Amos 1.3 index

COLLABORATORS			
	TITLE : Amos 1.3 index		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		November 15, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Amos 1.3 index

1.1 Index

1.2 the editor

```
Function keys: f1 - f10

 Run          Test         Indent       Blocks Menu    Search Menu
 Run Other    Edit Other   Overwrite    Fold/Unfold    Line Insert

(Function Keys)  With Shift or Right Mouse.
 Load         Save         Save As      Merge          Merge ASCII
 Ac.New/Load  Load Others  New Others   New            Quit

(Function Keys)  With Ctrl.
 Block Start  Block Cut    Block Move   Block Hide     Save ASCII
 Block End    Block Paste  Block Store  Block Save     Block Print

(Function Keys)  With Alt.
 Find         Find Next    Find Top     Replace         Replace All
 Low<>Up      Open All     Close All    Set Text B.     Set Tab

Special Editor Keys and Functions
 Esc                     Toggle Direct Mode and Edit Screen
 Shift+Back or Ctrl+Y    Delete current line and pull up text
 Ctrl+U                  Undo, when in Overwrite mode
 Ctrl+Q                  Erase text from cursor to end of line
 Ctrl+I                  Insert line at cursor

Cursor Keys
 Shift+Left              Previous word
 Shift+Right             Next word
 Shift+Up                Top of page
 Shift+Down              Bottom of page
 Ctrl+Up                 Up one page
 Ctrl+Down               Down one page
 Shift+Ctrl+Up           Top of text
 Shift+Ctrl+Down         Bottom of text

Program Control
 Amiga+P  Push program into memory and create a new one
 Amiga+F  Flip between two programs in memory
 Amiga+T  Display next program

Cut and Paste
 Ctrl+B   Set beginning of block
 Ctrl+E   Set end of block
 Ctrl+C   Cut block
 Ctrl+M   Move block
 Ctrl+S   Save block
 Ctrl+P   Paste block
 Ctrl+H   Hide block

Marks
 Ctrl+Shift+n    Set mark.  n = 0 to 9
 Ctrl+n          Goto mark n

Search/Replace
 Alt+Up     Search up for next label or procedure
 Alt+Down   Search down for next label or procedure
 Ctrl+F     Find text string
```

```
 Ctrl+N     Find next string
 Ctrl+R     Replace text

Tabs
 Tab        Move to next tab
 Shift+Tab  Move to last tab
 Ctrl+Tab   Set/Unset Tab
```

1.3 keyboard macros

```
Key$(n)=command$
command$=Key$(n)

Assign Macro in command$ to function key n.  Keys 11-20 are accessed
by holding down the left Amiga key at the same time.  Alt. + ' will be
interpreted as a return.
```

1.4 scan$

```
x$=Scan$(n[,m])

n is the scancode of a key to be used in a macro string and m is a
mask to set special keys.

   Bit   Special Key

    0    Left SHIFT key
    1    Right SHIFT key
    2    Caps Lock (ON or OFF)
    3    Control
    4    Left Alt
    5    Right Alt
    6    Left Amiga
    7    Right Amiga
```

1.5 close workbench

```
Saves about 40k of memory.
```

1.6 close editor

```
Saves about 28k of memory.
```

1.7 set buffer

```
Set Buffer n

n represents the variable buffer size in kilobytes.  Must be the first
line in your program excluding REMs
```

1.8 free

```
t=Free

Returns amount of available variable space
```

1.9 dim

```
Dim var(x,y,z,...)[,var$(x,y),var#(x)]

This creates a table of variables or strings for usage.  These tables
may have as many dimensions as you want, but each dimension is limited
to a maximum of 65,000 elements.

In order to access an element in an array you simply put the element's
(x,y) in brackets after the variable name.

Example:  a$=var$(x,y)  a=var(1,3)

See your manual pages 35-38 for variable defination.
```

1.10 data

```
                Data list of items[,more items]

This statement allows you to set up date fields to be read in by the

                READ
                 command.  Please Note, if the data field is without quotes, it
may be mistaken for a variable or expression.

For further notes on
                READ
                 and
                DATA
                 types check your manual page 256.
```

1.11 read

```
                Read list of variables

Read will read to the list of variables a list of data items.  These
items MUST be the same type [string or variable] otherwize an error will
occur.

Also see
                RESTORE
                 and
                DATA
                .  For further notes, see page 256 of your
manual.
```

1.12 left$

```
d$=Left$(s$,n)  or  Left$(d$,n)=s$
```

1.13 right$

```
d$=Right$(s$,n)  or  Right$(d$,n)=s$
```

1.14 mid$

```
d$=Mid$(s$,p,n)  or  Mid$(d$,p,n)=s$

If n is not specified then from p to end of string will be affected.
```

1.15 instr

```
f=Instr(d$,s$ [,p])

Search for s$ in d$.  p is the starting position of the search.
```

1.16 upper$

```
s$=Upper$(n$)   Convert n$ into all upper case.
```

1.17 lower$

```
s$=Lower$(n$)   Convert n$ into all lower case
```

1.18 flip$

```
f$=Flip$(n$)   Reverse order of n$
```

1.19 space$

```
s$=Space$(n)   s$ will be a string of n spaces
```

1.20 string$

```
s$=String$(a$,n)   s$ will be the first character of a$ repeated n times
```

1.21 chr$

```
s$=Chr$(n)   Return ASCII character n
```

1.22 asc

```
c=Asc(a$)   Return ASCII code for a$
```

1.23 len

```
l=Len(a$)   Return length of a$
```

1.24 val

```
v=Val(x$)   Convert string to a number
```

1.25 str$

```
s$=Str$(x)   Convert number to a string
```

1.26 sort

```
Sort a(0)  Sort a#(0)  Sort a$(0)

Sorts array a in ascending order.  "(0)" must be included
```

1.27 match

```
r=Match(t(0),s)  r=Match(t#(0),s#)  r=Match(t$(0),s$)

Search array t for s and return position to r
```

1.28 inc

```
Inc var    Add one to var  (faster then  var=var+1)
```

1.29 dec

```
Dec var    Subtract one from var  (faster then  var=var-1)
```

1.30 add

```
Add v,exp [,base To top]    (eg. Add v,150)  v must be an integer

The second version of ADD works like so:
    v=v+a
    If v<Base Then v=Top
    If v>Top Then v=Base
```

1.31 acos

```
                c#=Acos(n#)

The ACOS function takes a number between -1 & +1 and calculates the
angle which would be needed to generate this value with
                COS
                .
```

1.32 cos

```
                c#=Cos(a)
c#=Cos(a#)

The cosine function computes the cosine of an angle.  Normally all
angles are measured in
                RADIAN
                s.  This may be changed using the
                DEGREE
                command.
```

1.33 tan

```
                t#=Tan(a)
t#=Tan(a#)

TAN generates the tangent of an angle.  Examples:


                Degree
                 :
                Print
                 Tan(45)
 0.9999998

                Radian
                 :
                Print
                 Tan(
                Pi#
                /8)
 .04141
```

1.34 sin

```
s#=Sin(a)
s#=Sin(a#)

The Sin function calculates the sine of the angle in a.  Note that this
function always returns a floating point number.
```

1.35 atan

```
                t#=Atan(n#)

ATAN returns the Arc
                Tan
                gent of a number.
```

1.36 hsin

```
                s#=Hsin(a)
s#=Hsin(a#)

HSIN computes the hyperbolic
                sin
                e of angle a.
```

1.37 hcos

```
                c#=Hcos(a)
c#=Hcos(a#)

HCOS computes the hyperbolic co
                sin
                e of angle a.
```

1.38 htan

```
                t#=Htan(a)
t#=Htan(a#)

HTAN computes the hyperbolic
                tan
                gent of angle a.
```

1.39 degree

```
                Degree

Generally all angles are specified in
                RADIAN
                s.  Since radians are rather
difficult to work with, it's possible to instruct AMOS to accept angles
in degrees.  From the execution point of DEGREE, AMOS will expect
DEGREES instead of
                RADIAN
                S to all TRIG functions.
```

1.40 radian

```
                Radian

The Radian command instructs that AMOS is to receive all angles in
Radians.  [This is the default.]  Also see
                DEGREE
                .
```

1.41 log

```
r#=Log(v)
r#=Log(v#)

LOG returns the logarithm in base 10 (LOG10) of the expression in v/v#.
```

1.42 exp

```
r#=Exp(e#)

Calculates exponential of e#.
```

1.43 ln

```
                r#=Ln(l#)

LN computes the natural or naperian
                log
                arithm of l#.
```

1.44 pi#

```
a#=Pi#   Gives the value for pi.
```

1.45 sqr

```
(Square root)

s#=Sqr(v)  Returns the square root of v.
```

1.46 abs

```
(Absolute value)

r=Abs(v)
r#=Abs(v#)

Removes signs, so -1 would be 1.
```

1.47 int

```
(Convert floating point to integer)

i=Int(v#)  If v#=1.32 Then i=1
```

1.48 sgn

```
(Find the sign of a number)

s=Sgn(v)

Returns -1 if negative, 0 if 0, 1 if positive
```

1.49 rnd

```
v=Rnd(n)

Returns a random number between 0 and n inclusive. If n is less then 0
then the last random number will be repeated.
```

1.50 randomize

```
                Randomize seed

Set seed for random number generator.  Common pratice is Randomize

                Timer
                .
```

1.51 max

```
r=Max(x,y)   R#=Max(x#,y#)   r$=Max(x$,y$)

Returns the largest value of either x or y.
```

1.52 min

```
r=Min(x,y)   r#=Min(x#,y#)   r$=Min(x$,y$)

Returns smallest value of either x or y.
```

1.53 swap

```
Swap x,y   Swap x#,y#   Swap x$,y$

Swap data between two variables of the same type.
```

1.54 fix

```
Fix (n)

Changes how floating point numbers are displayed.
If 0<n<16 then n number of decimal places will be displayed.
If n>16 then printout will be proportional and trailing zeros removed.
If n=16 then format will be returned to normal
If n<0 then floating point numbers will be displayed in exponential
format and n determines the number of decimal places displayed.
```

1.55 def fn

```
Def Fn name [(list)]=expression

User defined function.  name is the name used to call the function,
list contains a list of variables separated by commas to be used in
the function, and expression is one line of functions.
(eg.  Def Fn Do_math (x,y,z)=10 * x + z / y  )
```

1.56 fn

```
Fn name [(variable list)]

Execute user defined function. name is the name of the function and
variable list contains values to be passed to the function.
```

1.57 poke

```
                Poke address,v
Will place value v [1 byte only] into memory location address.

Doke address,v
Will place value v [2 bytes long] into memory location address.
Address MUST be even or a crash will occur.

Loke address,v
Will place value v [4 bytes long] into memory location address.
Address MUST be even or a crash will occur.

WARNING:  Poking anything into the Amiga is DANGEROUS if you are NOT
          perfectly sure you're actually going where you want, so test
          your address in your program by printing it to the screen and
          doing a
                WAIT KEY
                 prior to the Poke/Doke/Loke, so you can
          Ctrl-C the program, just incase the Address is wrong.
```

1.58 peek

```
v=Peek(address)
Will return the value from address [1 byte only] into v.

v=Deek(address)
Will return the value from address [2 bytes long] into v.
Address MUST be even or a crash will occur.

v=Leek(address)
Will return the value from address [4 bytes long] into v.
Address MUST be even or a crash will occur.

Note for LEEK:  If bit 31 of the returned value is on, then v will show
                as being negative.  [Bit 31 is the sign bit, 0 for
                positive and 1 for negative.]
```

1.59 hunt

```
f=Hunt(start To finish,s$)

HUNT will search memory from start TO finish for s$.  If s$ is found, f
will hold the memory address of the start of s$, if not, f will hold 0.
```

1.60 rol

```
ROR  Rotate Bits Right
ROL  Rotate Bits Left

.B Byte.  [ 8 bits]
.W Word.  [16 bits]
.L Long.  [32 bits]

Rol.B n,v  Will rotate the lowest 8 bits left one.  %10010000=%00100001
Ror.B n,v  Will rotate the lowest 8 bits right one. %10010000=%01001000

With .W [Word] and .L [Long], you can rotate more bits.
```

1.61 hex$

```
h$=Hex$(v [,places])

Set h$ to the hex value of v with places digits in length.
```

1.62 bin$

```
b$=Bin$(v [,places])

Set b$ to the binary value of v with places digits in length.
```

1.63 varptr

```
                address=Varptr(v)
address=Varptr(v$)

Returns the address of variable v. Each type of variable is stored
using its own format:

 Integers: VARPTR finds the address of the four bytes containing the
contents of your variable.

 Floating point: VARPTR returns the location of four bytes which hold
the value of the variable in the IEEE single precision format.

 Strings: The VARPTR address points to the first character  of the
string.  Since AMOS Basic does not end its strings with a
                Chr$
                (0), you
must obtain the length of the string using something like:

Deek(Varptr(a$)-2), where a$ is the name of your variable.  You could
also use
                Len
                (a$).
```

1.64 copy

```
Copy start,finish To destination

Move a section of data in memory.  The addresses must be even.
```

1.65 fill

```
Fill start To finish, pattern

Fill an area of memory with the four bytes in pattern.  The addresses
must be even.
```

1.66 btst

```
                b=Btst(n,v)

Test the binary digit at position n in the variable v.  If it is 1,
then a value of -1(
                TRUE
                ) will be returned.
```

1.67 bset

```
Bset n,v

Set the bit at position n in variable v to 1.
```

1.68 bclr

```
Bclr n,v

Set the bit at position n in variable v to 0.
```

1.69 bchg

```
                Bchg n,v

NOT the value of the bit at position n in variable v.

 One byte      Two bytes      Four bytes

                PEEK
                          DEEK           LEEK

                POKE
                          DOKE           LOKE
```

1.70 areg

```
                Areg(r)=a
a=Areg(r)

Areg() is a PSEUDO register to the 68000 chip's A0-A6 Address Registers.

Amos allows writing (Areg(r)=a) only to registers 0-2, but allows
reading (a=Areg(r)) for all 0-6.

Dreg(r)=a
a=Dreg(a)

Dreg() is a PSEUDO register to the 68000 chip's D0-D7 Data Registers.
```

```
Amos allows reading and writing from all 8 of these registers.

Notes:  When the AMOS commands
                Call
                ,
                Doscall
                ,Execall,Gfxcall,Intcall
        are executed, Areg(0)->Areg(2) and Dreg(0)->Dreg(7) are passed
        to the 68000's REAL registers.  Upon exit of the routine, Amos
        will read the 68000's REAL registers back into Areg(0)->Areg(2)
        and Dreg(0)->Dreg(7).  Also, Areg(3)->Areg(7) are reserved for
        use by Amos.
```

1.71 pload

```
Pload "filename",bank

Reserves the selected memory bank and loads it with machine code.

bank is the bank number to be reserved for your machine code program.
If it's negative, then the bank will be calculated using the absolute
value of this number and the required memory area will be allocated in
Chip memory.

Once you've loaded a program in this way, you can save it on disk as a
normal ".Abk" file, since the banks created in this manner are
permanent.  It will always be saved with your Amos program.

Your program must consist of machine code in standard Amiga format with
the following restrictions:

o The code MUST be relocatable, as it will be positioned at the first free
  memory location which is available.  [And probably never the same place
  twice.]

o  Only the CODE chunk of your program will be loaded.

o  The program MUST terminate with a single RTS instruction.
```

1.72 call

```
                Call address[,params]
Call bank[,params]

Execute a machine code program at address or start of bank.

See
                Areg
                /Dreg for details on Register usage.

Note, when machine code is running, all registers are available for use,
```

except A7 [Areg(7)].

