNAL.ROM", $E000
Cpu8 6502 Load "C64/BASIC.ROM", $A000
Cpu8 6502 Load "C64/CHAR.ROM", $D000
C64 On
Td Load "C64SCR.OBJ"
Td Object 1, "C64SCR", 192, 112, 0, 0, 0, 0
Repeat
    For B=0 To 28
        C64 Border 0, 0, B, B
        Td Redraw
        Wait Vbl
    Next B
    For B=28 To 0 Step -1
        C64 Border 0, 0, B, B
        Td Redraw
        Wait Vbl
    Next B
Until Inkey$=Chr$(27)
```

See

C64 On, Vic20 Border, appendix L

C64 Off

Command · VMOS 0.24

Syntax

```
C64 Off
```

Action

Stops the C64: the 6502 back in reset, the VIC-II, the two CIAs and the PLA off, the SID the machine's again, the keyboard and the joystick the program's only. The texture C64 keeps the last picture. C64 On starts the C64 again from its reset. The program's end switches it off too.

Examples

Three seconds of the C64, then its last picture stays (run from EXAMPLES):

```
Cpu8 6502 Load "C64/KERNAL.ROM", $E000
Cpu8 6502 Load "C64/BASIC.ROM", $A000
Cpu8 6502 Load "C64/CHAR.ROM", $D000
C64 On
```

```
Td Load "C64SCR.OBJ"
Td Object 1, "C64SCR", 192, 112, 0, 0, 0, 0
For I=1 To 150
    Td Redraw
    Wait Vbl
Next I
C64 Off
Print "off"
Wait Key
```

Off and on again: the C64 restarts from its reset, with the ROMs it took the first time:

```
Cpu8 6502 Load "C64/KERNAL.ROM", $E000
Cpu8 6502 Load "C64/BASIC.ROM", $A000
Cpu8 6502 Load "C64/CHAR.ROM", $D000
Td Load "C64SCR.OBJ"
Td Object 1, "C64SCR", 192, 112, 0, 0, 0, 0
For K=1 To 3
    C64 On
    For I=1 To 100
        Td Redraw
        Wait Vbl
    Next I
    C64 Off
    Wait 25
Next K
Wait Key
```

See

C64 On, appendix L

C64 On

Command · VMOS 0.24

Syntax

```
C64 On
```

Action

Starts the C64 on the core's own chips (a core with the C64's: the 6502, the VIC-II, the two CIAs and the PLA; the SID is the machine's own). Load its ROMs first, as the VIC-20's (Cpu8 6502 Load): BASIC (8 KBytes) at $A000, the character ROM (4 KBytes) at $D000, the KERNAL (8 KBytes) at $E000. C64 On puts them into the C64's own ROM memory, where they stay for the next C64 On, and fills the 64 KB as after power-on, so files loaded before are gone: load a .prg after it. The 6502 runs from the KERNAL's reset vector at the C64's speed (985,249 Hz, on the VIC-II's clock), and with BASIC and the KERNAL loaded C64 On waits until BASIC shows READY. (at most 10 seconds). From then on the VIC-II's picture is the Td texture C64 (map_Kd C64 in a .mtl), new each Td Redraw; the SID is the C64's; palette entries 240-255 are its 16 colors (the machine's own C64 colors, as 0-15 start; until the program ends; pictures loaded after it set 16-239 only); and the keyboard and joystick

port 2 go to it too (the MEGA65's keyboard is the C64's key for key, cursor up and left with SHIFT, RESTORE is its RESTORE, RUN/STOP still stops the VMOS program). C64 On again restarts it. "The C64 did not reach READY." means a core without the C64's chips, or ROMs that are not a C64's.

Examples

The C64's READY. on a flat screen, pixel for pixel; type on it, ESC ends (run from EXAMPLES, the ROMs in its C64 folder):

```
Cpu8 6502 Load "C64/KERNAL.ROM", $E000
Cpu8 6502 Load "C64/BASIC.ROM", $A000
Cpu8 6502 Load "C64/CHAR.ROM", $D000
C64 On
Td Load "C64SCR.OBJ"
Td Object 1, "C64SCR", 192, 112, 0, 0, 0, 0
Repeat
    Td Redraw
    Wait Vbl
Until Inkey$=Chr$(27)
```

On a turning box of the screen's shape: C64BOX.MTL gives the front face map_Kd C64 and the other faces VICCRATE.BMP (CRATE.BMP on colors 16-19, as 240-255 are the C64's):

```
Degree
Cpu8 6502 Load "C64/KERNAL.ROM", $E000
Cpu8 6502 Load "C64/BASIC.ROM", $A000
Cpu8 6502 Load "C64/CHAR.ROM", $D000
C64 On
Td Load "C64BOX.OBJ"
Td Object 1, "C64BOX", 192, 112, 0, 0, 0, 0
Td Size 1, 240
Repeat
    Td Cls
    Td Angle 1, 20, A, 0
    Td Redraw
    Wait Vbl
    Add A, 1
Until Inkey$=Chr$(27)
```

See

C64 Off, C64 Run, C64 Border, Cpu8 6502 Load, Vic20 On, appendix L

C64 Run

Command · VMOS 0.24

Syntax

```
C64 Run
```

Action

Starts the last .prg loaded (Cpu8 6502 Load) as BASIC's program, as LOAD and RUN would: holds the 6502 for a moment, sets BASIC's end of program after it and types RUN. Use it after C64 On (which waits for READY and clears the 64 KB), with the .prg loaded after C64 On. The .prg must load where the C64's BASIC starts ($0801, its first 2 bytes 01 08): most C64 games are a BASIC line with a SYS and the game behind it, and start this way. Machine code saved elsewhere stops the program with "The .prg does not start where BASIC does".

Examples

HELLO64.PRG started on a flat screen; then type on it (run from EXAMPLES):

```
Cpu8 6502 Load "C64/KERNAL.ROM", $E000
Cpu8 6502 Load "C64/BASIC.ROM", $A000
Cpu8 6502 Load "C64/CHAR.ROM", $D000
C64 On
Cpu8 6502 Load "HELLO64.PRG"
C64 Run
Td Load "C64SCR.OBJ"
Td Object 1, "C64SCR", 192, 112, 0, 0, 0, 0
Repeat
    Td Redraw
    Wait Vbl
Until Inkey$=Chr$(27)
```

The same on a box swinging left and right:

```
Degree
Cpu8 6502 Load "C64/KERNAL.ROM", $E000
Cpu8 6502 Load "C64/BASIC.ROM", $A000
Cpu8 6502 Load "C64/CHAR.ROM", $D000
C64 On
Cpu8 6502 Load "HELLO64.PRG"
C64 Run
Td Load "C64BOX.OBJ"
Td Object 1, "C64BOX", 192, 112, 0, 0, 0, 0
Td Size 1, 260
Repeat
    Td Cls
    Td Angle 1, 15, Sin(A) * 40, 0
    Td Redraw
    Wait Vbl
    Add A, 1
Until Inkey$=Chr$(27)
```

See

C64 On, Cpu8 6502 Load, Vic20 Run, appendix L

Chr$

String function · VMOS 0.1

Syntax

```
Chr$(n)
```

Returns

a one-character string with character code n.

Examples

Characters from codes:

```
Print Chr$(65)              ' A
Print Chr$(65)+Chr$(66)     ' AB
Wait Key
```

The alphabet, built from codes 65 to 90:

```
A$=""
For C=65 To 90
    A$=A$+Chr$(C)
Next C
Print A$
Wait Key
```

A double quote inside a string (code 34):

```
Print "He said "+Chr$(34)+"hello"+Chr$(34)
Wait Key
```

See

Asc

Circle

Command · VMOS 0.3

Syntax

```
Circle x, y, r
```

Params

`x,y`	the center
`r`	radius in pixels, 0 or more (0 = a single pixel)

Action

Draws a circle outline in the Ink color. It draws into the current screen (both buffers with Double Buffer), in screen pixel coordinates (appendix C), clipped to the 512 x 256 screen. As with Print, anything drawn under a bob is lost when the bob moves.

Examples

One circle in the middle of the screen:

```
Ink 5
Circle 192, 112, 40
Wait Key
```

Rings growing out from the center:

```
For R=5 To 95 Step 10
    Ink R/10+1
    Circle 192, 112, R
Next R
Wait Key
```

Circles at random places and sizes:

```
For I=1 To 50
    Ink Rnd(14)+1
    Circle 40+Rnd(300), 20+Rnd(180), Rnd(30)
Next I
Wait Key
```

See

Ink, Plot

Cls

Command · VMOS 0.1

Syntax

```
Cls
```

Action

Fills the whole current screen (all 512 x 256 pixels, including the part under the border) with the Paper color and moves the cursor to the top-left of the text window. Bobs are removed first.

Examples

Clear to blue and print at the top:

```
Paper 6
Cls
Print "a blue screen"
Wait Key
```

Clear the screen between pages of text:

```
For P=1 To 3
    Cls
    Print "Page "; P
    Print "Press a key"
    Wait Key
Next P
```

See

Paper, Home

Color

Command · VMOS 0.3

Syntax

```
Color n, r, g, b
Color n, $RRGGBB
```

Params

`n`	palette entry 0-255
`r,g,b`	red, green, blue, 0-255 each
`$RRGGBB`	all three in one number, the format Color( ) returns

Action

Sets palette entry n in the palette Palette chose (default 0). Everything drawn in that color changes immediately: text, bobs, sprites, tiles, the border. The hardware stores 5 bits of red and blue and 6 bits of green (RGB565). The default palette: 0-15 the C64 colors, 16-31 a grey ramp, the rest black. Load Images and Load Tiles set entries 16-255 from the BMP's palette.

Examples

Define orange as color 20 and print with it:

```
Color 20, 255, 128, 0
Pen 20
Print "orange text"
Wait Key
```

The same with one $RRGGBB number:

```
Color 21, $00FF00
Ink 21
Bar 100, 60 To 280, 160
Wait Key
```

A red-to-yellow gradient in colors 32-63, drawn as bars:

```
For I=0 To 31
    Color 32+I, 255, I*8, 0
    Ink 32+I
    Bar 32+I*10, 60 To 41+I*10, 160
Next I
Wait Key
```

Fade a drawing out by changing its color, not redrawing it:

```
Color 40, 255, 255, 255
Ink 40
Circle 192, 112, 60
Bar 160, 100 To 224, 124
For L=255 To 0 Step -5
```

```
    Color 40, L, L, L
    Wait Vbl
Next L
Wait Key
```

See

Color( ), Color Back, Pen, Paper

Color( )

Function · VMOS 0.3

Syntax

```
Color(n)
```

Returns

palette entry n of the palette Palette chose (default 0) as $RRGGBB, as the hardware stores it (RGB565 expanded to 8 bits per channel).

Examples

Read back a C64 color:

```
Print Hex$(Color(2))            ' $8C3031, the C64 red
Wait Key
```

Set a color and read what the hardware kept (RGB565 drops the low bits):

```
Color 20, 100, 150, 200
Print Hex$(Color(20))           ' $6396CE
Wait Key
```

Copy one palette entry to another:

```
Color 21, Color(5)
Pen 21
Print "the same green as color 5"
Wait Key
```

See

Color

Color Back

Command · VMOS 0.3

Syntax

```
Color Back c
```

Params

`c`	color 0-255

Action

Sets the background color: shown wherever the program's screen, the tiles and the sprites are all transparent (color 0), and while the screen is hidden. Default: the Paper color.

Examples

Hide the screen and show only the background color:

```
Print "hello"
Color Back 6
Screen Hide
Wait Key
```

The background shows through Paper 0 (transparent):

```
Paper 0
Cls
Color Back 2
Print "red behind transparent paper"
Wait Key
```

See

Screen Hide, Paper, Color

Copper Back

Command · VMOS 0.4

Syntax

```
Copper Back y, c
```

Params

`y`	the raster line it takes effect on, 0-311, in sprite coordinates (0 = the top of the display, under the border)
`c`	color 0-255

Action

Sets the background color (Color Back) to c from line y down. Adds an entry to the program's copper list; the copper picks up the list at the next Wait Vbl or at Copper On (appendix F).

Examples

Ten background stripes (the paper must be transparent to see them):

```
Paper 0
Cls
For B=0 To 9
    Copper Back 40+B*17, B+1
Next B
Copper On
Wait Key
```

Sixteen background bands in the grey ramp (colors 16-31):

```
Paper 0
Cls
For Y=0 To 15
    Copper Back 12+Y*12, 16+Y
Next Y
Copper On
Print "grey bands behind the text"
Wait Key
```

Background bands that move down every frame:

```
Paper 0
Cls
Copper On
Repeat
    Copper Clear
    For B=0 To 9
        Copper Back ((B*20+P) mod 200)+12, B+2
    Next B
    Inc P
    Wait Vbl
Until Inkey$<>""
```

See

Color Back, Copper Color

Copper Border

Command · VMOS 0.4

Syntax

```
Copper Border y, c
```

Params

`y`	the raster line it takes effect on, 0-311, in sprite coordinates (0 = the top of the display, under the border)
`c`	color 0-255

Action

Sets the border color to c from line y down (raster bars). Adds an entry to the program's copper list; the copper picks up the list at the next Wait Vbl or at Copper On (appendix F).

Examples

A red band across the border:

```
Copper Border 60, 2
Copper Border 80, 0
```

```
Copper On
Wait Key
```

A rainbow border, one color every 14 lines:

```
For B=0 To 15
    Copper Border B*14, B
Next B
Copper On
Wait Key
```

A bar that moves down the border, rebuilt every frame:

```
Copper On
Repeat
    Copper Clear
    Copper Border Y, 7
    Copper Border Y+8, 0
    Y=(Y+2) mod 220
    Wait Vbl
Until Inkey$<>""
```

See

Border, Wait Raster

Copper Clear

Command · VMOS 0.4

Syntax

```
Copper Clear
```

Action

Empties the program's copper list. The copper keeps playing the previous list until the next Wait Vbl or Copper On, so a list rebuilt every frame (Copper Clear, the entries, Wait Vbl) is never shown half-built.

Examples

Rebuild the list every frame: a band of the screen sliding sideways:

```
For Y=0 To 199
    Ink Y/14+1
    Draw 32, 12+Y To 351, 12+Y
Next Y
Copper On
Repeat
    Copper Clear
    Copper Offset 100, P and 63
    Copper Offset 140, 0
    Inc P
    Wait Vbl
Until Inkey$<>""
```

Clear the list to remove all copper effects at once:

```
For B=0 To 15
    Copper Border B*14, B
Next B
Copper On
Print "a key clears the copper list"
Wait Key
Copper Clear
Wait Vbl
Wait Key
```

See

Copper On, appendix F

Copper Color

Command · VMOS 0.4

Syntax

```
Copper Color y, n, r, g, b
Copper Color y, n, $RRGGBB
```

Params

`y`	the raster line it takes effect on, 0-311, in sprite coordinates (0 = the top of the display, under the border)
`n`	palette entry 0-255
`r,g,b`	red, green, blue, 0-255 each (or one $RRGGBB)

Action

Sets palette entry n (in the palette Palette chose) to r,g,b from line y down: everything in color n changes from that line on. A gradient takes one entry per line. Adds an entry to the program's copper list; the copper picks up the list at the next Wait Vbl or at Copper On (appendix F).

Examples

A screen in one color that the copper turns into a gradient:

```
Paper 20
Cls
For Y=12 To 211
    Copper Color Y, 20, Y, 0, 255-Y
Next Y
Copper On
Wait Key
```

Text that changes color down the screen (color 1 is the default Pen):

```
For I=1 To 20
    Print "copper colored text, line "; I
```

```
Next I
For Y=12 To 211
    Copper Color Y, 1, 255, Y, 0
Next Y
Copper On
Wait Key
```

A gradient that scrolls, rebuilt every frame:

```
Paper 20
Cls
Copper On
Repeat
    Copper Clear
    For Y=12 To 211
        Copper Color Y, 20, (Y+P) and 255, 0, 128
    Next Y
    Add P, 2
    Wait Vbl
Until Inkey$<>""
```

See

Color, Copper Back

Copper Off

Command · VMOS 0.4

Syntax

```
Copper Off
```

Action

Stops playing the program's copper list (VMOS's own tile entries continue). From the next frame, everything the list changed is back to the values the program set.

Examples

Raster bars for two seconds, then back to normal:

```
For B=0 To 15
    Copper Border B*14, B
Next B
Copper On
Wait 100
Copper Off
Wait Key
```

Turn the copper effect on and off with a key:

```
For B=0 To 15
    Copper Border B*14, B
Next B
Print "o = on, f = off, q = quit"
```

```
Repeat
    K$=Inkey$
    If K$="o" Then Copper On
    If K$="f" Then Copper Off
Until K$="q"
```

See

Copper On

Copper Offset

Command · VMOS 0.4

Syntax

```
Copper Offset y, x [, sy]
```

Params

`y`	the raster line it takes effect on, 0-311, in sprite coordinates (0 = the top of the display, under the border)
`x`	horizontal screen offset from line y (as Screen Offset), 0-511
`sy`	vertical offset, 0-255; omitted: unchanged

Action

Scrolls the program's screen horizontally (and vertically) from line y down: waves, bands scrolling at different speeds. Adds an entry to the program's copper list; the copper picks up the list at the next Wait Vbl or at Copper On (appendix F).

Examples

A waving screen: every line gets its own horizontal offset from a sine table:

```
Degree
For I=1 To 24
    Print "VMOS COPPER WAVE  VMOS COPPER WAVE"
Next I
Dim W(63)
For I=0 To 63
    W(I)=8+Sin(I*360/64)*8
Next I
Copper On
Repeat
    Copper Clear
    For Y=12 To 211
        Copper Offset Y, W((Y+P) and 63)
    Next Y
    Inc P
    Wait Vbl
Until Inkey$<>""
```

Two bands that scroll at different speeds:

```
For I=1 To 24
    Print "<<< TOP BAND >>>  <<< BOTTOM BAND >>>"
Next I
Copper On
Repeat
    Copper Clear
    Copper Offset 12, P and 511
    Copper Offset 112, (P*3) and 511
    Inc P
    Wait Vbl
Until Inkey$<>""
```

See

Screen Offset, Copper Tile Scroll

Copper On

Command · VMOS 0.4

Syntax

```
Copper On
```

Action

Hands the program's list to the copper immediately; it plays from the next frame. After that, list changes are handed over at every Wait Vbl, during vertical blank, never mid-frame.

Examples

A red band in the border, started with Copper On:

```
Copper Border 100, 2
Copper Border 120, 0
Copper On
Wait Key
```

Build the list first, start it, then change it each frame:

```
Copper Border 100, 2
Copper On
Repeat
    Copper Clear
    Copper Border 100+P, 2
    Copper Border 120+P, 0
    P=(P+1) mod 80
    Wait Vbl
Until Inkey$<>""
```

See

Copper Off, Copper Clear, appendix F

Copper Palette

Command · VMOS 0.18

Syntax

```
Copper Palette y, n
```

Params

`y`	the raster line it takes effect on, 0-311, in sprite coordinates
`n`	a palette, 0-7

Action

The screen shows palette n from line y down: bands of one picture in different sets of colors. The tiles and the border keep the palettes Tiles Palette and Border Palette gave them. Adds an entry to the program's copper list; at the top of each frame the screen is back in its Screen Palette (appendix F).

Examples

One bar, the top half in palette 1 (red), the bottom half in palette 2 (blue):

```
Palette 1
Color 40, 255, 0, 0
Palette 2
Color 40, 0, 0, 255
Ink 40
Bar 40, 20 To 340, 200
Screen Palette 1
Copper Palette 112, 2
Copper On
Wait Key
```

Four bands of the same drawing, palettes 0-3:

```
For P=1 To 3
    Palette P
    For C=1 To 15
        V=Color(C)
       Color C, (V/65536)*(4-P)/4, ((V/256) mod 256)*(4-P)/4, (V mod 256)*(4-
P)/4
    Next C
Next P
For C=1 To 15
    Ink C
    Bar C*20, 12 To C*20+15, 211
Next C
Copper Palette 62, 1
Copper Palette 112, 2
Copper Palette 162, 3
```

```
Copper On
Wait Key
```

See

Screen Palette, Copper Color, Palette, appendix F

Copper Screen

Command · VMOS 0.4

Syntax

```
Copper Screen y, n
```

Params

`y`	the raster line it takes effect on, 0-311, in sprite coordinates (0 = the top of the display, under the border)
`n`	an open screen 0-7

Action

Displays screen n from line y down (a split screen); above y the display shows the front screen (Screen To Front). Adds an entry to the program's copper list; the copper picks up the list at the next Wait Vbl or at Copper On (appendix F).

Examples

A split screen: screen 0 at the top, screen 1 at the bottom:

```
Print "This is screen 0"
Screen Open 1
Paper 2
Cls
Print "This is screen 1"
Screen To Front 0
Copper Screen 112, 1
Copper On
Wait Key
```

A status panel at the bottom from its own screen, while the main screen is drawn on:

```
Screen Open 1
Paper 11
Cls
Text 40, 180, "STATUS PANEL - SCORE 001250"
Screen 0
Screen To Front 0
Copper Screen 170, 1
Copper On
Repeat
    Ink Rnd(15)+1
    Circle 32+Rnd(319), 12+Rnd(150), 10
```

```
    Wait Vbl
Until Inkey$<>""
```

See

Screen To Front, Screen Open

Copper Sprite

Command · VMOS 0.4

Syntax

```
Copper Sprite y, n, x, sy [, i]
```

Params

`y`	the raster line it takes effect on, 0-311, in sprite coordinates (0 = the top of the display, under the border)
`n`	sprite 0-14
`x,sy`	the sprite's top-left corner from line y on
`i`	the sprite's image from line y on; omitted: unchanged

Action

Repositions sprite n from line y down: the same sprite shown again further down (multiplexing: 15 hardware sprites, reused per line), or a new x on every line (a starfield). Above its first entry the sprite is where Sprite put it. Adds an entry to the program's copper list; the copper picks up the list at the next Wait Vbl or at Copper On (appendix F).

Examples

One sprite shown 8 times down the screen (run from EXAMPLES):

```
Load Images "BALLS.BMP", 1, 16, 16
Sprite 0, 100, 30, 1
For K=1 To 7
    Copper Sprite 30+K*22, 0, 100+K*25, 30+K*22
Next K
Copper On
Wait Key
```

A starfield from one sprite: a 1-pixel image, a new x on every 6th line:

```
Ink 1
Plot 0, 0
Get Image 1, 0, 0 To 0, 0
Cls
Sprite 0, 0, 12, 1
Copper On
Repeat
    Copper Clear
    For K=0 To 32
        Copper Sprite 12+K*6, 0, (F*(K mod 3+1)+K*37) mod 352+32, 12+K*6
```

```
    Next K
    Inc F
    Wait Vbl
Until Inkey$<>""
```

See

Sprite, appendix F

Copper Tile

Keyword part · VMOS 0.4

Syntax

```
Copper Tile Scroll y, x, ty
```

See

Copper Tile Scroll

Copper Tile Scroll

Command · VMOS 0.4

Syntax

```
Copper Tile Scroll y, x, ty
```

Params

`y`	the raster line it takes effect on, 0-311, in sprite coordinates (0 = the top of the display, under the border)
`x,ty`	the tile-map pixel at the display's left edge from line y on (as Tile Scroll)

Action

Scrolls the tile layer to a different position from line y down (parallax bands). Adds an entry to the program's copper list; the copper picks up the list at the next Wait Vbl or at Copper On (appendix F).

Examples

Five bands of tiles scrolling at five speeds (run from EXAMPLES):

```
Load Tiles "TILES.BMP", 1
For Y=0 To 31
    For X=0 To 63
        Tile X, Y, 1+(X+Y) mod 4
    Next X
Next Y
Tiles On
Copper On
Repeat
    Copper Clear
    For B=0 To 4
        Copper Tile Scroll 12+B*40, (P*(B+1)) and 511, 0
```

```
    Next B
    Inc P
    Wait Vbl
Until Inkey$<>""
```

Two halves of the tile layer scrolling in opposite directions:

```
Load Tiles "TILES.BMP", 1
For Y=0 To 31
    For X=0 To 63
        Tile X, Y, 1+(X+Y) mod 4
    Next X
Next Y
Tiles On
Copper On
Repeat
    Copper Clear
    Copper Tile Scroll 12, P and 511, 0
    Copper Tile Scroll 112, (512-P) and 511, 0
    Inc P
    Wait Vbl
Until Inkey$<>""
```

See

Tile Scroll, Copper Offset

Cos

Function · VMOS 0.9

Syntax

```
Cos(a)
```

Params

`a`	angle in radians, or in degrees after Degree

Returns

the cosine of a: a float (!) for a float angle, fixed-point (#) for a fixed-point or integer angle.

Examples

Cosine in degrees and as a float:

```
Degree
Print Cos(60)              ' 0.5
Print Cos(180!)            ' -1
Wait Key
```

A circle of dots with Sin and Cos:

```
Degree
For A=0 To 359 Step 6
    Plot 192+Cos(A)*80, 112+Sin(A)*80
```

```
Next A
Wait Key
```

A ball circling the middle of the screen (run from EXAMPLES):

```
Degree
Load Images "BALLS.BMP", 1, 16, 16
Repeat
    Bob 0, 184+Cos(A)*90, 104+Sin(A)*60, 1
    Add A, 3
    Wait Vbl
Until Inkey$<>""
```

See

Sin, Degree, Pi#

Cpu8 6502 Load, Cpu8 Z80 Load

Command · VMOS 0.16

Syntax

```
Cpu8 6502 Load "file" [, address]
Cpu8 Z80 Load "file" [, address]
```

Params

`file` a file in the program's folder

`address` where it goes in the 64 KB, $0000-$FFFF; may be left out for the files the Action names

Action

Copies a file into the 64 KB of the core's 8-bit CPUs (the 6502 and the Z80 share one 64 KB). Without an address: a .prg goes to the address in its first two bytes; a VICE cartridge file (it starts with "VIC20 CARTRIDGE") goes where its chips say; any other file that is 2 bytes over a whole KB (4098, 8194 ...) is a cartridge dump with its address in those 2 bytes; any other file needs the address. With an address, those 2 bytes are skipped, and a VICE cartridge file is refused (it holds its own addresses). A .crt or .rom file, and every VICE cartridge file, becomes read-only for the 8-bit CPU: a write there is lost, as on the real cartridge (copy protection that writes to its own ROM sees what it expects). The last .prg loaded is the one Vic20 Run starts. CTRL+B packs the file into the release.

Examples

The VIC-20's three ROMs, and the VIC-20 started on a flat screen (run from EXAMPLES; type on it, ESC ends):

```
Cpu8 6502 Load "KERNAL.ROM", $E000
Cpu8 6502 Load "BASIC.ROM", $C000
Cpu8 6502 Load "CHAR.ROM", $8000
Vic20 On
Td Load "VICSCR.OBJ"
```

```
Td Object 1, "VICSCR", 192, 112, 0, 0, 0, 0
Td Size 1, 184
Repeat
    Td Redraw
    Wait Vbl
Until Inkey$=Chr$(27)
```

A BASIC program: HELLO.PRG goes to $1001, where its first two bytes say, and Vic20 Run starts it:

```
Cpu8 6502 Load "KERNAL.ROM", $E000
Cpu8 6502 Load "BASIC.ROM", $C000
Cpu8 6502 Load "CHAR.ROM", $8000
Vic20 On
Cpu8 6502 Load "HELLO.PRG"
Vic20 Run
Td Load "VICSCR.OBJ"
Td Object 1, "VICSCR", 192, 112, 0, 0, 0, 0
Td Size 1, 184
Repeat
    Td Redraw
    Wait Vbl
Until Inkey$<>""
```

A cartridge: GAME.CRT (2 bytes of address, then 8 KB for $A000) is read-only and starts by itself; it writes to its own ROM and shows ROM PROTECTED:

```
Cpu8 6502 Load "KERNAL.ROM", $E000
Cpu8 6502 Load "BASIC.ROM", $C000
Cpu8 6502 Load "CHAR.ROM", $8000
Cpu8 6502 Load "GAME.CRT"
Vic20 On
Td Load "VICSCR.OBJ"
Td Object 1, "VICSCR", 192, 112, 0, 0, 0, 0
Td Size 1, 184
Repeat
    Td Redraw
    Wait Vbl
Until Inkey$<>""
```

A file with no address of its own, into the Z80's memory at $0000:

```
Cpu8 Z80 Load "CHAR.ROM", $0000
Print "CHAR.ROM is at $0000"
Wait Key
```

See

Cpu8 6502 Run, Vic20 On, Vic20 Run, appendix L

Cpu8 6502 Run, Cpu8 Z80 Run

Command · VMOS 0.17

Syntax

```
Cpu8 6502 Run [address]
Cpu8 Z80 Run [address]
```

Params

`address` where it starts, $0000-$FFFF; left out: the 6502 starts at its reset vector (the address in $FFFC-$FFFD), the Z80 at $0000

Action

Starts the 6502 or the Z80 on its own: no VIC-20 chips, at Cpu8 Speed (1 MHz unless set). It runs beside the VMOS program, at no cost to it, until Cpu8 Stop, the next Run or the end of the program. Every Run starts afresh, as after a reset. Its 64 KB are RAM, except where a .crt or .rom file was loaded (read-only for it). The program and the CPU share the 64 KB: Cpu8 Poke and Cpu8 Peek( ) read and write it while the CPU runs, and that is how the two talk. There is nothing else: the 6502 sees only memory, a Z80 IN reads $FF and an OUT does nothing. An error while the VIC-20 is on (Vic20 Off first).

Examples

A Z80 program adds 1 to 100, leaves the sum at $1000 and stops (HALT); it starts at $0000:

```
' LD HL,0 / LD B,100 / LD D,0
' loop: LD E,B / ADD HL,DE / DJNZ loop
' LD ($1000),HL / HALT
For I=0 To 14
    Read B
    Cpu8 Poke I, B
Next I
Cpu8 Z80 Run
Wait 1
Print "1+2+...+100 = "; Cpu8 Peek($1000)+256*Cpu8 Peek($1001)
Wait Key
Data $21,$00,$00,$06,$64,$16,$00,$58,$19,$10,$FC,$22,$00,$10,$76
```

A 6502 program at $0200, started there:

```
' CLD / LDX #10 / LDA #0
' loop: CLC / STX $10 / ADC $10 / DEX / BNE loop
' STA $1000 / JMP *
For I=0 To 18
    Read B
    Cpu8 Poke $0200+I, B
Next I
Cpu8 6502 Run $0200
Wait 1
Print "1+2+...+10 = "; Cpu8 Peek($1000)
Wait Key
Data $D8,$A2,$0A,$A9,$00,$18,$86,$10,$65,$10,$CA,$D0,$F8,$8D,$00,$10,$4C,
$10,$02
```

The same program started the 6502's own way, from its reset vector:

```
For I=0 To 18
    Read B
    Cpu8 Poke $0200+I, B
Next I
Cpu8 Poke $FFFC, $00
Cpu8 Poke $FFFD, $02
Cpu8 6502 Run
Wait 1
Print "1+2+...+10 = "; Cpu8 Peek($1000)
Wait Key
Data $D8,$A2,$0A,$A9,$00,$18,$86,$10,$65,$10,$CA,$D0,$F8,$8D,$00,$10,$4C,
$10,$02
```

The Z80 counts at $2000 without end while VMOS shows the count (a key ends):

```
' LD HL,$2000 / loop: INC (HL) / JR loop
For I=0 To 5
    Read B
    Cpu8 Poke I, B
Next I
Cpu8 Z80 Run
Repeat
    Locate 0, 0
    Print Cpu8 Peek($2000); "   "
    Wait Vbl
Until Inkey$<>""
Data $21,$00,$20,$34,$18,$FD
```

See

Cpu8 Stop, Cpu8 Speed, Cpu8 Poke, Cpu8 Peek( ), Cpu8 6502 Load, appendix L

Cpu8 Peek( )

Function · VMOS 0.17

Syntax

```
Cpu8 Peek(address)
```

Returns

the byte (0-255) at address ($0000-$FFFF) in the 64 KB of the 6502 and the Z80, as it is now, also while one of them runs.

Examples

Bytes put there with Cpu8 Poke, read back as text:

```
A$="VIXEN"
For I=1 To Len(A$)
    Cpu8 Poke $4000+I-1, Asc(Mid$(A$, I, 1))
Next I
For I=0 To 4
```

```
    Print Chr$(Cpu8 Peek($4000+I));
Next I
Print
Wait Key
```

A 16-bit number, low byte first as both CPUs keep it: the Z80 counts at $2000-$2001, VMOS shows it (a key ends):

```
' LD HL,0 / loop: INC HL / LD ($2000),HL / JR loop
For I=0 To 8
    Read B
    Cpu8 Poke I, B
Next I
Cpu8 Z80 Run
Repeat
    Locate 0, 0
    Print Cpu8 Peek($2000)+256*Cpu8 Peek($2001); "     "
    Wait Vbl
Until Inkey$<>""
Data $21,$00,$00,$23,$22,$00,$20,$18,$FA
```

The VIC-20's screen read from its memory: its first line, from screen codes (run from EXAMPLES):

```
Cpu8 6502 Load "KERNAL.ROM", $E000
Cpu8 6502 Load "BASIC.ROM", $C000
Cpu8 6502 Load "CHAR.ROM", $8000
Vic20 On
For I=0 To 21
    C=Cpu8 Peek($1E00+I)
    If C<32
        C=C+64
    End If
    Print Chr$(C);
Next I
Print
Wait Key
```

See

Cpu8 Poke, Cpu8 6502 Run, appendix L

Cpu8 Poke

Command · VMOS 0.17

Syntax

```
Cpu8 Poke address, byte
```

Params

address	$0000-$FFFF

`byte`	0-255

Action

Writes a byte into the 64 KB of the 6502 and the Z80, also while one of them runs: the next instruction that reads it sees it. It writes the read-only parts too (where a .crt or .rom was loaded). The way to give an 8-bit CPU its code and data, and to hand it a value while it runs.

Examples

A Z80 program doubles the number at $3000 into $3001, over and over; VMOS hands it the numbers:

```
' loop: LD A,($3000) / ADD A,A / LD ($3001),A / JR loop
For I=0 To 8
    Read B
    Cpu8 Poke I, B
Next I
Cpu8 Z80 Run
For N=1 To 10
    Cpu8 Poke $3000, N
    Wait 1
    Print N; " x 2 = "; Cpu8 Peek($3001)
Next N
Wait Key
Data $3A,$00,$30,$87,$32,$01,$30,$18,$F7
```

A 6502 program waits until VMOS writes a key's code into $0300, then writes the next letter to $0301 (ESC ends):

```
' CLD / loop: LDA $0300 / BEQ loop / CLC / ADC #1 / STA $0301
' LDA #0 / STA $0300 / JMP loop
For I=0 To 19
    Read B
    Cpu8 Poke $0200+I, B
Next I
Cpu8 6502 Run $0200
Print "Type a letter, ESC ends"
Repeat
    K$=Inkey$
    If K$<>"" and K$<>Chr$(27)
        Cpu8 Poke $0300, Asc(K$)
        Wait 1
        Print K$; " -> "; Chr$(Cpu8 Peek($0301))
    End If
    Wait Vbl
Until K$=Chr$(27)
Data $D8,$AD,$00,$03,$F0,$FB,$18,$69,$01,$8D,$01,$03,$A9,$00,$8D,$00,$03,
$4C,$01,$02
```

See

Cpu8 Peek( ), Cpu8 6502 Run, Cpu8 6502 Load

Cpu8 Speed

Command · VMOS 0.17

Syntax

```
Cpu8 Speed hz
```

Params

`hz`	the clock in Hz, 1 to 25000000 (default 1000000)

Action

The clock of the 6502 or Z80 started with Cpu8 6502 Run or Cpu8 Z80 Run, in Hz: 1000000 is 1 MHz, 3500000 a ZX Spectrum's Z80, 25000000 the most. It takes effect at once, also on a CPU that runs. Every program starts at 1 MHz. The VIC-20 (Vic20 On) does not use it: its 6502 runs at the VIC-20's own 1108405 Hz.

Examples

The same Z80 loop at three speeds: how often it goes round in one frame:

```
' LD HL,0 / loop: INC HL / LD ($2000),HL / JR loop (34 clocks)
For I=0 To 8
    Read B
    Cpu8 Poke I, B
Next I
Cpu8 Z80 Run
For S=1 To 3
    Read Hz
    Cpu8 Speed Hz
    Wait 1
    A=Cpu8 Peek($2000)+256*Cpu8 Peek($2001)
    Wait 1
    B=Cpu8 Peek($2000)+256*Cpu8 Peek($2001)
    Print Hz; " Hz: "; (B-A+65536) mod 65536; " a frame"
Next S
Wait Key
Data $21,$00,$00,$23,$22,$00,$20,$18,$FA
Data 1000000, 3500000, 25000000
```

Very slow: at 230 Hz this loop of 23 clocks counts 10 a second (a key ends):

```
' LD HL,$2000 / loop: INC (HL) / JR loop
For I=0 To 5
    Read B
    Cpu8 Poke I, B
Next I
Cpu8 Speed 230
Cpu8 Z80 Run
```

```
Repeat
    Locate 0, 0
    Print Cpu8 Peek($2000); "   "
    Wait Vbl
Until Inkey$<>""
Data $21,$00,$20,$34,$18,$FD
```

See

Cpu8 6502 Run, Vic20 On

Cpu8 Stop

Command · VMOS 0.17

Syntax

```
Cpu8 Stop
```

Action

Stops the 6502 or the Z80 between two instructions. It stays stopped, its 64 KB as it left them, until the next Cpu8 6502 Run or Cpu8 Z80 Run, which starts it afresh. With the VIC-20 on it stops the VIC-20's 6502 (Vic20 Off switches the VIC-20 off).

Examples

Stop a counting Z80 with a key; the count stays where it stopped:

```
' LD HL,$2000 / loop: INC (HL) / JR loop
For I=0 To 5
    Read B
    Cpu8 Poke I, B
Next I
Cpu8 Z80 Run
Print "The Z80 counts. Press a key to stop it."
Wait Key
Cpu8 Stop
Print "It stopped at "; Cpu8 Peek($2000)
Wait 50
Print "One second later: "; Cpu8 Peek($2000)
Wait Key
Data $21,$00,$20,$34,$18,$FD
```

Stop, change the program, and run it again: the Z80 sum, first to 10, then to 100:

```
' LD HL,0 / LD B,n / LD D,0
' loop: LD E,B / ADD HL,DE / DJNZ loop
' LD ($1000),HL / HALT
For I=0 To 14
    Read B
    Cpu8 Poke I, B
Next I
For N=10 To 100 Step 90
    Cpu8 Stop
```

```
    Cpu8 Poke $0004, N
    Cpu8 Z80 Run
    Wait 1
    Print "1 to "; N; " = "; Cpu8 Peek($1000)+256*Cpu8 Peek($1001)
Next N
Wait Key
Data $21,$00,$00,$06,$64,$16,$00,$58,$19,$10,$FC,$22,$00,$10,$76
```

See

Cpu8 6502 Run, Vic20 Off

Curs

Command · VMOS 0.1

Syntax

```
Curs On | Curs Off
```

Action

Shows or hides the text cursor while Input waits for a line. Default: On.

Examples

Input without a visible cursor:

```
Curs Off
Input "Your name"; N$
Print "Hello "; N$
Wait Key
```

Turn the cursor back on for the next question:

```
Curs Off
Input "Name (no cursor)"; N$
Curs On
Input "Age (cursor)"; A
Print N$; " is "; A
Wait Key
```

See

Input

D

Data

Command · VMOS 0.1

Syntax

```
Data value [, value ...]
```

Params

`value` integer or fixed-point constants (1.5) and "strings"

Action

Stores constants for Read. Data lines are never executed; they can be anywhere in the program.

Examples

Read numbers and strings from Data:

```
Read W, H, NAME$, SPEED#
Print W, H, NAME$, SPEED#
Wait Key
Data 320, 200, "screen", 1.5
```

A table read in a loop, ended by a marker value:

```
Do
    Read N$, AGE
    Exit If AGE<0
    Print N$; " is "; AGE
Loop
Wait Key
Data "Ann", 32, "Bob", 45, "Cid", 28, "", -1
```

See

Read, Restore

Dec

Command · VMOS 0.1

Syntax

```
Dec var
```

Action

var = var - 1.

Examples

Count down:

```
N=5
Repeat
    Print N
    Dec N
Until N=0
Print "lift off"
Wait Key
```

Lose a life on each key press:

```
LIVES=3
While LIVES>0
```

```
    Print "Lives: "; LIVES; " - press a key"
    Wait Key
    Dec LIVES
Wend
Print "Game over"
Wait Key
```

See

Inc, Add

Degree

Command · VMOS 0.9

Syntax

```
Degree
```

Action

Sin and Cos take angles in degrees (360 per turn) until Radian. Default: radians, as in AMOS.

Examples

The same angle in radians and in degrees:

```
Print Sin(Pi#/6)          ' 0.5 (radians, the default)
Degree
Print Sin(30)             ' 0.5
Wait Key
```

A clock hand turning in whole degrees:

```
Degree
For A=0 To 360 Step 6
    Cls
    Draw 192, 112 To 192+Sin(A)*80, 112-Cos(A)*80
    Wait 5
Next A
Wait Key
```

See

Radian, Sin

Dim

Command · VMOS 0.1

Syntax

```
Dim name(a [, b ...]) [, name(...) ...]
Dim name(a [, b ...]) As TYPE
Dim name As TYPE
```

Params

`a`	the highest index of the first dimension (indices start at 0)
`b ...`	the highest index of each further dimension; up to 255 dimensions
`TYPE`	a Type name: an array of records, or a single record without (a)

Action

Declares an array of a+1 (or (a+1) x (b+1) x ...) numbers, strings or Type records. An array must be declared with Dim before use; there is no implicit Dim. Large arrays are placed in chip RAM automatically; only free memory limits their size. Arrays of 1 or 2 dimensions are fastest; each further dimension costs one multiply, add and bounds check per access.

Examples

Arrays of integers, strings and fixed-point numbers:

```
Dim A(10), N$(5), V#(3)
A(10)=99
N$(0)="first"
V#(3)=2.5
Print A(10), N$(0), V#(3)
Wait Key
```

A 2D grid, filled and printed:

```
Dim GRID(9, 4)
For Y=0 To 4
    For X=0 To 9
        GRID(X, Y)=X*Y
    Next X
Next Y
For Y=0 To 4
    For X=0 To 9
        Print GRID(X, Y); " ";
    Next X
    Print
Next Y
Wait Key
```

An array of records:

```
Type ENEMY
    X, Y, HP
End Type
Dim E(20) As ENEMY
E(5).HP=100
Print E(5).HP
Wait Key
```

See

Type, appendix B

Do

Command · VMOS 0.1

Syntax

```
Do ... Loop
```

Action

Repeats the lines up to Loop forever. Leave it with Exit, Exit If, Goto, End or RUN/STOP.

Examples

A loop that ends when a key is pressed:

```
Do
    Print "press a key to stop"
    Wait 25
    Exit If Inkey$<>""
Loop
```

A main loop: move a dot with the joystick (port 2), fire ends it:

```
X=192
Y=112
Do
    J=Joy(2)
    If J and 1 Then Dec Y
    If J and 2 Then Inc Y
    If J and 4 Then Dec X
    If J and 8 Then Inc X
    Plot X, Y
    Exit If J and 16
    Wait Vbl
Loop
```

See

Loop, Exit, Repeat, While

Double Buffer

Command · VMOS 0.3

Syntax

```
Double Buffer
```

Action

Adds a second buffer to screen 0 so bobs never tear: at every Wait Vbl the bobs are drawn into the hidden buffer, which is then displayed. Print, Text, Cls, the drawing commands and text-window scrolling write into both buffers, so they stay steady. The tile map is double-buffered too: Tile changes go into the hidden map, which is displayed at the same Wait Vbl as Tile Scroll; the map is then 64 x 64. It stays on until the program ends; a second Double Buffer does nothing. Cost: 128 KB of chip RAM, and each bob's background is saved twice,

once per buffer. If the bobs take longer than a frame to draw, the display updates every second frame instead of tearing.

Examples

Many bobs without tearing:

```
Load Images "BALLS.BMP", 1, 16, 16
Reserve Bob 40
Double Buffer
Repeat
    For I=0 To 39
        Bob I, 40+I*7, 100+Sin(T#+I/4)*60, I mod 4+1
    Next I
    Add T#, 0.1
    Wait Vbl
Until Inkey$<>""
```

A tile map changed every frame without flicker:

```
Load Tiles "TILES.BMP", 1
Double Buffer
Tiles On
Repeat
    For X=0 To 63
        Tile X, 13, 1+(X+P) mod 4
    Next X
    Inc P
    Wait Vbl
Until Inkey$<>""
```

See

Bob, Wait Vbl, Reserve Bob

Draw

Command · VMOS 0.3

Syntax

```
Draw x1, y1 To x2, y2
Draw To x, y
```

Params

`x1,y1`	start point
`x2,y2`	end point (both end points are drawn)
`x,y`	Draw To: the end point; the line starts where the last Plot or Draw ended

Action

Draws a line in the Ink color, using the blitter. It draws into the current screen (both buffers with Double Buffer), in screen pixel coordinates (appendix C), clipped to the 512 x 256 screen. As with Print, anything drawn under a bob is lost when the bob moves.

Examples

A diagonal across the whole visible screen:

```
Draw 32, 12 To 351, 211
Wait Key
```

A triangle with Plot and Draw To:

```
Ink 7
Plot 100, 160
Draw To 192, 40
Draw To 284, 160
Draw To 100, 160
Wait Key
```

A fan of lines from one corner:

```
For X=32 To 351 Step 8
    Ink X/8 mod 15+1
    Draw 32, 211 To X, 12
Next X
Wait Key
```

A sine wave drawn as connected lines:

```
Degree
Plot 32, 112
For X=32 To 351
    Draw To X, 112-Sin(X*2)*60
Next X
Wait Key
```

See

Plot, Ink, Box, Polygon

E

Else

Keyword part · VMOS 0.1

Syntax

```
If c Then a Else b       (one line)
If c ... Else ... End If (block)
Else If c                (block)
```

Action

The branch that runs when the If condition is false.

Examples

One-line and block forms:

```
N=7
If N mod 2=0 Then Print "even" Else Print "odd"
If N>5
    Print "big"
Else
    Print "small"
End If
Wait Key
```

Else If chains:

```
Input "A number"; N
If N<0
    Print "negative"
Else If N=0
    Print "zero"
Else
    Print "positive"
End If
Wait Key
```

See

If

End

Command · VMOS 0.1

Syntax

```
End
```

Action

Ends the program and returns to the editor. It works anywhere, also inside a procedure.

Examples

End on a key:

```
Do
    K$=Inkey$
    If K$="q" Then End
    If K$<>"" Then Print "you typed "; K$; " - q quits"
Loop
```

End from inside a procedure:

```
For I=1 To 10
    CHECK[I]
Next I
```

```
Procedure CHECK[N]
    Print N
    If N=4
        Print "stopping at 4"
        Wait Key
        End
    End If
End Proc
```

See

Stop, End If, End Proc

End If

Keyword part · VMOS 0.1

Syntax

```
End If
```

Action

Closes a block If.

Examples

A block If:

```
X=12
If X>10
    Print "X is more than 10"
    Print "and this line too"
End If
Wait Key
```

See

If

End Proc

Command · VMOS 0.1

Syntax

```
End Proc [ [value] ]
```

Action

Closes a procedure definition. An optional value in square brackets is the procedure's result; the caller reads it with Param (integer), Param# (fixed-point), Param! (float) or Param$ (string).

Examples

A procedure that returns a number:

```
DOUBLE[21]
Print Param               ' 42
```

```
Wait Key
Procedure DOUBLE[N]
End Proc[N*2]
```

A procedure that returns a string:

```
GREET["Ann"]
Print Param$
Wait Key
Procedure GREET[N$]
End Proc["Hello, "+N$]
```

See

Procedure, Param, Pop Proc

End Type

Keyword part · VMOS 0.3

Syntax

```
End Type
```

Action

Closes a Type's field list.

Examples

A Type with two fields:

```
Type PAIR
    X, Y
End Type
Dim P As PAIR
P.X=10
P.Y=20
Print P.X, P.Y
Wait Key
```

See

Type

Envelope

Command · VMOS 0.7

Syntax

```
Envelope n [, a, d, s, r, wave, pulse]
```

Params

`n`	instrument 0-9 (Play's T0-T9)
`a`	attack, 0-15
`d`	decay, 0-15

`s`	sustain, 0-15
`r`	release, 0-15
`wave`	0 triangle, 1 sawtooth, 2 pulse, 3 noise, 4 ring modulation
`pulse`	pulse width, 0-4095 (wave 2 only)

Action

Defines a SID instrument for Play. An omitted (or empty: , ,) value keeps its current setting. Default instruments: BASIC 65's ten: 0 piano, 1 accordion, 2 calliope, 3 drum, 4 flute, 5 guitar, 6 harpsichord, 7 organ, 8 trumpet, 9 xylophone (appendix I).

Examples

Change instrument 9 and play a tune with it:

```
Envelope 9, 10, 5, 10, 5, 0, 300
Play "O5T9HCIDCDEHCG"
Wait 150
```

The same notes with a short pulse sound and a slow organ-like sound:

```
Envelope 0, 0, 4, 0, 2, 2, 1024
Play "O4T0QCEGO5C"
Wait 100
Envelope 0, 12, 0, 15, 10, 0
Play "O4T0QCEGO5C"
Wait 150
```

See

Play, appendix I

Every

Command · VMOS 0.13

Syntax

```
Every n Proc NAME
Every Off
Every On
```

Params

`n`	the interval in frames (1 = every frame, 50 per second)
`NAME`	a procedure without parameters

Action

Runs procedure NAME from the frame interrupt every n frames, whatever the main program is doing at that moment: a loop, a Wait, a Td Redraw, a Load Images or Track Load. Use it for things that must keep moving while the program loads or computes: a starfield,

raster bars, a status display. Every Off stops it, Every On restarts it. Only one Every runs at a time (a second Every n Proc replaces the first); it stops when the program ends.

Because it interrupts the main program, an Every procedure may only use what is safe there: integers and fixed-point (#) numbers, variables and arrays (Global or Shared ones are the main program's), If, For, Repeat, While, Do, the copper commands, Sprite and Color. Strings, floats (!), procedure calls, waits, drawing, Print, loading and sound are compile errors ("... is not allowed in an Every procedure"). Keep it short: the frame it runs in waits for it.

Its copper commands build a separate list (start it with Copper Clear). VMOS plays it together with the program's list (Copper On), merged by line; where both set the same register on the same line, the program's entry wins. The program's list is the one from its last Wait Vbl, so the Every list keeps playing while the program is loading.

A run-time error inside it stops the program and reports the line in the procedure.

Examples

A background color that keeps changing during Wait (palette entry 0 is the background):

```
Global F
Every 1 Proc TICK
Print "waiting 5 seconds - the background keeps changing"
Wait 250
Every Off
Procedure TICK
    F=F+1
    Color 0, (F*4) and 255, 0, 0
End Proc
```

A sprite starfield that moves while a song loads (run from EXAMPLES):

```
Global F
Ink 1
Plot 0, 0
Get Image 1, 0, 0 To 0, 0
Cls
Sprite 0, 0, 12, 1
Copper On
Every 1 Proc STARS
Track Load "MADNESS.MOD"
Track Play
Wait Key
Procedure STARS
    F=F+1
    Copper Clear
    For K=0 To 32
        Copper Sprite 12+K*6, 0, (F*(K mod 3+1)+K*37) mod 352+32, 12+K*6
    Next K
End Proc
```

See

Procedure, Copper Clear, appendix F

Exit

Command · VMOS 0.1

Syntax

```
Exit [n]
```

Action

Leaves the innermost loop (For, Repeat, While, Do); Exit n leaves n nested loops.

Examples

Leave a Do loop on a key:

```
Do
    Print "running"
    Wait 10
    If Inkey$<>"" Then Exit
Loop
Print "left the loop"
Wait Key
```

Leave two nested loops when a value is found:

```
For Y=1 To 10
    For X=1 To 10
        If X*Y=42 Then Exit 2
    Next X
Next Y
Print X; " x "; Y; " = 42"
Wait Key
```

See

Exit If

Exit If

Command · VMOS 0.1

Syntax

```
Exit If c [, n]
```

Action

Leaves the innermost loop (or n nested loops) if c is true.

Examples

Count until a limit:

```
Do
    Inc X
```

```
    Exit If X=10
Loop
Print X                     ' 10
Wait Key
```

Search a list of numbers:

```
Do
    Read N
    Exit If N=0
    Print N
Loop
Wait Key
Data 5, 8, 13, 21, 0
```

See

Exit

F

False

Function · VMOS 0.1

Syntax

```
False
```

Returns

0.

Examples

A flag that starts False:

```
DONE=False
Repeat
    Inc N
    If N=5 Then DONE=True
Until DONE
Print N                     ' 5
Wait Key
```

A procedure that returns True or False:

```
For N=1 To 10
    ISPRIME[N]
    If Param Then Print N; " is prime"
Next N
Wait Key
Procedure ISPRIME[N]
    If N<2 Then Pop Proc[False]
```

```
    For D=2 To N-1
        If N mod D=0 Then Pop Proc[False]
    Next D
End Proc[True]
```

See

True

Filter

Command · VMOS 0.7

Syntax

```
Filter [freq] [, lp, bp, hp, res]
```

Params

`freq`	cutoff frequency, 0-2047
`lp`	low-pass, 0 off or 1 on
`bp`	band-pass, 0 or 1
`hp`	high-pass, 0 or 1 (any combination of the three)
`res`	resonance, 0-15

Action

Sets the SID filter. An omitted value keeps its current setting. A Play voice goes through the filter when its string contains X1.

Examples

A filter sweep on a long note:

```
Play "T7X1O3P9WCL"
For F=50 To 1950 Step 50
    Filter F, 1, 0, 0, 15
    Wait 1
Next F
Play
```

The same note with low-pass, then high-pass:

```
Filter 300, 1, 0, 0, 8
Play "T7X1O3WC"
Wait 100
Filter 300, 0, 0, 1, 8
Play "T7X1O3WC"
Wait 100
```

See

Play, Sid

Fire

Function · VMOS 0.2

Syntax

```
Fire(2)
```

Returns

True while the fire button of the joystick in port 2 is held.

Examples

Wait for the fire button:

```
Print "press fire"
Repeat
    Wait Vbl
Until Fire(2)
Print "fire!"
Wait Key
```

Count fire presses (a press is counted once, not every frame it is held):

```
Repeat
    F=Fire(2)
    If F and Not LAST Then Inc SHOTS
    LAST=F
    Locate 0, 0
    Print "shots: "; SHOTS
    Wait Vbl
Until Inkey$<>""
```

See

Joy, Jup

Fm

Command · VMOS 0.21

Syntax

```
Fm r, v
```

Params

`r`	a YM2151 register, 0-255
`v`	0-255

Action

Writes a YM2151 register, as Sid r, v writes the SID's: the LFO ($18 frequency, $19 depths, $1B wave), the noise ($0F) and anything Fm Instrument, Fm Envelope and Fm Operator do not reach. Play's voices 8-15 write the registers of the voices they play. An error while a YM2151 song plays (Track Play).

Examples

A slow vibrato on every FM voice: the LFO at a slow rate, its pitch depth, and instrument 7's sensitivity to it:

```
Fm $18, 200
Fm $19, $80 + 40
Fm Instrument 7, , , 5
Play 8, "O4T7WCEG"
Wait 200
```

See

Fm Instrument, Sid, appendix I

Fm Envelope

Command · VMOS 0.21

Syntax

```
Fm Envelope t, op [, ar, dr, sr, sl, rr]
```

Params

`t`	the instrument, 0-9 (T0-T9 in a Play string on voices 8-15)
`op`	the operator, 1-4 (M1, C1, M2, C2)
`ar`	attack rate, 0-31 (31 the fastest)
`dr`	decay rate, 0-31: down to the sustain level
`sr`	sustain rate, 0-31: down from there while the key is held (0 = holds)
`sl`	sustain level, 0-15 (0 = no decay, 15 the quietest)
`rr`	release rate, 0-15 (15 the fastest)

Action

Sets an operator's envelope, in the YM2151's own values. A value left out (, ,) keeps the one it had. A voice playing the instrument takes it at its next note. Play alone puts the instruments back as they start (appendix I).

Examples

Instrument 1's two carriers die away slowly, like a bell:

```
Fm Envelope 1, 2, 31, 4, 2, 15, 3
Fm Envelope 1, 4, 31, 4, 2, 15, 3
Play 8, "O5T1HCEGO6C"
Wait 200
```

See

Fm Instrument, Fm Operator

Fm Instrument

Command · VMOS 0.21

Syntax

```
Fm Instrument t [, algorithm, feedback, pms, ams, slide]
```

Params

`t`	the instrument, 0-9 (T0-T9 in a Play string on voices 8-15)
`algorithm`	0-7: how the four operators connect (0 = 1 into 2 into 3 into 4; 7 = all four heard)
`feedback`	0-7: operator 1 into itself (0 = none)
`pms`	0-7: how much the LFO bends the pitch (Fm $19)
`ams`	0-3: how much the LFO changes the loudness
`slide`	−128 to 127 (0.21, b120): the pitch moves this many sixteenths of a semitone every frame of a note, from the frame after it starts − a drum falls (T3: −64)

Action

Sets an FM instrument's algorithm, feedback and LFO sensitivity, in the YM2151's own values; Fm Envelope and Fm Operator set its four operators. A value left out keeps the one it had. The instruments start as the ten in appendix I.

Examples

Instrument 0 (piano) with all four operators heard, then each one's level, a simple organ:

```
Fm Instrument 0, 7, 0
Fm Operator 0, 1, 20, 1
Fm Operator 0, 2, 26, 2
Fm Operator 0, 3, 32, 4
Fm Operator 0, 4, 38, 8
For E = 1 To 4
    Fm Envelope 0, E, 31, 0, 0, 0, 10
Next E
Play 8, "O4T0QCEGO5C", "O3T0HCG"
Wait 150
```

See

Fm Envelope, Fm Operator, Play, appendix I

Fm Operator

Command · VMOS 0.21

Syntax

```
Fm Operator t, op [, tl, mul, dt, dt2, ks, am]
```

Params

`t`	the instrument, 0-9
`op`	the operator, 1-4 (M1, C1, M2, C2)
`tl`	total level, 0-127 (0 the loudest, 127 off)
`mul`	frequency multiple, 0-15 (0 = half the note, 1 = the note, 2 = twice ...)
`dt`	detune, 0-7
`dt2`	coarse detune, 0-3
`ks`	key scaling, 0-3: higher notes have faster envelopes
`am`	0-1: the LFO changes its loudness (Fm Instrument's ams)

Action

Sets an operator's level and frequency, in the YM2151's own values. A value left out keeps the one it had. A voice playing the instrument takes it at its next note.

Examples

Instrument 2 brighter and brighter: its first modulator louder each time:

```
For L = 60 To 20 Step -10
    Fm Operator 2, 1, L
    Play 8, "O4T2HC"
    Wait 50
Next L
```

See

Fm Envelope, Fm Instrument

For

Command · VMOS 0.1

Syntax

```
For var = a To b [Step s] ... Next [var]
```

Action

Runs the lines up to Next with var = a, a+s, a+2s ... while var has not passed b. Step defaults to 1 and may be negative. A fixed-point (#) loop variable counts in fixed-point.

Examples

Counting up, and down in steps of 2:

```
For I=1 To 5
    Print I
Next I
For I=10 To 0 Step -2
    Print I
```

```
Next
Wait Key
```

A fixed-point loop:

```
For A#=0 To 1 Step 0.25
    Print A#
Next A#
Wait Key
```

Nested loops: a multiplication table:

```
For Y=1 To 9
    For X=1 To 9
        Print X*Y; Space$(4-Len(Str$(X*Y)));
    Next X
    Print
Next Y
Wait Key
```

See

Next, To, Step, Exit

G

Get Image

Command · VMOS 0.4

Syntax

```
Get Image i, x1, y1 To x2, y2
```

Params

`i`	image 1-1024
`x1,y1`	one corner of a rectangle on the current screen
`x2,y2`	the opposite corner (both inclusive; at most 64 x 64 for images 1-240, 1024 x 1024 for 241-1024)

Action

Copies a rectangle of the current screen into image i, for Sprite, Bob and Paste Bob, so images drawn with Plot, Draw, Bar etc. need no BMP. Color 0 stays transparent; any part outside the screen becomes color 0. Images 241-1024 (VMOS 0.22) are for bobs only (Bob, Paste Bob).

Examples

Draw an image, grab it and use it as a sprite:

```
Ink 7
Bar 0, 0 To 15, 15
Ink 2
Circle 7, 7, 5
Get Image 1, 0, 0 To 15, 15
Sprite 0, 180, 100, 1
Wait Key
```

Grab a drawn star and stamp copies of it as bobs:

```
Ink 7
Polygon 8, 0 To 10, 6 To 16, 6 To 11, 10 To 13, 16 To 8, 12 To 3, 16 To 5,
10 To 0, 6 To 6, 6
Get Image 1, 0, 0 To 16, 16
Cls
For I=0 To 9
    Paste Bob 40+I*30, 100, 1
Next I
Wait Key
```

See

Load Images, Sprite

Global

Command · VMOS 0.1

Syntax

```
Global var [, var ...]
```

Action

In the main program: makes these variables shared by every procedure.

Examples

A score kept by a procedure:

```
Global SCORE
ADDPOINTS[10]
ADDPOINTS[25]
Print SCORE                ' 35
Wait Key
Procedure ADDPOINTS[N]
    Add SCORE, N
End Proc
```

Global arrays and strings used by several procedures:

```
Global NAME$, HIGH
NAME$="nobody"
NEWSCORE["Ann", 120]
NEWSCORE["Bob", 340]
NEWSCORE["Cid", 200]
SHOWHIGH
```

```
Wait Key
Procedure NEWSCORE[N$, S]
    If S>HIGH
        HIGH=S
        NAME$=N$
    End If
End Proc
Procedure SHOWHIGH
    Print "high score: "; HIGH; " by "; NAME$
End Proc
```

See

Shared, Procedure

Gosub

Command · VMOS 0.1

Syntax

```
Gosub LABEL
```

Action

Jumps to the label; Return continues after the Gosub.

Examples

A subroutine called twice:

```
Gosub LINE
Print "between the lines"
Gosub LINE
Wait Key
End
LINE:
Print "------------------"
Return
```

A subroutine called in a loop to draw a star at X, Y:

```
For I=1 To 10
    X=40+Rnd(280)
    Y=20+Rnd(170)
    Gosub STAR
Next I
Wait Key
End
STAR:
Ink 7
Draw X-4, Y To X+4, Y
Draw X, Y-4 To X, Y+4
Return
```

See

Return, Goto, Procedure

Goto

Command · VMOS 0.1

Syntax

```
Goto LABEL
```

Action

Jumps to the label (a name followed by a colon at the start of a line).

Examples

A loop made with Goto:

```
N=1
AGAIN:
Print N
Inc N
If N<=5 Then Goto AGAIN
Wait Key
```

Jump past a part of the program:

```
Input "Skip the greeting (y/n)"; A$
If A$="y" Then Goto MAIN
Print "Hello!"
MAIN:
Print "main part"
Wait Key
```

See

Gosub

H

Hex$

String function · VMOS 0.1

Syntax

```
Hex$(n [, digits])
```

Returns

n in hexadecimal with a "$" prefix, zero-padded to at least digits digits.

Examples

Hex numbers, with and without padding:

```
Print Hex$(255)            ' $FF
Print Hex$(10, 4)          ' $000A
```

```
Print Hex$(65535)          ' $FFFF
Wait Key
```

A table of numbers in decimal and hex:

```
For N=0 To 255 Step 32
    Print N, Hex$(N, 2)
Next N
Wait Key
```

See

Bin$, Val

Hide

Command · VMOS 0.2

Syntax

```
Hide
```

Action

Hides the mouse pointer. Default: hidden.

Examples

Show the pointer, then hide it again:

```
Show
Print "move the mouse, then press a key"
Wait Key
Hide
Print "the pointer is hidden"
Wait Key
```

Hide the pointer while the mouse is not moving, show it when it moves:

```
OX=X Mouse
OY=Y Mouse
Repeat
    If X Mouse<>OX or Y Mouse<>OY
        Show
        T=Timer
    End If
    If Timer-T>100 Then Hide
    OX=X Mouse
    OY=Y Mouse
    Wait Vbl
Until Inkey$<>""
```

See

Show, X Mouse

Home

Command · VMOS 0.1

Syntax

```
Home
```

Action

Moves the text cursor to the top-left of the text window without clearing the screen.

Examples

Overwrite the top line every frame (a counter):

```
Repeat
    Home
    Print "frames: "; Timer
    Wait Vbl
Until Inkey$<>""
```

A status line at the top that is updated in place, while text is printed below it:

```
Locate 0, 2
For I=1 To 20
    Print "working on item "; I
    Home
    Print "progress: "; I*5; "%   "
    Locate 0, 2+I
    Wait 10
Next I
Wait Key
```

See

Locate, Cls, Window

I

If

Command · VMOS 0.1

Syntax

```
If c Then commands [Else commands]      (one line)
If c                                    (block)
   ...
Else If c
   ...
Else
   ...
End If
```

Action

Runs the commands if c is true (non-zero).

Examples

One-line If with Else:

```
X=8
If X>5 Then Print "big" Else Print "small"
Wait Key
```

A block If with Else If:

```
Input "Your age"; AGE
If AGE<13
    Print "child"
Else If AGE<20
    Print "teenager"
Else
    Print "adult"
End If
Wait Key
```

Conditions with and / or:

```
Input "A number from 1 to 10"; N
If N<1 or N>10 Then Print "out of range" Else Print "ok"
Wait Key
```

See

Then, Else, End If

Inc

Command · VMOS 0.1

Syntax

```
Inc var
```

Action

var = var + 1 (1.0 for a fixed-point variable).

Examples

Increment integers, array elements and fixed-point variables:

```
Dim A(3)
Inc SCORE
Inc A(3)
X#=0.5
Inc X#
Print SCORE, A(3), X#      ' 1         1         1.5
Wait Key
```

Count key presses:

```
Repeat
    K$=Inkey$
    If K$<>"" Then Inc N
    Home
    Print "keys pressed: "; N; " (q quits)"
Until K$="q"
```

See

Dec, Add

Ink

Command · VMOS 0.3

Syntax

```
Ink c
```

Params

`c`	color 0-255 (default 1, white)

Action

Sets the color used by Plot, Draw, Box, Bar, Circle, Polygon and Path Record's trail. Print uses Pen instead.

Examples

Draw in red, then in yellow:

```
Ink 2
Bar 40, 40 To 120, 120
Ink 7
Circle 200, 80, 40
Wait Key
```

Lines in all 16 C64 colors:

```
For C=0 To 15
    Ink C
    Draw 40, 20+C*12 To 340, 20+C*12
Next C
Wait Key
```

See

Draw, Plot, Pen

Inkey$

String function · VMOS 0.1

Syntax

```
Inkey$
```

Returns

the next key from the keyboard buffer, or "" if none is waiting. It does not wait.

Examples

Print every key typed until q:

```
Print "type keys, q quits"
Repeat
    K$=Inkey$
    If K$<>"" Then Print K$;
Until K$="q"
```

A loop that keeps running and checks keys without stopping:

```
Repeat
    Inc N
    Home
    Print N
    K$=Inkey$
    Wait Vbl
Until K$<>""
Print "you pressed "; K$
Wait Key
```

Move a dot with the cursor keys (Scancode gives the key number):

```
X=192
Y=112
Repeat
    K$=Inkey$
    If K$<>""
        If Scancode=74 Then Add X, -4
        If Scancode=2 Then Add X, 4
        If Scancode=73 Then Add Y, -4
        If Scancode=7 Then Add Y, 4
    End If
    Plot X, Y
    Wait Vbl
Until K$="q"
```

See

Wait Key, Scancode, Key State

Input

Command · VMOS 0.1

Syntax

```
Input ["prompt";] var [, var ...]
```

Action

Reads a line of text, ended by RETURN. Comma-separated parts go into the variables in order. Without a prompt it shows "? ". A fixed-point (#) variable accepts decimals (-2.5).

Examples

Ask for a name:

```
Input "Your name"; NAME$
Print "Hello, "; NAME$
Wait Key
```

Two numbers on one line, separated by a comma:

```
Input "Two numbers (a,b)"; A, B
Print A; " + "; B; " = "; A+B
Wait Key
```

A decimal number into a fixed-point variable:

```
Input "Speed"; SPEED#
Print "half of it: "; SPEED#/2
Wait Key
```

See

Curs, Inkey$

Instr

Function · VMOS 0.1

Syntax

```
Instr(a$, b$ [, start])
```

Returns

the position (from 1) of the first occurrence of b$ in a$, searching from position start (default 1); 0 if not found.

Examples

Find a substring:

```
Print Instr("HELLO", "LL")       ' 3
Print Instr("HELLO", "X")        ' 0
Print Instr("ABCABC", "B", 3)    ' 5
Wait Key
```

Split a string at a comma:

```
S$="Ann,Smith"
P=Instr(S$, ",")
Print "first: "; Left$(S$, P-1)
Print "last: "; Mid$(S$, P+1, Len(S$)-P)
Wait Key
```

Count the spaces in a sentence:

```
S$="the quick brown fox"
P=Instr(S$, " ")
While P>0
```

```
    Inc N
    P=Instr(S$, " ", P+1)
Wend
Print N; " spaces"
Wait Key
```

See

Mid$

Instrument

Command · VMOS 0.9

Syntax

```
Instrument t, s [, note$]
```

Params

`t`	instrument 0-9 (T0-T9 in a Play string)
`s`	sample 1-64 (Load Sample)
`note$`	the pitch the sample was recorded at: "A4", "#F3", "$B2" (sharp or flat sign before the letter, as in Play; octave 0-7 after it). Default "C4"

Action

Makes sample s instrument Tt for Play's PCM voices (Play strings 4-7); on the SID voices Tt stays an Envelope instrument. A note plays the sample faster or slower to reach its pitch: one octave above note$ plays it at twice its own rate. Default: T0-T9 are samples 1-10, each taken as C4. Play with no arguments keeps these settings.

Examples

A plucked A4 sample as instrument 0, playing a scale (run from EXAMPLES):

```
Load Sample "PLUCK.WAV", 1
Instrument 0, 1, "A4"
Play , , , "O4T0QCDEFGABO5C"
Wait 150
```

A chord on four PCM voices:

```
Load Sample "PLUCK.WAV", 1
Instrument 0, 1, "A4"
Play , , , "O4T0WC", "O4T0WE", "O4T0WG", "O5T0WC"
Wait 150
```

See

Play, Load Sample, appendix I

J

Jdown

Function · VMOS 0.2

Syntax

```
Jdown(2)
```

Returns

True while the joystick in port 2 is pushed down.

Examples

Show the direction held:

```
Repeat
    Home
    If Jdown(2) Then Print "down" Else Print "    "
    Wait Vbl
Until Inkey$<>""
```

Move a bar down while the joystick is held down, up otherwise:

```
Y=60
Repeat
    If Jdown(2) Then Y=Min(Y+2, 190) Else Y=Max(Y-1, 20)
    Cls
    Bar 180, Y To 204, Y+10
    Wait Vbl
Until Inkey$<>""
```

See

Joy, Jup, Jleft, Jright, Fire

Jleft

Function · VMOS 0.2

Syntax

```
Jleft(2)
```

Returns

True while the joystick in port 2 is pushed left.

Examples

Move a ball left and right with the joystick (run from EXAMPLES):

```
Load Images "BALLS.BMP", 1, 16, 16
X=184
Repeat
    If Jleft(2) Then Add X, -2
    If Jright(2) Then Add X, 2
    Bob 0, X, 180, 1
    Wait Vbl
Until Inkey$<>""
```

Count left presses (counted once per press):

```
Repeat
    L=Jleft(2)
    If L and Not LAST Then Inc N
    LAST=L
    Home
    Print "left presses: "; N
    Wait Vbl
Until Inkey$<>""
```

See

Joy

Joy

Function · VMOS 0.2

Syntax

```
Joy(port)
```

Returns

the joystick state in port 2 as a bit mask: 1 up, 2 down, 4 left, 8 right, 16 fire. Joy(1) is always 0: port 1 has the mouse.

Examples

Print the joystick value as it changes:

```
Repeat
    Home
    Print "Joy(2) = "; Joy(2); "   "
    Wait Vbl
Until Inkey$<>""
```

Move a sprite in 8 directions, fire to stop (run from EXAMPLES):

```
Load Images "SHIP.BMP", 1
X=184
Y=104
Repeat
    J=Joy(2)
    If J and 1 Then Add Y, -2
    If J and 2 Then Add Y, 2
    If J and 4 Then Add X, -2
    If J and 8 Then Add X, 2
    Sprite 0, X, Y, 1
    Wait Vbl
Until J and 16
```

See

Jup, Jdown, Jleft, Jright, Fire

Jright

Function · VMOS 0.2

Syntax

```
Jright(2)
```

Returns

True while the joystick in port 2 is pushed right.

Examples

Count how many frames right is held:

```
Repeat
    If Jright(2) Then Inc N
    Home
    Print "frames held right: "; N
    Wait Vbl
Until Inkey$<>""
```

A ship that only moves while right is held (run from EXAMPLES):

```
Load Images "SHIP.BMP", 1
X=40
Repeat
    If Jright(2) and X<320 Then Add X, 3
    Sprite 0, X, 104, 1
    Wait Vbl
Until Inkey$<>""
```

See

Joy

Jup

Function · VMOS 0.2

Syntax

```
Jup(2)
```

Returns

True while the joystick in port 2 is pushed up.

Examples

Jump when up is pushed: a dot that rises and falls back:

```
Y#=180
Repeat
    If Jup(2) and Y#>=180 Then VY#=-6
    Add VY#, 0.3
    Add Y#, VY#
    If Y#>180
        Y#=180
```

```
        VY#=0
    End If
    Cls
    Bar 188, Y# To 196, Y#+8
    Wait Vbl
Until Inkey$<>""
```

Show all four directions at once:

```
Repeat
    Home
    U$="  "
    D$="  "
    L$="  "
    R$="  "
    If Jup(2) Then U$="UP"
    If Jdown(2) Then D$="DN"
    If Jleft(2) Then L$="LT"
    If Jright(2) Then R$="RT"
    Print U$; " "; D$; " "; L$; " "; R$
    Wait Vbl
Until Inkey$<>""
```

See

Joy

K

Key

Keyword part · VMOS 0.1

Syntax

```
Key State(k)     Wait Key
```

See

Key State, Wait Key

Key State

Function · VMOS 0.1

Syntax

```
Key State(k)
```

Returns

True while key number k (appendix D) is held down.

Examples

Show whether SPACE is held (key 60):

```
Repeat
    Home
    If Key State(60) Then Print "SPACE down" Else Print "          "
    Wait Vbl
Until Key State(71)
```

Move a dot with the cursor keys while they are held (several at once):

```
X=192
Y=112
Repeat
    If Key State(74) Then Dec X
    If Key State(2) Then Inc X
    If Key State(73) Then Dec Y
    If Key State(7) Then Inc Y
    Plot X, Y
    Wait Vbl
Until Key State(71)
```

See

Scancode, Inkey$

L

Left$

String function · VMOS 0.1

Syntax

```
Left$(s$, n)
```

Returns

the first n characters of s$.

Examples

The start of a string:

```
Print Left$("HELLO", 2)          ' HE
Print Left$("HELLO", 10)         ' HELLO
Wait Key
```

A word printed one more letter each line:

```
S$="VIXEN-32"
For I=1 To Len(S$)
    Print Left$(S$, I)
Next I
Wait Key
```

See

Right$, Mid$

Len

Function · VMOS 0.1

Syntax

`Len(s$)`

Returns

the length of s$ in characters (0-32767).

Examples

String lengths:

```
Print Len("abc")                ' 3
Print Len("")                   ' 0
Wait Key
```

Center a text on a 40-column line:

```
T$="CENTERED"
Locate (40-Len(T$))/2, 10
Print T$
Wait Key
```

See

Left$, Mid$

Let

Command · VMOS 0.1

Syntax

`[Let] var = value`

Action

Assigns a value to a variable. The word Let is optional.

Examples

With and without Let:

```
Dim A(5)
Let X = 5
NAME$ = "Ann"
A(3) = X*2
Print X, NAME$, A(3)
Wait Key
```

Assignments with expressions and strings:

```
Let A = 10
Let B = A*A + 5
Let N$ = "result: " + Str$(B)
```

```
Print N$
Wait Key
```

Load Images

Command · VMOS 0.2

Syntax

```
Load Images "file", first [, w, h]
```

Params

`file`	an 8-bit (256-color) BMP in the program's folder
`first`	the first image number, 1-1024
`w, h`	the size of each image: up to 64 x 64 for images 1-240, up to 1024 x 1024 for 241-1024; omitted: the whole picture is one image

Action

Cuts the picture into images of w x h, left to right, top to bottom, numbered from first. Color 0 is transparent. The BMP's palette entries 16-255 are copied to the palette (0-15 stay the C64 colors). Sprites and bobs use images 1-240; images 241-1024 (VMOS 0.22) are the bobs' own, any size up to 1024 x 1024, kept in chip RAM (Bob, Paste Bob).

Examples

One picture as one image, shown as a sprite (run from EXAMPLES):

```
Load Images "SHIP.BMP", 1
Sprite 0, 180, 100, 1
Wait Key
```

A strip of four 16 x 16 balls cut into images 1-4:

```
Load Images "BALLS.BMP", 1, 16, 16
For I=1 To 4
    Paste Bob 100+I*30, 100, I
Next I
Wait Key
```

A whole picture as one image of the bobs' own, 241, drawn three times its size (run from EXAMPLES):

```
Load Images "SHIP.BMP", 241
Paste Bob 100, 60, 241, 48, 48
Wait Key
```

See

Sprite, Bob, Load Tiles

Load Sample

Command · VMOS 0.2

Syntax

```
Load Sample "file", s
```

Params

`file`	a WAV file: 8 or 16 bits, mono or stereo, any sample rate
`s`	sample number 1-64

Action

Loads the sound into chip RAM (stereo is mixed down to mono).

Examples

Load and play a sample (run from EXAMPLES):

```
Load Sample "BOOM.WAV", 1
Sam Play 1
Wait 100
```

Two samples on keys: b for boom, p for pluck:

```
Load Sample "BOOM.WAV", 1
Load Sample "PLUCK.WAV", 2
Print "b = boom, p = pluck, q = quit"
Repeat
    K$=Inkey$
    If K$="b" Then Sam Play 1
    If K$="p" Then Sam Play 2
Until K$="q"
```

See

Sam Play

Load Tiles

Command · VMOS 0.3

Syntax

```
Load Tiles "file", first
```

Params

`file`	an 8-bit BMP in the program's folder
`first`	the first tile number, 1-1023

Action

Cuts the picture into 8 x 8 tiles, left to right, top to bottom, numbered from first. Tile 0 is always empty; color 0 in a tile is transparent. The BMP's palette entries 16-255 are copied to the palette.

Examples

Load the four tiles of TILES.BMP and fill the screen with them (run from EXAMPLES):

```
Load Tiles "TILES.BMP", 1
For Y=0 To 31
    For X=0 To 63
        Tile X, Y, Rnd(3)+1
    Next X
Next Y
Tiles On
Wait Key
```

Load the tiles at number 10 instead of 1:

```
Load Tiles "TILES.BMP", 10
For X=0 To 63
    Tile X, 12, 10+X mod 4
Next X
Tiles On
Wait Key
```

See

Tile, Tiles

Locate

Command · VMOS 0.1

Syntax

```
Locate x, y
```

Params

`x`	column in the text window, from 0 (0-39 with the default window)
`y`	row in the text window, from 0 (0-24 with the default window)

Action

Moves the text cursor to column x, row y of the text window (Window). Values beyond the window are clamped to its edge.

Examples

Print in the middle of the screen:

```
Locate 16, 12
Print "Middle"
Wait Key
```

A diagonal of stars:

```
For I=0 To 24
    Locate I, I
    Print "*"
```

```
Next I
Wait Key
```

A score that updates in place in the top-right corner:

```
For S=0 To 500 Step 10
    Locate 30, 0
    Print "SCORE "; S
    Wait 2
Next S
Wait Key
```

See

Home, Print, Window

Loop

Keyword part · VMOS 0.1

Syntax

```
Do ... Loop
```

See

Do

Lower$

String function · VMOS 0.1

Syntax

```
Lower$(s$)
```

Returns

s$ converted to lower case.

Examples

Lower case, and a case-insensitive comparison:

```
Print Lower$("HELLO World")      ' hello world
Input "Continue (yes/no)"; A$
If Lower$(A$)="yes" Then Print "continuing"
Wait Key
```

Turn a typed command into lower case before comparing it:

```
Repeat
    Input "Command (help/quit)"; C$
    C$=Lower$(C$)
    If C$="help" Then Print "type quit to end"
Until C$="quit"
```

See

Upper$

M

Max

Function · VMOS 0.1

Syntax

```
Max(a, b)
```

Returns

the larger of a and b; fixed-point if either argument is.

Examples

The larger of two numbers:

```
Print Max(3, 7)            ' 7
Print Max(-2, -9)          ' -2
Print Max(1.5, 1)          ' 1.5
Wait Key
```

Keep a value from going below 0 (a health bar):

```
HP=10
For HIT=1 To 5
    HP=Max(HP-4, 0)
    Print "HP: "; HP
Next HIT
Wait Key
```

See

Min

Mid$

String function · VMOS 0.1

Syntax

```
Mid$(s$, p [, n])
```

Returns

n characters of s$ starting at position p (the first character is 1); the rest of the string if n is omitted.

Examples

Parts of a string:

```
Print Mid$("HELLO", 2, 3)        ' ELL
Print Mid$("HELLO", 3)           ' LLO
Wait Key
```

A string reversed, one character at a time:

```
S$="VIXEN"
R$=""
For I=Len(S$) To 1 Step -1
    R$=R$+Mid$(S$, I, 1)
Next I
Print R$                          ' NEXIV
Wait Key
```

A scrolling message on one line:

```
M$="  WELCOME TO VMOS ON THE VIXEN-32 ...  "
Repeat
    Locate 0, 12
    Print Mid$(M$+M$, P+1, 40)
    P=(P+1) mod Len(M$)
    Wait 5
Until Inkey$<>""
```

See

Left$, Right$, Instr

Min

Function · VMOS 0.1

Syntax

```
Min(a, b)
```

Returns

the smaller of a and b; fixed-point if either argument is.

Examples

The smaller of two numbers:

```
Print Min(3, 7)            ' 3
Print Min(-2, -9)          ' -9
Wait Key
```

Keep a position inside the screen (clamp with Min and Max):

```
X=400
X=Min(Max(X, 32), 335)
Print X                    ' 335
Wait Key
```

See

Max

mod

Operator · VMOS 0.1

Syntax

```
a mod b
```

Returns

the remainder of a / b.

Examples

Remainders:

```
Print 17 mod 5             ' 2
Print 20 mod 5             ' 0
Wait Key
```

Even or odd:

```
For N=1 To 6
    If N mod 2=0 Then Print N; " even" Else Print N; " odd"
Next N
Wait Key
```

Wrap a position around the screen:

```
Repeat
    X=(X+3) mod 320
    Cls
    Bar 32+X, 100 To 40+X, 108
    Wait Vbl
Until Inkey$<>""
```

See

appendix A

Mouse Key

Function · VMOS 0.2

Syntax

```
Mouse Key
```

Returns

the mouse buttons held: 1 left, 2 right, 3 both, 0 none.

Examples

Show which button is held:

```
Show
Repeat
    Home
    Print "buttons: "; Mouse Key
    Wait Vbl
Until Inkey$<>""
```

A simple paint program: draw with the left button, clear with the right:

```
Show
Repeat
    If Mouse Key=1 Then Plot X Mouse, Y Mouse
    If Mouse Key=2 Then Cls
    Wait Vbl
Until Inkey$<>""
```

See

X Mouse, Y Mouse

N

Net Ip$

String function · VMOS 0.20

Syntax

```
Net Ip$
```

Returns

this machine's address on the network, as text ("192.168.5.69"); "" while the network is off or has no address yet. The network is switched on with SHIFT+POUND, in the editor or at BASIC's READY prompt, never by a program (appendix M).

Examples

Check the network before using it:

```
If Net Ip$ = ""
    Print "The network is off: SHIFT+POUND"
Else
    Print "This machine is "; Net Ip$
End If
Wait Key
```

See

Net Lookup$, Ping, Appendix M

Net Lookup$

String function · VMOS 0.20

Syntax

```
Net Lookup$(host$)
```

Params

`host$`	a name ("example.com") or an address ("192.168.5.1")

Returns

the address of host$, as text; "" when the DNS server knows no such name. An address comes back as it is. A name is asked of the DNS server once and kept for the rest of the run, as in every command that takes a host.

Examples

Where a name is:

```
Print "example.com is "; Net Lookup$("example.com")
Wait Key
```

See

Net Ip$, Appendix M

Next

Keyword part · VMOS 0.1

Syntax

```
Next [var]
```

Action

Closes a For loop. The variable name is optional.

Examples

Next with and without the variable:

```
For I=1 To 3
    For J=1 To 2
        Print I; ","; J
    Next
Next I
Wait Key
```

See

For

not

Operator · VMOS 0.1

Syntax

```
not a
```

Returns

the bitwise NOT of a. not True (-1) = False (0), and not False = True.

Examples

Invert a condition:

```
Print "release fire to go on"
While not Fire(2)
    Wait Vbl
```

```
Wend
Wait Key
```

A flag toggled with not:

```
FLAG=False
For I=1 To 4
    FLAG=not FLAG
    Print I; ": "; FLAG
Next I
Wait Key
```

See

and, or

O

Off

Keyword part · VMOS 0.1

Syntax

```
Curs Off, Tiles Off, Sam Loop Off, Copper Off, Every Off, Beep Off ...
```

See

Curs, Tiles, Sam Loop

On

Keyword part · VMOS 0.1

Syntax

```
Curs On, Tiles On, Sam Loop On, Copper On, Every On ...
```

See

Curs, Tiles, Sam Loop

or

Operator · VMOS 0.1

Syntax

```
a or b
```

Returns

the bitwise OR of a and b. With comparisons (True −1, False 0) it works as a logical OR.

Examples

Either condition:

```
X=400
If X<32 or X>351 Then Print "off the screen"
Wait Key
```

Set bits with or:

```
F=0
F=F or 1
F=F or 4
Print Bin$(F, 4)           ' %0101
Wait Key
```

See

and, xor

P

Palette

Command · VMOS 0.18

Syntax

```
Palette n
```

Params

`n`	a palette, 0-7 (default 0)

Action

The machine has eight palettes of 256 colors. Palette chooses the one that Color, Color( ), Copper Color, Load Images, Load Tiles and Td's colors set and read from now on. What a layer shows is chosen apart from it: Screen Palette, Tiles Palette, Border Palette, Sprite Palette, Copper Palette and a tile's own palette (Tile). All eight start as the default palette (0-15 the C64 colors, 16-31 a grey ramp, the rest black); while the VIC-20 is on, 240-255 are its colors in all eight.

Examples

One drawing, two palettes: the screen shows palette 0 and palette 1 in turn:

```
Palette 1
Color 40, 255, 128, 0
Palette 0
Color 40, 0, 128, 255
Ink 40
Circle 192, 112, 60
Bar 100, 180 To 284, 200
For I=1 To 8
    Screen Palette I mod 2
    Wait 25
```

```
Next I
Wait Key
```

Color( ) reads the palette Palette chose:

```
Palette 3
Color 50, $FF8000
Print Hex$(Color(50))            ' $FF8200, as the hardware keeps it
Palette 0
Print Hex$(Color(50))            ' $0: palette 0's color 50 is black
Wait Key
```

See

Screen Palette, Tiles Palette, Border Palette, Sprite Palette, Copper Palette, Color

Paper

Command · VMOS 0.1

Syntax

```
Paper c
```

Params

`c`	color 0-255 (default 0)

Action

Sets the background color for printed text and the color Cls fills the screen with. Color 0 is transparent: the tile layer (Tiles Back) or the background color shows through. Paper does not change the border.

Examples

A blue screen:

```
Paper 6
Cls
Print "white text on blue"
Wait Key
```

Text with a highlighted word:

```
Print "This is ";
Paper 2
Print "important";
Paper 0
Print " text."
Wait Key
```

See

Pen, Cls

Param

Function · VMOS 0.1

Syntax

```
Param
```

Returns

the numeric result of the last procedure call (End Proc[x]), as an integer; a fixed-point result is truncated (Param# keeps the fraction).

Examples

Use a procedure's result:

```
SQUARE[12]
Print Param                ' 144
Wait Key
Procedure SQUARE[N]
End Proc[N*N]
```

A procedure that returns the larger of three numbers:

```
BIGGEST[4, 9, 2]
Print "the biggest: "; Param
Wait Key
Procedure BIGGEST[A, B, C]
    M=Max(A, Max(B, C))
End Proc[M]
```

See

End Proc, Param$

Param$

String function · VMOS 0.1

Syntax

```
Param$
```

Returns

the string result of the last procedure call.

Examples

A procedure that builds a string:

```
STARS[5]
Print Param$               ' *****
Wait Key
Procedure STARS[N]
    S$=""
    For I=1 To N
        S$=S$+"*"
```

```
    Next I
End Proc[S$]
```

A procedure that pads a number with zeros:

```
For S=5 To 5000 Step 995
    PAD[S, 6]
    Print Param$
Next S
Wait Key
Procedure PAD[N, W]
    S$=Str$(N)
    While Len(S$)<W
        S$="0"+S$
    Wend
End Proc[S$]
```

See

Param

Param#

Function · VMOS 0.5

Syntax

```
Param#
```

Returns

the numeric result of the last procedure call as fixed-point (an integer result is converted).

Examples

A fixed-point result:

```
HALF[3]
Print Param#               ' 1.5
Wait Key
Procedure HALF[X#]
End Proc[X#/2]
```

The average of three numbers, with its fraction:

```
AVERAGE[3, 4, 6]
Print Param                ' 4
Print Param#               ' 4.3333
Wait Key
Procedure AVERAGE[A, B, C]
End Proc[(A+B+C)/3.0]
```

See

Param, Param!, appendix G

Param!

Function · VMOS 0.6

Syntax

```
Param!
```

Returns

the numeric result of the last procedure call as a float.

Examples

A float result:

```
AREA[2.5]
Print Param!                ' 19.635
Wait Key
Procedure AREA[R!]
End Proc[Pi!*R!*R!]
```

A procedure that returns a float:

```
HYPOT[3, 4]
Print Param!                ' 5
Wait Key
Procedure HYPOT[A!, B!]
    H!=A!*A!+B!*B!
    R!=H!/2
    For I=1 To 10
        R!=(R!+H!/R!)/2
    Next I
End Proc[R!]
```

See

Param, appendix H

Paste Bob

Command · VMOS 0.3

Syntax