A3 holds the address to the start of the params list.

All params will be pushed onto the stack at address A3. Retrieving them requires you to read pull them in reverse. [See Manual Page 285 for more information.]

A5 holds the address to the start of the MAIN Amos data area.

1.73 doscall

Before attempting these commands, be sure to observe warnings and guidelines written in the Amiga ROM Kernel Manuals [RKMs].

x=Doscall(offset) Execute a Dos Library call.
x=Execall(offset) Execute an Exec Library call.
x=Gfxcall(offset) Execute a Graphics Library call.
x=Intcall(offset) Execute an Intuition Library call.

For information on Registers, see
Areg
/Dreg.

offset should be negative. Although the RKMs show them as being positive, they are indeed supposed to be negative.

You should only attempt these commands if you're familar with the Amiga's ROMs.

1.74 restore

Restore label Restore lable$ Restore line Restore number

1.75 wait

Wait n

n is measured in 50ths of a second.

1.76 timer

v=Timer
Timer=v

TIMER is a reserved variable which is incremented by 1 every 50th of a second. It's most commonly used to help bring a random number:

```
                Randomize
                 Timer
```

1.77 not

```
                v=Not(d)

This does the same as an "Exclusive OR" which changes all the bits in a
digit to their opposite.

d=%101 : v=Not(d) :
                Print
                 d

d will have %1111111111111010
```

1.78 true

```
                v=True      Returns a value of -1.  [Try- a=0 :
                Print
                 a=0]
```

1.79 false

```
v=False     Returns a value of 0.
```

1.80 procedure

```
HELLO           Procedure HELLO            pass no values to procedure
HELLO[n$,25]    Procedure HELLO[Name$,x]   pass values to procedure
```

1.81 end proc

```
                End Proc
End Proc[variable]
End Proc[variable$]

End a Procedure.  Optional variables can be passed [ONLY one].  See

                PARAM
                 or PARAM$ for more information.
```

1.82 global

```
Global variable list

Let procedures use all variables in the variable list.
```

1.83 shared

```
                Shared variable list

Used inside of a procedure, lets procedure use program variables.
Similiar to
                GLOBAL
                 but is contained to current procedure only.
```

1.84 param

```
                Param, Param#, Param$

Lets a procedure store a value into the appropriate PARAM that can be
accessed by the main program.  This value is entered by the
                End Proc
                command.   (eg.
                End Proc
                [a$+b$+g$])  Only one value can be returned.
```

1.85 pop proc

```
Pop Proc

Jumps out of procedure
```

1.86 goto

```
Goto label   Goto line number    Goto variable    Goto exp
(eg. Goto START_UP    Goto 210    Goto x    Goto a$ + "Hello" )
```

1.87 gosub

```
Gosub label   Gosub line number   Gosub variable  Gosub exp
```

1.88 return

```
Return

Exit from a subroutine.  Must be present in all subroutines.
```

1.89 pop

```
               Pop

Removes the return address generated by a
               GOSUB
                and allows you to exit
the subroutine any way you like.  (eg. IF x=1 Then Pop : Goto label  )
```

1.90 if...then...[else]

```
               If conditions Then statements 1 [Else statements 2]

If...[Else]...End If

If tests=
               TRUE
                   list of statements 1
Else
    list of statements 2
End If

Note: It is illegal to use an IF...THEN...ELSE inside an
IF...ELSE...END IF structured test.
```

1.91 for...next

```
For index = first To last [Step inc] : list of statements : Next index
```

1.92 while...wend

```
While condition : list of statements : Wend
```

1.93 repeat...until

```
Repeat : list of statements : Until condition
```

1.94 do...loop

```
DO : list of statements : LOOP
```

1.95 exit

```
                Exit [n]
Jumps out of the following structures:
                FOR...NEXT
                ,
                REPEAT...UNTIL
                ,

                WHILE...WEND
                 and
                DO...LOOP
                .  If n is stated then EXIT will jump out of
n number of nested structures.

Exit If
Exit If expression [,n]
```

1.96 edit

```
Stops program and enters AMOS Basic editor.
```

1.97 direct

```
Stops program and enters DIRECT mode.
```

1.98 system

```
Exits Amos immediately.  Closing any open files/screens/etc in the
process.
```

1.99 end

```
Stops program.
```

1.100 on...proc

```
On v Proc proc1, proc2, proc3, ... proc n

Note: You can not pass parameters to a procedure using this command.
```

1.101 on...goto

```
On v
Goto
 line1, line2, line3, ... line n
```

1.102 on...gosub

```
On v
Gosub
 line1, line2, line3, ... line n
```

1.103 every n gosub

```
Every n
Gosub
 label
```

```
Subroutine label will be executed every n 50ths of a second.  Your
subroutine must be completed in less then this time.  After a
subroutine has been entered, the system will be automatically
disabled.  The EVERY ON command must be used before the
      RETURN
      statement in your subroutine.

Every n Proc
Every n Proc name

Every On   Every Off  Toggles automatic procedures and subroutines.
```

1.104 break on-off

```
Break On   Break Off

Activate/deactivate the Ctrl+C function.
```

1.105 on error goto

```
      On Error Goto label

On Error Proc
On Error Proc name

When an error occurs, the program will jump to the
      Goto
      /Proc.
```

```
Also see
                RESUME
                .
```

1.106 resume

```
Resume          Continue with same command program errored with.
Resume Next     Continue 1 command after the command that caused the error.
Resume line     Continue at "line".  [Program must have line numbers.]
Resume label    Continue at label.
```

1.107 errn

```
e=Errn

Returns the last error number.
```

1.108 error

```
Error n

Creates error number n.
```

1.109 memory banks

```
Types of memory banks

  Class       Stores            Restrictions       Type

  Sprites     Sprites and Bobs  Only bank 1        Permanent
  Icons       Icons             Only bank 2        Permanent
  Music       Sound Tracks      Only bank 3        Permanent
  Amal        Amal data         Only bank 4        Permanent
  Samples     Sample data       Banks 1 - 15       Permanent
  Menu        Menu definition   Banks 1 - 15       Permanent
  Chip Work   Temp workspace    Banks 1 - 15       Temporary
  Chip Data   Perm workspace    Banks 1 - 15       Permanent
  Work        Temp workspace    Banks 1 - 15       Temporary
  Data        Perm workspace    Banks 1 - 15       Permanent
```

1.110 reserve

```
                Reserve AS type,bank,length     [See
                Memory Banks
                 for more.]
```

1.111 listbank

```
Listbank   List banks in use    [See
Memory Banks
 for more.]
```

1.112 erase

```
Erase b    Erase data in bank b   [See
Memory Banks
 for more.]
```

1.113 start

```
s=Start(b)   s will hold the starting address of bank b

[See
Memory Banks
 for more.]
```

1.114 length

```
l=Length(b)  l will hold the length of bank b.  If bank b contains
sprites or BOBs then the number of images will be returned.

[See
Memory Banks
 for more.]
```

1.115 load

```
Load "filename"[,n]

If filename contains more then one bank then ALL existing banks will
be erased.  If n is given then only bank n will be overwritten.  If
filename is sprite or bob data then if n=0 current sprite data will be
lost; if n=1 then new sprite data will be appended to current data.

[See
Memory Banks
 for more.]
```

1.116 save

```
                Save "filename"[,n]

Save all banks unless n is specified.  Use extension ".ABK"

                [See
                Memory Banks
                 for more.]
```

1.117 bsave

```
                Bsave file$, start To end

eg. Bsave "Test",
                Start
                (7) To
                Start
                (7)+
                Length
                (7)

Saves a chunk of memory.
                Start
                (7) to
                Start
                (7)+
                Length
                (7).
```

1.118 bload

```
                Bload file$, address
  or
Bload file$, bank

If BLOADing into a bank it must already exist.

                [See
                Memory Banks
                 for more.]
```

1.119 screen open

```
Screen Open n,w,h,nc,mode

Open screen number n (0-7) , with a size of w pixels wide by h pixels
high.  The size of the screen can be larger then the display.  nc sets
the number of colours to be used (2,4,8,16,32,64,4096). mode sets
```

```
Lowres or Hires.  If screen number n already exists it will be
replaced by new screen. (eg. Screen Open 3,640,200,16,Hires)
```

1.120 screen close

```
Screen Close n

Delete screen number n
```

1.121 auto view

```
                Auto View On   Auto View Off

When AUTO VIEW is on then any new
                screen
                 that is opened or any changes
to the current screen will be automatically displayed.  Auto View Off
prevents this.
```

1.122 default

```
Default

Closes all currently open screens and returns to the default display.
```

1.123 view

```
                View

Display any changes to the current screen at the next vertical blank
period.  For use when
                AUTO VIEW
                 is OFF.
```

1.124 load iff

```
Load Iff "filename"[,s]

Load Iff picture called filename into screen number s.  If s is not
specified then picture will be loaded into the current screen.
```

1.125 save iff

```
Save Iff "filename"[,compression]
```

Save current screen to disk and call it filename. If compression is set to 1 then standard Amiga file compression is used. If compression is set to 0 then extra AMOS data containing screen settings such as Screen Display, Screen Offset and Screen Hide/Show will be added to your file. The default is 0.

1.126 screen display

```
Screen Display n [,x,y,w,h]
```

Display screen number n. The following modifiers can be added:
x,y sets the screens location in hardware coordinates. x is rounded to the nearest 16-pixel boundary.
w,h sets the width and height of current screen in pixels. w is rounded to the nearest 16-pixel boundary.

1.127 screen offset

```
Screen Offset n,x,y
```

Sets display offset of screen n by x,y pixels. x and y may be negative.

1.128 screen clone

```
Screen Clone n
```

Clones the current screen to screen n. This cloned screen can only be manipulated by the

```
SCREEN DISPLAY
 and
SCREEN OFFSET
 commands.
```

1.129 dual playfield

```
Dual Playfield screen1, screen2
```

Let screen 2 show through colour 0 of screen 1. The two screens must be the same resolution. The palette for both screens will be taken from screen 1. The colours for screen 2 will be taken from the half of the colour registers that are not being used by screen 1. If screen 1 and screen 2 both had 8 colours then screen 1 would use

```
colour registers 0-7 and screen two would use colour registers 8-15.
Anything drawn to screen 2 will automatically have it's colour
register converted to the appropriate number (eg. if you draw to
screen 2 with INK set to register 2, then register 9 will actually be
used.)  The possible screen colour combinations are:

   Screen 1         Screen 2
 No of colours    No of colours
      2                2
      4                2
      4                4
      8                4        Lowres only
      8                8        Lowres only
Note: never set SCREEN OFFSET for both screens to 0.
```

1.130 dual priority

```
Dual Priority screen1,screen2

Let screen 2 be in front of screen 1. The colour palette will still be
taken from screen 1.
```

1.131 screen

```
s=Screen

Return current screen number.  Current screen may or may not be
visible.
```

1.132 screen to front

```
                Screen To Front [s]

Move screen s to front of display.  If s is omitted then the current
screen is move to front.
Note: if
                AUTO VIEW
                 is OFF then the
                VIEW
                 command must be used before
the effect can been seen.
```

1.133 screen to back

```
Screen To Back [n]

Move screen n back of display.  Default is current screen.
```

1.134 screen hide

```
Screen Hide [n]
```

Hide screen n completely from view. Default is current screen.

1.135 screen show

```
Screen Show n
```

Show a screen that was previously hidden with Screen HIDE.

1.136 screen height

```
h=Screen Height [n]
```

Returns height of screen n. Default is current screen.

1.137 screen width

```
w=Screen Width [n]
```

Returns width of screen n. Default is current screen.

1.138 screen colour

```
c=Screen Colour
```

Returns maximum number of colours in current screen.

1.139 scin

```
s=Scin(x,y)
```

Returns screen number at hardware coordinates x,y.

1.140 default palette

```
Default Palette c1,c2,c3,... c32
```

Set colours in a default palette that will be applied to any
subsequent screens created. c is the $RGB values for each colour
register.

1.141 get palette

```
Get Palette n [,mask]

Load the colour palette from screen n to the current screen.  mask is
a 32-bit binary number that selects which colours you want to copy to
the current screen.  When a bit is set to 1 then the corresponding
colour is copied.
(eg. to get just the first 4 colours  mask=%000001111 )
```

1.142 cls

```
Cls
Fill current screen with colour 0 and clear any windows that are open.

Cls col
Fill current screen with colour col.

CLS col,x1,y1 To x2,y2
Fill area x1,y1 To x2,y2 with colour col.
```

1.143 screen copy

```
Screen Copy scr1 To scr2
Screen Copy screen scr1 To screen scr2

Screen Copy scr1,x1,y1,x2,y2 To scr2,x3,y3 [,mode]

Copy an area of screen scr1 to location x3,y3 on screen scr2. Mode is
a binary number that can set as follows:

  Mode     Effect                                      Bit pattern
  REPLACE  Copy image completely over destination      %11000000
  INVERT   Invert image and copy over destination      %00110000
  AND      Use logical AND                             %10000000
  OR       Use logical OR                              %11100000
  XOR      Use logical XOR                             %01100000
```

1.144 screen base

```
table=Screen Base

Returns the base address of the internal table used to hold the number
and position of your AMOS screen.  See EXAMPLE 20.2 for a simple
demonstration.  [On your AMOS 1.2 Program disk within the Manual Folder.]
```

1.145 def scroll

```
                Def Scroll n,x1,y1 To x2,y2,dx,dy

Define scroll zone number n (1-16).  x1,y1 to x2,y2 defines the area
to be scrolled. dx is the number of pixels the zone will be scrolled
to the right if positive or to the left if negative.  dy is the number
of pixel the zone will be scrolled down if positive or up if negative.
The scroll is performed every time the
                SCROLL
                 command is called.
```

1.146 scroll

```
                Scroll n

Scroll zone number n as defined in the
                DEF SCROLL
                 command.
```

1.147 screen swap

```
Screen Swap [n]

Swap the physical and logical screens.

This command deals with logical and physical screens.    A physical
screen is the one that is being shown at any given time and a
logical is the screen that all current drawing commands are being
sent to and is not visible.  After all drawing to the logical screen
is done, it can then be swapped with the physical screen and the
process repeats.
```

1.148 logbase

```
address=Logbase(p)

Returns the address for bit-plane p of the logical screen.

This command deals with a logical screen ONLY.    A physical
screen is the one that is being shown at any given time and a
logical is the screen that all current drawing commands are being
sent to and is not visible.  After all drawing to the logical screen
is done, it can then be swapped with the physical screen and the
process repeats.
```

1.149 phybase

```
address=Phybase(p)

Returns the address for bit-plane p of the physical screen.

This command deals with a physical screen ONLY.    A physical
screen is the one that is being shown at any given time and a
logical is the screen that all current drawing commands are being
sent to and is not visible.  After all drawing to the logical screen
is done, it can then be swapped with the physical screen and the
process repeats.
```

1.150 physic

```
               x=Physic   x=Physic(s)

Returns an identification number for the current physical screen to be
used in place of a screen number in the
               ZOOM
               ,
               APPEAR
                and
               SCREEN COPY
               commands.

This command deals with a physical screen ONLY.    A physical
screen is the one that is being shown at any given time and a
logical is the screen that all current drawing commands are being
sent to and is not visible.  After all drawing to the logical screen
is done, it can then be swapped with the physical screen and the
process repeats.
```

1.151 logic

```
x=Logic   x=Logic(s)

Returns an identification number for the current logical screen.

This command deals with a logical screen ONLY.    A physical
screen is the one that is being shown at any given time and a
logical is the screen that all current drawing commands are being
sent to and is not visible.  After all drawing to the logical screen
is done, it can then be swapped with the physical screen and the
process repeats.
```

1.152 wait vbl

```
                Wait Vbl

Wait for the next vertical blank and then continue.  Also see

                MULTI WAIT
                .
```

1.153 appear

```
                Appear source To destination,effect [,pixels]

Make
                screen
                 source appear on screen destination.  effect (1-number of
pixels on screen) determines the type of fade.  pixel sets the number
of pixels starting from the top of the screen that will be affected.
```

1.154 fade

```
Fade speed [,colour list]
Fade the current palette to back or to [colour list].  Speed is the
number of vertical blank periods used to complete the fade.

Fade speed To s [,mask]
Fade the current palette to the palette of screen s.  If s is negative
then it represents the palette of a sprite.  mask is a bit pattern
that specifies which colours should be changed.
(eg. just fade the first 5 colours  FADE 20 To 2,%00011111 )
```

1.155 flash

```
Flash index,"($RGB,delay)($RGB,delay)($RGB,delay)..."
Make colour register index cycle through each $RGB colour value
listed.  delay is the time measured in 50ths of a second that each
colour will be displayed.  FLASH operates as an interrupt.
Note:  A WAIT command should be used after a FLASH command.  It is
calculated like so:  wait value = fade speed * 15.

Flash Off
Deactivate the FLASH command.
```

1.156 shift up

```
Shift Up delay,first,last,flag

Shift colours from register first to register last up one position at
a time.  delay is the time measured in 50ths of a second between each
```

shift. If flag is 1 then the colours loop. If flag is 0 then the
contents of the first and last registers will be discarded, and the
region between will gradually be replaced by a copy of the first
colour in the list. SHIFT operates as an interrupt.

1.157 shift down

```
                Shift Down delay,first,last,flag

Shift colours down.  See
                SHIFT UP
                .
```

1.158 shift off

```
                Shift Off

Deactivate the
                SHIFT UP
                 and
                SHIFT DOWN
                 command.
```

1.159 zoom

```
Zoom source,x1,y1,x2,y2 To dest,x3,y3,x4,y4

Take area defined by x1,y1 to x2,y2 in screen source and make it fit
into area x3,y3 to x4,y4 on screen dest.
```

1.160 cop logic

```
addr=Cop Logic

Returns the absolute address to the logical copper list in memory.  This
allows you to poke your copper instructions directly into the buffer,
possibly using assembly language.
```

1.161 cop move

```
                (Write a MOVE instruction into the logical copper list.)

Cop Move addr,value
Generates a MOVE instruction into the copper list.
```

addr is an address of a 16 bit register to be changed. This must lie
within the normal copper DATAZONE ($7F-$1BE).

value is a word-sized (2bytes) integer to be loaded into the requested
register.

(See
                COP LOGIC
                 for the addr value.)

1.162 cop movel

                (Write a long MOVE instruction into the copper list.)

Cop Movel addr,value

This is identical to the standard
                COP MOVE
                 command, except that addr
now refers to a 32bit copper register.

value contains a long word (4bytes) integer.

1.163 cop reset

(Reset copper list pointer.)

Cop Reset

COP RESET restores the address used by the next copper instruction to
the start of the copper list.

1.164 cop wait

(Copper WAIT instruction)

Cop Wait x,y[,x mask,y mask]

COP WAIT writes a WAIT instruction into your copper list. The copper
waits until the hardware coordinates x,y have been reached and returns
control to the main processor.

Note that line 255 is managed automatically by AMOS. So, you don't have
to worry about it at all.

x mask and y mask are bitmaps which allow you to wait until just a
certain combination of bits in the screen coordinates have been set. As
a default, both masks are automatically assigned to $1FF.

1.165 copper off

This freezes the current AMOS copper list and turns off the screen display completely. You can now create your own display using a series of
COP MOVE
and
COP WAIT
instructions.

As a default, all user-defined copper lists are limited to a maximum of 12k. On average, each copper instruction takes up 2 bytes. So, there is space for around 6000 instructions. This may be increased if required, using a special option from the AMOS CONFIG utility.

Note that all copper instructions are written to a separate logical list which is not displayed on the screen. This stops your program from corrupting the display while the copper list is being constructed. To activate your new screen, you'll need to swap the physical and logical lists around with the
COP SWAP
command.

It's also important to generate your copper lists in strict order, starting from the top left of the screen, progressing downward to the bottom right. See EXAMPLE 10.15 in the MANUAL folder of your AMOS1.2 PROGRAM disk.

1.166 cop swap

This will cause the logical and physical copper lists to swap immediately. [This command is not in the manual!]

1.167 copper on

COPPER ON restarts the AMOS copper list calculations and displays the current AMOS screens. Providing you haven't drawn anything since the COPPER OFF instruction, the screen will be restored to precisely it's original state.

1.168 spack

Spack s To n [x1,y1,x2,y2]

Compress screen number s into bank number n. x1,y1,x2,y2 defines the area to be saved. The default is the whole screen. SPACK also saves the screen's mode, size, offset and display position. All x coordinates are rounded to the nearest 8 pixel boundary.

1.169 pack

```
                Pack s To n [x1,y1,x2,y2]

Same as
                SPACK
                 but does not save the screen's mode, size, offset or
position.
```

1.170 unpack

```
                Unpack b to s
For use with unpacking
                SPACK
                ed screens.  Unpacks the image in bank b
to screen number s.

Unpack b [,x,y]
For use with unpacking
                PACK
                ed screens.  Unpacks the image in bank b to
the current screen at coordinates x,y.
                PACK
                ed screen are really
intended to be used with
                DOUBLE BUFFER
                ing on.
```

1.171 pen

```
Pen index
Sets colour of text to the colour stored in register index (0-63).

=PEN$(n)
a$=Pen(n)
Allows a colour change within a string.
(eg.  c$ = Pen$(2) + "White " + Pen$(6) + "Blue" )
```

1.172 paper

```
Paper index
Sets background colour of text to the colour in register index (0-63).

=PAPER$(n)
x$=Paper$(n)
Allows a colour change within a string.
```

1.173 inverse on-off

```
Inverse On   Inverse Off
```

Swaps text background and foreground colours.

1.174 shade on-off

```
Shade On   Shade Off
```

Reduces brightness of text.

1.175 under on-off

```
Under On   Under Off
```

Turns on-off text underlining.

1.176 writing

```
Writing w1 [,w2]
```

Changes text writing mode as follows:

```
  w1=0  REPLACE    New text overwrites anything underneath
  w1=1  OR         Merge text using logical OR
  w1=2  XOR        Merge text using logical XOR
  w1=3  AND        Merge text using logical AND
  w1=4  IGNORE     All printing will be ignored

  w2=0  NORMAL     Text is printed along with its background
  w2=1  PAPER      Only the text background is printed
  w2=3  PEN        Prints text with a background of colour 0
```

1.177 locate

```
Locate x,y   Locate x,   Locate ,y
```

Place text cursor at location x,y measured from top left of screen.

1.178 cmove

```
Cmove w,h
```

MOVE text cursor relative to current position.

1.179 at

```
                x$=At(x,y)

Allows you to place text from within a string.
(eg.  x$=At(10,10) :
                Print
                 x$ + "Over Here" )
```

1.180 x text

```
t=X Text(x)
Converts a normal x coordinate into a text coordinate relative to the
current window.  If x lies outside this window a negative value will
be returned.

t=Y Text(y)
Converts a y coordinate form the standard screen format into a text
coordinate relative to the current window.
```

1.181 x graphic

```
g=X Graphic(x)
g=Y Graphic(y)

Converts a relative window coordinate into a screen coordinate.
```

1.182 home

```
Home

Move text cursor to 0,0
```

1.183 cdown

```
Cdown

Moves cursor down one line.
```

1.184 cdown$

```
x$=Cdown$

Allows a cursor down movement within a string.
```

1.185 cup

```
Cup
```

```
Move text cursor up one line.
```

1.186 cup$

```
x$=Cup$
```

```
Allows a cursor up movement within a string.
```

1.187 cleft

```
Cleft
```

```
Move text cursor left one space.
```

1.188 cleft$

```
x$=Cleft$
```

```
Allows a cursor left movement within a string.
```

1.189 cright

```
Cright
```

```
Move text cursor right one space.
```

1.190 cright$

```
x$=Cright$
```

```
Allows a cursor right movement within a string.
```

1.191 x curs

```
x=X Curs
y=X Curs
```

```
Returns x or y [respective] coordinate of text cursor.
```

1.192 set curs

```
Set Curs L1,L2,L3,L4,L5,L6,L7,L8

Changes shape of cursor.  Each parameter is an eight-bit binary number
representing a line of the cursor starting from the top.
```

1.193 curs on-off

```
Curs On   Curs Off

Turn the current screen's cursor ON or OFF.
```

1.194 memorize

```
               Memorize X   Memorize Y

Saves current cursor position in memory.

               [Also see
               REMEMBER
               .]
```

1.195 remember

```
               Remember X   Remember Y

Recalls previous
               MEMORIZE
               d cursor position.
```

1.196 cline

```
Cline [n]

Clear current line of text.  If n is stated then text is erased from
current position to n number of characters.
```

1.197 curs pen

```
Curs Pen n

Change cursor colour.
```

1.198 centre

```
Centre a$
```

Prints a$ centered on current line.

1.199 set tab

```
Set Tab n
```

Sets tab distance to n spaces.

1.200 tab$

```
x$=Tab$
```

Returns tab control character.

1.201 repeat$

```
x$=Repeat$(a$,n)
```

Creates a repeat ESC string for a$ n times and stores result in x$.

1.202 inkey$

```
k$=Inkey$
```

Returns whichever ASCII key is pressed on the keyboard. INKEY$ does not wait for a key to be pressed.