```
Paste Bob x, y, i [, w, h [, a]]
```

Params

`x,y`	top-left corner
`i`	image 1-1024 (Load Images, Get Image; 241-1024 since VMOS 0.22)
`w,h`	(VMOS 0.14) size on screen in pixels, larger or smaller than the image; 0 draws nothing
`a`	(VMOS 0.23) angle, in radians or in degrees after Degree, as for Sin; positive turns clockwise on the screen; omitted: 0

Action

Draws image i into the current screen permanently, color 0 transparent. Unlike a bob, nothing is saved under it and it does not move. It draws into the current screen (both buffers with Double Buffer), in screen pixel coordinates (appendix C), clipped to the 512 x 256 screen. As with Print, anything drawn under a bob is lost when the bob moves.

With w, h the image is scaled to w x h (nearest-pixel: each screen pixel takes the image pixel under its center). The image is loaded into the blitter's texture cache and stays there for the next paste of the same image, so many scaled pastes of one image are fast: about 6 microseconds each up to 16 x 16. The cache holds 256 x 256, so a bigger image (241-1024) is scaled in pieces of 256 x 256.

With a (VMOS 0.23) the w x h box is turned by a about its middle, as Bob turns one: clockwise on the screen, x, y the top-left corner of the box before it is turned. Angle 0 draws exactly as without one.

Examples

Paste images 1 to 4 in a row (run from EXAMPLES):

```
Load Images "BALLS.BMP", 1, 16, 16
For I=1 To 4
    Paste Bob 60+I*40, 100, I
Next I
Wait Key
```

The same image at five sizes:

```
Load Images "BALLS.BMP", 1, 16, 16
For S=1 To 5
    Paste Bob 40+S*S*8, 100, 1, S*8, S*8
Next S
Wait Key
```

A field of balls getting bigger toward the bottom:

```
Load Images "BALLS.BMP", 1, 16, 16
For Y=20 To 190 Step 10
    For X=40 To 320 Step 30
        Paste Bob X, Y, 2, Y/10, Y/10
    Next X
Next Y
Wait Key
```

Twelve ships in a row, each turned 30 degrees more:

```
Degree
Load Images "SHIP.BMP", 1
For K=0 To 11
    Paste Bob 30+K*26, 100, 1, 20, 20, K*30
Next K
Wait Key
```

See

Bob, Load Images

Path Record

Command · VMOS 0.15

Syntax

```
Path Record "file"
```

Params

`file`	name of the recorded path, e.g. "SPIN.PTH", saved in the program's folder

Action

Records the mouse movement until SPACE is pressed, then saves it. The mouse is sampled once per frame, so playback runs at the speed it was recorded. The trail is drawn from the pointer in the Ink color. While the right button is held, moving the mouse away/toward you goes to z (away = deeper) instead of y, and the trail is drawn in the next color (Ink + 1). The file also stores the start position. Play it with Td Path, Bob Path or Sprite Path ... Play. Works only from the editor (F1), not in a release. A recording takes 6 bytes per frame (18 KB per minute).

Examples

Record a path (move the mouse, then press SPACE):

```
Ink 7
Path Record "SPIN.PTH"
Print "saved"
Wait Key
```

Play it back on a 3D cube that leaves a trail (run from EXAMPLES, after recording):

```
Td Load "BOXC.OBJ"
Td Object 1, "BOXC", 192, 112, 0, 0, 0, 0
Td Size 1, 30
Td Path 1, Play "SPIN.PTH"
Repeat
    Td Redraw
    Wait Vbl
Until Not Td Moving(1)
Wait Key
```

See

Td Path

Pen

Command · VMOS 0.1

Syntax

```
Pen c
```

Params

`c`	color 0-255 (default 1, white; 0-15 = the C64 colors)

Action

Sets the text color for Print and Text.

Examples

Text in yellow, then in red:

```
Pen 7
Print "yellow"
Pen 2
Print "red"
Wait Key
```

Every C64 color on its own line:

```
For C=1 To 15
    Pen C
    Print "color "; C
Next C
Wait Key
```

See

Paper

Pi#

Function · VMOS 0.9

Syntax

```
Pi#
```

Returns

pi as a fixed-point number (3.1416). Assigned to a float (!) variable it is the float pi.

Examples

Sine of pi/2 in radians:

```
Print Pi#                  ' 3.1416
Print Sin(Pi#/2)           ' 1
Wait Key
```

The circumference of a circle:

```
R#=10
Print "circumference: "; 2*Pi#*R#
Wait Key
```

See

Pi!, Sin

Pi!

Function · VMOS 0.9

Syntax

```
Pi!
```

Returns

pi as a float (3.14159).

Examples

Area of a circle in floats:

```
R!=2.5
A!=Pi!*R!*R!
Print A!                    ' 19.635
Wait Key
```

A circle drawn with float math:

```
For I=0 To 99
    A!=I*2*Pi!/100
    Plot 192+Cos(A!)*80, 112+Sin(A!)*80
Next I
Wait Key
```

See

Pi#, Cos

Ping

Function · VMOS 0.20

Syntax

```
Ping(host$)
```

Params

`host$`	a name or an address

Returns

the round trip to host$ in milliseconds; −1 when no answer comes within a second, or there is no such name. Waits for the answer.

Examples

Five pings, half a second apart:

```
For I=1 To 5
    Print "1.1.1.1: "; Ping("1.1.1.1"); " ms"
    Wait 25
```

```
Next I
Wait Key
```

See

Net Ip$, Appendix M

Play

Command · VMOS 0.7

Syntax

```
Play s1$ [, s2$ [, s3$]]
Play s1$, s2$, s3$, s4$ [, s5$ ... [, s15$]]
Play voice, s$ [, s2$ ...]
Play
```

Params

`s1$` notes for SID voice 1 (BASIC 65 format, appendix I); s2$, s3$ for SID voices 2 and 3; an omitted string (, ,) leaves that voice unchanged

`s4$` notes for voice 4 = PCM voice 0 (0.9); s5$-s7$ for voices 5-7 = PCM voices 1-3, with samples as instruments (Instrument)

`s8$` notes for voice 8 = YM2151 voice 0 (0.21); s9$-s15$ for voices 9-15 = YM2151 voices 1-7, with FM instruments (Fm Instrument)

`voice` 1-15 (0.21): the first string goes to this voice, the next to the one after it (default 1)

Action

Plays music in the background: the program continues immediately while the frame interrupt plays the notes. A new string for a voice replaces what it was playing. Play with no arguments stops all voices and resets Tempo, the instruments and the filter. While a Sound uses a voice, that voice's music keeps time but is silent. Voices 4-7 (0.9) play samples on the PCM voices: a note sounds until the next one (no gap); a rest or the end of the string stops it; a Sam Play on one of these voices takes it over until Play's next note there.

Examples

A tune on one voice; the program waits 4 seconds while it plays:

```
Tempo 30
Play "O5T9HCIDCDEHCG IGAGFEFDEWCL"
Wait 200
```

A tune with a bass line on a second voice:

```
Tempo 30
Play "O5T9HCIDCDEHCG IGAGFEFDEWCL", "O2T0QCGEGCGEG DBGB CGEGL"
Wait 250
```

A tune on the YM2151 (voice 8) with an FM bass under it (voice 9):

```
Tempo 20
Play 8, "O4T0QCEGECEGE DFAFDFAF", "O2T8HCCDD"
Wait 250
```

Music that keeps playing while the program does other things:

```
Tempo 20
Play "O4T7QCEGECEGE DFAFDFAF CEGECEGE"
Repeat
    Ink Rnd(15)+1
    Plot 32+Rnd(319), 12+Rnd(199)
Until Inkey$<>""
Play
```

See

Rplay, Tempo, Envelope, Fm Instrument, appendix I

Plot

Command · VMOS 0.3

Syntax

```
Plot x, y [, c]
```

Params

`x,y`	the pixel
`c`	its color; omitted: the Ink color

Action

Sets one pixel and sets the start point for Draw To. It draws into the current screen (both buffers with Double Buffer), in screen pixel coordinates (appendix C), clipped to the 512 x 256 screen. As with Print, anything drawn under a bob is lost when the bob moves.

Examples

Two pixels, one in the Ink color and one in red:

```
Plot 160, 100
Plot 162, 100, 2
Wait Key
```

Random stars:

```
For I=1 To 500
    Plot 32+Rnd(319), 12+Rnd(199), Rnd(15)+1
Next I
Wait Key
```

A parabola plotted point by point:

```
For X=-80 To 80
    Plot 192+X, 200-X*X/40
```

```
Next X
Wait Key
```

See

Point, Draw, Ink

Point

Function · VMOS 0.3

Syntax

```
Point(x, y)
```

Returns

the color of the pixel at x,y on the displayed screen; −1 outside the 512 x 256 screen.

Examples

Read back colors that were drawn:

```
Ink 2
Bar 100, 100 To 150, 150
Print Point(120, 120)      ' 2
Print Point(50, 50)        ' 0
Print Point(600, 50)       ' -1
Wait Key
```

A dot that stops when it reaches a wall:

```
Ink 7
Bar 300, 40 To 310, 180
Ink 1
X=40
While Point(X+1, 100)<>7
    Plot X, 100
    Inc X
    Wait Vbl
Wend
Print "hit the wall at X = "; X
Wait Key
```

See

Plot

Polygon

Command · VMOS 0.14

Syntax

```
Polygon x1, y1 To x2, y2 To x3, y3 [To x4, y4 ...]
```

Params

`x1,y1`	3 to 64 corners, in either winding order

Action

Draws a filled polygon in the Ink color. Triangles and convex polygons are drawn by the core's triangle unit; other shapes are filled scanline by scanline with the even-odd rule (a self-intersecting star has an empty center). A pixel is filled when its center is inside the shape; pixels whose center lies exactly on a right or bottom edge are not, so polygons sharing an edge meet without gaps or overlap: Polygon 0, 0 To 10, 0 To 10, 10 To 0, 10 fills 10 x 10 pixels. Like Bar, it draws into the current screen (both buffers with Double Buffer), in screen pixel coordinates, clipped to the 512 x 256 screen. A small triangle takes about 12 microseconds.

Examples

A filled triangle:

```
Ink 5
Polygon 192, 40 To 260, 160 To 124, 160
Wait Key
```

A house: a square and a roof:

```
Ink 10
Polygon 140, 110 To 240, 110 To 240, 190 To 140, 190
Ink 2
Polygon 130, 110 To 190, 60 To 250, 110
Wait Key
```

A five-pointed star; even-odd leaves its center empty:

```
Ink 7
Polygon 192, 40 To 228, 150 To 134, 82 To 250, 82 To 156, 150
Wait Key
```

A spinning triangle, redrawn every frame:

```
Degree
Repeat
    Cls
    Ink 13
    Polygon 192+Cos(A)*80, 112+Sin(A)*80 To 192+Cos(A+120)*80,
112+Sin(A+120)*80 To 192+Cos(A+240)*80, 112+Sin(A+240)*80
    Add A, 3
    Wait Vbl
Until Inkey$<>""
```

See

Bar, Draw, Ink

Pop

Keyword part · VMOS 0.1

Syntax

```
Pop Proc [ [value] ]
```

See

Pop Proc

Pop Proc

Command · VMOS 0.1

Syntax

```
Pop Proc [ [value] ]
```

Action

Returns from the procedure immediately, with an optional result.

Examples

Leave early for bad input:

```
ROOT[-4]
Print Param                ' 0
ROOT[49]
Print Param                ' 7
Wait Key
Procedure ROOT[N]
    If N<0 Then Pop Proc[0]
    R=0
    While (R+1)*(R+1)<=N
        Inc R
    Wend
End Proc[R]
```

Return as soon as a value is found:

```
FINDFIRST[50]
Print "first multiple of 7 over 50: "; Param
Wait Key
Procedure FINDFIRST[N]
    For I=N To N+7
        If I mod 7=0 Then Pop Proc[I]
    Next I
End Proc[-1]
```

See

End Proc

Print

Command · VMOS 0.1

Syntax

```
Print [value] [; value] [, value] ...
```

Action

Prints values at the text cursor. `;` prints the next value directly after the previous one; `,` moves to the next 10-column tab stop. A trailing `;` or `,` keeps the cursor on the line. Numbers are printed without a leading space. Print writes into the current screen (Screen n), inside the text window: long lines wrap, and the window scrolls up below its last line.

Examples

Text, numbers and the separators:

```
SCORE=1250
Print "Score: "; SCORE
Print 1, 2, 3
Print "no new line";
Print " - continued"
Print
Print "after an empty line"
Wait Key
```

A table in 10-column tab stops:

```
Print "N", "N*N", "N*N*N"
For N=1 To 8
    Print N, N*N, N*N*N
Next N
Wait Key
```

Text scrolling up when the window is full:

```
For I=1 To 60
    Print "line "; I
    Wait 3
Next I
Wait Key
```

See

Locate, Window, Text, Pen, Paper, Str$

Proc

Command · VMOS 0.1

Syntax

```
Proc NAME[params]
```

Action

Calls a procedure. The word Proc is optional.

Examples

The same call with and without Proc:

```
Proc HELLO["Ann"]
HELLO["Bob"]
Wait Key
```

```
Procedure HELLO[N$]
    Print "Hello "; N$
End Proc
```

Procedures called with numbers, strings and expressions:

```
Proc BOXAT[50, 50, 7]
BOXAT[150, 80, 2]
For I=0 To 3
    BOXAT[60+I*60, 150, I+3]
Next I
Wait Key
Procedure BOXAT[X, Y, C]
    Ink C
    Bar X, Y To X+40, Y+30
End Proc
```

See

Procedure

Procedure

Command · VMOS 0.1

Syntax

```
Procedure NAME[a, b$ ...] ... End Proc [ [value] ]
```

Action

Defines a procedure with up to 8 parameters (numbers or strings). Its variables are local; recursion works; labels inside it are local to it. Procedures can be defined anywhere, also after the code that calls them; the main program does not run into them.

Examples

A procedure with a parameter:

```
HELLO["Ann"]
HELLO["Bob"]
Wait Key
Procedure HELLO[N$]
    Print "Hello "; N$
End Proc
```

A procedure that draws something at a position:

```
For I=0 To 4
    FLAG[50+I*60, 80, I+2]
Next I
Wait Key
Procedure FLAG[X, Y, C]
    Ink 1
    Draw X, Y To X, Y+60
    Ink C
```

```
    Bar X+1, Y To X+30, Y+20
End Proc
```

Recursion: factorial:

```
FACT[6]
Print Param                ' 720
Wait Key
Procedure FACT[N]
    If N<=1 Then Pop Proc[1]
    FACT[N-1]
End Proc[N*Param]
```

See

End Proc, Proc, Param, Global, Shared, Pop Proc

R

Radian

Command · VMOS 0.9

Syntax

```
Radian
```

Action

Sin and Cos take angles in radians (2 x Pi# per turn). This is the default.

Examples

Switch to degrees and back:

```
Degree
Print Sin(90)              ' 1
Radian
Print Sin(Pi#/2)           ' 1
Wait Key
```

A full turn in radians: 0 to 2 x Pi#:

```
Radian
For A#=0 To 2*Pi# Step 0.05
    Plot 192+Cos(A#)*90, 112+Sin(A#)*90
Next A#
Wait Key
```

See

Degree, Sin

Randomize

Command · VMOS 0.1

Syntax

```
Randomize n
```

Action

Seeds the random number generator. The same seed gives the same sequence of Rnd values. Without Randomize the seed comes from the frame counter.

Examples

The same five numbers twice:

```
Randomize 42
For I=1 To 5
    Print Rnd(99); " ";
Next I
Print
Randomize 42
For I=1 To 5
    Print Rnd(99); " ";
Next I
Wait Key
```

A star map that is the same every run:

```
Randomize 7
For I=1 To 300
    Plot 32+Rnd(319), 12+Rnd(199)
Next I
Wait Key
```

See

Rnd

Raster

Function · VMOS 0.3

Syntax

```
Raster
```

Returns

the raster line being displayed now: 0-223 is the visible picture (0 = the top of the display, under the border), 224-311 the vertical blank.

Examples

Wait for a line and read it back:

```
Wait Raster 100
Print Raster               ' 99: it returns just before line 100
Wait Key
```

Measure how many lines some work takes:

```
Wait Vbl
A=Raster
For I=1 To 2000
    X=X+I
Next I
B=Raster
Print "took "; B-A; " lines"
Wait Key
```

See

Wait Raster, Wait Vbl

Read

Command · VMOS 0.1

Syntax

```
Read var [, var ...]
```

Action

Reads the next Data values into the variables, in order. A number is converted to the variable's type (an integer variable takes the integer part of a fixed-point value).

Examples

Values of every type:

```
Read W, H, NAME$, SPEED#
Print W; " x "; H; " "; NAME$; " "; SPEED#
Wait Key
Data 320, 200, "pixels", 1.5
```

Fill an array from Data:

```
Dim DAYS(11)
For M=0 To 11
    Read DAYS(M)
Next M
Print "days in June: "; DAYS(5)
Wait Key
Data 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31
```

Draw a shape from coordinates in Data:

```
Read N
Read X, Y
Plot X, Y
For I=2 To N
    Read X, Y
    Draw To X, Y
Next I
Wait Key
```

```
Data 5
Data 192, 40, 260, 180, 100, 90, 284, 90, 124, 180
```

See

Data, Restore

Rem

Command · VMOS 0.1

Syntax

```
Rem text     or     ' text
```

Action

A comment to the end of the line. Both forms can follow other statements on a line.

Examples

Comments on their own lines and after code:

```
Rem draw the frame
Box 40, 20 To 340, 200     ' the outline
' wait for a key
Wait Key
```

Comment out a line to switch it off:

```
Ink 2
Bar 40, 40 To 100, 100
' Bar 120, 40 To 180, 100
Rem Bar 200, 40 To 260, 100
Wait Key
```

Repeat

Command · VMOS 0.1

Syntax

```
Repeat ... Until c
```

Action

Runs the lines until c is true. The body runs at least once.

Examples

Count to 10:

```
Repeat
    Inc X
    Print X
Until X=10
Wait Key
```

Ask until the answer is valid:

```
Repeat
    Input "A number from 1 to 9"; N
Until N>=1 and N<=9
Print "thanks: "; N
Wait Key
```

The typical main loop: run until a key is pressed:

```
Repeat
    Ink Rnd(15)+1
    Circle 32+Rnd(319), 12+Rnd(199), 10
    Wait Vbl
Until Inkey$<>""
```

See

Until, While, Do

Reserve Bob

Command · VMOS 0.3

Syntax

```
Reserve Bob n
```

Params

`n`	number of bobs, 1-1024

Action

Allocates the bob table for bobs 0 to n-1 (40 bytes per bob). Use it once, before the first Bob. Without it, the first Bob allocates bobs 0-63; with it, exactly n (Reserve Bob 16 gives bobs 0-15). The table stays until the program ends. Background save buffers grow as needed: each bob's copy is the size of the area it covers (256 bytes for a 16 x 16 bob; twice with Double Buffer). Each visible bob costs blitter time at every Wait Vbl; reserved but unused bobs cost nothing.

Examples

200 bobs on a grid (run from EXAMPLES):

```
Load Images "BALLS.BMP", 1, 16, 16
Reserve Bob 200
For I=0 To 199
    Bob I, 32+(I mod 20)*16, 12+(I/20)*16, I mod 4+1
Next I
Wait Vbl
Wait Key
```

Reserve exactly the bobs a program needs: 8 bobs, numbered 0 to 7:

```
Degree
Load Images "BALLS.BMP", 1, 16, 16
Reserve Bob 8
```

```
Repeat
    For I=0 To 7
        Bob I, 184+Cos(A+I*45)*90, 104+Sin(A+I*45)*70, I mod 4+1
    Next I
    Add A, 2
    Wait Vbl
Until Inkey$<>""
```

See

Bob, Bob Off

Restore

Command · VMOS 0.1

Syntax

```
Restore [LABEL]
```

Action

Sets the Data pointer: the next Read starts at the first Data statement, or at the first Data after LABEL.

Examples

Read the same Data twice:

```
For I=1 To 3
    Read N
    Print N
Next I
Restore
Read N
Print "again: "; N
Wait Key
Data 10, 20, 30
```

Pick a level's data by label:

```
Input "Level (1 or 2)"; L
If L=1 Then Restore LEVEL1 Else Restore LEVEL2
Read W, H, NAME$
Print NAME$; ": "; W; " x "; H
Wait Key
LEVEL1:
Data 20, 10, "the cave"
LEVEL2:
Data 40, 30, "the castle"
```

See

Data, Read

Return

Command · VMOS 0.1

Syntax

```
Return
```

Action

Returns from a Gosub subroutine to the statement after the Gosub.

Examples

A subroutine that draws a box at X:

```
X=60
Gosub FRAME
X=200
Gosub FRAME
Wait Key
End
FRAME:
Box X, 80 To X+80, 160
Return
```

Nested subroutines: one Gosub calls another:

```
Gosub OUTER
Wait Key
End
OUTER:
Print "outer start"
Gosub INNER
Print "outer end"
Return
INNER:
Print "  inner"
Return
```

See

Gosub

Right$

String function · VMOS 0.1

Syntax

```
Right$(s$, n)
```

Returns

the last n characters of s$.

Examples

The end of a string:

```
Print Right$("HELLO", 2)         ' LO
Wait Key
```

Numbers right-aligned in 6 columns:

```
For N=1 To 100000 Step 9999
    Print Right$("      "+Str$(N), 6)
Next N
Wait Key
```

The extension of a file name:

```
F$="SHIP.BMP"
Print "extension: "; Right$(F$, 3)
Wait Key
```

See

Left$, Mid$

Rnd

Function · VMOS 0.1

Syntax

```
Rnd(n)
```

Returns

a random integer from 0 to n (inclusive).

Examples

Roll a die ten times:

```
For I=1 To 10
    Print Rnd(5)+1; " ";
Next I
Wait Key
```

A random choice from a list:

```
Dim C$(3)
C$(0)="red"
C$(1)="green"
C$(2)="blue"
C$(3)="yellow"
Print "today: "; C$(Rnd(3))
Wait Key
```

Random rectangles all over the screen:

```
Repeat
    Ink Rnd(15)
    X=32+Rnd(300)
    Y=12+Rnd(180)
```

```
    Bar X, Y To X+Rnd(20), Y+Rnd(20)
Until Inkey$<>""
```

See

Randomize

Rplay

Function · VMOS 0.7

Syntax

```
Rplay(v)
```

Params

v	a Play voice, 1-15 (4-7 the PCM voices, 8-15 the YM2151)

Returns

True while voice v is playing its string; False when it has ended or never started.

Examples

Wait until a tune has finished:

```
Play "O4ICDEFGABO5C"
Repeat
    Wait Vbl
Until Not Rplay(1)
Print "the tune has ended"
Wait Key
```

Animate while the music plays:

```
Play "O4QCEGO5CO4GEC"
Repeat
    Ink Rnd(15)+1
    Circle 192, 112, Rnd(90)
    Wait Vbl
Until Not Rplay(1)
```

See

Play

S

Sam Loop

Command · VMOS 0.2

Syntax

```
Sam Loop On | Sam Loop Off
```

Action

Sets whether samples started by Sam Play from now on loop (On) or play once (Off, the default).

Examples

A looping sample until a key is pressed (run from EXAMPLES):

```
Load Sample "PLUCK.WAV", 1
Sam Loop On
Sam Play 1
Wait Key
Sam Stop
```

Loop on, then off: the second sample plays once:

```
Load Sample "PLUCK.WAV", 1
Load Sample "BOOM.WAV", 2
Sam Loop On
Sam Play 0, 1
Wait 100
Sam Stop 0
Sam Loop Off
Sam Play 1, 2
Wait 100
```

See

Sam Play

Sam Play

Command · VMOS 0.2

Syntax

```
Sam Play s
Sam Play v, s
Sam Play v, s, rate
```

Params

`s`	sample 1-64 (Load Sample)
`v`	voice 0-3; omitted: the next voice in rotation
`rate`	playback rate in Hz; omitted: the sample's own rate

Action

Starts playing a sample. It plays in the background; the program continues immediately.

Examples

Play a sample (run from EXAMPLES):

```
Load Sample "BOOM.WAV", 1
Sam Play 1
Wait 100
```

The same sample at three rates (pitches), on voice 0:

```
Load Sample "PLUCK.WAV", 1
For R=11025 To 44100 Step 11025
    Sam Play 0, 1, R
    Wait 40
Next R
```

A sound on every key press; several can overlap on the four voices:

```
Load Sample "BOOM.WAV", 1
Print "press keys, q quits"
Repeat
    K$=Inkey$
    If K$<>"" Then Sam Play 1
Until K$="q"
```

See

Load Sample, Sam Stop, Sam Loop, Volume

Sam Stop

Command · VMOS 0.2

Syntax

```
Sam Stop [v]
```

Action

Stops voice v, or all four voices if v is omitted.

Examples

Stop a looping sample after two seconds:

```
Load Sample "PLUCK.WAV", 1
Sam Loop On
Sam Play 0, 1
Wait 100
Sam Stop 0
```

Stop all voices at once:

```
Load Sample "PLUCK.WAV", 1
Sam Loop On
For V=0 To 3
    Sam Play V, 1, 11025*(V+1)
Next V
Wait 100
Sam Stop
Wait Key
```

See

Sam Play

Scancode

Function · VMOS 0.1

Syntax

`Scancode`

Returns

the key number (appendix D) of the last key read by Inkey$ or Wait Key.

Examples

Print the number of each key pressed (ESC quits):

```
Repeat
    Wait Key
    Print "key number "; Scancode
Until Scancode=71
```

React to F1 and F3 (keys that type nothing visible):

```
Print "F1 = red, F3 = blue, q = quit"
Repeat
    K$=Inkey$
    If K$<>""
        If Scancode=4 Then Border 2
        If Scancode=5 Then Border 6
    End If
Until K$="q"
```

See

Key State, Inkey$

Screen

Command · VMOS 0.3

Syntax

`Screen n`

Params

`n`	an open screen 0-7 (screen 0 is always open)

Action

Selects the screen that Print, Text, Cls, the drawing commands and Paste Bob write to, whether it is displayed or not. Each screen has its own text cursor; the text window, Pen, Paper and Ink are shared.

Examples

Draw on a hidden screen, then show it:

```
Screen Open 1
Screen To Front 0
Print "screen 0 is shown - press a key"
Screen 1
Paper 6
Cls
Print "this was drawn on screen 1"
Wait Key
Screen To Front 1
Wait Key
```

Flicker-free animation with two screens: draw one while the other is shown:

```
Screen Open 1
Screen Open 2
S=1
Repeat
    Screen S
    Cls
    Circle 192+Sin(T#)*100, 112, 20
    Screen To Front S
    Wait Vbl
    S=3-S
    Add T#, 0.05
Until Inkey$<>""
```

See

Screen Open, Screen To Front

Screen Copy

Command · VMOS 0.3

Syntax

```
Screen Copy a, x1, y1, x2, y2 To b, x3, y3
```

Params

`a`	source screen
`x1,y1`	one corner of the rectangle
`x2,y2`	the opposite corner (both inclusive)
`b`	destination screen (may be a)
`x3,y3`	destination top-left corner

Action

Copies a rectangle of pixels with the blitter, clipped to both screens. Source and destination may overlap within one screen. With Double Buffer, copying from screen 0 reads the displayed buffer, and copying to it writes both buffers.

Examples

Copy a drawing from a hidden screen to the visible one:

```
Screen Open 1
Ink 7
Circle 32, 32, 30
Screen To Front 0
Screen 0
For I=0 To 4
    Screen Copy 1, 0, 0, 63, 63 To 0, 40+I*64, 80
Next I
Wait Key
```

Duplicate part of the screen within the same screen:

```
Ink 2
Bar 40, 20 To 100, 60
Text 44, 36, "COPY"
Screen Copy 0, 40, 20, 100, 60 To 0, 200, 140
Wait Key
```

See

Screen, Paste Bob

Screen Hide

Command · VMOS 0.3

Syntax

```
Screen Hide
```

Action

Hides the program's screen (text, bobs, drawing); only the tiles, the sprites and Color Back remain visible. The screen keeps its contents; Screen Show displays it again.

Examples

Hide and show the screen:

```
Print "now you see me"
Wait 50
Screen Hide
Wait 50
Screen Show
Wait Key
```

Build a picture while the screen is hidden, then show it all at once:

```
Screen Hide
For I=1 To 200
    Ink Rnd(15)+1
    Circle 32+Rnd(319), 12+Rnd(199), Rnd(20)
Next I
Screen Show
Wait Key
```

See

Screen Show, Color Back

Screen Offset

Command · VMOS 0.3

Syntax

```
Screen Offset x, y
```

Params

`x`	0-511 (wraps around)
`y`	0-255 (wraps around)

Action

Scrolls the display over the program's 512 x 256 screen: pixel x,y of the screen appears at the display's top-left. Text, bobs and drawing keep the screen's own coordinates; sprites and the tile layer do not scroll with it. The offset changes during vertical blank, so it never tears; with Double Buffer it changes at the Wait Vbl that shows the bobs.

Examples

Scroll the screen sideways forever (it wraps at 512):

```
For I=1 To 25
    Print "SCROLLING TEXT - SCROLLING TEXT - SCROLLING"
Next I
Repeat
    Inc X
    Screen Offset X and 511, 0
    Wait Vbl
Until Inkey$<>""
```

Shake the screen for an explosion:

```
Print "BOOM!"
For I=1 To 30
    Screen Offset Rnd(8), Rnd(8)
    Wait Vbl
Next I
Screen Offset 0, 0
Wait Key
```

See

Tile Scroll, Double Buffer, appendix C

Screen Open

Command · VMOS 0.3

Syntax

```
Screen Open n
```

Params

`n`	screen 0-7

Action

Opens screen n: its own 512 x 256 pixels (128 KB of chip RAM), cleared to the Paper color. It becomes the current screen and the displayed screen, as in AMOS. Opening it again clears it and keeps its memory. Screen 0 is open from the start.

Examples

A second screen in blue:

```
Screen Open 1
Paper 6
Cls
Print "screen 1"
Wait Key
```

Three pre-drawn screens shown in turn:

```
For N=1 To 3
    Screen Open N
    Ink N+1
    Circle 192, 112, N*30
Next N
Repeat
    For N=1 To 3
        Screen To Front N
        Wait 10
    Next N
Until Inkey$<>""
```

See

Screen, Screen To Front, Screen Copy

Screen Palette

Command · VMOS 0.18

Syntax

```
Screen Palette n
```

Params

`n` a palette, 0-7 (default 0)

Action

The screen (everything drawn, printed and pasted, and Color Back) shows its colors from palette n, at once. A whole new set of colors in one step: prepared in another palette (Palette, Color), then shown. Copper Palette changes it from a line down.

Examples

A flash: palette 1 all white, shown for a few frames at a time:

```
Ink 2
Bar 60, 40 To 320, 180
Palette 1
For C=0 To 15
    Color C, 255, 255, 255
Next C
For I=1 To 5
    Screen Palette 1
    Wait 4
    Screen Palette 0
    Wait 20
Next I
Wait Key
```

Night: palette 1 is the C64 colors at a third of their light; a key switches:

```
For C=0 To 15
    V=Color(C)
    Palette 1
    Color C, (V/65536)/3, ((V/256) mod 256)/3, (V mod 256)/3
    Palette 0
Next C
For C=1 To 15
    Ink C
    Bar C*20, 40 To C*20+15, 180
Next C
Print "A key: night"
Wait Key
Screen Palette 1
Wait Key
```

See

Palette, Copper Palette

Screen Show

Command · VMOS 0.3

Syntax

```
Screen Show
```

Action

Displays the program's screen again after Screen Hide. The screen is shown by default.

Examples

Blink the screen three times:

```
Print "blink"
For I=1 To 3
    Screen Hide
    Wait 15
    Screen Show
    Wait 15
Next I
Wait Key
```

Show the screen again after a hidden setup (sprites stay visible meanwhile):

```
Load Images "SHIP.BMP", 1
Sprite 0, 184, 104, 1
Screen Hide
Print "this text appears after two seconds"
Wait 100
Screen Show
Wait Key
```

See

Screen Hide

Screen To Front

Command · VMOS 0.3

Syntax

```
Screen To Front n
```

Params

`n`	an open screen 0-7

Action

Displays screen n; the switch happens between two frames, never mid-frame. The bobs move with it: they leave the old screen as it is, and the next Wait Vbl draws them on the new one. Screen Offset and the border apply to whichever screen is displayed.

Examples

A three-frame animation from three screens:

```
For N=1 To 3
    Screen Open N
    Ink 7
```

```
    Bar 100+N*40, 100 To 130+N*40, 130
Next N
Repeat
    For N=1 To 3
        Screen To Front N
        Wait 8
    Next N
Until Inkey$<>""
```

Switch between two pages with a key:

```
Screen Open 1
Paper 2
Cls
Print "PAGE 1 - press a key"
Screen Open 2
Paper 6
Cls
Print "PAGE 2 - press a key"
For I=1 To 4
    Screen To Front 1+(I mod 2)
    Wait Key
Next I
```

See

Screen, Screen Open, Double Buffer

Sgn

Function · VMOS 0.1

Syntax

```
Sgn(n)
```

Returns

the sign of n: −1, 0 or 1.

Examples

Signs:

```
Print Sgn(-7)              ' -1
Print Sgn(0)               ' 0
Print Sgn(3.5)             ' 1
Wait Key
```

Move one step toward a target, from either side:

```
X=40
TARGET=300
While X<>TARGET
    Add X, Sgn(TARGET-X)
    Plot X, 100
```

```
Wend
Wait Key
```

See

Abs

Shared

Command · VMOS 0.1

Syntax

```
Shared var [, var ...]
```

Action

Inside a procedure: makes these variables refer to the main program's variables.

Examples

A procedure that changes a main-program variable:

```
LIVES=3
LOSELIFE
LOSELIFE
Print LIVES                ' 1
Wait Key
Procedure LOSELIFE
    Shared LIVES
    Dec LIVES
End Proc
```

Shared arrays: a procedure that fills a main-program array:

```
Dim A(9)
FILL[7]
For I=0 To 9
    Print A(I); " ";
Next I
Wait Key
Procedure FILL[V]
    Shared A
    For I=0 To 9
        A(I)=V*I
    Next I
End Proc
```

See

Global, Procedure

Show

Command · VMOS 0.2

Syntax

```
Show
```

Action

Shows the mouse pointer.

Examples

Show the pointer and follow it with a circle:

```
Show
Repeat
    Cls
    Circle X Mouse, Y Mouse, 20
    Wait Vbl
Until Inkey$<>""
```

Show the pointer only while the right button is held:

```
Repeat
    If Mouse Key=2 Then Show Else Hide
    Wait Vbl
Until Inkey$<>""
```

See

Hide, X Mouse

Sid

Command · VMOS 0.7

Syntax

```
Sid r, v
```

Params

`r`	SID register 0-28 (the C64's $D400 + r)
`v`	value 0-255

Action

Writes a SID register directly, for anything the other sound commands do not cover. Play and Sound also write the registers of the voices they use.

Examples

A 440 Hz pulse tone on voice 1, written register by register:

```
Sid 0, 7382 and 255        ' frequency, low byte
Sid 1, 7382/256            ' frequency, high byte
Sid 2, 0                   ' pulse width, low byte
Sid 3, 8                   ' pulse width, high byte
Sid 5, 9                   ' attack 0, decay 9
Sid 6, 9*16                ' sustain 9, release 0
Sid 24, 15                 ' volume 15
Sid 4, $41                 ' pulse wave, gate on
Wait 50
```

```
Sid 4, $40                  ' gate off: the note fades
Wait 25
```

A noise "wind" sound whose frequency drifts:

```
Sid 5, 0
Sid 6, $F0
Sid 24, 15
Sid 4, $81                  ' noise, gate on
For F=0 To 255
    Sid 1, F
    Wait 1
Next F
Sid 4, $80
```

See

Sid Volume, appendix I

Sid Volume

Command · VMOS 0.7

Syntax

```
Sid Volume n
```

Params

`n`	0 (silent) to 15 (loudest)

Action

Sets the SID master volume (Play's U0-U9 also sets it). Samples have their own volume: Volume.

Examples

A tune that fades out:

```
Play "O4T7QCDEFGABO5CO4BAGFEDC"
For V=15 To 0 Step -1
    Sid Volume V
    Wait 10
Next V
Play
Sid Volume 15
```

The same note at four volumes:

```
For V=15 To 3 Step -4
    Sid Volume V
    Play "O4T7HC"
    Wait 40
Next V
Sid Volume 15
```

See

Volume, Sid

Sin

Function · VMOS 0.9

Syntax

`Sin(a)`

Params

`a`	angle in radians, or in degrees after Degree

Returns

the sine of a: a float (!) for a float angle (computed on the float unit, as C's sinf), fixed-point (#) for a fixed-point or integer angle (from a table, accurate to the last of the 16 fraction bits). In degrees, Sin(30) is exactly 0.5 and Sin(180!) exactly 0.

Examples

Sines in degrees:

```
Degree
Print Sin(30)              ' 0.5
Print Sin(90)              ' 1
Print Sin(270)             ' -1
Wait Key
```

A circle of dots:

```
Degree
For A=0 To 359 Step 5
    Plot 192+Cos(A)*70, 112+Sin(A)*70
Next A
Wait Key
```

A sine wave across the screen:

```
Degree
For X=32 To 351
    Plot X, 112+Sin((X-32)*360/160)*60
Next X
Wait Key
```

A ball bobbing up and down smoothly (run from EXAMPLES):

```
Degree
Load Images "BALLS.BMP", 1, 16, 16
Repeat
    Bob 0, 184, 104+Sin(A)*80, 1
    Add A, 4
```

```
    Wait Vbl
Until Inkey$<>""
```

See

Cos, Degree, Radian, Pi#

Sound

Command · VMOS 0.7

Syntax

```
Sound v, freq, time [, dir, min, sweep, wave, pulse]
```

Params

`v`	SID voice 1-3
`freq`	SID frequency value, 0-65535 = Hz x 16.777 (7382 = 440 Hz)
`time`	duration in frames, 0-32767 (50 per second); 0 stops the voice's sound
`dir`	sweep direction: 0 up, 1 down, 2 up and down
`min`	lower end of the sweep, 0-65535
`sweep`	added to (or subtracted from) the frequency each frame, 0-65535
`wave`	0 triangle, 1 sawtooth, 2 pulse (default), 3 noise
`pulse`	pulse width, 0-4095 (default 2048)

Action

Plays a sound effect in the background, as in BASIC 65: the frequency sweeps between min and freq, wrapping around to the other end (or reversing, with dir 2). Values may be left empty: Sound 1, 7382, 50, , , , 1. The voice's music (Play) is muted while the sound plays.

Examples

A short beep at 440 Hz:

```
Sound 1, 7382, 10
Wait 20
```

A laser: a sawtooth sweeping down fast:

```
Sound 1, 20000, 50, 1, 2000, 800, 1
Wait 60
```

An explosion: noise sweeping down:

```
Sound 1, 6000, 80, 1, 200, 70, 3
Wait 90
```

A siren: up and down:

```
Sound 1, 12000, 200, 2, 6000, 200, 2
Wait 210
```

See

Sound Clr, Beep, Sam Play

Sound Clr

Command · VMOS 0.7

Syntax

```
Sound Clr
```

Action

Stops all SID sounds and music and resets the music system to its defaults.

Examples

Stop a long sound early:

```
Sound 1, 12000, 1000, 2, 6000, 200, 2
Wait 50
Sound Clr
Wait Key
```

Stop music and effects together with a key:

```
Tempo 20
Play "O4T7QCEGECEGEL"
Sound 2, 12000, 1000, 2, 6000, 200, 2
Print "a key stops all sound"
Wait Key
Sound Clr
Wait Key
```

See

Sound, Play

Space$

String function · VMOS 0.1

Syntax

```
Space$(n)
```

Returns

a string of n spaces (0-32767).

Examples

Indent text:

```
For I=0 To 5
    Print Space$(I*2); "level "; I
Next I
Wait Key
```

Erase a line of text by overwriting it with spaces:

```
Print "this line will be erased"
Wait 50
Locate 0, 0
Print Space$(40)
Wait Key
```

See

String$

Sprite

Command · VMOS 0.2

Syntax

```
Sprite n, x, y [, i]
```

Params

`n`	sprite 0-14 (higher numbers are in front; the mouse pointer is in front of all)
`x,y`	top-left corner in screen coordinates (appendix C)
`i`	image 1-240 (Load Images; images 241-1024 are for bobs only); omitted: keeps the current image

Action

Displays hardware sprite n at x,y. Sprites take no CPU or blitter time to draw; move them right after Wait Vbl for smooth motion.

Examples

A sprite in the middle of the screen (run from EXAMPLES):

```
Load Images "SHIP.BMP", 1
Sprite 0, 184, 104, 1
Wait Key
```

A sprite following the mouse:

```
Load Images "SHIP.BMP", 1
Repeat
    Sprite 0, X Mouse, Y Mouse, 1
    Wait Vbl
Until Inkey$<>""
```

Four sprites circling, each with its own image:

```
Degree
Load Images "BALLS.BMP", 1, 16, 16
Repeat
    For N=0 To 3
        Sprite N, 184+Cos(A+N*90)*80, 104+Sin(A+N*90)*80, N+1
    Next N
    Add A, 2
```

```
    Wait Vbl
Until Inkey$<>""
```

See

Sprite Off, Sprite Col, Load Images

Sprite Behind

Command · VMOS 0.3

Syntax

```
Sprite Behind n
```

Params

`n`	sprite 0-14

Action

Moves sprite n behind the program's screen and the tiles: it is visible only where both are transparent (color 0). Sprite Front puts it back in front (the default).

Examples

A sprite passing behind a bar:

```
Load Images "SHIP.BMP", 1
Ink 2
Bar 160, 12 To 220, 211
Sprite Behind 0
For X=60 To 320
    Sprite 0, X, 104, 1
    Wait Vbl
Next X
Wait Key
```

Sprites going behind and in front of a drawing as they move up and down:

```
Load Images "BALLS.BMP", 1, 16, 16
Ink 2
Bar 32, 90 To 351, 130
Repeat
    For N=0 To 3
        Y=100+Sin(T#+N)*60
        Sprite N, 80+N*60, Y, N+1
        If Y>110 Then Sprite Behind N Else Sprite Front N
    Next N
    Add T#, 0.05
    Wait Vbl
Until Inkey$<>""
```

See

Sprite Front, Sprite, appendix C

Sprite Col

Function · VMOS 0.2

Syntax

```
Sprite Col(n)
```

Returns

True if sprite n overlapped another sprite (or the mouse pointer) in the last displayed frame, pixel-exact.

Examples

Two sprites flying into each other:

```
Load Images "BALLS.BMP", 1, 16, 16
X=40
Repeat
    Sprite 0, X, 104, 1
    Sprite 1, 330-X, 104, 2
    Inc X
    Wait Vbl
Until Sprite Col(0)
Print "crash"
Wait Key
```

Touch the sprite with the mouse pointer:

```
Load Images "SHIP.BMP", 1
Sprite 0, 184, 104, 1
Show
Repeat
    Wait Vbl
Until Sprite Col(0)
Print "touched"
Wait Key
```

See

Sprite, Bob Col

Sprite Flip

Command · VMOS 0.3

Syntax

```
Sprite Flip n, h, v
```

Params

`n`	sprite 0-14
`h`	non-zero = mirror left-right
`v`	non-zero = upside down

Action

Flips sprite n's image; its position stays the same. Sprite Flip n, 0, 0 restores the normal orientation.

Examples

A ship that faces the way it flies:

```
Load Images "SHIP.BMP", 1
X=40
D=2
Repeat
    Add X, D
    If X>320 or X<40 Then D=-D
    Sprite Flip 0, D<0, 0
    Sprite 0, X, 104, 1
    Wait Vbl
Until Inkey$<>""
```

The four orientations side by side:

```
Load Images "SHIP.BMP", 1
For N=0 To 3
    Sprite N, 80+N*60, 104, 1
    Sprite Flip N, N and 1, N and 2
Next N
Wait Key
```

See

Sprite, Tile

Sprite Front

Command · VMOS 0.3

Syntax

```
Sprite Front n
```

Params

`n`	sprite 0-14

Action

Moves sprite n in front of the program's screen and the tiles (the default).

Examples

Behind, then in front, of a bar:

```
Load Images "SHIP.BMP", 1
Ink 2
Bar 150, 80 To 230, 140
Sprite 0, 180, 104, 1
Sprite Behind 0
```

```
Wait 50
Sprite Front 0
Wait Key
```

Toggle a sprite in front of and behind a panel with a key:

```
Load Images "SHIP.BMP", 1
Ink 6
Bar 120, 70 To 260, 140
Sprite 0, 180, 96, 1
Print "f = front, b = behind, q = quit"
Repeat
    K$=Inkey$
    If K$="f" Then Sprite Front 0
    If K$="b" Then Sprite Behind 0
Until K$="q"
```

See

Sprite Behind

Sprite Moving

Function · VMOS 0.15

Syntax

```
Sprite Moving(n)
```

Returns

True (-1) while sprite n's path (Sprite Path) is running; False (0) when it has ended, was stopped, or none was set.

Examples

Start a new path each time the last one has ended:

```
Load Images "SHIP.BMP", 1
Sprite 0, 184, 104, 1
Repeat
    If Not Sprite Moving(0) Then Sprite Path 0, "M
"+Str$(Rnd(100)-50)+","+Str$(Rnd(100)-50)+",25"
    Wait Vbl
Until Inkey$<>""
```

Wait for a sprite's path to end, then remove the sprite:

```
Load Images "BALLS.BMP", 1, 16, 16
Sprite 0, 40, 104, 2
Sprite Path 0, "M 290,0,120"
Repeat
    Wait Vbl
Until Not Sprite Moving(0)
Sprite Off 0
```

```
Print "the sprite arrived and was removed"
Wait Key
```

See

Sprite Path

Sprite Off

Command · VMOS 0.2

Syntax

```
Sprite Off [n]
```

Action

Turns sprite n off, or all sprites if n is omitted.

Examples

Show three sprites, then remove them one by one:

```
Load Images "BALLS.BMP", 1, 16, 16
For N=0 To 2
    Sprite N, 120+N*50, 104, N+1
Next N
For N=0 To 2
    Wait 50
    Sprite Off N
Next N
Wait Key
```

Remove all sprites at once:

```
Load Images "BALLS.BMP", 1, 16, 16
For N=0 To 9
    Sprite N, 40+N*30, 104, N mod 4+1
Next N
Wait 100
Sprite Off
Wait Key
```

See

Sprite

Sprite Palette

Command · VMOS 0.18

Syntax

```
Sprite Palette n, p
```

Params

`n`	sprite 0-14

`p` a palette, 0-7 (default 0)

Action

Sprite n shows its image in palette p, at once if it is on: one image in another set of colors, or the sprites in colors of their own, apart from the screen's.

Examples

One image, two sprites: palette 1 has blue for the ball's red (run from EXAMPLES):

```
Load Images "BALLS.BMP", 1, 16, 16
Palette 1
Color 2, 0, 0, 255
Sprite Palette 0, 1
Sprite 0, 150, 100, 1
Sprite 1, 230, 100, 1
Wait Key
```

The ball changes color each second through palettes 0-3 (run from EXAMPLES):

```
Load Images "BALLS.BMP", 1, 16, 16
For P=1 To 3
    Palette P
    Color 2, P*80, 255-P*80, 128
Next P
Sprite 0, 180, 100, 1
For I=0 To 11
    Sprite Palette 0, I mod 4
    Wait 50
Next I
Wait Key
```

See

Sprite, Palette

Sprite Path

Command · VMOS 0.15

Syntax

```
Sprite Path n, "steps"
Sprite Path n, Play "file" [, scale [, times]]
Sprite Path n Off
```

Params

`n` sprite 0-14

`steps`, `file`, `scale`, `times` as for Bob Path

Action

Moves sprite n automatically, one step per frame, like Bob Path does for bobs. The path advances at Wait Vbl, so the program loop must contain one.

Examples

A ship patrolling up and down, pausing at each end:

```
Load Images "SHIP.BMP", 1
Sprite 0, 184, 40, 1
Sprite Path 0, "M 0,120,40 W 25 M 0,-120,40 W 25 L"
Repeat
    Wait Vbl
Until Inkey$<>""
```

A diamond path:

```
Load Images "BALLS.BMP", 1, 16, 16
Sprite 0, 184, 30, 3
Sprite Path 0, "M 80,75,30 M -80,75,30 M -80,-75,30 M 80,-75,30 L"
Repeat
    Wait Vbl
Until Inkey$<>""
```

See

Bob Path, Td Path, Sprite Moving

State

Keyword part · VMOS 0.1

Syntax

```
Key State(k)
```

See

Key State

Step

Keyword part · VMOS 0.1

Syntax

```
For var = a To b Step s
```

Examples

Steps up, down and in fractions:

```
For I=0 To 20 Step 5
    Print I; " ";
Next I
Print
For I=3 To 1 Step -1
    Print I; " ";
```

```
Next I
Print
For A#=0 To 1 Step 0.5
    Print A#; " ";
Next A#
Wait Key
```

See

For

Stop

Command · VMOS 0.1

Syntax

```
Stop
```

Action

Ends the program, like End.

Examples

Stop when a condition is met:

```
For I=1 To 100
    Print I
    If I=5 Then Stop
Next I
```

Stop from inside a procedure:

```
For I=1 To 10
    TEST[I]
Next I
Procedure TEST[N]
    If N=3 Then Stop
    Print "test "; N
End Proc
```

See

End

Str$

String function · VMOS 0.1

Syntax

```
Str$(n)
```

Returns

n converted to a string, without a leading space; fixed-point numbers with up to 4 decimals (Str$(1/3.0) = "0.3333").

Examples

Numbers into strings:

```
SCORE=1250
A$="Score "+Str$(SCORE)
Print A$
Print Str$(1.5)+" meters"
Wait Key
```

Count the digits of a number:

```
N=123456
Print N; " has "; Len(Str$(N)); " digits"
Wait Key
```

A score drawn with Text at a pixel position:

```
For S=0 To 990 Step 10
    Text 260, 20, "SCORE "+Str$(S)+"  "
    Wait 2
Next S
Wait Key
```

See

Val

String$

String function · VMOS 0.1

Syntax

```
String$(s$, n)
```

Returns

the first character of s$ repeated n times (0-32767 characters).

Examples

A line of dashes:

```
Print String$("-", 40)
Wait Key
```

A text bar chart:

```
For I=1 To 6
    Read N$, V
    Print N$; " "; String$("#", V)
Next I
Wait Key
Data "Mon", 5, "Tue", 12, "Wed", 8, "Thu", 20, "Fri", 15, "Sat", 3
```

See

Space$

T

Tcp Close

Command · VMOS 0.20

Syntax

```
Tcp Close c
```

Params

`c`	the connection, 1 or 2 (from Tcp Open)

Action

Closes connection c. A connection that is not open is let be. The program's end closes its connections.

Examples

Connect and close again:

```
C = Tcp Open("example.com", 80)
Print "Connection "; C
If C > 0
    Tcp Close C
End If
Wait Key
```

See

Tcp Open

Tcp Connected

Function · VMOS 0.20

Syntax

```
Tcp Connected(c)
```

Params

`c`	the connection, 1 or 2

Returns

True (-1) while connection c is open; False (0) once the server has closed it, or when it is not open. What the server sent before it closed is still there for Tcp Recv$( ). Not in an Every procedure.

Examples

Read a whole web page: until the server closes, then once more:

```
C = Tcp Open("example.com", 80)
If C = 0
    Print "No connection"
Else
    Tcp Send C, "GET / HTTP/1.0" + Chr$(13) + Chr$(10) + "Host: example.com"
+ Chr$(13) + Chr$(10) + Chr$(13) + Chr$(10)
    A$ = ""
    Repeat
        A$ = A$ + Tcp Recv$(C)
        Wait Vbl
    Until Not Tcp Connected(C)
    A$ = A$ + Tcp Recv$(C)
    Tcp Close C
    Print Left$(A$, 15)
    Print Len(A$); " characters"
End If
Wait Key
```

See

Tcp Recv$, Tcp Open

Tcp Open

Function · VMOS 0.20

Syntax

```
Tcp Open(host$, port)
```

Params

`host$`	a name or an address
`port`	1-65535

Returns

the connection, 1 or 2 (two can be open at a time); 0 when it could not connect: no such name, nothing listening on that port, or no answer within 5 seconds. Waits until the server answers; RUN/STOP stops the program while it waits. A client only: nothing can connect to the machine.

Examples

Save a high score on your own score server (its name and port are yours) and show its answer, one line:

```
C = Tcp Open("scores.example.com", 4000)
If C = 0
    Print "The score server did not answer"
Else
    Tcp Send C, "ACE 12345" + Chr$(10)
    R$ = ""
    T = Timer
```

```
    Repeat
        R$ = R$ + Tcp Recv$(C)
        Wait Vbl
    Until Instr(R$, Chr$(10)) > 0 or Timer - T > 250
    Tcp Close C
    Print "The server said: "; R$
End If
Wait Key
```

See

Tcp Send, Tcp Recv$, Tcp Close, Appendix M

Tcp Recv$

String function · VMOS 0.20

Syntax

```
Tcp Recv$(c)
```

Params

`c`	the connection, 1 or 2

Returns

what has arrived on connection c since the last Tcp Recv$( ), up to 2048 characters at a time; "" when nothing has, or the connection is not open. Never waits. The machine keeps 2048 characters for a connection: call it every frame while data comes.

Examples

A web page's first line:

```
C = Tcp Open("example.com", 80)
If C = 0
    Print "No connection"
Else
    Tcp Send C, "GET / HTTP/1.0" + Chr$(13) + Chr$(10) + "Host: example.com"
+ Chr$(13) + Chr$(10) + Chr$(13) + Chr$(10)
    A$ = ""
    Repeat
        A$ = A$ + Tcp Recv$(C)
        Wait Vbl
    Until Not Tcp Connected(C)
    A$ = A$ + Tcp Recv$(C)
    Tcp Close C
    Print Left$(A$, 15)
    Print Len(A$); " characters"
End If
Wait Key
```

See

Tcp Connected, Tcp Send

Tcp Send

Command · VMOS 0.20

Syntax

```
Tcp Send c, s$
```

Params

`c`	the connection, 1 or 2
`s$`	what to send, any length

Action

Sends s$ on connection c, 536 characters a block, each acknowledged by the server before the next; it returns when the last block has gone. Stops the program with an error if the connection is not open or closes meanwhile.

Examples

Ask a web server for a page:

```
C = Tcp Open("example.com", 80)
If C = 0
    Print "No connection"
Else
    Tcp Send C, "GET / HTTP/1.0" + Chr$(13) + Chr$(10) + "Host: example.com"
+ Chr$(13) + Chr$(10) + Chr$(13) + Chr$(10)
    A$ = ""
    Repeat
        A$ = A$ + Tcp Recv$(C)
        Wait Vbl
    Until Not Tcp Connected(C)
    A$ = A$ + Tcp Recv$(C)
    Tcp Close C
    Print Left$(A$, 15)
    Print Len(A$); " characters"
End If
Wait Key
```

See

Tcp Open, Tcp Recv$

Td Angle

Command · VMOS 0.10

Syntax

```
Td Angle n, a, b, c
```

Params

`n`	object 0-255

`a`	rotation about X (tilts it: the top moves away from you)
`b`	rotation about Y (turns it left/right)
`c`	rotation about Z (rolls it)

Action

Sets object n's rotation, in degrees or radians as set by Degree/Radian. The angles are absolute (not added to the current ones) and applied in the order Td Order sets (default c, then a, then b).

Examples

A cube turning about all three axes (run from EXAMPLES):

```
Degree
Td Load "BOXC.OBJ"
Td Object 1, "BOXC", 192, 112, 0, 0, 0, 0
Td Size 1, 90
Repeat
    Td Cls
    Td Angle 1, A, A*2, A/2
    Td Redraw
    Wait Vbl
    Add A, 2
Until Inkey$<>""
```

The same cube at fixed angles: front, tilted, turned:

```
Degree
Td Load "BOXC.OBJ"
For I=0 To 2
    Td Object I, "BOXC", 90+I*100, 112, 0, 0, 0, 0
    Td Size I, 60
Next I
Td Angle 1, 30, 0, 0
Td Angle 2, 30, 45, 0
Td Redraw
Wait Key
```

See

Td Object, Td Order, appendix K

Td Camera

Command · VMOS 0.10

Syntax

```
Td Camera x, y, z, a, b, c
Td Camera Off
```

Params

`x,y,z`	camera position in world coordinates (Y up)
`a`	pitch: positive looks down
`b`	yaw: positive turns right
`c`	roll

Action

From now on objects are drawn as seen by the camera: their x, y, z are world coordinates, not screen positions. Td Camera Off returns to screen-positioned objects. Use it with Td Clip, so objects passing beside or behind the camera are clipped instead of dropped.

Examples

A camera circling a floor of crate tiles with three crates on it (run from EXAMPLES):

```
Degree
Td Load "FLOOR8.OBJ"
Td Load "BOXT.OBJ"
Screen Open 1
Screen Open 2
Td Clip 32, 12 To 351, 211
Td Object 1, "FLOOR8", 0, 0, 0, 0, 0, 0
Td Size 1, 16
Td Object 2, "BOXT", -2, 0.5, 3, 0, 20, 0
Td Object 3, "BOXT", 2, 0.5, 4, 0, 50, 0
Td Object 4, "BOXT", 0, 0.5, -2, 0, 0, 0
S=1
Repeat
    Screen S
    Td Cls
    Td Camera Sin(A)*6, 1.5, -Cos(A)*6, 12, -A, 0
    Td Redraw
    Screen To Front S
    Wait Vbl
    S=3-S
    Add A, 1
Until Inkey$<>""
```

Walk forward toward a crate:

```
Td Load "BOXT.OBJ"
Td Clip 32, 12 To 351, 211
Td Object 1, "BOXT", 0, 0, 0, 0, 0, 0
For Z#=-10 To -2 Step 0.1
    Td Cls
    Td Camera 0, 0, Z#, 0, 0, 0
    Td Redraw
    Wait Vbl
Next Z#
Wait Key
```

See

Td Clip, appendix K

Td Center

Command · VMOS 0.12

Syntax

```
Td Center x, y
Td Center Off
```

Params

`x,y`	the view's center point, in screen coordinates

Action

Sets the projection center (the point straight ahead of the eye) to x, y, independent of the Td Clip window. Default: the center of the Td Clip window, or 192, 112 without Td Clip; Td Center Off restores the default. Without a camera, object x, y are still screen coordinates.

Examples

The same two cubes seen with the center in the middle, then low on the screen (perspective changes):

```
Degree
Td Load "BOXC.OBJ"
Td Object 1, "BOXC", 120, 112, 100, 30, 30, 0
Td Size 1, 70
Td Object 2, "BOXC", 264, 112, 100, 30, 30, 0
Td Size 2, 70
Td Redraw
Wait Key
Cls
Td Center 192, 211
Td Redraw
Wait Key
```

Move the projection center with the mouse, while a cube stays in the middle:

```
Degree
Td Load "BOXC.OBJ"
Td Object 1, "BOXC", 192, 112, 100, 20, 20, 0
Td Size 1, 80
Repeat
    Td Center X Mouse, Y Mouse
    Td Cls
    Td Redraw
    Wait Vbl
Until Inkey$<>""
```

See

Td Clip

Command · VMOS 0.10

Syntax

```
Td Clip x1, y1 To x2, y2
Td Clip Off
```

Params

`x1,y1`	one corner of the window, in screen coordinates
`x2,y2`	the opposite corner (both inclusive)

Action

Restricts 3D drawing to the window and clips triangles at the near plane, so objects can leave the window's edges and come right up to the viewer. The window's center is the projection center (unless Td Center is set); the window is clamped to the 384 x 224 display (the Z buffer's size). Without Td Clip, 3D is still limited to the 384 x 224 display, but triangles reaching behind the eye are dropped.

Examples

A crate sliding through a window; the parts outside are cut off (run from EXAMPLES):

```
Degree
Td Load "BOXT.OBJ"
Ink 1
Box 79, 49 To 301, 191
Td Clip 80, 50 To 300, 190
Td Object 1, "BOXT", 0, 120, 0, 20, 0, 0
Td Size 1, 80
Repeat
    Td Cls
    Td Move 1, 192+Sin(A)*170, 120, 0
    Td Angle 1, 20, A*2, 0
    Td Redraw
    Wait Vbl
    Add A, 2
Until Inkey$<>""
```

Two clip windows: one cube drawn into each half of the screen:

```
Degree
Td Load "BOXC.OBJ"
Td Object 1, "BOXC", 0, 0, 0, 0, 0, 0
Td Size 1, 90
Repeat
    Td Clip 32, 12 To 191, 211
    Td Cls
    Td Move 1, 111, 112, 0
```

```
    Td Angle 1, A, A, 0
    Td Redraw
    Td Clip 192, 12 To 351, 211
    Td Cls
    Td Move 1, 271, 112, 0
    Td Angle 1, -A, A*2, 0
    Td Redraw
    Wait Vbl
    Add A, 3
Until Inkey$<>""
```

See

Td Cls, Td Camera, appendix K

Td Cls

Command · VMOS 0.10

Syntax

```
Td Cls
```

Action

Clears the 3D area to the Paper color: the Td Clip window, or the whole 384 x 224 display area of the current screen without Td Clip. Text and drawings outside the window stay.

Examples

Clear only the 3D window each frame; the text outside stays:

```
Degree
Td Load "BOXC.OBJ"
Print "the text above the window stays"
Td Clip 100, 60 To 284, 200
Paper 6
Td Object 1, "BOXC", 192, 130, 0, 0, 0, 0
Td Size 1, 80
Repeat
    Td Cls
    Td Angle 1, A, A, 0
    Td Redraw
    Wait Vbl
    Add A, 3
Until Inkey$<>""
```

Without Td Clip, Td Cls clears the whole display area:

```
Degree
Td Load "BOXC.OBJ"
Td Object 1, "BOXC", 192, 112, 0, 0, 0, 0
Td Size 1, 80
Paper 2
For A=0 To 360 Step 4
    Td Cls
```

```
    Td Angle 1, A, A, 0
    Td Redraw
    Wait Vbl
Next A
Wait Key
```

See

Td Redraw, Td Clip

Td Color

Command · VMOS 0.12

Syntax

```
Td Color n, c
Td Color n, m$, c
Td Color n Off
Td Color n, m$ Off
```

Params

`n`	object 0-255
`m$`	a material, by its newmtl name in the .mtl (case-insensitive)
`c`	palette entry 0-255

Action

Draws object n's flat-shaded faces in color c, unlit. Change that palette entry with Color to fade the object, or with Copper Color to vary it down the screen. Without m$ it applies to all the object's flat faces; with m$ only to that material's faces, so one object can have several unlit colors. Textured faces keep their texture. Off restores the materials' own colors, lit from the upper left. Td Object resets all of them.

Examples

A cube in one unlit color that fades in and out (run from EXAMPLES):

```
Degree
Td Load "BOXC.OBJ"
Td Object 1, "BOXC", 192, 112, 0, 30, 30, 0
Td Size 1, 100
Td Color 1, 40
Repeat
    Color 40, 0, (Sin(A)+1)*120, 255
    Td Cls
    Td Redraw
    Wait Vbl
    Add A, 4
Until Inkey$<>""
```

Lit (default) and unlit side by side:

```
Degree
Td Load "BOXC.OBJ"
Td Object 1, "BOXC", 120, 112, 0, 30, 30, 0
Td Size 1, 80
Td Object 2, "BOXC", 264, 112, 0, 30, 30, 0
Td Size 2, 80
Td Color 2, 7
Td Redraw
Wait Key
```

See

Td Object, Color, appendix K

Td Cull

Command · VMOS 0.12

Syntax

```
Td Cull n On
Td Cull n Off
```

Params

`n`	object 0-255

Action

Off also draws object n's back-facing faces, so a flat object (a logo, text) shows its back side without listing every face twice in the .obj. Lit back faces are lit from their other side. On (the default; Td Object resets it) skips back-facing faces again. Closed objects such as a cube should keep it On: their back faces are hidden anyway.

Examples

A floor tile turning: with culling it disappears when seen from below; with Td Cull Off it does not (run from EXAMPLES):

```
Degree
Td Load "FLOOR1.OBJ"
Td Object 1, "FLOOR1", 120, 112, 0, 0, 0, 0
Td Size 1, 100
Td Object 2, "FLOOR1", 264, 112, 0, 0, 0, 0
Td Size 2, 100
Td Cull 2 Off
Repeat
    Td Cls
    Td Angle 1, A, 0, 0
    Td Angle 2, A, 0, 0
    Td Redraw
    Wait Vbl
    Add A, 2
Until Inkey$<>""
```

A flat floor tile spinning around Y with Td Cull Off: its back side shows too:

```
Degree
Td Load "FLOOR1.OBJ"
Td Object 1, "FLOOR1", 192, 112, 0, 90, 0, 0
Td Size 1, 140
Td Cull 1 Off
Repeat
    Td Cls
    Td Angle 1, 90, A, 0
    Td Redraw
    Wait Vbl
    Add A, 2
Until Inkey$<>""
```

See

Td Color, Td Object, appendix K

Td Focus

Command · VMOS 0.12

Syntax

```
Td Focus f
```

Params

`f`	eye distance in front of the screen, in pixels: 16 to 4095 (default 256)

Action

Sets the eye distance (the focal length). A larger value gives a narrower view: less perspective distortion as objects turn, and less size change as they approach. Without a camera, an object at z = 0 is drawn at 1 pixel per unit, and one at z = f at half that size.

Examples

The same turning cube with strong (64) and weak (1024) perspective:

```
Degree
Td Load "BOXC.OBJ"
Td Object 1, "BOXC", 192, 112, 0, 0, 0, 0
Td Size 1, 100
For F=1 To 2
    If F=1 Then Td Focus 64 Else Td Focus 1024
    For A=0 To 180 Step 2
        Td Cls
        Td Angle 1, 20, A, 0
        Td Redraw
        Wait Vbl
    Next A
Next F
Wait Key
```

Change the focus with the mouse's x position:

```
Degree
Td Load "BOXC.OBJ"
Td Object 1, "BOXC", 192, 112, 0, 0, 0, 0
Td Size 1, 100
Repeat
    Td Focus Max(16, (X Mouse-32)*8)
    Td Cls
    Td Angle 1, 20, A, 0
    Td Redraw
    Home
    Print "focus: "; Max(16, (X Mouse-32)*8); "    "
    Wait Vbl
    Add A, 2
Until Inkey$<>""
```

See

Td Center, Td Object, appendix K

Td Kill

Command · VMOS 0.10

Syntax

```
Td Kill n
```

Params

`n`	object 0-255

Action

Removes object n. Its loaded definition stays, so Td Object can use it again.

Examples

Three cubes; one disappears after a second:

```
Td Load "BOXC.OBJ"
For I=0 To 2
    Td Object I, "BOXC", 90+I*100, 112, 0, 30, 30, 0
    Td Size I, 60
Next I
Td Redraw
Wait 50
Td Kill 1
Td Cls
Td Redraw
Wait Key
```

Remove five cubes one by one, one per second:

```
Degree
Td Load "BOXC.OBJ"
For I=0 To 4
    Td Object I, "BOXC", 60+I*66, 112, 0, 30, I*20, 0
    Td Size I, 50
Next I
For I=0 To 4
    Td Cls
    Td Redraw
    Wait 50
    Td Kill I
Next I
Td Cls
Wait Key
```

See

Td Object

Td Load

Command · VMOS 0.10

Syntax

```
Td Load "BOX.OBJ"
```

Params

`file`	a Wavefront .obj file in the program's folder (1 MB max)

Action

Loads an object definition, named after the file without .OBJ ("BOX"). It reads vertices (v), texture coordinates (vt) and faces (f, any number of corners); mtllib names the .mtl file, from which it reads newmtl, Kd (the color) and map_Kd (the texture). A texture is an 8-bit BMP up to 256 x 256; its palette entries 16-255 are copied to the palette, as Load Images does. Textures whose width and height are powers of two (2, 4, 8 ... 256, not necessarily equal) repeat when texture coordinates go beyond 0..1 (vt 0 to 4: four times in each direction); "map_Kd -clamp on CRATE.BMP" in the .mtl disables the repeat. Other sizes do not repeat: keep their texture coordinates within 0..1. "map_Kd VIC20" is no file: that face shows the VIC-20's screen (Vic20 On, appendix L); "map_Kd C64" the C64's (C64 On; it is two textures, its left and right half, as the texture cache is 256 wide, and Td Load cuts every face of it where u = 0.5); Vic20 Border and C64 Border add as much of their border as you ask. Up to 32 definitions and 64 textures; loading the same name again replaces it. CTRL+B packs the .obj, its .mtl and their textures into the release.

Examples

Load a colored cube and a textured crate (run from EXAMPLES):

```
Degree
Td Load "BOXC.OBJ"
```

```
Td Load "BOXT.OBJ"
Td Object 1, "BOXC", 120, 112, 0, 30, 30, 0
Td Size 1, 80
Td Object 2, "BOXT", 264, 112, 0, 30, 30, 0
Td Size 2, 80
Td Redraw
Wait Key
```

A sphere of 960 triangles, turning:

```
Degree
Td Load "SPHERE.OBJ"
Td Object 1, "SPHERE", 192, 112, 0, 0, 0, 0
Td Size 1, 160
Repeat
    Td Cls
    Td Angle 1, A, A*2, 0
    Td Redraw
    Wait Vbl
    Add A, 2
Until Inkey$<>""
```

See

Td Object, appendix K

Td Move

Command · VMOS 0.10

Syntax

```
Td Move n, x, y, z
```

Params

`n`	object 0-255
`x,y,z`	new position (as in Td Object)

Action

Moves object n to x, y, z.

Examples

A cube moving in a circle and coming closer and further (run from EXAMPLES):

```
Degree
Td Load "BOXC.OBJ"
Td Object 1, "BOXC", 192, 112, 0, 30, 30, 0
Td Size 1, 60
Repeat
    Td Cls
    Td Move 1, 192+Cos(A)*100, 112+Sin(A)*50, 150+Sin(A*2)*140
    Td Redraw
```

```
    Wait Vbl
    Add A, 2
Until Inkey$<>""
```

Move a crate with the joystick (run from EXAMPLES):

```
Degree
Td Load "BOXT.OBJ"
Td Object 1, "BOXT", 192, 112, 0, 20, 30, 0
Td Size 1, 60
X=192
Y=112
Z=0
Repeat
    If Jleft(2) Then Add X, -3
    If Jright(2) Then Add X, 3
    If Jup(2) Then Add Z, 8
    If Jdown(2) Then Add Z, -8
    Td Move 1, X, Y, Z
    Td Cls
    Td Redraw
    Wait Vbl
Until Inkey$<>""
```

See

Td Move Rel, Td Object

Td Move Rel

Command · VMOS 0.10

Syntax

```
Td Move Rel n, dx, dy, dz
```

Params

`n`	object 0-255
`dx`	added to x
`dy`	added to y
`dz`	added to z

Action

Moves object n by dx, dy, dz relative to its current position.

Examples

A cube flying away into the distance:

```
Td Load "BOXC.OBJ"
Td Object 1, "BOXC", 192, 112, 0, 30, 30, 0
Td Size 1, 80
```

```
For I=1 To 150
    Td Cls
    Td Move Rel 1, 0, 0, 10
    Td Redraw
    Wait Vbl
Next I
Wait Key
```

A cube that bounces between the screen edges with relative moves:

```
Degree
Td Load "BOXC.OBJ"
Td Object 1, "BOXC", 192, 112, 0, 30, 30, 0
Td Size 1, 50
X=192
DX=3
Repeat
    If X<60 or X>324 Then DX=-DX
    Add X, DX
    Td Move Rel 1, DX, 0, 0
    Td Cls
    Td Redraw
    Wait Vbl
Until Inkey$<>""
```

See

Td Move

Td Moving

Function · VMOS 0.15

Syntax

```
Td Moving(n)
```

Returns

True (-1) while object n's path (Td Path) is running; False (0) when it has ended, was stopped, or none was set.

Examples

Wait until a path has finished:

```
Td Load "BOXC.OBJ"
Td Object 1, "BOXC", 60, 112, 0, 0, 0, 0
Td Size 1, 50
Td Path 1, "MT 260,0,0,0,360,0,100"
Repeat
    Td Cls
    Td Redraw
    Wait Vbl
Until Not Td Moving(1)
```

```
Print "done"
Wait Key
```

Start the next path when the last one has ended: a cube that walks a square:

```
Td Load "BOXC.OBJ"
Td Object 1, "BOXC", 100, 60, 0, 0, 0, 0
Td Size 1, 40
Dim P$(3)
P$(0)="M 180,0,0,50"
P$(1)="M 0,100,0,50"
P$(2)="M -180,0,0,50"
P$(3)="M 0,-100,0,50"
Repeat
    If Not Td Moving(1)
        Td Path 1, P$(N mod 4)
        Inc N
    End If
    Td Cls
    Td Redraw
    Wait Vbl
Until Inkey$<>""
```

See

Td Path

Td Object

Command · VMOS 0.10

Syntax

```
Td Object n, "BOX", x, y, z, a, b, c
```

Params

`n`	object 0-255
`name`	a loaded definition (Td Load)
`x,y,z`	without a camera: x, y = the object's center in screen coordinates, z = distance behind the screen (at z 0, one .obj unit is one pixel); with a camera: world coordinates
`a,b,c`	rotation angles (Td Angle)

Action

Creates object n from a definition, at its own size (Td Size scales it). Td Redraw draws all objects.

Examples

One cube in the middle of the screen (run from EXAMPLES):

```
Degree
Td Load "BOXC.OBJ"
Td Object 1, "BOXC", 192, 112, 0, 20, 30, 0
Td Size 1, 100
Td Redraw
Wait Key
```

A row of cubes going into the distance:

```
Degree
Td Load "BOXC.OBJ"
For I=0 To 7
    Td Object I, "BOXC", 192, 140, I*150, 20, I*10, 0
    Td Size I, 60
Next I
Td Redraw
Wait Key
```

See

Td Load, Td Size, Td Redraw, appendix K

Td Order

Command · VMOS 0.12

Syntax

```
Td Order n, o$
```

Params

`n`	object 0-255
`o$`	"X", "Y" and "Z" once each, the first applied first (default "ZXY")

Action

Sets the order in which object n's three rotations (Td Angle: a about X, b about Y, c about Z) are applied. "ZYX" rotates about Z first, then Y, then X, as many old demos did. Td Object resets it to "ZXY".

Examples

The same angles in two orders give different results:

```
Degree
Td Load "BOXC.OBJ"
Td Object 1, "BOXC", 120, 112, 0, 40, 40, 40
Td Size 1, 80
Td Object 2, "BOXC", 264, 112, 0, 40, 40, 40
Td Size 2, 80
Td Order 2, "XYZ"
Td Redraw
Wait Key
```

Turning an object with each of the orders (watch how the same angles move it):

```
Degree
Td Load "BOXC.OBJ"
Td Object 1, "BOXC", 192, 112, 0, 0, 0, 0
Td Size 1, 90
For O=0 To 2
    Read O$
    Td Order 1, O$
    For A=0 To 90 Step 3
        Td Cls
        Td Angle 1, A, A, A
        Td Redraw
        Home
        Print O$
        Wait Vbl
    Next A
Next O
Wait Key
Data "ZXY", "XYZ", "YZX"
```

See

Td Angle, Td Object, appendix K

Td Path

Command · VMOS 0.15

Syntax

```
Td Path n, "steps"
Td Path n, Play "file" [, scale [, times]]
Td Path n Off
```

Params

`n`	object (Td Object)
`steps`	the script: steps separated by spaces (below)
`file`	a recorded path (Path Record)
`scale`	scale factor for the recording (default 1; 0.5 = half size)
`times`	number of plays (default 1; 0 = loop forever)

Step	What it does
`M x,y,z,n`	move by x, y, z over n frames (Bob/Sprite Path: `M x,y,n`, pixels)
`T a,b,c,n`	rotate by a, b, c degrees over n frames (Td Path only)
`MT x,y,z,a,b,c,n`	move and rotate together (Td Path only)
`W n`	wait n frames

Step	What it does
`L`	restart from the first step, forever

Action

Moves and rotates object n automatically, one step per frame, like AMOS's AMAL does for sprites; the program only has to draw (Td Redraw). Values may have fractions (0.5); frame counts are integers. Moves are relative to the object's current position, so Td Move and Td Angle still work; each step adds up to exactly its total. Paths run on the frame counter, not on the drawing: Wait Vbl, Td Redraw and the reading functions (Td Moving, Td X ...) bring every path up to the current frame, so a program drawing at 25 fps still moves its objects on time. The script is parsed when Td Path executes; an error reports the line. Play first moves the object to where the recording started (x, y), so it plays where it was drawn on screen; with Td Camera on, screen positions are not world positions, so it plays from where the object is. To play a recording elsewhere, Td Move Rel the object after Play. A new Td Path replaces the object's previous one; Off stops it where it is. Limits: 64 paths at once, 64 steps per script, 16 recordings loaded.

Examples

A cube that slides right while turning, then comes back, forever (run from EXAMPLES):

```
Td Load "BOXC.OBJ"
Td Object 1, "BOXC", 80, 112, 0, 20, 0, 0
Td Size 1, 60
Td Path 1, "MT 220,0,0,0,180,0,100 W 20 MT -220,0,0,0,180,0,100 W 20 L"
Repeat
    Td Cls
    Td Redraw
    Wait Vbl
Until Inkey$<>""
```

An object flying in from the distance, spinning, then flying away:

```
Td Load "BOXC.OBJ"
Td Object 1, "BOXC", 192, 112, 2000, 0, 0, 0
Td Size 1, 80
Td Path 1, "MT 0,0,-2000,90,180,0,100 T 0,360,0,100 MT 0,0,2000,90,180,0,100"
Repeat
    Td Cls
    Td Redraw
    Wait Vbl
Until Not Td Moving(1)
```

Play a recording made with Path Record, at half size, forever:

```
Td Load "BOXC.OBJ"
Td Object 1, "BOXC", 192, 112, 0, 0, 0, 0
Td Size 1, 40
Td Path 1, Play "SPIN.PTH", 0.5, 0
```

```
Repeat
    Td Cls
    Td Redraw
    Wait Vbl
Until Inkey$<>""
```

See

Td Moving, Td X, Td Y, Td Z, Path Record, Bob Path

Td Redraw

Command · VMOS 0.10

Syntax

```
Td Redraw
```

Action

Draws all objects into the current screen. The hardware Z buffer keeps the nearest face per pixel, so intersecting faces are drawn correctly; textured faces are perspective-correct, flat faces are shaded by a light from the upper left. Pixels where no face is drawn are left unchanged: clear first with Td Cls (or Cls, Bar). For smooth animation draw into a hidden screen and display it when done (Screen To Front).

Examples

Draw once:

```
Td Load "BOXT.OBJ"
Td Object 1, "BOXT", 192, 112, 0, 30, 30, 0
Td Size 1, 100
Td Redraw
Wait Key
```

Smooth animation with two screens: draw one while the other is displayed (run from EXAMPLES):

```
Degree
Td Load "BOXT.OBJ"
Screen Open 1
Screen Open 2
Td Object 1, "BOXT", 192, 112, 0, 0, 0, 0
Td Size 1, 100
S=1
Repeat
    Screen S
    Td Cls
    Td Angle 1, A, A*2, 0
    Td Redraw
    Screen To Front S
    Wait Vbl
    S=3-S
```

```
    Add A, 2
Until Inkey$<>""
```

Without clearing, every frame stays: a trail of cubes:

```
Degree
Td Load "BOXC.OBJ"
Td Object 1, "BOXC", 192, 112, 0, 0, 0, 0
Td Size 1, 30
For A=0 To 360 Step 10
    Td Move 1, 192+Cos(A)*120, 112+Sin(A)*70, 0
    Td Angle 1, A, A, 0
    Td Redraw
    Wait Vbl
Next A
Wait Key
```

See

Td Cls, appendix K

Td Size

Command · VMOS 0.10

Syntax

```
Td Size n, s
```

Params

`n`	object 0-255
`s`	the size of its largest dimension: pixels without a camera, world units with one

Action

Scales object n so that its largest dimension is s.

Examples

Five cubes of growing size:

```
Degree
Td Load "BOXC.OBJ"
For I=0 To 4
    Td Object I, "BOXC", 50+I*70, 112, 0, 30, 30, 0
    Td Size I, 20+I*12
Next I
Td Redraw
Wait Key
```

A pulsing cube:

```
Degree
Td Load "BOXC.OBJ"
Td Object 1, "BOXC", 192, 112, 0, 30, 30, 0
```

```
Repeat
    Td Size 1, 90+Sin(A)*40
    Td Cls
    Td Redraw
    Wait Vbl
    Add A, 5
Until Inkey$<>""
```

See

Td Object

Td X, Td Y, Td Z

Function · VMOS 0.15

Syntax

```
Td X(n)
Td Y(n)
Td Z(n)
```

Returns

object n's current position (fixed-point), with its path brought up to the current frame.

Examples

Follow a moving object's position:

```
Td Load "BOXC.OBJ"
Td Object 1, "BOXC", 60, 112, 0, 0, 0, 0
Td Size 1, 40
Td Path 1, "M 260,0,500,150"
Repeat
    Td Cls
    Td Redraw
    Locate 0, 0
    Print Td X(1); "  "; Td Z(1); "   "
    Wait Vbl
Until Not Td Moving(1)
Wait Key
```

Darken an object as it moves away (Td Color + its distance):

```
Td Load "BOXC.OBJ"
Td Object 1, "BOXC", 192, 112, 0, 30, 30, 0
Td Size 1, 80
Td Color 1, 40
Td Path 1, "M 0,0,1000,100 M 0,0,-1000,100 L"
Repeat
    L=255-Td Z(1)/4
    Color 40, L, L, 0
    Td Cls
    Td Redraw
```

```
    Wait Vbl
Until Inkey$<>""
```

See

Td Path, Td Move

Tempo

Command · VMOS 0.7

Syntax

```
Tempo t
```

Params

`t`	1-255; a whole note lasts 24 / t seconds (default 8)

Action

Sets the speed of Play for all voices, immediately.

Examples

The same scale slow and fast:

```
Tempo 10
Play "O4QCDEFGABO5C"
Wait 200
Tempo 40
Play "O4QCDEFGABO5C"
Wait 60
```

A tune that speeds up while it plays:

```
Play "O4QCEGECEGECEGECEGECEGECEGECEGE"
For T=10 To 60
    Tempo T
    Wait 4
Next T
Play
```

See

Play

Text

Command · VMOS 0.3

Syntax

```
Text x, y, s$
```

Params

`x,y`	top-left corner of the first character, any pixel position (appendix C)

`s$`	the text

Action

Draws s$ in the 8 x 8 font, Pen on Paper, at any pixel position of the current screen, inside or outside the text window, including under the border. Clipped to the screen. The text cursor does not move.

Examples

Text at pixel positions, not character cells:

```
Text 37, 50, "five pixels in"
Text 100, 103, "anywhere"
Wait Key
```

A status line over the border area:

```
Border Dims 0, 0, 0, 0
Text 0, 1, "SCORE 001250"
Text 300, 214, "LIVES 3"
Wait Key
```

Text moving smoothly across the screen, one pixel at a time:

```
For X=351 To 32 Step -1
    Text X, 104, "SMOOTH TEXT  "
    Wait Vbl
Next X
Wait Key
```

See

Print, Window, Pen, Paper

Type

Command · VMOS 0.3

Syntax

```
Type NAME
   field [, field ...]
   ...
End Type
```

Params

`NAME`	the type name
`field`	a name (with $, # or ! like a variable), a fixed-size array `VX#(99)`, `M(3, 3)`, or a record of an earlier Type `POS As VEC3`, `PTS(9) As VEC3`; on as many lines as needed, up to 256 fields

Action

Defines a record type: a group of named values, like a struct in C. Dim creates one record or an array of them, and E(i).F or HERO.F accesses a field anywhere a variable can be used: expressions, Print, assignments, Inc, Add, Input, Read. Nested records: E(i).POS.X#, E(i).PTS(2).Y#; array fields take an index: E(i).VX#(5). Each numeric or string field takes 4 bytes. A Type must be defined before its first use, outside any procedure; up to 256 Types.

Examples

A record type for enemies, in an array:

```
Type ENEMY
    X, Y, HP
    NAME$
End Type
Dim E(3) As ENEMY
For I=0 To 3
    E(I).X=50+I*80
    E(I).HP=100-I*20
    Read E(I).NAME$
Next I
For I=0 To 3
    Print E(I).NAME$, E(I).X, E(I).HP
Next I
Wait Key
Data "orc", "troll", "bat", "rat"
```

Nested records and an array field:

```
Type VEC3
    X#, Y#, Z#
End Type
Type SHIP
    POS As VEC3
    TRAIL#(9)
End Type
Dim S As SHIP
S.POS.X#=1.5
S.TRAIL#(9)=0.25
Print S.POS.X#, S.TRAIL#(9)
Wait Key
```

See

Dim, As, End Type

Then

Keyword part · VMOS 0.1

Syntax

```
If c Then ...
```

See

If

Tile

Command · VMOS 0.3

Syntax

```
Tile x, y, t [, flip [, palette]]
```

Params

`x`	map column 0-63
`y`	map row 0-127 (0-63 with Double Buffer)
`t`	tile 0-1023 (0 = empty)
`flip`	1 = mirrored left-right, 2 = upside down, 3 = both; default 0
`palette`	0-7, added to Tiles Palette (default 0)

Action

Sets map cell x,y to tile t, in its own palette added to the layer's (Tiles Palette; 7 + 1 is 0). With Double Buffer it goes into the hidden map, which is displayed at the next Wait Vbl together with the bobs, so map changes never show half-done.

Examples

A row of tiles across the screen (run from EXAMPLES):

```
Load Tiles "TILES.BMP", 1
For X=0 To 63
    Tile X, 20, 1
Next X
Tiles On
Wait Key
```

A checkerboard of two tiles:

```
Load Tiles "TILES.BMP", 1
For Y=0 To 31
    For X=0 To 63
        Tile X, Y, 1+(X+Y) mod 2
    Next X
Next Y
Tiles On
Wait Key
```

The same tile in its four orientations:

```
Load Tiles "TILES.BMP", 1
For F=0 To 3
    Tile 10+F*2, 10, 3, F
Next F
```

```
Tiles On
Wait Key
```

See

Tile( ), Load Tiles, Tiles, Tile Scroll

Tile( )

Function · VMOS 0.3

Syntax

```
Tile(x, y)
```

Returns

the tile number at map cell x,y.

Examples

Read tiles back from the map:

```
Load Tiles "TILES.BMP", 1
Tile 5, 5, 3
Print Tile(5, 5)           ' 3
Print Tile(6, 5)           ' 0
Wait Key
```

Collision with the map: a dot falls until it lands on a row of tiles:

```
Load Tiles "TILES.BMP", 1
For X=0 To 63
    Tile X, 20, 2
Next X
Tiles On
X=100
Y=20
While Tile(X/8, (Y+1)/8)=0
    Inc Y
    Cls
    Plot X, Y
    Wait Vbl
Wend
Print "landed at Y = "; Y
Wait Key
```

See

Tile

Tile Scroll

Command · VMOS 0.3

Syntax

```
Tile Scroll x, y
```

Action

Scrolls the tile layer: map pixel x,y appears at the display's top-left (0,0, under the border). It wraps at 512 horizontally and 1024 vertically (512 with Double Buffer). The scroll position changes during vertical blank, so it never tears; with Double Buffer it changes at the Wait Vbl that shows the bobs, so tiles and bobs move together.

Examples

Scroll the tile layer sideways (run from EXAMPLES):

```
Load Tiles "TILES.BMP", 1
For Y=0 To 31
    For X=0 To 63
        Tile X, Y, Rnd(3)+1
    Next X
Next Y
Tiles On
Repeat
    Inc S
    Tile Scroll S and 511, 0
    Wait Vbl
Until Inkey$<>""
```

Scroll diagonally:

```
Load Tiles "TILES.BMP", 1
For Y=0 To 127
    For X=0 To 63
        Tile X, Y, 1+(X xor Y) mod 4
    Next X
Next Y
Tiles On
Repeat
    Inc S
    Tile Scroll S and 511, S and 1023
    Wait Vbl
Until Inkey$<>""
```

See

Tile, Tiles

Tiles

Command · VMOS 0.3

Syntax

```
Tiles On | Tiles Off
```

Action

Shows or hides the tile layer: 8 x 8 tiles in 256 colors on a map 64 wide and 128 high (64 high with Double Buffer, which uses the hardware's two maps as the two buffers), drawn and scrolled by hardware. It is turned off when the program ends.

Examples

Show and hide a tile pattern:

```
Load Tiles "TILES.BMP", 1
For Y=0 To 31
    For X=0 To 63
        Tile X, Y, 1+(X+Y) mod 4
    Next X
Next Y
Tiles On
Wait 50
Tiles Off
Wait 50
Tiles On
Wait Key
```

Toggle the tile layer with a key:

```
Load Tiles "TILES.BMP", 1
For Y=0 To 31
    For X=0 To 63
        Tile X, Y, 1+(X*Y) mod 4
    Next X
Next Y
Print "t = tiles on/off, q = quit"
SHOWN=False
Repeat
    K$=Inkey$
    If K$="t"
        SHOWN=Not SHOWN
        If SHOWN Then Tiles On Else Tiles Off
    End If
Until K$="q"
```

See

Tile, Tiles Back, Tiles Front, Load Tiles

Tiles Back

Command · VMOS 0.3

Syntax

```
Tiles Back
```

Action

Puts the tile layer behind the program's screen (the default). Where the screen is color 0 (Paper 0) the tiles show through; text, bobs and sprites appear on top of them.

Examples

Text on top of a tile background:

```
Load Tiles "TILES.BMP", 1
For Y=0 To 31
    For X=0 To 63
        Tile X, Y, 1+(X+Y) mod 4
    Next X
Next Y
Tiles On
Tiles Back
Paper 0
Print "text over the tiles"
Wait Key
```

Bobs moving over a tile background (the tiles are behind, the default):

```
Load Images "BALLS.BMP", 1, 16, 16
Load Tiles "TILES.BMP", 1
For Y=0 To 31
    For X=0 To 63
        Tile X, Y, 1+(X+Y) mod 4
    Next X
Next Y
Tiles On
Tiles Back
For X=32 To 336
    Bob 0, X, 100, 1
    Wait Vbl
Next X
Wait Key
```

See

Tiles Front, Paper

Tiles Front

Command · VMOS 0.3

Syntax

```
Tiles Front
```

Action

Puts the tile layer in front of the program's screen; the screen shows only where tiles are transparent (tile 0, or color 0 inside a tile).

Examples

A row of tiles in front of a drawing:

```
Load Tiles "TILES.BMP", 1
For X=0 To 63
```

```
    Tile X, 13, 2
Next X
Tiles On
Tiles Front
Ink 2
Bar 32, 90 To 351, 130
Wait Key
```

Switch the tiles between front and back every second:

```
Load Tiles "TILES.BMP", 1
For X=0 To 63
    Tile X, 13, 1
Next X
Tiles On
Ink 2
Bar 32, 90 To 351, 130
For I=1 To 6
    If I mod 2 Then Tiles Front Else Tiles Back
    Wait 50
Next I
Wait Key
```

See

Tiles Back

Tiles Palette

Command · VMOS 0.18

Syntax

```
Tiles Palette n
```

Params

`n` a palette, 0-7 (default 0)

Action

The tile layer shows its colors from palette n, at once. A tile's own palette (Tile) is added to it: tile palette 1 under Tiles Palette 2 is palette 3 (counting on from 7 to 0).

Examples

Bricks in palette 1, where their red is green; a key switches (run from EXAMPLES):

```
Load Tiles "TILES.BMP", 1
Palette 1
Color 2, 0, 200, 0
For X=0 To 63
    Tile X, 18, 1
Next X
Tiles On
Print "A key: Tiles Palette 1"
```

```
Wait Key
Tiles Palette 1
Wait Key
```

Two rows, the second with its own palette 1 added to the layer's 2: palettes 2 and 3 (run from EXAMPLES):

```
Load Tiles "TILES.BMP", 1
Palette 3
Color 2, 0, 255, 0
For X=0 To 63
    Tile X, 18, 1
    Tile X, 20, 1, 0, 1
Next X
Tiles On
Tiles Palette 2
Wait Key
```

See

Tile, Palette

Timer

Function · VMOS 0.1

Syntax

```
Timer           Timer = n
```

Returns

the number of frames (1/50 s) since the program started; Timer = n sets it.

Examples

Measure two seconds:

```
Timer=0
Wait 100
Print Timer                ' 100
Wait Key
```

A reaction test: how fast do you press a key?