1.203 scancode

```
s=Scancode
```

Returns the scancode for the key that was entered using the last

```
INKEY$
 function.
```

1.204 key state

```
                t=Key State(s)

Returns a value of -1(
                TRUE
                ) if key number s has been pressed.  s is a
keyboard
                scancode
                .
```

1.205 key shift

```
k=Key Shift

Returns a bit pattern representing which control keys have been
pressed.
   Bit   Key
    0    Left Shift
    1    Right Shift
    2    Caps Lock (on of off)
    3    Ctrl
    4    Left Alt
    5    Right Alt
    6    Left Amiga
    7    Right Amiga (or Commodore key)
```

1.206 set input

```
Set Input c1,c2

This sets the End-Of-Line characters which will be used to terminate all
input statements.  Normally c1 holds 10 and c2 holds -1.

c1 and c2 hold ascii values which will be used as input terminators.  If
you want to use a single character, set c2 to -1.
```

1.207 input$(n)

```
x$=Input$(n)

Returns n number of characters from the keyboard.  This command does
not echo these characters to the screen.
```

1.208 wait key

```
Wait Key

Wait for a key to be pressed and then continue.
```

1.209 key speed

```
Key Speed lag,speed

Set the speed of the key repeats.  Both lag and speed are measured in
50ths of a second.  lag is the amount of time between the key press
and the first repeat.  speed is the amount of time between each
successive repeat.
```

1.210 clear key

```
Clear Key

Clears the keyboard buffer.
```

1.211 put key

```
Put Key a$

Load a string into the keyboard buffer.  Often used for setting
defaults for INPUTs.
```

1.212 input

```
Input [a$];[var1,var2,...];

a$ is the optional text to be displayed as a prompt.
```

1.213 line input

```
Line Input [a$];[var1,var2,...];

Inputs a number of variables separated by the user by pressing return.

a$ is the optional text to be displayed as a prompt.
```

1.214 print

```
Print variable list

Print displays the variable list to the current window or screen.

NOTE: all variables must separate themselves with semi colons [;].
      Example:  Print "Your name is";realname$
```

1.215 print using

```
Print Using format$;variable list

Here is a list of the possible formatting controls:

 ~  Print a single character from a string.
 #  Print a single digit from a variable.  If no digit then space.
 +  Add a plus sign if positive or a minus sign if negative.
 -  Add a minus sign if negative.
 .  Place decimal point.
 ;  Centre a number but don't print a decimal point.
 ^  Print a number in exponential form.
```

1.216 zone$

```
x$=Zone$(a$,n)

Creates zone number n around a$ and stores in x$.  When x$ is printed
it will automatically activate it's zone.
```

1.217 border$

```
x$=Border(a$,n)

Creates border number n around a$ and stores in x$.  When x$ is
printed it will automatically have a boarder.
```

1.218 hscroll

```
Hscroll n

Scroll text in current window horizontally by one character. n can be
one of four values:

   1 = Move current line to the left
   2 = Scroll screen to the left
   3 = Move current line to the right
```

```
   4 = Scroll screen to the right

Blank lines are left where gaps are created.
```

1.219 vscroll

```
                Vscroll n

Same as
                HSCROLL
                 except this scroll is vertical and the options are
different.  n can be one of the following:

   1 = Any text at the cursor line and below are scrolled down.
   2 = Text at the cursor line and below (if any) are scrolled up.
   3 = Only text from the top of the screen to the cursor line is scrolled
       up.
   4 = Text from top of the screen to the current cursor position is
       scrolled down.

Blank lines are left where gaps are created.
```

1.220 text

```
                Text x,y,t$

Prints graphic text in t$ at x,y.  All coordinates are measured
relative to the characters baseline which can be determined by the

                TEXT BASE
                 command.
```

1.221 get fonts

```
                Get Fonts

Makes an internal list of available fonts from the current start-up
disk.  This list can be examined using the
                FONT$
                 function.  Get FONTS
must be called before
                SET FONT
                .
```

1.222 get disc fonts

```
Get Disk Fonts

Search for fonts in the font folder.
```

1.223 get rom fonts

```
Get Rom Fonts

Searches for ROM fonts.
```

1.224 font$

```
a$=Font$(n)

Returns a string of 38 characters describing font number n.  If font
does not exist then a null string will be returned, otherwise the
string will be in the following format:

  Character   Description

   1-29        Font name
   30-33       Font height
   34-37       Identifier (disk or ROM)
```

1.225 set font

```
Set Font n

Change current font to font number n
```

1.226 set text

```
Set Text style

Selects font style. style is a bit pattern:
   Bit  Effect

    0   Underline
    1   Bold
    2   Italic
```

1.227 text styles

```
                s=Text Styles

Returns current text style in bit pattern style as in
                SET TEXT
                .
```

1.228 text length

```
w=Text Length(t$)

Returns the width of t$ in pixels using the current font.
```

1.229 text base

```
b=Text Base

Returns the position of the current font's base line in pixels.
```

1.230 wind open

```
                Wind Open n,x,y,w,h [,border [,set]]

Open window number n at graphic coordinates x,y.  x is rounded to the
nearest multiple of 16.    w,h specify the size of the window in
characters.  w and h must be divisible by 2.  border selects one of 16
border styles.  Any borders added to a window will be outside the
defined text area.  set selects the character set number defined by
the
                SET FONT
                 command.
```

1.231 wind save

```
Wind Save

This feature saves the display under the window created and is
replaced when the window is moved or closed.
```

1.232 border

```
Border n,paper,pen

Set the border of the current window. n is the border style (1-16).
paper and pen select the foreground and background colours of the
border.
```

1.233 title top

```
Title Top t$
```

Display t$ in the top border of current window. Only bordered windows can be titled this way.

1.234 title bottom

```
Title Bottom b$
```

Display b$ in the bottom border of current window. Only bordered windows can be titled this way.

1.235 window

```
Window n
```

Activates window number n.

1.236 =windon

```
w=Windon
```

Returns current window number.

1.237 wind close

```
Wind Close
```

Close current window.

1.238 wind move

```
Wind Move x,y
```

Move current window to graphic coordinates x,y. x will be rounded to the nearest 16-pixel boundary.

1.239 wind size

```
Wind Size sx,sy

Changes window size to sx,sy measured in characters.  After a window's
size has been changed the text cursor returns to 0,0.
```

1.240 clw

```
Clw

Clear current window and fill with present PAPER colour.
```

1.241 hslider

```
Hslider x1,y1 To x2,y2,total,pos,size

Draw a horizontal slider in area x1,y1 To x2,y2.  total is the number
of units the slider will be divided into.  pos sets the position of
the slider from the start of the slider in units defined by total.
size sets the size of the slider box in units defined by total.
```

1.242 vslider

```
                Vslider x1,y1 To x2,y2,total,pos,size

Similar to
                HSLIDER
                .
```

1.243 set slider

```
Set Slider b1,b2,b3,pb,s1,s2,s3,ps

Set colours and patterns for slider boxes.  b1,b2,b3 set the ink,paper
and outline colours for the background of the box.  pb chooses the
fill pattern to be used for these regions.  s1,s2,s3 set the colours
for the slider box and ps selects the pattern it is to be filled with.
bp and ps can be any fill patterns you wish (0-24).
```

1.244 ink

```
                Ink col[,paper][,border]

Selects colours for all subsequent drawing operations.  col is the
register number (0-63), paper is the colour for the background fill
patterns generated by the
                SET PATTERN
                 command, border is the colour
for outlines added to bars and polygons.  The border is toggled by the

                SET PAINT
                 command.  Include commas for excluded parameters.
(eg. INK ,,10)
```

1.245 colour

```
Colour index,$RGB

Loads colour register index (0-31) with $RGB.  $RGB represents the
red, green and blue intensity of the colour stored in register index.
```

1.246 =colour

```
c=Colour(index)

Return colour stored in register index.
```

1.247 palette

```
Palette reg0,reg1,reg2,... reg31

Load all colour registers with new colours.  Any register not to be
changed can be skipped by leaving in the corresponding comma.
```

1.248 gr locate

```
Gr Locate x,y

Set position of the graphics cursor in screen coordinates.  Any
drawing commands with the starting coordinates omitted will default to
the current graphic cursor position.
```

1.249 xgr

```
x=Xgr  y=Ygr

Returns the current graphics cursor location.
```

1.250 plot

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

Draws one pixel at coordinates x,y using colour c. If c is included in this statement then all following drawing commands will use this colour; otherwise the pixel will be drawn with the current colour.

1.251 point

```
c=Point(x,y)
```

Returns the colour register at point x,y

1.252 draw

```
Draw [x1,y1] To x2,y2
```

Draws a line from x1,y1 to x2,y2. x1 and y1 default to the graphics cursor.

1.253 box

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

Draws a box from x1,y1 to x2,y2.

1.254 polyline

```
                Polyline x1,y1 To x2,y2 To x3,y3 ...
```

POLYLINE is simular to
DRAW
 except it draws several lines at once. It's capable of drawing complex hollow polygons with one statement.

1.255 circle

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

Draw a circle at point x,y with a radius of r.

1.256 ellipse

```
Ellipse x,y,r1,r2

Draw an ellipse at point x,y  r1 wide and r2 high.
```

1.257 set line

```
                Set Line mask

mask is a 16-bit binary number that describes how lines made by the

                DRAW
                ,
                BOX
                 and
                POLYLINE
                 commands will look.  (eg. for a dotted line,
SET LINE %0101010101010101)  This command does not affect CIRCLE or
ELLIPSE.
```

1.258 paint

```
                Paint x,y,mode

Fills an enclosed area starting at point x,y with the current fill
pattern made with the
                SET PATTERN
                 command.  If mode is 0, filling will
stop at the current border colour. mode 1 will stop filling at any
colour different from the current
                INK
                 colour.
```

1.259 bar

```
Bar x1,y1 To x2,y2

Draws a filled rectangle.
```

1.260 polygon

```
                Polygon x1,y1 To x2,y2 To x3,y3 ...

Draw a filled polygon.  The last x and y coordinates should be the
same as the first.
```

```
[Also see
POLYLINE
.]
```

1.261 set pattern

```
Set Pattern pattern

Sets fill pattern.  Default 0 is a solid in the current
INK
 colour.
If pattern > 0 then one of 34 built-in fill styles is used. The first
three of which are used for the mouse.  If pattern < 0 the absolute
value of pattern will relate to the sprite number in bank one. Fill
sprites will be truncated as follows:  the width will be clipped to 16
pixels wide, the height will be rounded to the nearest power of 2.
Two-coloured images will be drawn in the current
INK
 colour.
Multi-coloured images' foreground colour will be merged with the
current ink colour using a logical AND; the paper colour of your
pattern is ORed with the image background colour.  If you want to use
your images original colour then set your colours to
INK
 31,0.  Don't
forget to load your images pallet with the
GET SPRITE PALETTE
 command.
```

1.262 set paint

```
Set Paint n

If n = 1 then outline mode is activated.  All
POLYGON
 and
BAR
instructions will be outlined in the boarder colour set with the
INK
command.  If n = 0 then outline mode is turned off.
```

1.263 gr writing

```
Gr Writing bitpattern

All graphics will be drawn in the style determined by the bitpattern.
The possibilities are:

JAM1 Bit 0=0
Only draws the part of your image that are set to the current ink
```

colour. Any parts drawn in the paper colour are ignored. This
is ideal for merging text over an existing background.

JAM2 Bit 0=1
This is the default. Any existing graphics will be replaced by
the new image (foreground and background).

XOR Bit 1=1
Changes the colour of the areas of a drawing which overlap an
existing picture. You can erase an image by XORing it in the
same position.

INVERSEVID Bit 2=1
Reverse image before it is drawn. Swaps foreground and
background colours of image.

Note: This command does not affect the PRINT and CENTRE commands
which are set by the WRITING command.

It is possible to combine one or more of these styles.
(eg. GR WRITING %101   use JAM2 and inversevid)

1.264 clip

Clip [x1,y1 To x2,y2]

Limits all drawing operations to a region of the screen specified by
by x1,y1 TO x2,y2. It is acceptable to use coordinates outside of the
normal screen boundaries.

1.265 sprites

Important info:

Four-colour sprites use the colours stored in registers 16 to 31
like so:

Sprite Number	Colour Registers
0/1	17/18/19
2/3	21/22/23
4/5	25/26/27
6/7	29/30/31

If your are using 32 or 64 colours on a screen then the sprites
will share the above color registers. This does not apply to
fifteen-colour sprites.

Note: If computed sprites are in use then make sure each sprite uses
the same colours.

1.266 sprite

```
Sprite n,x,y,i

Display sprite number n (0-63) at hardware coordinates x,y using image
number i.  n values that are grater then 7 relate to computed sprites.
```

1.267 get sprite palette

```
Get Sprite Palette [mask]

Loads sprite palette into current palette.  mask is a bit-pattern used
to select certain colours.
```

1.268 set sprite buffer

```
Set Sprite Buffer n

Eliminate any redundant memory used by the sprite buffer. n (16-256)
should be set to the number of pixels in your longest sprite.
```

1.269 sprite off

```
Sprite Off [n]

Turn off all sprites or just sprite number n.
```

1.270 sprite update

```
Sprite Update Off
Turn off automatic sprite updating.

Sprite Update
Update any sprites that have been moved.  For use when Sprite Update
is OFF.

Sprite Update On
Turn automatic sprite updating back on.
```

1.271 x sprite

```
x=X Sprite(n)
y=Y Sprite(n)

Returns current x or y [repsective] hardware coordinate of sprite n.
```

1.272 get sprite

```
Get Sprite [s,] i,x1,y1 To x2,y2

Grab area x1,y1 TO x2,y2 form screen s (default is current screen) and
store in sprite bank as image i.
```

1.273 del sprite

```
                Del Sprite s [TO f]

Delete
                sprites
                /
                bob
                s from bank.
```

1.274 x screen

```
x=X Screen([s,] xcoord)
y=Y Screen([s,] ycoord)

Translate a hardware coordinate into a screen coordinate relative to
the current screen or screen s.
```

1.275 x hard

```
x=X Hard([s,] xcoord)
y=Y Hard([s,] Xcoord)

Translate a screen coordinate into a hardware coordinate.
```

1.276 i sprite

```
image=I Sprite(n)

Returns the current image number for sprite number n.  0 will be
returned if the sprite is not displayed.
```

1.277 sprite base

```
table=Sprite Base(n)

Provides the address of the internal data list for sprite n.  If sprite
n does not exist, then the address of table will be 0.

Negative values for n return the address of the optional MASK associated
with your sprite.  table will now contain one of three possible values
depending on the status of this mask:

table<0   Indicates that there is no mask for this sprite.

table=0   Sprite n does have a mask, but the system has yet to generate
          it.

table>1   This is the address of the MASK in memory.  The first Long
          Word [see LEEK] of this area holds the lenght of the mask
          and the following locations is the actual mask defination.

See EXAMPLE 20.3 for a simple demonstration.  [On your AMOS 1.2 Program
disk within the Manual Folder.]
```

1.278 bob

```
Bob n,x,y,i

Place BOB number n at screen coordinates x,y using image number i.  n
is normally limited to 63 at the most but this value can be changed
with the AMOS setup program.
```

1.279 double buffer

```
Double Buffer

Create a logical screen for the current physical screen.
```

1.280 set bob

```
                Set Bob n,back,planes,minterms

Set drawing parameters for
                BOB
                 number n.  If back = 0 then the
                BOB
                 can
be moved around without corrupting the background.  If back > 0 then
then area beneath the
                BOB
                 will be replaced by colour back-1.  This is
useful if a
```

```
                BOB
                 is moving on a solid coloured background.  If back < 0
then the redrawing process is turned off and its up to the user to
replace and backgrounds destroyed by the
                BOB
                .  planes is a binary mask
representing which bit-planes the
                BOB
                 will be drawn on.  minterm is
the blitter mode used to draw the
                BOB
                .  Normal settings are %11100010
if the
                BOB
                 is used with a mask or %11001010 if no mask has been set.
Note: it is a good idea to use the SET BOB command before turning a
BOB on.
```

1.281 no mask

```
                No Mask [n]

Turn of masks for all
                BOB
                s or just
                BOB
                 number n.
```

1.282 autoback

```
                Autoback n

Coordinate drawing functions with
                BOB
                s.  n sets the AUTOBACK mode.  If
n = 0 then AUTOBACK is turned off.  All drawing commands are sent to
the logical screen only.  If n = 1 then graphical operations are sent
to the physical and logical screens at the same time.  No account is
taken of the
                BOB
                 on the screen.  It is best to keep graphical
operations away from
                BOB
                 when in this mode.  If n = 2 then graphical
operations are now synced with the
                BOB
                 updates and will appear behind
them.  However, in this mode, graphical operations will take twice as
long to complete.
```

1.283 bob update

```
                Bob Update On
Turn on the automatic
                BOB
                 update function.
                BOB
                s will be drawn every
50th of a second.

Bob Update Off
Turn off the automatic
                BOB
                 update function.  This allows all
                BOB
                s to
be placed before they are drawn.  To draw
                BOB
                s use Bob Update (below).

Bob Update
Draws all
                BOB
                s.  Note:  Bob UPDATE draws all
                BOB
                s to the logical
screen.  In order to see them a
                SCREEN SWAP
                 command must be called.
```

1.284 bob clear

```
                Bob Clear

Clear all BOBs from the screen and redraw the background regions
underneath.  This is intended for use with
                BOB DRAW
                 to provide an
alternative to the standard
                BOB UPDATE
                 command.
```

1.285 bob draw

```
                Bob Draw

Draw all
                BOB
                s.
                BOB CLEAR
                 and BOB DRAW give finer control over
                BOB
```

```
                updates then the
                BOB UPDATE
                 command.
```

1.286 x bob

```
                x1 =X Bob(n)
y1 =Y Bob(n)

Returns the x or y [respective] screen coordinate of
                BOB
                 number n.
```

1.287 i bob

```
                im=I Bob(n)

Returns the current image number used by
                BOB
                 number n.  The value will
be 0 if
                BOB
                 number n is not displayed.
```

1.288 limit bob

```
                Limit Bob [n,] x1,y1 To x2,y2

Limit the visibility of all
                BOB
                s or just
                BOB
                 number n to an area
defined by x1,y1 to x2,y2 in screen coordinates.  x will be rounded to
the nearest 16 pixel boundary.  This area must be greater then the
width of your
                BOB
                s or an error message will be returned.
```

1.289 get bob

```
                Get Bob [s,] i,x1,y1 To x2,y2

Grab an an image in area x1,y1 to x2,y2 on screen s and store it as

                BOB
                 image number i.
```

1.290 put bob

```
                Put Bob n

Draw
                BOB
                 number n to the screen at its current location for good.
Background image is not preserved.  Note:  Its a good idea to use a

                WAIT VBL
                 after this command.
```

1.291 paste bob

```
                Paste Bob x,y,i

Draw image number i to the current screen at screen coordinates x,y.
A
                WAIT VBL
                 command is not needed after this command.
```

1.292 bob off

```
                Bob Off [n]

Turn off all
                BOB
                s or just
                BOB
                 number n.
```

1.293 hide

```
                Hide [On]

Hide the mouse pointer.  A record of the number of times this function
has been used is kept and it takes an equal number of
                SHOW
                 commands
before the mouse is visible again.  HIDE On will hide the pointer
regardless of how many times the
                SHOW
                 command has been called.  Even
though the pointer is invisible, its location can still be read with
the
                X MOUSE
                 and Y MOUSE functions.
```

1.294 show

```
                Show [On]

Show the mouse pointer.  A record of the number of times this function
has been used is kept and it takes an equal number of
                HIDE
                 commands
before the pointer will be hidden.  SHOW On will show the pointer
regardless of how many times the
                HIDE
                 command has been called.
```

1.295 change mouse

```
Change Mouse m

Change the pointer image number m.  If m = 1 then pointer is an arrow.
If m = 2 then pointer is a crosshair.  If m = 3 then pointer is a
clock.  If m > 3 then pointer will be image number m-3.  The mouse
image can not be wider then 16 pixels and contain no more then four
colours.
```

1.296 mouse key

```
k=Mouse Key

Returns a bit pattern representing which key(s) of the mouse have been
pressed.  Bits 0-2 represent mouse buttons 1-3.
```

1.297 mouse click

```
c=Mouse Click

Returns a bit pattern representing which key(s) of the mouse have been
clicked.  The register is set back to zero after it has been checked
so it will only detect one key press at a time.
```

1.298 x mouse

```
x1=X Mouse    X Mouse=x1
Returns current x hardware coordinate of the mouse pointer, or assigns
pointer to coordinate x1.

y1=Y Mouse    Y Mouse=y1
Returns current y hardware coordinate of the mouse pointer, or assigns
pointer to coordinate y1.
```

1.299 limit mouse

```
Limit Mouse x1,y1 To x2,y2

Restrict mouse movement to area x1,y1 to x2,y2.
```

1.300 joy

```
d=Joy(j)

Returns a bit pattern that represents the direction of joystick j
(0-1).  The pattern is as follows:

    Bit Number    Direction
        0            up
        1           down
        2           left
        3           right
        4         fire button
```

1.301 jleft

```
                x=Jleft(j)

Returns a value of -1(
                TRUE
                ) if joystick number j is moved to the left,
otherwise 0(
                FALSE
                ) is returned.
```

1.302 jright

```
                x=Jright(j)

Returns a value of -1(
                TRUE
                ) if joystick number j is moved to the
right, otherwise 0(
                FALSE
                ) is returned.
```

1.303 jup

```
                x=Jup(j)

Returns a value of -1(
                TRUE
                ) if joystick number j is moved up,
otherwise 0(
                FALSE
                ) is returned.
```

1.304 jdown

```
                x=Jdown(j)

Returns a value of -1(
                TRUE
                ) if joystick number j is moved down,
otherwise 0(
                FALSE
                ) is returned.
```

1.305 fire

```
                x=Fire(j)

Returns a value of -1(
                TRUE
                ) if fire button on joystick j is down,
otherwize 0(
                FALSE
                ) is returned.
```

1.306 sprite col

```
                c=Sprite Col(n [,s To e])

Returns -1(
                TRUE
                ) if sprite number n has collided with any other
sprites or just a range of sprites from s to e.
Note: the
                MAKE MASK
                 command must be called before this function can be
used.
```

1.307 bob col

```
                c=Bob Col(n [,s To e])

Returns -1(
                TRUE
                ) if
                BOB
                 number n has collided with any other
                BOB
                 or
just a range of
                BOB
                s from s to e.
```

1.308 spritebob col

```
                c=Spritebob Col(n [,s To e])

Returns -1(
                TRUE
                ) if sprite number n has collided with any
                BOB
                s or just
a range of
                BOB
                s form s to e.

Note: this function only works in low res.
```

1.309 bobsprite col

```
                c=Bobsprite Col(n [,s To e])

Returns -1(
                TRUE
                ) if
                BOB
                 number n has collided with any sprites or just
a range of sprites form s to e.

Note: this function only works in low res.
```

1.310 col

```
                c=Col(n)

COL is an array containing the collision detection data.  Each element
relates to a sprite or
                BOB
                .  The first element will be set to -1 if a
```

```
collision has been detected with sprite or
                BOB
                 number 1 and so on
through the array.

Also see
                Author Note on =COL(Bob)
                .
```

1.311 hot spot

```
Hot Spot i,x,y

Sets the hot spot for image number i to x,y measured in pixel from the
top left corner of the image.  A hot spot is the "handle" by which a
image is positioned.
```

1.312 make mask

```
                Make Mask [n]

Make a mask for all images in the
                SPRITE
                /
                BOB
                 bank or just image n.
Masks are used for collision detection.
```

1.313 reserve zone

```
                Reserve Zone [n]

Reserve memory for n number of
                zone
                s.  If n is not provided then all
zones are removed from memory.
```

1.314 set zone

```
                Set Zone z,x1,y1 To x2,y2

Create zone area number n at screen coordinates x1,y1 to x2,y2.  The

                RESERVE ZONE
                 command must be called before any zones are set.
```

1.315 zone

```
t=Zone([s,] x,y)

Returns the zone number at screen coordinates x,y on screen s.  This
only detects the first zone at these coordinates.
```

1.316 hzone

```
t=Hzone([s,] x,y)

Returns the zone number at hardware coordinates x,y on screen s.
```

1.317 mouse zone

```
x=Mouse Zone

Returns the zone number at the current mouse position.
```

1.318 reset zone

```
                Reset Zone [z]

Deactivate all zones or just zone number z.  This does not fee any
memory used by the
                RESERVE ZONE
                 command.
```

1.319 priority on-off

```
                Priority On   Priority Off

Normally
                BOB
                s are drawn in the order they are numbered.  When Priority
in ON then the
                BOB
                s with the greater y coordinates are given priority.
This means that the
                BOB
                 lower on the screen have priority over the

                BOB
                s higher on the screen.  PRIORITY OFF set
                BOB
                 priority calculation
back to normal.
```

1.320 update

```
                Update   Update On   Update Off

Same as
                SPRITE UPDATE
                 and
                BOB UPDATE
                 except this command controls both

                sprite
                s and
                BOB
                s at the same time.
```

1.321 paste icon

```
                Paste Icon x,y,n

Draw icon number n at graphic coordinates x,y.
Note: if
                DOUBLE BUFFER
                 is on then icons will be drawn to both the
physical and logical screens.  To speed this up, turn
                AUTOBACK
                 to 0
before drawing icons.  This way they are only drawn to the logical
screen.
```

1.322 get icon

```
Get Icon [s,] i,x1,y1 To x2,y2

Grab icon number i from area x1,y1 to x2,y2 form screen s.
```

1.323 get icon palette

```
Get Icon Palette

Use icon palette.
```

1.324 del icon

```
Del Icon n [TO m]

Delete icon n to m.
```

1.325 make icon mask

```
Make Icon Mask [n]

Make a mask for all icons or just icon number n.
```

1.326 icon base

```
                table=Icon Base(n)

Returns the address for icon n.  The format for this information is
exactly the same as the
                SPRITE BASE
                 command.
```

1.327 get block

```
Get Block n,x,y,w,h [,mask]

Grab block number n from location x,y to width w and height h.  If
mask is set to 1 then a mask will be made for the block.
```

1.328 put block

```
Put Block n [,x,y]
Draw block number n at its original coordinates or at x,y.  x and y
are rounded to the nearest 16-pixel boundary.

Put Block n,x,y,planes [,minterms]
Draw block number n at x,y.  planes is a bit pattern dictating which
planes the block is drawn on.  minterms selects the blitter mode.
```

1.329 del block

```
Del Block [n]

Delete all blocks from memory or just block number n.
```

1.330 get cblock

```
Get Cblock n,x,y,w,h

Grab compressed block number n from location x,y to width w and height
h.  x will be rounded to the nearest multiple of eight.
```

1.331 put cblock

```
Put Cblock n [,x,y]

Draw compressed block number n at its original coordinates or at x,y.
x will be rounded to the nearest multiple of eight.
```

1.332 del cblock

```
Del Cblock [n]

Delete all compressed blocks or just number n.
```

1.333 boom

```
Boom

Make a boom sound.
```

1.334 shoot

```
Shoot

Make a gun shot sound.
```

1.335 bell

```
Bell

Make a bell sound.
```

1.336 volume

```
Volume [v,] intensity

Change volume of all sound channels or just channel number v.
intensity can be set from 0(off) to 63(full volume).
```

1.337 sam play

```
Sam Play s    Sam Plap v,s    Sam Play v,s,f

Play sample number s with voice v at a frequency of f.  v is a bit
pattern representing the 4 voice channels.
```

1.338 sam bank

```
Sam Bank n
```

Select a new memory bank to be used for samples.