```
Print "press a key when the screen turns red"
Wait 50+Rnd(150)
Paper 2
Cls
Timer=0
Wait Key
Print "your time: "; Timer*20; " ms"
Wait Key
```

Time how long a loop takes:

```
T=Timer
For I=1 To 100000
    X=X+1
Next I
Print "frames: "; Timer-T
Wait Key
```

See

Wait

To

Keyword part · VMOS 0.1

Syntax

```
For var = a To b
Draw x1, y1 To x2, y2
```

See

For

Track Load

Command · VMOS 0.8

Syntax

```
Track Load "SONG.MOD"
Track Load "SONG.PST"
Track Load "SONG.PSY"
Track Load "SONG.VGZ"
```

Params

`file` — a ProTracker MOD (4 channels, 31 samples: M.K., M!K!, FLT4, 4CHN), a PROSID TRACKER song (.PST), a PROSIDYM TRACKER song (.PSY, 0.21: with eight YM2151 tracks) or a YM2151 song (.VGM, or .VGZ: the same packed with gzip), 1 MB max, in the program's folder

Action

Loads a song into chip RAM, ready for Track Play; any playing song stops. The file type is detected from its contents, not its name. A MOD is converted to a PROSID song while loading (as the tracker imports one), so both formats play on the tracker's own player. A YM2151 song (VMOS 0.19) keeps only the YM2151's part: other chips in the file (SegaPCM drums, for one) are not played (appendix J). A file name given as a plain "string" is packed into a release (CTRL+B).

Examples

Load and play a MOD (run from EXAMPLES):

```
Track Load "MADNESS.MOD"
Track Play
```

```
Print "playing - press a key to stop"
Wait Key
Track Stop
```

Load a PROSID song:

```
Track Load "DEMO.PST"
Track Play
Wait Key
```

A YM2151 song, with a sample over it – the PCM voices stay free (run from EXAMPLES):

```
Load Sample "BOOM.WAV", 1
Track Load "YMSONG.VGM"
Track Play
Print "a key: BOOM, q quits"
Repeat
    K$=Inkey$
    If K$<>"" and K$<>"q" Then Sam Play 1
Until K$="q"
```

See

Track Play, appendix J

Track Loop

Command · VMOS 0.8

Syntax

```
Track Loop On
Track Loop Off
```

Action

On (the default): the song loops, as the tracker plays it. Off: Track Play plays the song once; it stops by itself where it would jump back to an earlier position (its end, or a backward jump). Loops inside a pattern (E6x) still play. A YM2151 song loops from its loop point, or from its start when it has none; Off plays it to its end.

Examples

Play a song once and end the program when it is over (run from EXAMPLES):

```
Track Load "MADNESS.MOD"
Track Loop Off
Track Play
Print "playing once..."
Repeat
    Wait Vbl
Until Not Track Playing
Print "the end"
Wait Key
```

Loop the song (the default) and stop it with a key:

```
Track Load "MADNESS.MOD"
Track Loop On
Track Play
Print "looping - press a key"
Wait Key
Track Stop
```

See

Track Play, Track Playing

Track Play

Command · VMOS 0.8

Syntax

```
Track Play [pos]
```

Params

`pos`	order position to start at: 0 (the default) to the song length - 1; a YM2151 song has one, 0

Action

Plays the loaded song in the background: the program continues immediately while the frame interrupt plays it. If a song is playing, it restarts at pos. While it plays it uses the four PCM voices, and a .PST with notes on its SID tracks uses the SID too (appendix J). A YM2151 song uses the YM2151 only and plays from the timer interrupt; it starts 2 frames after Track Play (the chip is reset first).

Examples

Start a song from its third position (run from EXAMPLES):

```
Track Load "MADNESS.MOD"
Track Play 2
Wait Key
```

Music while the program animates:

```
Track Load "MADNESS.MOD"
Track Play
Repeat
    Ink Rnd(15)+1
    Circle 192, 112, Rnd(100)
    Wait Vbl
Until Inkey$<>""
Track Stop
```

See

Track Load, Track Stop, Track Loop

Track Playing

Function · VMOS 0.14

Syntax

```
Track Playing
```

Returns

True (-1) while a song is playing; False (0) before Track Play, after Track Stop, and after a Track Loop Off song has reached its end. With Track Loop On it stays True until Track Stop.

Examples

Wait for the end of the song (run from EXAMPLES):

```
Track Load "MADNESS.MOD"
Track Loop Off
Track Play
Repeat
    Wait Vbl
Until Not Track Playing
Print "song finished"
Wait Key
```

Show "MUSIC" while a song is playing, toggled with a key:

```
Track Load "MADNESS.MOD"
Print "a key starts/stops, q quits"
Repeat
    K$=Inkey$
    If K$<>"" and K$<>"q"
        If Track Playing Then Track Stop Else Track Play
    End If
    Locate 0, 2
    If Track Playing Then Print "MUSIC" Else Print "     "
    Wait Vbl
Until K$="q"
```

See

Track Loop, Track Pos

Track Pos

Function · VMOS 0.14

Syntax

```
Track Pos
```

Returns

the order position being played (0 = the first, the same numbering Track Play uses); −1 if no song is playing. A YM2151 song is at 0 while it plays.

Examples

Show the song position while it plays (run from EXAMPLES):

```
Track Load "MADNESS.MOD"
Track Play
Repeat
    Home
    Print "position: "; Track Pos; "  "
    Wait Vbl
Until Inkey$<>""
```

Change the border color at each new position (sync effects to the music):

```
Track Load "MADNESS.MOD"
Track Play
LAST=-1
Repeat
    If Track Pos<>LAST
        LAST=Track Pos
        Border LAST mod 15+1
    End If
    Wait Vbl
Until Inkey$<>""
```

See

Track Play, Track Playing

Track Stop

Command · VMOS 0.8

Syntax

```
Track Stop
```

Action

Stops the song. Every voice it used is silenced and free again for Sam Play (and, after a .PST, for Play and Sound); a YM2151 song's chip goes quiet at once. The song stays loaded: Track Play starts it again. The song also stops when the program ends.

Examples

Stop and restart a song with keys:

```
Track Load "MADNESS.MOD"
Print "p = play, s = stop, q = quit"
Repeat
    K$=Inkey$
    If K$="p" Then Track Play
    If K$="s" Then Track Stop
Until K$="q"
```

Stop the music when a game ends:

```
Track Load "MADNESS.MOD"
Track Play
Print "game running - press a key for game over"
Wait Key
Track Stop
Print "GAME OVER"
Wait Key
```

See

Track Play

True

Function · VMOS 0.1

Syntax

`True`

Returns

-1 (all bits set), the value of a true comparison.

Examples

Comparisons are True or False:

```
Print True                ' -1
Print 5>3                 ' -1
Print 5<3                 ' 0
Wait Key
```

True as a flag, and a loop that runs forever until Exit:

```
RUNNING=True
While RUNNING
    Inc N
    If N=5 Then RUNNING=False
Wend
Print N                   ' 5
Do
    Inc M
    Exit If M=3
Loop
Print M                   ' 3
Wait Key
```

See

False

U

Udp Close

Command · VMOS 0.20

Syntax

```
Udp Close
```

Action

Closes Udp Open's port: packets sent to it are no longer kept. Nothing happens when none is open. The program's end closes it.

Examples

Open, then close:

```
Udp Open 6502
Print "Listening on 6502"
Wait Key
Udp Close
```

See

Udp Open

Udp Ip$

String function · VMOS 0.20

Syntax

```
Udp Ip$
```

Returns

the address of whoever sent the packet Udp Recv$ gave last; "" before the first.

Examples

An echo:

```
' whatever comes in on port 7 goes back to whoever sent it (a key ends)
Udp Open 7
Repeat
    R$ = Udp Recv$
    If R$ <> ""
        Print Udp Ip$; ":"; Udp Port; " "; R$
        Udp Send Udp Ip$, Udp Port, R$
    End If
    Wait Vbl
Until Inkey$ <> ""
```

See

Udp Port, Udp Recv$

Udp Open

Command · VMOS 0.20

Syntax

```
Udp Open port
```

Params

port	1-65535

Action

Makes port the program's UDP port: packets sent to this machine on it wait for Udp Recv$, up to 16 (more are dropped), and Udp Send sends from it. A second Udp Open closes the first port and drops what waited there.

Examples

Two machines, one joystick each:

```
' two VIXENs: each sends its joystick to the other (the other's address here)
OTHER$ = "192.168.5.70"
Udp Open 6502
Repeat
    Udp Send OTHER$, 6502, Str$(Joy(2))
    R$ = Udp Recv$
    If R$ <> ""
        Locate 0, 0
        Print "The other joystick: "; R$; "   "
    End If
    Wait Vbl
Until Inkey$ <> ""
```

See

Udp Send, Udp Recv$, Udp Close, Appendix M

Udp Port

Function · VMOS 0.20

Syntax

```
Udp Port
```

Returns

the port the packet Udp Recv$ gave last came from; 0 before the first.

Examples

An echo:

```
' whatever comes in on port 7 goes back to whoever sent it (a key ends)
Udp Open 7
Repeat
    R$ = Udp Recv$
    If R$ <> ""
        Print Udp Ip$; ":"; Udp Port; " "; R$
        Udp Send Udp Ip$, Udp Port, R$
```

```
    End If
    Wait Vbl
Until Inkey$ <> ""
```

See

Udp Ip$, Udp Recv$

Udp Recv$

String function · VMOS 0.20

Syntax

```
Udp Recv$
```

Returns

the oldest packet waiting on Udp Open's port, which it takes; "" when none waits. Never waits.

Examples

Two machines, one joystick each:

```
' two VIXENs: each sends its joystick to the other (the other's address here)
OTHER$ = "192.168.5.70"
Udp Open 6502
Repeat
    Udp Send OTHER$, 6502, Str$(Joy(2))
    R$ = Udp Recv$
    If R$ <> ""
        Locate 0, 0
        Print "The other joystick: "; R$; "   "
    End If
    Wait Vbl
Until Inkey$ <> ""
```

See

Udp Ip$, Udp Port, Udp Open

Udp Send

Command · VMOS 0.20

Syntax

```
Udp Send host$, port, s$
```

Params

`host$`	a name or an address
`port`	1-65535
`s$`	up to 1472 characters

Action

Sends s$ to host$ on port as one packet, from Udp Open's port (with none open, from port). A packet can be lost on the way and nothing says so: a game sends its state every frame, or the other side answers. The first packet to a new address waits for the network to find it, up to a second.

Examples

An echo:

```
' whatever comes in on port 7 goes back to whoever sent it (a key ends)
Udp Open 7
Repeat
    R$ = Udp Recv$
    If R$ <> ""
        Print Udp Ip$; ":"; Udp Port; " "; R$
        Udp Send Udp Ip$, Udp Port, R$
    End If
    Wait Vbl
Until Inkey$ <> ""
```

See

Udp Open, Udp Recv$, Appendix M

Until

Keyword part · VMOS 0.1

Syntax

```
Repeat ... Until c
```

See

Repeat

Upper$

String function · VMOS 0.1

Syntax

```
Upper$(s$)
```

Returns

s$ converted to upper case.

Examples

Upper case:

```
Print Upper$("hello world")      ' HELLO WORLD
Wait Key
```

Accept y or Y as yes:

```
Input "Quit (y/n)"; A$
If Upper$(A$)="Y" Then Print "bye" Else Print "staying"
Wait Key
```

See

Lower$

V

Val

Function · VMOS 0.1

Syntax

```
Val(s$)
```

Returns

the integer value of the number in s$; hex ($FF) and binary (%101) are accepted.

Examples

Strings to numbers:

```
Print Val("12")+1          ' 13
Print Val("$FF")           ' 255
Print Val("%101")          ' 5
Wait Key
```

Add up numbers typed as text:

```
Input "Numbers separated by commas"; A$
A$=A$+","
P=Instr(A$, ",")
While P>0
    Add T, Val(Left$(A$, P-1))
    A$=Mid$(A$, P+1)
    P=Instr(A$, ",")
Wend
Print "total: "; T
Wait Key
```

See

Str$

Vbl

Keyword part · VMOS 0.1

Syntax

```
Wait Vbl
```

See

Wait Vbl

Vic20 Border

Command · VMOS 0.24

Syntax

```
Vic20 Border l, r, t, b
```

Params

`l, r`	pixels of the VIC-20's border shown left and right of its screen (default 0)
`t, b`	pixels shown above and below it, together 0-72 (default 0)

Action

Sets how much of the VIC-20's border the Td texture VIC20 (map_Kd VIC20) shows round its screen, on each side, from the next Td Redraw on; it can change at any time. A side shows at most what the VIC-20's picture has there (with the VIC-20's own screen of 176 x 184, about 48 pixels left and 32 right); more is cut to that. Top and bottom together are at most 72 pixels, so the picture still fits the texture cache in one piece ("Top + bottom is 72 pixels at most"), but they need not be equal. A face keeps its texture coordinates 0..1, which then cover the screen and the border asked for; 0, 0, 0, 0 shows the screen alone. C64 Border does the same for the C64.

Examples

16 pixels of border all round, on the 176 x 184 face (run from EXAMPLES):

```
Cpu8 6502 Load "KERNAL.ROM", $E000
Cpu8 6502 Load "BASIC.ROM", $C000
Cpu8 6502 Load "CHAR.ROM", $8000
Vic20 On
Vic20 Border 16, 16, 16, 16
Td Load "VICSCR.OBJ"
Td Object 1, "VICSCR", 192, 112, 0, 0, 0, 0
Td Size 1, 184
Repeat
    Td Redraw
    Wait Vbl
Until Inkey$=Chr$(27)
```

Only the border above and below, more below than above:

```
Cpu8 6502 Load "KERNAL.ROM", $E000
Cpu8 6502 Load "BASIC.ROM", $C000
Cpu8 6502 Load "CHAR.ROM", $8000
Vic20 On
Vic20 Border 0, 0, 20, 40
Td Load "VICSCR.OBJ"
Td Object 1, "VICSCR", 192, 112, 0, 0, 0, 0
Td Size 1, 184
Repeat
    Td Redraw
```

```
    Wait Vbl
Until Inkey$=Chr$(27)
```

See

Vic20 On, C64 Border, appendix L

Vic20 Off

Command · VMOS 0.16

Syntax

```
Vic20 Off
```

Action

Stops the VIC-20: the 6502 back in reset, the VIC and the two VIAs off, its sound silent, the keyboard and the joystick the program's only. The texture VIC20 keeps the last picture. Vic20 On starts the VIC-20 again from its reset. The program's end switches it off too.

Examples

Three seconds of the VIC-20, then its last picture stays (run from EXAMPLES):

```
Cpu8 6502 Load "KERNAL.ROM", $E000
Cpu8 6502 Load "BASIC.ROM", $C000
Cpu8 6502 Load "CHAR.ROM", $8000
Vic20 On
Td Load "VICSCR.OBJ"
Td Object 1, "VICSCR", 192, 112, 0, 0, 0, 0
Td Size 1, 184
For I=1 To 150
    Td Redraw
    Wait Vbl
Next I
Vic20 Off
Print "off"
Wait Key
```

Off and on again: the VIC-20 restarts from its reset:

```
Cpu8 6502 Load "KERNAL.ROM", $E000
Cpu8 6502 Load "BASIC.ROM", $C000
Cpu8 6502 Load "CHAR.ROM", $8000
Td Load "VICSCR.OBJ"
Td Object 1, "VICSCR", 192, 112, 0, 0, 0, 0
Td Size 1, 184
For K=1 To 3
    Vic20 On
    For I=1 To 100
        Td Redraw
        Wait Vbl
    Next I
    Vic20 Off
```

```
    Wait 25
Next K
Wait Key
```

See

Vic20 On, appendix L

Vic20 On

Command · VMOS 0.16

Syntax

```
Vic20 On
```

Action

Starts the VIC-20 on the core's own chips: the 6502 runs from its reset vector ($FFFC) at the VIC-20's speed (1,108,405 Hz, on the VIC's clock), with the VIC and the two VIAs at $9000-$93FF and the memory Vic20 Ram fits. Load the ROMs first (Cpu8 6502 Load). With BASIC ($C000) and the KERNAL ($E000) loaded and no cartridge at $A000 that starts itself, it waits until BASIC shows READY. (at most 5 seconds). From then on the VIC's picture is the Td texture VIC20 (map_Kd VIC20 in a .mtl), new each Td Redraw; its sound plays with the machine's; palette entries 240-255 are its 16 colors (until the program ends; pictures loaded after it set 16-239 only); and the keyboard and joystick port 2 go to it too (the cursor keys and CTRL are also its joystick, RESTORE is its RESTORE, RUN/STOP still stops the VMOS program). Vic20 On again restarts it.

Examples

The VIC-20's READY. on a flat screen; type on it, ESC ends (run from EXAMPLES):

```
Cpu8 6502 Load "KERNAL.ROM", $E000
Cpu8 6502 Load "BASIC.ROM", $C000
Cpu8 6502 Load "CHAR.ROM", $8000
Vic20 On
Td Load "VICSCR.OBJ"
Td Object 1, "VICSCR", 192, 112, 0, 0, 0, 0
Td Size 1, 184
Repeat
    Td Redraw
    Wait Vbl
Until Inkey$=Chr$(27)
```

On a turning cube: VICBOX.MTL gives the front face map_Kd VIC20 and the other faces VICCRATE.BMP (CRATE.BMP on colors 16-19, as 240-255 are the VIC-20's):

```
Degree
Cpu8 6502 Load "KERNAL.ROM", $E000
Cpu8 6502 Load "BASIC.ROM", $C000
Cpu8 6502 Load "CHAR.ROM", $8000
Vic20 On
Td Load "VICBOX.OBJ"
```

```
Td Object 1, "VICBOX", 192, 112, 0, 0, 0, 0
Td Size 1, 120
Repeat
    Td Cls
    Td Angle 1, 20, A, 0
    Td Redraw
    Wait Vbl
    Add A, 1
Until Inkey$=Chr$(27)
```

See

Vic20 Ram, Vic20 Off, Vic20 Run, Vic20 Border, Cpu8 6502 Load, appendix L

Vic20 Ram

Command · VMOS 0.16

Syntax

```
Vic20 Ram None
Vic20 Ram 3k
Vic20 Ram 8k
Vic20 Ram 16k
Vic20 Ram 24k
Vic20 Ram Full
```

Params

Size	The VIC-20's expansion RAM	BASIC's bytes free
`None`	none: an unexpanded VIC-20 (default)	3583
`3k`	$0400-$0FFF	6655
`8k`	block 1, $2000-$3FFF	11775
`16k`	blocks 1-2, $2000-$5FFF	19967
`24k`	blocks 1-3, $2000-$7FFF	28159
`Full`	3K, blocks 1-3 and block 5 ($A000-$BFFF)	28159

Action

Fits the VIC-20's expansion RAM. Where nothing is fitted the 6502 reads what a VIC-20 reads there and its writes are lost, so the KERNAL's RAM test finds exactly this memory. Give it before Vic20 On (the KERNAL tests the RAM as it starts). A .crt or .rom loaded into one of these blocks stays read-only.

Examples

An 8K VIC-20: 11775 BYTES FREE (run from EXAMPLES):

```
Cpu8 6502 Load "KERNAL.ROM", $E000
Cpu8 6502 Load "BASIC.ROM", $C000
Cpu8 6502 Load "CHAR.ROM", $8000
```

```
Vic20 Ram 8k
Vic20 On
Td Load "VICSCR.OBJ"
Td Object 1, "VICSCR", 192, 112, 0, 0, 0, 0
Td Size 1, 184
Repeat
    Td Redraw
    Wait Vbl
Until Inkey$<>""
```

Three sizes in turn, the VIC-20 restarted for each:

```
Cpu8 6502 Load "KERNAL.ROM", $E000
Cpu8 6502 Load "BASIC.ROM", $C000
Cpu8 6502 Load "CHAR.ROM", $8000
Td Load "VICSCR.OBJ"
Td Object 1, "VICSCR", 192, 112, 0, 0, 0, 0
Td Size 1, 184
Vic20 Ram None
Vic20 On
LOOK[100]
Vic20 Ram 3k
Vic20 On
LOOK[100]
Vic20 Ram Full
Vic20 On
LOOK[100]
Wait Key
Procedure LOOK[F]
    For I=1 To F
        Td Redraw
        Wait Vbl
    Next I
End Proc
```

See

Vic20 On, appendix L

Vic20 Run

Command · VMOS 0.16

Syntax

```
Vic20 Run
```

Action

Starts the last .prg loaded (Cpu8 6502 Load) as BASIC's program, as LOAD and RUN would: holds the 6502 for a moment, moves the program to where BASIC starts if it was saved for another memory size (and links its lines again), sets BASIC's end of program after it, and types RUN. Use it after Vic20 On, which waits for READY. A program made for an expanded VIC-20 needs that Vic20 Ram first.

Examples

HELLO.PRG started on a flat screen (run from EXAMPLES):

```
Cpu8 6502 Load "KERNAL.ROM", $E000
Cpu8 6502 Load "BASIC.ROM", $C000
Cpu8 6502 Load "CHAR.ROM", $8000
Vic20 On
Cpu8 6502 Load "HELLO.PRG"
Vic20 Run
Td Load "VICSCR.OBJ"
Td Object 1, "VICSCR", 192, 112, 0, 0, 0, 0
Td Size 1, 184
Repeat
    Td Redraw
    Wait Vbl
Until Inkey$<>""
```

The same on an 8K VIC-20, where BASIC starts at $1201: the program saved at $1001 is moved there, then runs, on a turning cube:

```
Degree
Cpu8 6502 Load "KERNAL.ROM", $E000
Cpu8 6502 Load "BASIC.ROM", $C000
Cpu8 6502 Load "CHAR.ROM", $8000
Vic20 Ram 8k
Vic20 On
Cpu8 6502 Load "HELLO.PRG"
Vic20 Run
Td Load "VICBOX.OBJ"
Td Object 1, "VICBOX", 192, 112, 0, 0, 0, 0
Td Size 1, 120
Repeat
    Td Cls
    Td Angle 1, 20, A, 0
    Td Redraw
    Wait Vbl
    Add A, 1
Until Inkey$<>""
```

See

Cpu8 6502 Load, Vic20 On, appendix L

Volume

Command · VMOS 0.2

Syntax

```
Volume n
```

Params

`n` 0-63 (default 63)

Action

Sets the sample (PCM) volume, for sounds already playing and those started later. The SID has its own: Sid Volume.

Examples

A sample repeated at falling volume (run from EXAMPLES):

```
Load Sample "PLUCK.WAV", 1
For V=63 To 3 Step -10
    Volume V
    Sam Play 1
    Wait 25
Next V
Volume 63
```

Volume set by the mouse's y position while a sample loops:

```
Load Sample "PLUCK.WAV", 1
Sam Loop On
Sam Play 0, 1
Repeat
    Volume Max(0, Min(63, (211-Y Mouse)/3))
    Wait Vbl
Until Inkey$<>""
Sam Stop
```

See

Sam Play

W

Wait

Command · VMOS 0.1

Syntax

`Wait n`

Action

Waits n frames (Wait 50 = one second).

Examples

A countdown, one number per second:

```
For I=5 To 1 Step -1
    Print I
    Wait 50
```

```
Next I
Print "go!"
Wait Key
```

Slow text: one character every 5 frames:

```
T$="This text appears letter by letter."
For I=1 To Len(T$)
    Print Mid$(T$, I, 1);
    Wait 5
Next I
Wait Key
```

See

Wait Vbl, Wait Key, Timer

Wait Key

Command · VMOS 0.1

Syntax

```
Wait Key
```

Action

Waits for a key that produces a character (a letter, a digit, RETURN, SPACE, a cursor key) and consumes it, as Inkey$ would. SHIFT, CTRL and similar keys alone do not count.

Examples

Keep the result on screen until a key is pressed:

```
Print "Hello!"
Wait Key
```

Pages of text, a key between them:

```
For P=1 To 3
    Cls
    Print "page "; P; " of 3"
    Print "press a key"
    Wait Key
Next P
```

See

Inkey$

Wait Raster

Command · VMOS 0.3

Syntax

```
Wait Raster n
```

Params

`n` raster line 0-311

Action

Waits until just before the display starts line n (in the horizontal blank at the end of line n-1), so a register change made next takes effect from line n's first pixel. Used for changes partway down the picture. Called again for the same line, it waits for the next frame.

Examples

A red band in the border, made with the CPU (the copper does this without CPU time: Copper Border):

```
Repeat
    Wait Raster 100
    Border 2
    Wait Raster 120
    Border 0
Until Inkey$<>""
```

Several bands of color:

```
Repeat
    For B=0 To 7
        Wait Raster 40+B*20
        Border B+1
    Next B
    Wait Raster 200
    Border 0
Until Inkey$<>""
```

See

Raster, Wait Vbl

Wait Vbl

Command · VMOS 0.1

Syntax

```
Wait Vbl
```

Action

Waits for the next frame (1/50 s). Bobs are drawn here (with Double Buffer into the hidden buffer, which is then displayed), the copper list is handed over, and paths advance. Put one Wait Vbl in every animation loop.

Examples

A sprite moving one pixel per frame (run from EXAMPLES):

```
Load Images "SHIP.BMP", 1
For X=32 To 336
    Sprite 0, X, 104, 1
    Wait Vbl
```

```
Next X
Wait Key
```

Count frames for one second:

```
T=Timer
For I=1 To 50
    Wait Vbl
Next I
Print Timer-T              ' 50
Wait Key
```

See

Bob, Wait

Wend

Keyword part · VMOS 0.1

Syntax

```
While c ... Wend
```

See

While

While

Command · VMOS 0.1

Syntax

```
While c ... Wend
```

Action

Runs the lines while c is true. The condition is tested first, so the body may not run at all.

Examples

Count while below a limit:

```
While X<10
    Inc X
Wend
Print X                    ' 10
Wait Key
```

The body is skipped if the condition is false at the start:

```
N=100
While N<10
    Print "never printed"
Wend
Print "done"
Wait Key
```

Halve a number until it is below 1:

```
X#=100
While X#>=1
    Print X#
    X#=X#/2
Wend
Wait Key
```

See

Wend, Repeat, Do

Window

Command · VMOS 0.3

Syntax

```
Window x, y, w, h
```

Params

`x,y`	top-left corner of the window, in pixels (appendix C)
`w,h`	size in characters; the window must fit inside 512 x 256 pixels

Action

Sets the text window: Locate, the cursor, line wrap and scrolling all stay inside it, and the cursor moves to its top-left. Default: Window 32, 12, 40, 25, the 320 x 200 area inside the default border. Changing the border does not move the window. The screen is not cleared.

Examples

Print over the whole display:

```
Border Dims 0, 0, 0, 0
Window 0, 0, 48, 28
For I=1 To 28
    Print "line "; I; " of the full display"
Next I
Wait Key
```

A small window in the middle that scrolls on its own:

```
Ink 6
Bar 112, 60 To 271, 163
Window 112, 60, 20, 13
Paper 6
For I=1 To 40
    Print "message "; I
    Wait 5
Next I
Wait Key
```

See

Print, Locate, Text, Border Dims

X

X Mouse

Function · VMOS 0.2

Syntax

```
X Mouse
```

Returns

the mouse pointer's x position in screen coordinates (appendix C).

Examples

Show the mouse position:

```
Show
Repeat
    Home
    Print X Mouse; ", "; Y Mouse; "     "
    Wait Vbl
Until Inkey$<>""
```

A sprite that follows the mouse (run from EXAMPLES):

```
Load Images "SHIP.BMP", 1
Repeat
    Sprite 0, X Mouse, Y Mouse, 1
    Wait Vbl
Until Mouse Key=1
```

See

Y Mouse, Mouse Key

xor

Operator · VMOS 0.1

Syntax

```
a xor b
```

Returns

the bitwise exclusive OR: bits set in a or in b, but not in both.

Examples

Bitwise xor:

```
Print 12 xor 10            ' 6    (%1100 xor %1010 = %0110)
Wait Key
```

Toggle a value between two numbers with xor:

```
C=2
For I=1 To 6
    Print C
    C=C xor 7                  ' 2, 5, 2, 5 ...
Next I
Wait Key
```

A pattern from x xor y:

```
For Y=0 To 127
    For X=0 To 127
        Plot 128+X, 48+Y, (X xor Y) and 15
    Next X
Next Y
Wait Key
```

See

and, or

Y

Y Mouse

Function · VMOS 0.2

Syntax

```
Y Mouse
```

Returns

the mouse pointer's y position in screen coordinates (appendix C).

Examples

Draw where the mouse is while the left button is held:

```
Show
Repeat
    If Mouse Key=1 Then Circle X Mouse, Y Mouse, 3
    Wait Vbl
Until Inkey$<>""
```

A horizontal line that follows the mouse up and down:

```
Repeat
    Cls
    Draw 32, Y Mouse To 351, Y Mouse
    Wait Vbl
Until Inkey$<>""
```

See

X Mouse

Appendix A: Operators and Precedence

Highest precedence first:

Operator	Meaning
`-x`	negation
`* / mod`	multiply, divide, remainder (appendix G)
`+ -`	add, subtract; `+` also concatenates strings
`= <> < > <= >=`	comparison: −1 (true) or 0 (false); strings too
`not`	bitwise NOT
`and`	bitwise AND
`or xor`	bitwise OR / exclusive OR

Because comparisons give −1 or 0, `and`, `or` and `not` work as logical operators on them. `not` binds less tightly than comparisons, so `not A=B` is `not (A=B)`.

Symbol	Meaning
`( )`	grouping, any depth
`:`	separates statements on one line
`'`	comment to the end of the line (same as `Rem`)

Appendix B: Values and Names

Numbers

32-bit integers, −2147483648 to 2147483647: `123 -5 $FF` (hex) `%1010` (binary). Fixed-point (0.5): `1.5 -0.25 .75`, −32768 to 32767.9999 (appendix G). Floating-point (0.6): `1.5! 1E10! 2.5E-3!` (appendix H).

Strings

`"text"`, up to 32767 characters (0.6), allocated at the length needed.

Names

A letter or _, then letters, digits and _, up to 23 characters. A `$` suffix makes a string variable, `#` a fixed-point one, `!` a float (`SPEED#`, `ANGLE!`; `A`, `A$`, `A#` and `A!` are four different variables). Numeric variables start as 0, strings as `""`. Simple variables need no declaration; arrays and records need `Dim` (Dim).

Labels

A name followed by a colon at the start of a line: `START:`

Truth

True = −1, False = 0; any non-zero value counts as true.

Appendix C: Screen Coordinates

Sprites, bobs, the mouse and the tile map share one coordinate system, as on the C64: 0,0 is the top-left corner of the whole display, under the border. The display is 384 x 224 pixels; with the default border (`Border Dims 32, 32, 12, 12`) the visible 320 x 200 area is x 32 to 351, y 12 to 211. Negative coordinates are allowed (the object is then partly or fully off screen). `Print` writes in the text window: 40 x 25 characters at 32,12 by default, anywhere with `Window` (`Locate` counts characters within it, not pixels); `Text` writes at any pixel. The program's screen is 512 x 256 pixels; the display shows 384 x 224 of it, from 0,0 unless `Screen Offset` sets another origin.

The display layers, back to front:

1. the background color (`Color Back`)
2. sprites set to `Sprite Behind`
3. the tile layer (`Tiles Back`)
4. the program's screen (`Print`, `Cls`, drawing, bobs)
5. the tile layer (`Tiles Front`)
6. sprites
7. the mouse pointer

Appendix D: Key Numbers

For `Key State` and `Scancode`.

Key	No.	Key	No.	Key	No.	Key	No.	Key	No.
A	10	H	29	O	38	V	31	2	59
B	28	I	33	P	41	W	9	3	8
C	20	J	34	Q	62	X	23	4	11
D	18	K	37	R	17	Y	25	5	16
E	14	L	42	S	13	Z	12	6	19
F	21	M	36	T	22	0	35	7	24
G	26	N	39	U	30	1	56	8	27
								9	32

Key	No.	Key	No.	Key	No.
RETURN	1	UP	73	F1	4
SPACE	60	DOWN	7	F3	5
DEL	0	LEFT	74	F5	6
TAB	65	RIGHT	2	F7	3
ESC	71	HOME	51	RUN/STOP	63
CTRL	58	MEGA (C=)	61	left SHIFT	15
				right SHIFT	52

Typing \ ~ | _ { } and the backquote. Hold the MEGA key to type the character printed on the front of a key: / gives \, , gives ~, . gives |, = gives _, : gives { and ; gives }; the left arrow gives the backquote. This works in the editor and for Inkey$.

Appendix E: Run-Time Errors

Syntax errors are found when the program is compiled (F1 / F2). These errors stop a running program and report the line:

- Division by zero
- Array index out of range
- Array used before its Dim
- This array has had its Dim already
- Array too big
- Out of memory for arrays
- Out of chip memory (bobs, samples, Double Buffer)
- String longer than 32767 characters
- Out of Data
- Read wants a number, the Data is a string
- Division by zero (fixed point too)
- Too many procedure calls inside each other
- Sprite / bob / image / sample / tile numbers out of range
- Reserve Bob comes once, before the first Bob
- Border Dims are 0 to 255
- The window must fit in 512 x 256 pixels
- Color numbers are 0 to 255, Raster lines are 0 to 311, Tile flip is 0 to 3
- A circle's radius is 0 or more
- That image (or sample) is not loaded
- Cannot open: FILE (a picture or sound file that does not exist)
- Stopped (RUN/STOP was pressed)
- Copper lines are 0 to 311, The copper list is full (1020 entries)

- Voices are 1 to 3, Tempo is 1 to 255, Envelope numbers are 0 to 9, Sid registers are 0 to 28
- Play's voices are 1 to 15, Instruments are 0 to 9, Play has 15 voices: 1-3 SID, 4-7 PCM, 8-15 FM, Give the note as "A4", "#F3" or "$B2", A YM2151 song has the YM2151 (Track Stop), Operators are 1 to 4, Algorithm, feedback, pms 0 to 7, ams 0 to 3, The slide is −128 to 127, ar, dr, sr are 0 to 31; sl, rr 0 to 15, tl 0-127, mul 0-15, dt 0-7, dt2/ks 0-3, am 0-1, Fm registers and values are 0 to 255
- Objects are 0 to 255, There is no such object (Td Object), BOX is not loaded (Td Load), Too many objects loaded (32), Too many textures (64), BOX.OBJ has no faces, textures are up to 256 x 256, Too much for one Td Redraw (65000 corners, triangles)
- Td Path / Bob Path / Sprite Path: the steps are M, T, MT, W and L, M takes x,y,z,frames, frames are whole, 1 or more, L needs a step that takes frames first, up to 64 paths at a time, a path has up to 64 steps, Not a recorded path (Path Record), Up to 16 recorded paths (.PTH)
- No track is loaded (Track Load), That position is not in the song, A track is playing: it has the PCM voices (Track Stop), A .PST track is playing: it has the SID (Track Stop), FILE is not a MOD, PST, PSY or VGM, FILE is not a PROSIDYM song, FILE has no YM2151 music, FILE is damaged (a .VGZ that does not unpack)
- Image numbers are 1 to 240 (Sprite), Image numbers are 1 to 1024 (bobs), Images 1-240 are 1x1 to 64x64, Images 241-1024 are 1x1 to 1024x1024, A bob's size is 0 or more
- FILE: give the load address, FILE does not fit in the 64 KB there, FILE holds its own addresses, FILE is not a plain cartridge, The address is $0000 to $FFFF, Vic20 On first, Load a .prg first (Cpu8 6502 Load), The VIC-20 did not reach READY., The program does not fit where BASIC starts
- Network is off (SHIFT+POUND), A port is 1 to 65535, A host name has 1 to 127 characters, No such host name (Udp Send), Udp Send sends 1472 characters at most, The network has no UDP port free, A Tcp connection is 1 or 2, No Tcp connection is free (two at most), That Tcp connection is not open, The Tcp connection has closed

Appendix F: The Copper

The copper is VIXEN-32's display-list coprocessor: at a given raster line it writes a value into a display register while the picture is being drawn, so a color, a sprite position or a scroll value can differ on every line.

The list. Each Copper command adds an entry to the program's list: from line y on, this register has this value. Lines are in sprite coordinates (0 = the top of the display, under the border). Entries may be added in any order, and any number may share a line; up to 1020 entries.

When it takes effect. `Copper On` hands the list to the copper and starts it. After that the list is handed over again at every `Wait Vbl`, during vertical blank, so a list rebuilt every frame (`Copper Clear`, the entries, `Wait Vbl`) is never shown half-built. `Copper Off` stops it.

Above the first entry. A copper write would persist until something else writes the register, so at the top of every frame VMOS restores the program's own values: Border, Color Back, the

palette (Color), Screen Offset, the sprites (Sprite) and Tile Scroll. A copper change therefore only affects its own lines, and the normal commands keep working.

Below the last entry. A change lasts until the next entry for the same register: a Copper Offset wave ending at line 196 leaves line 196's offset on every line below; end it with an explicit entry (`Copper Offset 197, 0`). The screen's first two lines take their scroll from the screen's last line, so when the list scrolls the screen, VMOS restores the program's own offset on the screen's second-to-last line (from Border Dims), after that line has been fetched, so nothing visible changes and the top stays in place.

What it cannot change. The border size, the tile layer on/off and the sprites on/off are owned by the CPU once it has set them, so the copper does not offer them.

Exact timing. Border, background and palette colors and the tile scroll change on their own line. The screen scroll, the screen pointer and the sprites are used two lines ahead of the beam (graphics are fetched and sprites positioned for the line after next), so VMOS issues those entries two lines early; each then changes on its own line on the MEGA65 (the PC emulator shows the screen and sprites two lines early). STEP4.VMS has rulers for the screen, the sprites and the tiles. A sprite or scroll entry at line 0 or 1 (Copper Sprite, Copper Offset) would be due before the frame starts, so VMOS also makes it the frame's initial state; it then holds from the top. Copper Screen at line 0 or 1 starts at line 2.

Appendix G: Fixed-Point Numbers

A name ending in `#` holds a fixed-point number: `SPEED# = 0.5`, `Dim X#(100)`, a Type field `X#`, a procedure parameter `[A#]`. The format is Q16.16 (16 integer bits, 16 fraction bits): −32768 to 32767.9999 in steps of 1/65536, the format the 3D commands use.

Literals. `1.5`, `-0.25`, `.75`, `3.0`. A decimal is rounded to the nearest 1/65536, so 0.1 is 0.100006 and `Print 0.1*10` shows 1.0001.

Mixed arithmetic. An integer combined with a fixed-point value gives fixed-point: `X# + 2`, `3 * A#`, `A# / 4`. Two integers stay integer: `7/2` is 3, `7.0/2` is 3.5.

Conversion to integer (an integer variable, an array index, a command parameter such as `Sprite 0, X#, Y#`): the value is rounded down (floor): 1.5 gives 1, −1.5 gives −2.

Speed. `+`, `-`, comparisons, and `*` or `/` by an integer cost the same as for integers. Fixed-point times fixed-point takes a few more instructions; fixed-point divided by fixed-point is a 64-bit division (the slowest).

Text. Print and Str$ show up to 4 decimals, rounded, without trailing zeros: 1.5, 0.3333, −2. Input and Read accept decimals for a # variable; Val reads integers only.

Procedure results. `End Proc[x]` sets both `Param` (the integer part) and `Param#` (the fixed-point value).

Expression	Result
`X# = 7.0 / 2`	3.5
`X# = 7 / 2`	3 (integer division first)
`I = 2.75`	2
`I = -2.75`	−3
`If X# Then`	true when X# is not 0

Appendix H: Floating-Point Numbers

A name ending in `!` holds a floating-point number (0.6): `ANGLE! = 0.5`, `Dim V!(100)`, a Type field `X!`, a procedure parameter `[A!]`. The format is IEEE single precision: about 7 significant digits, 1.2E-38 to 3.4E+38, computed on the machine's floating-point unit.

Literals. `1.5!`, `3!`, `1E10!`, `2.5E-3!`. A decimal without `!` (`0.1`) is fixed-point by itself, but where it meets a float it is converted from its exact text: `X! = 0.1` and `X! * 0.1` use 0.1 exactly as a float represents it.

Mixed arithmetic. A float combined with an integer or fixed-point value gives a float: `X! + 2`, `F# * A!`, `7 / 2!` is 3.5. A float division by 0 stops the program (Division by zero).

Conversion to integer (or fixed-point): rounded down (floor): 1.5 gives 1, −1.5 gives −2.

Speed. Each float `+ - * /` and comparison is one round trip to the FPU: four instructions plus the unit's latency (about 25 cycles; division slightly more). Fixed-point (`#`) is faster where its range is enough.

Text. Print and Str$ show 6 significant digits without trailing zeros, with E+nn for very large or small numbers: 1.5, 0.333333, 1E+10, 1.5E-07. Input and Read accept floats for a `!` variable; Val reads integers only.

Procedure results. `End Proc[x]` sets `Param`, `Param#` and `Param!`.

Expression	Result
`X! = 7 / 2!`	3.5
`X! = 0.1 * 3`	0.3
`I = -2.5!`	−3
`Print 1/3!`	0.333333
`If X! Then`	true when X! is not 0

Appendix I: SID, PCM and FM Music (Play)

VMOS's Play, Envelope, Tempo, Filter, Rplay and Sound follow BASIC 65 (the MEGA65), so its music strings play unchanged, on three voices (VIXEN-32 has one SID; the MEGA65 has four). The notes are played in the background by the frame interrupt while the program continues.

Play string syntax. Spaces are ignored; a setting stays in effect until changed.

Code	Meaning
`A - G`	a note
`# $`	before a note: sharp, flat
`.`	before a note: dotted (1.5 x the length)
`W H Q I S`	note length from here on: whole, half, quarter, eighth, sixteenth
`R`	a rest
`O0 - O6`	octave (default 4: middle C)
`T0 - T9`	instrument (Envelope)
`U0 - U9`	volume
`X0 X1`	filter off / on for this voice
`M0 - M9`	vibrato (0 = off)
`P0 - P9`	portamento: notes glide into each other (0 = off)
`L`	loop back to the start of the string

The default instruments (Envelope changes them):

T	A	D	S	R	Wave	Pulse	Instrument
0	0	9	0	0	pulse	1536	piano
1	12	0	12	0	saw		accordion
2	0	0	15	0	tri		calliope
3	0	5	5	0	noise		drum
4	9	4	4	0	tri		flute
5	0	9	2	1	saw		guitar
6	0	9	0	0	pulse	512	harpsichord
7	0	9	9	0	pulse	2048	organ
8	8	9	4	1	pulse	512	trumpet
9	0	9	0	0	tri		xylophone

Timing. A whole note lasts 24 / Tempo seconds; the fraction of a frame left over carries into the next note, so long tunes keep time. A note's gate closes one frame before it ends, so the next note restarts its envelope.

Pitch. The SID runs at 1 MHz: its frequency value is Hz x 16.777 (440 Hz = 7382). Sound and Sid take that value; Play takes note names.

The beeper (Beep) and the samples (Sam Play) are separate sound sources and use none of the SID's three voices.

Play on the PCM voices (0.9). Play strings 4 to 7 drive the four PCM voices, in the same string syntax. An instrument there is a sample: Instrument t, s, "A4" makes sample s instrument Tt and states the pitch it was recorded at (default C4). T0-T9 default to samples 1-10, each taken as C4. A note plays the sample at the rate for its pitch (up to 48000 Hz) and sounds until the next note on that voice (no gap) or until a rest or the end of the string stops it; a sample shorter than the note ends by itself, and Sam Loop On makes it loop. U0-U9 is that voice's own volume (multiplied by Volume); M and P bend its pitch as on the SID; X has no effect there. A Sam Play on one of the four voices takes it over until Play's next note on it. While a track plays (Track Play) it owns the PCM voices, and Play strings 4-7 give an error.

Play on the YM2151 (0.21). Play strings 8 to 15 drive the YM2151's eight FM voices, in the same string syntax; `Play 8, "O4T0QCDE"` starts at voice 8. T0-T9 are FM instruments: the ten above remade in FM at the start (the same names), changed with Fm Instrument (algorithm, feedback, the LFO's depths), Fm Envelope (each operator's envelope) and Fm Operator (its level, frequency multiple, detune ...), all in the chip's own values, as Furnace and DefleMask show them. Play alone puts them back. A note keys the voice on; its key goes off the frame before the note ends, as the SID's gate, so the next note starts the envelope again. U0-U9 lowers the instrument's carriers 3 dB a step below 9; M and P bend the pitch as on the SID; X has no effect there. Fm r, v writes any YM2151 register (the LFO, the noise). While a YM2151 song plays (Track Play) it owns the chip: Play's voices 8-15 stop, and Play strings 8-15 give an error until Track Stop.

The FM instruments' operators (AR DR SR SL RR TL MUL DT; algorithm and feedback; operators 1-4 are M1, C1, M2, C2, so algorithm 0 runs them 1 to 4):

T	Instrument	Algorithm	Feedback	Operators 1 / 2 / 3 / 4
0	piano	4	3	31 12 4 5 7 34 1 3 / 31 9 3 4 7 10 1 0 / 31 14 5 6 7 42 3 5 / 31 9 3 4 7 12 1 7
1	accordion	4	6	22 2 0 1 7 30 1 0 / 20 1 0 1 7 10 1 0 / 22 2 0 1 7 36 3 2 / 20 1 0 1 7 12 2 6
2	calliope	4	0	31 0 0 0 8 52 2 0 / 28 0 0 0 8 10 1 0 / 31 0 0 0 8 62 4 0 / 28 0 0 0 8 18 2 0

T	Instrument	Algorithm	Feedback	Operators 1 / 2 / 3 / 4
3	drum	7	0	31 0 0 0 15 127 1 0 / 31 31 0 15 15 20 8 0 / 31 0 0 0 15 127 1 0 / 31 14 0 15 12 0 1 0
4	flute	4	4	18 4 0 2 8 44 1 0 / 16 2 0 1 8 8 1 0 / 31 0 0 0 8 70 7 0 / 16 2 0 1 8 34 2 0
5	guitar	4	5	31 14 6 7 9 28 3 0 / 31 7 4 3 9 8 1 0 / 31 16 6 8 9 38 1 3 / 31 8 5 4 9 14 2 0
6	harpsichord	4	6	31 12 7 6 10 22 4 0 / 31 9 6 5 10 8 1 0 / 31 12 7 6 10 30 8 0 / 31 9 6 5 10 14 2 0
7	organ	7	0	31 0 0 0 12 22 0 0 / 31 0 0 0 12 14 1 0 / 31 0 0 0 12 20 2 0 / 31 0 0 0 12 28 4 0
8	trumpet	4	6	16 4 0 2 8 26 1 0 / 18 2 0 1 8 8 1 0 / 16 4 0 3 8 32 1 3 / 18 2 0 1 8 12 1 3
9	xylophone	4	0	31 20 0 15 12 30 7 0 / 31 12 8 15 12 8 1 0 / 31 22 0 15 12 40 11 0 / 31 14 10 15 12 16 4 0

The drum is the ProSidYM tracker's FM KICK: a click (operator 2) and a body (operator 4), its pitch falling with slide −64 (4 semitones a frame); every other instrument has slide 0. Every DT2 is 0.

Appendix J: Tracker Music and YM2151 Songs (Track)

Track Load, Track Play, Track Stop and Track Loop play songs made in a tracker: ProTracker MODs, PROSID TRACKER songs (.PST, VIXEN-32's own tracker) and PROSIDYM TRACKER songs (.PSY, its successor, with eight YM2151 FM tracks; VMOS 0.21). The song plays in the background, from the frame interrupt at the start of vertical blank, while the program continues. Since VMOS 0.19 they play YM2151 songs too (below).

One player for every format. VMOS plays every song on the tracker's own player (pst_player.h), so a .PST or a .PSY sounds exactly as in the tracker. A MOD is converted to a PROSID song while loading, as the tracker imports one: its 31 samples become instruments 1-31, its four channels the four PCM tracks, its notes and effects unchanged, starting at 125 BPM and speed 6.

Memory. The file is read whole (1 MB max). The song (patterns and instruments) stays in chip RAM, where the player reads it, and each sample is copied once into chip RAM with silence appended. The next Track Load reuses the same chip RAM if the new song fits.

Voices. A playing song owns the voices it needs; commands that would take them give an error until Track Stop:

The song	It owns	Errors while it plays
a MOD, or a .PST without SID notes	the four PCM voices	Sam Play
a .PST with SID notes	the PCM voices and the SID	Sam Play, Play, Sound, Filter, Sid, Sid Volume
a .PSY	as a .PST, and the YM2151 when its FM tracks have notes	as a .PST, and Play's voices 8-15 and Fm
a YM2151 song (.VGM, .VGZ)	the YM2151	Play's voices 8-15 and Fm (0.21)

So SID sound effects (Sound, Play) can play over MOD music, and samples and SID sounds over a YM2151 song. Sam Stop, Volume, Play without arguments and Sound Clr do not affect a song's voices. The beeper (Beep) is a separate sound source and is always free.

YM2151 songs (VMOS 0.19). The YM2151 is the machine's FM chip: eight voices of four operators, as in many arcade machines and the X68000. Its songs come as VGM files (.VGM, or .VGZ packed with gzip): every write a game made to the chip, with the time between, recorded. Track Load keeps only the YM2151's writes and the waits; another chip's part (SegaPCM's drums in Out Run's songs, for one) is not played. A song made for a chip at another clock (4 MHz is common) is raised to the right pitch on this one at 3,579,545 Hz; its envelopes stay a little slow. The song plays from the timer interrupt, which runs 2,000 times a second while it plays, and it keeps time by the raster beam, to a 44,100th of a second. With Track Loop On (the default) it loops from the loop point the file gives, or from its start. YMSONG.VGM in EXAMPLES is VMOS's own: four bars in C, eight seconds, looping.

Appendix K: 3D Objects (Td)

VMOS draws 3D objects made in any 3D program and saved as Wavefront .obj files, using AMOS 3D's command names. There are three levels; each costs something only when the program uses it.

1. Objects on the screen. Td Load, Td Object, Td Redraw. No camera and no clipping: x, y are the object's center in screen coordinates (the same as Plot and Sprite), z its distance behind the screen. At z 0 one .obj unit is one pixel; Td Size scales it. Use this to show an object, or for vector bobs: draw it on a screen of its own and Screen Copy it where it is needed.

2. Td Clip. Triangles are clipped to a window and at the near plane, so objects can leave the window's edges and come right up to the viewer.

3. Td Camera. x, y, z become world coordinates, Y up, seen from the camera (which needs Td Clip).

Angles follow Degree/Radian like Sin and Cos: a tilts an object (its top moves away from you), b turns it, c rolls it. They are applied c, then a, then b; Td Order changes the order per object.

The view. The eye is 256 pixels in front of the screen (Td Focus changes it) and looks at the center of the Td Clip window (Td Center moves the center anywhere). Td Color draws an object, or one of its materials, in one unlit palette color. Td Cull Off shows a flat object's back side.

Rendering. Each Td Redraw transforms every object's vertices with one matrix, skips back-facing faces and draws the rest in any order: the hardware Z buffer (one depth value per 2 horizontal pixels, cleared by each Td Redraw) keeps the nearest face, so intersecting faces are drawn correctly; where two faces intersect, the dividing line can step by 2 pixels horizontally. The blitter draws each triangle in one operation, stepping its rows itself: textured faces from the blitter's 256 x 256 texture cache, perspective-correct (recomputed every 32 pixels), so large floors and walls seen at an angle keep straight textures; flat faces in one color. 3D is drawn inside the 384 x 224 display area, the Z buffer's size.

Colors. Flat faces are shaded by a light from the upper left: from half the color (unlit) to the full color (fully lit), each shade mapped to the nearest palette entry (recomputed when the palette changes). The default palette has only the 16 C64 colors and 16 greys, so define your own colors (Color) for smooth shading, or use textures.

The .obj file may contain v, vt, f (any number of corners; v, v/vt, v//vn and v/vt/vn), usemtl and mtllib; the .mtl newmtl, Kd and map_Kd (with -clamp on). Normals are computed from the faces. Limits: 32 definitions and 64 textures, 65534 vertices per definition, 256 objects, 65000 triangles per Td Redraw.

Textures. Td Redraw draws the faces grouped by texture and loads each texture into the blitter's 256 x 256 cache once per redraw. A texture with power-of-two width and height repeats: it is stored as a repeated 256 x 256 image (64 KB of chip RAM) and its load takes about 40 microseconds; the cache wraps at 256, so texture coordinates beyond 0..1 repeat it. Any other texture is stored at its own size plus a one-texel border (copies of the edge texels, so a face seen almost edge-on samples its own edge); it loads faster and does not repeat. The texture VIC20 is the VIC-20's picture, loaded new each redraw (appendix L).

Memory. Definitions, textures and per-frame work arrays are in chip RAM; the code runs from fast RAM.

Appendix L: The 8-Bit CPUs, the VIC-20 and the C64 (Cpu8, Vic20, C64)

The core has an 8-bit CPU block: a 6502 and a Z80 sharing one 64 KB, and the VIC-20's VIC (a PAL 6561) and two 6522 VIAs for the 6502. VMOS loads files into the 64 KB (Cpu8 6502 Load, Cpu8 Z80 Load) and runs a VIC-20 on these chips (Vic20 Ram, Vic20 On, Vic20 Off, Vic20 Run). Nothing of the VIC-20 is emulated: the 6502 runs the original KERNAL and BASIC at the VIC-20's own speed, beside the VMOS program, at no cost to it.

Either CPU also runs on its own, without the VIC-20's chips (Cpu8 6502 Run, Cpu8 Z80 Run; 0.17), at any clock from 1 Hz to 25 MHz (Cpu8 Speed). The VMOS program and the CPU talk through the 64 KB: Cpu8 Poke and Cpu8 Peek( ) read and write it while the CPU runs; Cpu8 Stop stops it. Nothing else is there: the 6502 sees only memory, a Z80 IN reads $FF and an OUT does nothing.

A VIC-20 in a program:

```
Cpu8 6502 Load "KERNAL.ROM", $E000
Cpu8 6502 Load "BASIC.ROM", $C000
Cpu8 6502 Load "CHAR.ROM", $8000
Vic20 Ram 8k
Vic20 On
Cpu8 6502 Load "GAME.PRG"
Vic20 Run
```

A cartridge game is loaded before Vic20 On (`Cpu8 6502 Load "GAME.CRT"`) and starts by itself; it needs no Vic20 Run.

Where a file goes (Cpu8 6502 Load / Cpu8 Z80 Load):

File	Without an address	With an address
.prg	its first 2 bytes	there; its first 2 bytes skipped
VICE cartridge ("VIC20 CARTRIDGE")	where its chips say	refused
2 bytes over a whole KB (4098, 8194 ...)	its first 2 bytes	there; its first 2 bytes skipped

File	Without an address	With an address
anything else	error: give the load address	there, whole

A .crt or .rom file and a VICE cartridge are read-only for the 8-bit CPU.

The VIC-20's memory. $0000-$03FF and $1000-$1FFF RAM; $0400-$0FFF, $2000-$7FFF and $A000-$BFFF as Vic20 Ram fits them; $8000-$8FFF the character ROM; $9000-$93FF the VIC and the VIAs; $9400-$97FF color RAM; $9800-$9FFF absent (I/O2, I/O3) unless a file is loaded there; $C000-$DFFF BASIC; $E000-$FFFF the KERNAL.

The picture. The VIC writes each frame into chip RAM by itself (no CPU time). A .mtl line `map_Kd VIC20` makes a face show it: the screen inside the VIC's border, as its registers place and size it ($9000-$9003; 176 x 184 pixels for the VIC-20's own 22 x 23 characters), so a game with another screen size still fills the face, with two texels of the border around it. Keep its texture coordinates within 0..1 (it does not repeat). Each Td Redraw loads the newest whole picture into the texture cache (about 30 KB, some 20 microseconds); VICSCR.OBJ in EXAMPLES is a face of 176 x 184 that shows it pixel for pixel at Td Size 184. Vic20 Border (0.24) adds as much of the border as you ask on each side, top and bottom 72 pixels at most together, and the texture coordinates 0..1 then cover the screen and that border.

Colors. From Vic20 On to the program's end, palette entries 240-255 are the VIC-20's 16 colors (240 black, 241 white … 255 light yellow); Load Images, Load Tiles and Td's textures set 16-239 only.

Keys. While the VIC-20 is on, every key goes to it and to the program (Inkey$ still sees them): the MEGA65's keys as on a VIC-20, cursor up and left with SHIFT, RESTORE as RESTORE. Joystick port 2 is its joystick too, and the cursor keys and CTRL are also its stick and fire. RUN/STOP stops the VMOS program, as always.

Sound. The VIC's three voices and noise play with the machine's sound.

Releases. CTRL+B packs every file Cpu8 6502 Load names (not VIC20 or C64, which are no files).

The C64 (0.24). A core with the C64's chips runs a Commodore 64 on them the same way: the 6502, the VIC-II (a PAL 6569), the two 6526 CIAs, the PLA and the machine's own SID, with nothing of the C64 emulated. Its ROMs are loaded as the VIC-20's are, where a C64 has them: BASIC at $A000, the character ROM at $D000, the KERNAL at $E000. C64 On puts them into the C64's own ROM memory and fills the 64 KB as at power-on, so a .prg is loaded after C64 On:

```
Cpu8 6502 Load "C64/KERNAL.ROM", $E000
Cpu8 6502 Load "C64/BASIC.ROM", $A000
Cpu8 6502 Load "C64/CHAR.ROM", $D000
C64 On
```

```
Cpu8 6502 Load "GAME.PRG"
C64 Run
```

C64 Run starts a .prg saved at $0801, where the C64's BASIC starts. There are no C64 cartridges and no disk drive. Vic20 On and C64 On take the same 6502: each switches the other off.

The C64's picture. The VIC-II writes each frame into chip RAM by itself, its border too. `map_Kd C64` makes a face show the 320 x 200 screen inside the border, with two texels of the border around it. As the texture cache is 256 texels wide, it is two textures of 160 x 200, the left and right half; Td Load cuts each face of such a material where u = 0.5, so a face of any shape keeps texture coordinates 0..1 across the whole screen. Each Td Redraw loads both halves (twice the VIC-20's, some 40 microseconds). C64SCR.OBJ in EXAMPLES is a face of 320 x 200 that shows it pixel for pixel at its own size; C64BOX.OBJ is a box of the screen's shape. C64 Border adds as much of the border as you ask on each side, 64 pixels at most left and right, top and bottom 56 at most together: the picture then stays two halves of at most 224 x 256.

The C64's colors and keys. Palette entries 240-255 are its 16 colors (the machine's own C64 colors, as palette 0-15 start), as for the VIC-20. The MEGA65's keyboard is the C64's key for key (cursor up and left with SHIFT, RESTORE as RESTORE); joystick port 2 is its port 2. RUN/STOP stops the VMOS program, as always.

Appendix M: The Network (Net, Ping, Udp, Tcp)

Switching it on. SHIFT+POUND switches the network on: in the VMOS editor (the status line shows "Network on: IP 192.168.5.69"), or at BASIC's READY prompt before VMOS or a release starts. A second SHIFT+POUND switches it off. A program never switches it on: it checks Net Ip$, and every network command but Net Ip$, Udp Close and Tcp Close stops it with "Network is off (SHIFT+POUND)" while the machine has no address. While the network is on, the editor answers pings.

The address comes from /VIXEN-32/NETWORK.TXT, the file BASIC reads too; no file means DHCP:

```
ip:   192.168.5.77        or   ip: dhcp
mask: 255.255.255.0
gw:   192.168.5.1
dns:  192.168.5.1
mac:  02:56:33:32:00:02   (optional)
```

Hosts. Every command that takes a host takes a name ("example.com") or an address ("192.168.5.1"). A name is asked of the DNS server once and kept for the rest of the run.

UDP or TCP:

	UDP	TCP
What	single packets, up to 1472 characters	a stream, any length
Arrives	maybe, maybe out of order	all of it, in order, or the connection closes
Speed	at once	a block of 536 characters per round trip
For	a game's state every frame	high scores, a server's answer, files

Limits. One UDP port, 16 packets waiting on it. Two TCP connections, as a client: nothing can connect to the machine. 2048 characters waiting on a connection.

Keep it going. The machine holds one arriving packet at a time. Udp Recv$, Tcp Recv$( ), Tcp Connected( ) and Wait Vbl take what has arrived, so a program that uses the network calls one of them every frame.

The end. The program's end closes its UDP port and its connections; the network stays on. A release uses the network the same way.