1.339 sam raw

```
Sam Raw v,addr,len,freq
```

Play a raw sound sample from anywhere in memory using voice v.
Starting at address addr, play len number of samples at a frequency of
freq.

1.340 sam loop

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

All subsequent samples are to be continuously looped.

1.341 play

```
Play [voice,] pitch,delay
```

Plays the single note pitch for a length of delay. The optional voice
is a bit pattern which allows you to select one or more voices.

For full details, it's recommended that you read page 240-241 in your
manual for the listing of the notes and voice settings.

1.342 set wave

```
Set Wave wave,shape$
```

This provides you with the ability to design your own instruments. The

```
WAVE
```

command then allows you to use the wave with a voice.

In most musical programs [Sonix, DMCS, etc], you have the ability to use
a WAVEform, so you now do with AMOS. As usual, these waveforms are a
list of values from 0 to 255 and of a length of 256 values.

For more information, it is suggested to read pages 241-243 of your
manual to see what a WAVEform is.

1.343 wave

```
                Wave w To v

This instructs the voice(s) v that it is to use the WAVEform w while
playing notes with the
                PLAY
                 command.  v contains the standard bitmap
format for selecting voices.
```

1.344 noise

```
Noise To v

This instructs the voice(s) v that it is to use the Amos WHITE NOISE
WAVEform for the selected voice(s) v.  v contains the standard bitmap
format for selecting voices.
```

1.345 del wave

```
                Del Wave n

This deletes user created WAVEforms that were created using the
                SET WAVE
                command.  All voices using deleted WAVEform will be reset to the
standard SINE WAVEform (default).
```

1.346 sample

```
                Sample n To v

This allows a digitized WAVEform to be used as a WAVEform for v.  The
sample is a standard sample that the
                SAM PLAY
                 would play.

For more details, see page 245 of your manual.
```

1.347 set envel

```
Set Envel wave,phase To duration,volume

wave is the WAVEform that you wish to change the Envelope of.  This also
immediately will change any voice playing this WAVEform to the current
settings.

It is recommended to read the information in the manual on page 245-246
for further information on Attack, Decay, Sustain and Release.
```

1.348 say

```
Say t$ [,mode]

SAY will cause your amiga to speak t$ to you.

mode can be one of two modes:

0  (default) means AMOS must wait for the SAY command to finish speaking.

1  allows AMOS to continue while the SAY command speaks, but this slows
   down AMOS, so use as you see fit.
```

1.349 set talk

```
Set Talk sex,mode,pitch,rate

The following is the format for:

sex   is:      0 for male [default], 1 is for female

mode  is:      a rythmic effect added to the speech.
                 0  (default) is off
                 1  turns on the rythmic effect.

pitch is:      changes the pitch of the voice from 65 (low) to
               320 (incredibly high)

rate  is:      amount of words/minute ranging from 40 to 400.

Any of the above can be omitted, as long as commas are left as required.

Example:  Set Talk ,,,65
```

1.350 music

```
Music n

Play music number n.  Only one piece of music can be played at a time,
but up to four can be active and waiting on a stack.  All music files
must first be translated to AMOS format using one of the conversion
programs.
```

1.351 music stop

```
Music Stop

Stop playing the current piece of music.
```

1.352 music off

```
Music Off
```

Turn off all music. If restarted, the music will play from the beginning.

1.353 tempo

```
Tempo s
```

Set tempo for music to s (1-100).

1.354 mvolume

```
Mvolume n
```

Change the music volume to n (0-63).

1.355 voice

```
Voice mask
```

Set which voices are to be used by the music. mask is a bit pattern representing the four voice channels.

1.356 vumeter

```
s=Vumeter(v)
```

Returns the volume of the current note being played. The value returned can range form 0 to 63.

1.357 led

```
Led On   Led Off
```

Turn the audio filter on or off. This also controls the power light on the Amiga.

1.358 menu$

```
Menu$(n)=title$
Set menu titles that will appear in the menu bar.  Leave a space at
the end of each title to space them out.
(eg.  Menu$(1)="First menu " : Menu$(2)="Second menu " )

Menu$(t,o)
Menu$(t,o)=normal$ [,selected$][,inactive$][,background$]
Set option number o under title number t to normal$.
(eg. Menu$(1,1)="First option" : Menu$(1,2)="Second option" )
selected$ sets what the option will look like when it is selected.
The default is inverse text.  inactive$ sets what the option will look
like when it is inactive.  The default is italic text.  background$
sets the background for the menu.  background sets any back ground
that can be drawn with embedded drawing commands explained later.
```

1.359 menu on

```
Menu On
Activate menu defined in the Menu$ commands.

Menu On [bank]
Activates a menu that has been defined.  If bank number is included
then the menu in the appropriate memory bank will be used.
```

1.360 choice

```
                select=Choice
Returns a value of -1(
                TRUE
                ) if something has been selected from the
menu.  CHOICE is set to 0(
                FALSE
                ) after every check.

item=Choice(c)
Returns the option number that was selected at level number c.
```

1.361 on menu proc

```
On Menu Proc proc1 [,proc2][,proc3]...

Executes the procedure in the list that corresponds with menu title
that has been selected.  (eg. if the 3rd menu item was selected then
the 3rd procedure in the list will be executed.)  This is an interrupt
command and is checked every 50th of a second.
```

1.362 on menu gosub

```
                On Menu Gosub label1 [,label2][,label3]...

Similar to
                On Menu Proc
                .
```

1.363 on menu goto

```
                On Menu Goto label1 [,label2][,label3]...

Similar to
                On Menu Proc
                .
```

1.364 on menu on-off

```
On Menu On
Activate the automatic menuing system created by the On Menu
Proc/Gosub/Goto commands.  Every time an On Menu Proc/Gosub/Goto
function is called, the automatic system is turned off.  It must be
turned back on with an On Menu On command before the end of a
procedure or subroutine if you want it to stay on.

On Menu Off
On Menu Off turns off the automatic menuing system.  Its a good idea
to do this before disk access.
```

1.365 on menu del

```
Clear the current automatic menuing system so that another can be
defined.  A On Menu Off command must be used before this one.
```

1.366 menu key

```
Menu Key(,,) To c$
Create a keyboard short cut to a menu selection.  c$ is a single
character.

Menu Key(,,) To scan [,shift]
Create a keyboard short cut to non ASCII keys.  scan is a keyboard
scancode:

    Scancode     Keys
     80-89        Function keys F1-F10
     95           Help
```

```
     69           Esc
shift is an optional bitmap that checks for control keys:

  Bit    Key
   0     Left Shift Key
   1     Right Shift Key
   2     Caps Lock (On or Off)
   3     Ctrl
   4     Left Alt
   5     Right Alt
   6     Left Amiga
   7     Right Amiga (or Commodore key)
```

1.367 menu off

```
Menu Off

Turn off the menu bar.
```

1.368 menu del

```
Menu Del [(,,)]

Delete all menus or just those listed.
```

1.369 menu to bank

```
Menu To Bank n

Save current menu tree to bank number n.  Bank number n must not
already exist.  When a menu tree has been saved to a bank, it will
automatically be saved and loaded along with the main program.  This
means that the original program lines defining the menu can be
deleted, saving memory.
```

1.370 bank to menu

```
                Bank To Menu n

Load a menu tree saved in bank n.  To activate the new menu, use the

                Menu On
                 command.
```

1.371 menu calc

```
                Menu Calc

Rearranging a large menu can take some time.  Wait for an appropriate
time, then turn off the menu with
                MENU OFF
                , rearrange the menu, use
MENU CALC, and then turn the menu back on with
                MENU ON
                .
```

1.372 menu inactive

```
Menu Inactive level    Menu Inactive(,,)

Set an entire level of the menu to inactive or an individual option by
addressing its position in the menu tree.
```

1.373 menu active

```
Menu Active level    Menu Active(,,)

Activate an entire menu level or an individual option by addressing
its position in the menu tree.
```

1.374 menu line

```
Menu Line level    Menu Line(,,)

Display menu as a horizontal line.
```

1.375 menu tline

```
Menu Tline level    Menu Tline(,,)

Display menu as a horizontal line that goes right across the screen.
```

1.376 menu bar

```
Menu Bar level    Menu Bar(,,)

Display menu as a vertical bar.
```

1.377 menu movable

```
Menu Movable level    Menu Movable(,,)
```

Allow menus to be moved by the user.

1.378 menu static

```
Menu Static level    Menu Static(,,)
```

Stop user from rearranging the menus.

1.379 menu separate

```
Menu Separate level    Menu Separate(,,)
```

Allow each item in a menu to be treated as a separate element.

1.380 menu link

```
                Menu Link level    Menu Link(,,)
```

Reverse the

```
                MENU SEPARATE
                 command.
```

1.381 menu base

```
Menu Base x,y
```

Set the starting point of the first level of menus to x,y.

1.382 set menu

```
Set Menu (,,) To x,y
```

Place the top left corner of a menu to x,y measured relative to the previous level.

1.383 menu mouse

```
Menu Mouse On    Menu Mouse Off

Display menus at the current mouse pointer location.  The position of
the menus can be offsetted by the menu base command.
```

1.384 menu called

```
Menu Called(,,)

This automatically redraws the selected menu item 50 times a second
whenever it is displayed on the screen.  It's usually used in junction
with a menu procedure to generate animated menu items which change
infront of your eyes.

See EXAMPLE 16.11 in the MANUAL folder of your AMOS1.2 PROGRAM disk.
```

1.385 menu item movable

```
                Menu Item Movable level
Menu Item Movable(,,)

This command is simular to the
                MENU MOVABLE
                 command except that it
allows you to re-arrange the various options in a particular level.  So,
all the items in a menu bar may be individually repositioned by the
user.  See your manual page 230 for detailed information on this
command.
```

1.386 menu item static

```
Menu Item Static level
Menu Item Static(,,)

This command locks one or more menu items firmly into place and is the
default setting.
```

1.387 menu once

```
                Menu Once(,,)

Turns off the automatic updating system started using the command

                MENU CALLED
                .  From that point on, the menu item selected will only be
drawn once when on screen.
```

1.388 menu x

```
x=Menu X(,,)
This function allows you to retrieve the position of a menu item
relative to the previous option on the screen.  You can use this
information to implement powerful menus such as the one found in EXAMPLE
16.13 in the MANUAL folder on your AMOS1.2 PROGRAM disk.

y=Menu Y(,,)
Returns the Y coordinate of the menu option.  Note that all coordinates
are measured relative to the previous item.  So this is NOT a standard
screen coordinate.
```

1.389 embedded menu commands

```
                Important Info:

The following embedded menu commands can be added to your menus
by including them in round brackets () in the menu definition
string. (eg.
                Menu$
                (1)="(locate 10,10 : INK 1,1) Hello"  )

BOB
BOb n
Display BOB number n at the current cursor location.  The HOT SPOT of
the BOB is not taken into account.

ICON
ICon n
Draw icon number n at the current cursor location.

LOCATE
LOcate x,y
Place the graphics cursor at x,y measured relative to the top left
corner of the menu line.  After a graphics operation the new cursor
location will be at the bottom right corner of the last object drawn.

INK
INk n,index
Change the colour register of the PEN, PAPER or OUTLINE to index.  If
n= 1 then the PEN is affected, 2 for the PAPER and 3 for the OUTLINE.

SFONT
SFont n
Set current font to graphics font number n.

SSTYLE
SStyle n
Set font style to n.  n is a bit pattern.  Bit 0 sets underline, 1
sets bold and 2 sets italic.

LINE
LIne x,y
```

```
Draw a line from the current cursor position to graphics coordinates
x,y.

SLINE
SLine p
Set line style to bit pattern in p.  p should be entered as a decimal.

BAR
BAr x,y
Draw a rectangular bar from the current cursor location to x,y.

PATTERN
PAttern n
Set fill pattern to n.

OUTLINE
OUtline flag
If flag is 1 then outline mode is on.  0 turns it off.

ELLIPSE
ELlipse r1,r2
Draw an ellipse with the center at the current cursor position radii
or r1 and r2.
```

1.390 dir

```
Dir [PATH$][/W]

W will cause the directory to be printed in two columns.
```

1.391 dir$

```
s$=Dir$   Dir$=s$

Return or set the current directory.
```

1.392 parent

```
Parent

Jump up one directory.
```

1.393 set dir

```
Set Dir n [,filter$]

Set directory listings to n characters wide.  filter$ contains a list
```

```
of path names separated by / to be excluded from the directory list.
(eg. SET DIR 10,".INFO/*.INFO/*.*.INFO" )
```

1.394 dfree

```
f=Dfree

Returns the number of bytes free on the current disk.
```

1.395 mkdir

```
Mkdir f$

Make a new folder.
```

1.396 kill

```
Kill f$

Erase a file on disk.
```

1.397 rename

```
Rename old$ To new$
```

1.398 fsel$

```
f$=Fsel$(path$ [,default$][,title1$,title2$])

Calls up the file requester and returns the selected file name or ""
if QUIT was selected.
```

1.399 run

```
Run f$

Run program f$ from disk.
```

1.400 exist

```
                flag=Exist(f$)

Returns a value of -1(
                TRUE
                ) if file f$ exists on disk.
```

1.401 dir first$

```
file$=Dir First$(path$)

Returns the first file name at location path$.
```

1.402 dir next$

```
                file$=Dir Next$

Returns the name of the next file in the directory.
                DIR FIRST$
                 must
be called before this command.  If there is not another file, then ""
is returned.
```

1.403 open out

```
Open Out c,n$

Open channel number c for output to file n$.  If the file already
exists then the old one will be erased.
```

1.404 append

```
Append c,n$

Open channel number c for appending output to file n$.
```

1.405 open in

```
Open In c,n$

Open channel number c for input from file n$.
```

1.406 open port

```
Open Port channel,"PAR:"  (Opens channel to the Parallel interface.)
Open Port channel,"SER:"  (Opens channel to the Serial interface.)
Open Port channel,"PRT:"  (Opens channel to the Printer interface
                           chosen from Workbench Preferences.)

OPEN PORT allows you to communicate with external devices such as
printers.  All the standard sequential file commands can be preformed as
normal, except for commands like LOF or POF which are obviously only
relevant to disc operations.
```

1.407 port

```
                n=Port(channel)

Tests to see if an input device is ready to send you some information.
If the device is ready for you to read it, n will have a -1(
                TRUE
                )
value or a 0(
                FALSE
                ) value otherwize.
```

1.408 open random

```
                Open Random channel,f$

Opens a RANDOM ACCESS file called f$.  When you're using this
instruction, you should always define the record structure immediately
after using the
                FIELD
                 command.
```

1.409 field

```
Field channel,length1 As field1$,length2 As field2$,...

Field allows you to define a record which will be used for a random
access file.  This record can be up to 65535 bytes in length.
```

1.410 get

```
                Get channel,r

This loads the
                FIELD
```

selected strings with the information of out of
the random access file channel from record r. Note that you can only
use GET to retrieve records which are actually on disc. If you try to
grab a record which does not exist, then an error will be generated.

1.411 put

Put channel,r

PUT moves a record from the Amiga's memory [following the
FIELD
command's list of strings and sizes] to the record number in r to the
random access file channel. Before using the PUT command, the strings
defined by the
FIELD
command should be defined.

1.412 close

Close [n]

Close channel number n. IF n isn't supplied, Close will close all open
files.

1.413 print#

Print#c,variable list

Print data to channel c.

1.414 input#

Input#c,variable list

Input data from channel c.

1.415 line input#

Line Input#c,variable list LINE Input#c,separator,variable list
Same as
INPUT#
, but allows you separate your list of data using any
character you wish. If separator is omitted then the default is the
return character.

1.416 input$

```
x$=Input$(f,count)

Read count number of characters from device number f.
```

1.417 eof

```
                flag=Eof(c)

Returns -1(
                TRUE
                ) if the end of the file in channel c has been reached.
```

1.418 lof

```
length=Lof(c)

Returns the length of the file in channel c.
```

1.419 pof

```
pos=Pof(c)

Reads the current reading or writing position of the file in channel
number c.
```

1.420 lprint

```
Lprint variable list

Print a list of variables.
```

1.421 ldir

```
Ldir [path$][/W]

Print a directory.
```

1.422 amal important info

```
                Up to 16
                AMAL
                 programs can be run at the same time using
interrupts.  more may be executed but they will not be running on
interrupts.  Only the capital letters of an AMAL command are
significant.  Each
                AMAL
                 program has its own set of 10 internal
registers.  Each register starts with the letter R and is
followed by a number from 0 to 9 (eg. R1,R2,R3...).There are 26
external registers that can be accessed by other
                AMAL
                 programs or
directly from Basic.  They begin with the letter R and are
followed by another letter from A to Z (eg. RA,RB,RC...).

Special Registers
X and Y are internal registers and always contain the location of the
object being controlled by an
                AMAL
                 program.  Another internal register
is A.  It contains the number of the image that is displayed by a

                sprite
                 or
                BOB
                .

Operators

                AMAL
                 expressions can include all the normal arithmetic operations,
except MOD.  You can also use the following logical operations:
   &   logical AND
   |   logical OR
```

1.423 (amal) move

```
Move deltaX, deltaY, n

Move object to deltaX,deltaY relative to the objects current position
in n number of steps.  If deltaX is positive the object will move to
the right, else to the left.  If deltaY is positive then the object
will move down, else up.  The smoothest movements are made when both
deltaX and deltaY are multiples of n.
```

1.424 (amal) anim

```
                Anim cycles,(image,delay)(image,delay)...
```

```
Animate an object.  cycles is the number of times the animation will
be repeated.  If cycles is 0 then the animation will be looped until
told to stop.  image is the image number to be displayed and delay is
the time measured in 50ths of a second that it will remain until the
next image is displayed.  After an animation command has been
initialized,
                AMAL
                 will continue with the next instruction.
```

1.425 (amal) let

```
                Let reg=exp

Save a value in an
                AMAL
                 register.  Possible
                AMAL
                 registers are R0-R9
and RA-RZ.
```

1.426 (amal) jump

```
Jump L

Jump to label L.  Labels are defined as a single letter followed by a
colon.  A label name can be padded with lowercase letters to help them
read easier but make sure that the first letter of each label in an
AMAL program is different. (eg.  Start:  and  Shoot:  would be
considered the same thing and would cause an error.)
```

1.427 (amal) if

```
                If exp Jump L

Perform a test and if the result is
                TRUE
                , then Jump to label L.
exp can be any logical expression  =,  <>,  <,  >.  Its common
practice to pad out this instruction with lowercase commands like
"then" or "else". (eg. If X>100 then Jump Label else Let X=X+1 ).
 If exp Direct L     If exp eXit
The above variations of the If command are used by the
                (Amal)  AUTOTEST
                feature explained later.
```

1.428 (amal) for to next

```
                For reg=start To end ...Next reg

It is legal to have nested loops in an
                AMAL
                 program but the step size
of the loop is always set to one.  Only one step of a loop is
performed every vertical blank period.
```

1.429 (amal) play

```
                PLay n

Play series number n of movements stored in the
                AMAL
                 bank.  These
movements are created using the
                AMAL
                 accessory.  When the PLay command
is used R0 holds the tempo of the movements.  larger the number in R0,
the faster the object will move.  R1 controls the direction of the
movements.  If R1=1 then the movements are play forward.  If R1=0 then
the movements are played backward.  If R1= -1 then the movements are
stopped and
                AMAL
                 continues with the next command.
```

1.430 (amal) end

```
                End

Terminates the
                AMAL
                 program and turns off the
                (Amal)  AUTOTEST
                 feature if
its been defined.
```

1.431 (amal) pause

```
                Pause

Wait for the next vertical blank period.  It is good practice to use a
Pause command before a
                (Amal)  Jump
                .
```

1.432 (amal) autotest

```
                AUtotest (list of test)
AUtotest is a list of commands that are executed every 50th of a
second just before the main
                AMAL
                 program is run.  The possible
commands that can be used in an AUtotest list are:

Let
Let reg=exp
Same as the regular
                AMAL
                 command

Jump
Jump label
Same as the regular
                AMAL
                 command except label must lie inside the
Autotest list.

eXit
eXit
Leave Autotest and continue with the main program.

Wait
Wait
Turn off the main
                AMAL
                 program and only execute the AUtotest.

If
If exp Jump label   If exp Direct label   If exp eXit
Jump is the same as the regular
                AMAL
                 command.  Direct will Jump
to a part of the main
                AMAL
                 program after the AUtotest.  eXit will
leave AUtotest and continue with the main program.

On
On
Restart the main
                AMAL
                 program after a Wait command.

Direct
Direct label
After AUtotest is complete, the main
                AMAL
                 program will continue
at label.  label must lie outside of AUtotest.
```

1.433 (amal function) =xm

Returns the X hardware coordinate of the mouse.

1.434 (amal function) =ym

Returns the Y hardware coordinate of the mouse.

1.435 (amal function) =k1

Returns a value of -1(
TRUE
) if the left mouse key has been pressed.

1.436 (amal function) =k2

Returns a value of -1(
TRUE
) if the right mouse key has been pressed.

1.437 (amal function) =j0

Returns a bit map containing the right joystick status. See
JOY
for
more details.

1.438 (amal function) =j1

Returns a bit map containing the left joystick status. See
JOY
for
more details.

1.439 (amal function) =z(n)

Returns a random number from 0 to n.

1.440 (amal function) =xh (s,x)

Converts screen coordinate x on screen s to a hardware coordinate.

1.441 (amal function) =yh (s,y)

```
Converts screen coordinate y on screen s to a hardware coordinate.
```

1.442 (amal function) =xs(s,x)

```
Converts hardware coordinate x to a screen coordinate on screen s.
```

1.443 (amal function) =ys(s,x)

```
Converts hardware coordinate y to a screen coordinate on screen s.
```

1.444 (amal function) =bob col(n,s,e)

```
                Identical to the
                BOB COL
                 command.  Returns a value of -1(
                TRUE
                ) if
                BOB
                number n has collided with
                BOB
                s s to e.  This command can not be used
with an interrupt driven
                AMAL
                 program.  See
                SYNCHRO
                 for information on
non-interrupt
                AMAL
                 programs.
```

1.445 (amal function) =sprite col(n,s,e)

```
                Similar to
                (Amal Function)  =Bob Col(n,s,e)
                , above.
```

1.446 (amal function) =c(n)

```
                Returns a value of -1(
                TRUE
                ) if object n has collided with another
object.  For use after an SpriteCol or BobCol command.
```

```
See
                (Amal Function)  =Bob Col(n,s,e)
                 or
                (Amal Function)  =Sprite Col(n,s,e)
                .
```

1.447 (amal function) =v(v)

```
Returns the volume (0-255) of the current voice.
```

1.448 amal

```
Amal n,a$
Assign a$ to AMAL channel number n.  If n > 16 then the AMAL program
will not be interrupt driven.

Amal n,p
Assign AMAL program number p in the AMAL bank to AMAL channel n.

Amal n,a$ To address
Copy the contents of registers X, Y and A into an area of memory
starting at address.  The information will be saved like so:

  Location     Effect
  Address      Bit 0 is set to 1 if the X has changed.
               Bit 1 is set to 1 if the Y has changed.
               Bit 2 is set to 1 if the A has changed.
  Address+2    Is a word containing the latest value of X.
  Address+4    Holds the current value of Y.
  Address+6    Stores the value of A.
```

1.449 amal on

```
                Amal On [n]    Amal Off [n]

Turn on all
                AMAL
                 programs or just program number n.
```

1.450 amal freeze

```
                Amal Freeze [n]

Stops all
                AMAL
                 programs or just number n.  The
```

```
                AMAL
                 programs can be
restarted with the
                AMAL ON
                 command.  Its a good idea to freeze all

                AMAL
                 programs before disk access.
```

1.451 amreg

```
                r=Amreg(n [,c])    Amreg(n [,c])=exp

Read an
                AMAL
                 register or pass a value to an
                AMAL
                 register. n is the
number of the register.  Possible values range from 0 to 25
representing RA to RZ.  If c is present then the internal registers of
the
                AMAL
                 program in channel c can be accessed.  In this case n must be
between 0 and 9 representing R0 to R9.
```

1.452 amplay

```
                Amplay tempo,direction[s To e]

Set parameters for
                AMAL
                 play sequences in all channels or just
channels s to e.  tempo set the delay in 50ths of a second between
each movement.  direction is as follows:

    Value   Direction
     >0      Forwards
      0      Backwards
     -1      Stop movement pattern and continue with next
                AMAL
                 command
```

1.453 chanan

```
                s=Chanan(c)

Returns a value of -1(
                TRUE
                ) if the Anim sequence in channel c is still
active.
```

1.454 chanmv

```
                s=Chanmv(c)

Returns a value of -1(
                TRUE
                ) if the Move command in channel c is still
active.
```

1.455 amalerr

```
                p=Amalerr

Returns the position of an error in the current
                AMAL
                 program.
```

1.456 channel

```
                Assign an
                AMAL
                 channel to an object.
```

1.457 channel n to sprite s

```
                Assign sprite number s to
                AMAL
                 channel n.  The X and Y registers
in the
                AMAL
                 program will now control the hardware coordinates of
the sprite.
```

1.458 channel n to bob b

```
                Assign BOB number b to
                AMAL
                 channel n.  The X and Y registers in
the
                AMAL
                 program will now control the screen coordinates of the
BOB.
```

1.459 channel n to screen display d

```
                Assign screen number d to
                AMAL
                 channel n.  The X and Y registers
in the
                AMAL
                 program now control the screen position in hardware
coordinates.
```

1.460 channel n to screen offset d

```
                Assign screen number d to
                AMAL
                 channel n.  The X and Y registers
in the
                AMAL
                 program now control the screen offset.
```

1.461 channel n to screen size s

```
                Assign screen number s to
                AMAL
                 channel n.  The X and Y registers
in the
                AMAL
                 program now control the width and height of the
screen.
```

1.462 channel n to rainbow r

```
                Assign rainbow number r to
                AMAL
                 channel n.  The X register
controls the BASE of the rainbow and Y controls the starting line
and A stores the height.  X and Y are hardware coordinates.
```

1.463 update every

```
                Update Every n

Force
                AMAL
                 programs to update only every n 50ths of a second.  This
may free up some time for the main program and a result could make
things run faster.
```

1.464 rain

```
Rain(n,line)=c
c=Rain(n,line)

Set or Read the colour at line from rainbow n at any time.
```

1.465 rainbow

```
Rainbow n,base,y,h

RAINBOW activates Rainbow n starting base lines into the rainbow
defination at hardware screen location y with a length of h lines.

See your manual page 141 for further information reguarding safeguards.
```

1.466 set rainbow

```
                Set Rainbow n,colour,length,r$,g$,b$

SET RAINBOW defins a rainbow effect which can be subsequently displayed
using the
                RAINBOW
                 command.  It works by changing the shade of a colour
according to a series of simple rules.

n is the number of your rainbow.  Possible values range from 0 to 3.

colour is the colour index which will be changed by the instruction.
Note only 15 colours can be manipulated in this manner.

length is the size of the table used to store your colours.  There's one
entry in this table for each colour value on the screen.  The size can
range from 16 to 65500.  If length is less than the physical height of
your rainbow, then the colour pattern will be repeated for the full
length.

r$, g$, b$ are command strings which change the intensities of red,
green and blue respectively.  Each entry controls a single screen line.

Each string can contain a whole list of commands.  These will be cycled
until the final rainbow pattern is produced.  The format is:

"(n,step,count)(n,step,count)..."

n sets the number of lines to be assigned to the specific colour value
in the rainbow.

step holds the number to be added to the colour component [RGB] mod 15.
     [Each component can go from 0 to 15, if a value exceeds 15, it is
      set to 0.  As well, if a value goes below 0, it's set to 15.]
```

count is the number of times the current operation is to be repeated.

1.467 synchro

```
                Synchro Off   Synchro On    Synchro

Release
                AMAL
                 updating to Basic control.  If more then 16
                AMAL
                 channels
are going to be used then SYNCHRO must be used.  First, call SYNCHRO OFF
before defining the
                AMAL
                 programs.  Then use SYNCHRO to activate the
next step in all the
                AMAL
                 programs.  To return to normal interrupts
use SYNCHRO On.
```

1.468 move x

```
                Move X n,m$
Move Y n,m$

Define a movement for animation channel n.

m$ is the defination of movements.

Definations are as follows:

"(speed,step,count)(speed,step,count)...[E#/L]"

The E directive allows you to have your object stop when it reaches a
certain location on the screen, such as "E100" will ensure that the
current x or y stops at 100.

The L directive allows the MOVE command to loop back to the first entry.

speed is the delay in 50ths of a second between each step.  speed's range
is 1 (mega fast) and 32767 (super mega slow).

step is the amount the object will move.  [Positive or negative pixels.]

count is the amount of times you want to do this movement.

Example:  Move X 1,"(3,-4,10)E100"  will move the X -4 pixels every
          3/50th of a second for 10 times.  E100 will ensure that X never
          goes beyond 100.

See
                MOVE ON-OFF
```

for more details.

1.469 move on-off

```
                Move On-Off [n]
Move On [n]
Starts MOVE n or all movements.  [Previously created using either the

                MOVE X
                 or MOVE Y command.]

Move Off [n]
Stops MOVE n or all movements.  [They can't be resumed after a stop.]
```

1.470 move freeze

```
Move Freeze [n]

This will stop movement n or all movements.  [This is not perminant like
the MOVE OFF command.]

To resume movements, use the MOVE ON command.
```

1.471 movon

```
                x=Movon(n)

This returns a -1(
                TRUE
                ) if the current channel has movement [a MOVE
command is running, not an AMAL movement].  Otherwize a 0(
                FALSE
                ) is
returned.
```

1.472 anim

```
                Anim n,a$

Define an image animation for animation channel n.

a$ is the defination of the image animation, which is executed every 50th
of a second like AMAL animations.  Except these animations change the
object's image # [in a
                Sprite
                /
                Bob
```

```
                 Bank].

Defination are as follows:

"(image,delay)(image,delay)...[L]"

The L directive allows the ANIM command to loop back to the first entry.

image is the image # within a
                Sprite
                /
                Bob
                 Bank to be displayed for
the delay period.

delay specifies the amount of time the image remains (in 50ths of a
second).

Also see
                ANIM ON-OFF
                .
```

1.473 anim on-off

```
                Anim On-Off [n]
Anim On [n]
Starts ANIM n or all image animations.  [Previously created using the

                ANIM
                 command.]

Anim Off [n]
Stops ANIM n or all image animations.  [They can't be resumed
after a stop.]
```

1.474 anim freeze

```
Anim Freeze [n]

This will stop movement n or all movements.  [This is not perminant like
the ANIM OFF command.]

To resume movements, use the ANIM ON command.
```

1.475 track load

```
Track Load "Name_Of_The_Module",Bank

Load up a tracker module into a chip memory bank. It will of course it
reserve the bank for you, and choose the correct bank size.
```

If AMOS detects a Startracker synthetic instrument file (which must have
a ".NT" extension appended to the file name), it will load it AS WELL
into the bank. In fact you don't have to worry about it.

1.476 track play

Track Play [Bank],[Pattern]

Plays a tracker module loaded into a bank.

"Bank" is the number of the bank to be played. If omitted, it is the last
loaded with the
TRACK LOAD
instruction, or upon running, bank number 5.

"Pattern" is the first pattern to be played. Use this with caution, as NO
CHECK is done on the number of the pattern. You can very simply crash the
computer by giving a bad value. This parameter is intended to allow you
to have more than one music in a tracker bank...

1.477 track loop on-off

Track Loop On-Off

Enable or disable looping when the tracker music is finished.

1.478 track stop

Track Stop

Stops a tracker music being played.

1.479 important tracker notes:

The Tracker-playing instructions are implemented to give you a ↩
quick way
of playing modules. It is not as powerful as the normal AMOS music system.

For example:

* Do not play a normal AMOS music while playing a tracker module,
this can lead to unpredictable results.

* Do not start any sample, or sound effect when a Tracker module is played.

* A Tracker module uses and initialises all four voices, even if your music

```
  is only on 3 or 2. So do not play any sample on the other voices you think
  are free. They are not!

* VOLUME instructions do not have any effect on the Tracker music, but
  =
                VUMETER
                 works fine with a Tracker module.

... if you want to make sound effects while a music is playing, then you
should use the Soundtracker converter, and the normal AMOS Music system...
```

1.480 sload

```
                Sload File_Number,Length To Address

A new instruction intended to load parts of a sample, but it can be used
in many other ways.

This instruction is an extended
                BLOAD
                .

"File_Number" is the number of a file opened previously with the

                OPEN IN
                 File_Number,"Name" instruction.

"Length" is the number of bytes to load. If this number is bigger than
the actual size of the file, then only the remaining bytes are loaded,
without errors. You'll get an error if you try to load once more after
the end has been reached.

"Address" is the destination address. Of course, the memory must have
been previously reserved.

The advantage of this instruction, is that you can set the position of
the file pointer with the
                POF
                ()= instruction before using SLOAD. As you
can see, this instructions can have a lot more usage than just loading
samples...
```

1.481 sam swap

```
                Sam Swap Voices To Address,Length

This instruction initialise the sound-swapping. The actual swapping will
only occur when the actual buffer has been totally played through the
speaker. The swap is done under interrupts, so you will not hear any tick
in the sample.

"Voice" is a bit pattern to define the voice concerned, just like in the
```

```
                SAM RAW
                 instruction.

"Address" is the address of the next buffer to play. Of course, it must
be in chip memory.

"Length" is the number of bytes to play.
```

1.482 sam swapped

```
                =Sam Swapped(Voice_Number)

This function returns a boolean value (
                TRUE
                 -1, or
                FALSE
                 0). It is the
key function in synchronising double buffer players.

"Voice_Number" is the number of the voice you want to have information
about (0 to 3). Do not make mistake, is it NOT a bitpattern.

It returns
                TRUE
                 if the sample swapping has occurred, it means the new
buffer you have initialised with the
                SAM SWAP
                 instruction is being
played at the very moment. It returns
                FALSE
                 if not.

Practically, you can only load a new part of the sample in the free
buffer, when the SAM SWAPPED instruction returns you a
                TRUE
                 value. It
returns
                FALSE
                 if the sample swapping has NOT happened.
```

1.483 sam stop

```
                Sam Stop [Voice_Pattern]

This simple instruction seems to have been forgotten in the instruction
set since the beginning. The only way to stop a sample playing, was to
use the PLAY instruction!

"Voice_Pattern" is a bit pattern defining the voices to be stopped, like
in the
                SAM RAW
```

instruction. All voices will be affected if you omit it.

NOTE: it is perfectly possible to have an AMOS music bank playing on 2
voices, and double-buffered samples playing on the last 2. (hey, but not
a Tracker module, if you remember what I told you!)

1.484 author note on =col(bob)

Well, in fact, it is not a new instruction, but a good enhancement ↩
to the
collision detection method.
To detect a collision, I remind you, you have to use one of the collision
detection functions (=
Bob Col
(), =
Sprite Col
(), =
Bobsprite Col
(),
=
Spritebob Col
()). When this function returns a
TRUE
value, you have
to explore the =
Col
() reserved array to find out which bob or sprite
created the collision.
The problem, is that you had to write a loop exploring sequentially all
the
Col
() array. This was eating a lot of processor time. You could see
certain games slowing down when some bobs were colliding.

I wanted to do something to correct it. But what? The problem with AMOS,
is that if I change the syntax or the behaviour of one instruction to
please certain people, it may (and surely will) not be compatible with
thousand of existing programs!
So I found a -rather tricky- solution to this problem.

=
Col
(Number) behaves normally if you send it a POSITIVE number as an
argument, this keeps it compatible with all existing programs.

If "Number" is negative, AMOS will first remove the sign, i.e. turn it
into a positive number.
Then it will explore the
Col
() array himself, and find the first
non-zero value higher than -Number. Then it will not return
TRUE
or
FALSE

```
                ;
as it used to, but the actual number of the bob colliding.
To get the next bob colliding, simply call it again with a this number,
minus one..

Example, this small and fast loop, will give all bobs colliding with bob
zero:

B=
                Bob Col
                (0)
BB=0

                Repeat
                   BB=
                Col
                (-(BB+1))

                If
                 BB

                Print
                 "Bob";BB;" is colliding..."

                End if

                Until
                 BB=0
```

You can certainly remark that with this method, it is impossible to get the collision of bob number zero. That’s why I say this was not perfect...

1.485 disc info$

```
                =Disc Info$("Name")
```

...is a new instruction that returns information on any disc.

"Name" is the name of a file or a directory of the disc you want to have information about. The string returned has the following form:

"NAME_OF_THE_DISC:XXXXXXX" , where XXXXXXX is the free space on the disc.

To get both, use this simple method:

```
A$=DiscInfo$("Df0:")
C=
                Instr
                (A$,":")
N$=
                Left$
                (A$,C)
F=
                Val
                (A$,C+1)
```

```
Print
 "Name of the disc :";N$;" Free space:";F
```

1.486 prg state

```
=Prg State (returns the current status of a program)

=Prg State

This little handy function let you know how your program was launched.
It returns three possible values:

  0 : if your program was run under the AMOS interpreter.
  1 : if your program was run under RAMOS run-only.
 -1 : if your program is compiled.
```

1.487 bgrab

```
Bgrab b

b is the bank that Bgrab "borrows" from the current program being
edited.  [This only works from within an Accessory.]  If there is a bank
already in the accessory, it's erased and replaced with the new one.
When you exit the Accessory, the "borrowed" bank will be returned to the
main program along with any changes.  [Bank #'s 1 to 15.]

Note:  This instruction can only be done in an Accessory.  If you
attempt it otherwize, you'll get an appropriate error.
```

1.488 prun

```
Prun "filename"

This is identical to choosing the Run Other from the Editor's menu.

Also, all screens/sprites/bobs/music will need to be kept prior to using
the PRUN command and restored by your program after the PRUN has
finished, to ensure your data remains intact.  See EXAMPLE 3.3 in the
MANUAL folder on your AMOS1.2 PROGRAM disk.
```

1.489 prg first$

```
p$=Prg First$

This returns the name of the first AMOS basic program in memory [loaded
with the Load Other editor option].  It's used in conjunction with the
```

```
                PRG NEXT$
                 command to create a full list of available programs.
```

1.490 prg next$

```
                p$=Prg Next$

This is used after the
                PRG FIRST$
                 command to continue to read the
available AMOS basic programs in memory [loaded with the Load Other
editor option].  When the list is complete, a string of "" will be
returned.
```

1.491 psel$

```
                n$=Psel$("filter",[default$,title1$,title2$])

PSEL$ calls up a program selector simular to the one used by Run Other,
Load Others, Edit Others and New Others.  This can be used to select a
program in the usual manner.  The selected file will be returned in n$,
which can be
                PRUN
                .  If QUIT was selected, n$ will be set to "".

filter sets the type of programs which will be listed by this
instruction.  These can be:

"*.ACC"   List Accessories.
"*.AMOS"  List Amos programs.
"*.*"     List All files.

For further details, see the
                FSEL$
                 and
                DIR
                 commands.
```

1.492 getting the system time

```
                You will find on your updated disc (1.34) a program called " ↩
                   GET_TIME.AMOS".
This program includes two procedures to get the time and date from
the system.

  Time.
  -----
Call the procedure, and you'll have in Param$ the current clock time under
the following format: 00:00:00
```

```
_TIME$

                Print
                 Param$
...
11:04:04
```

1.493 getting the system date

```
                You will find on your updated disc (1.34) a program called " ↩
                   GET_TIME.AMOS".
This program includes two procedures to get the time and date from
the system.

  Date.
  -----
This procedure returns the current date in Param$:

_DATE$

                Print
                 Param$
...
21/06/1991
```

1.494 safe amigados execute

```
                Safe AmigaDos EXECUTE.

It is perfectly possible to launch an external program from AMOS. But in
order to do so, you have to know some of the AmigaDos internal functions.
That's why you will find on your updated disc a small program with a
procedure called _EXECUTE.
Just transmit an AmigaDos command to this procedure, and it will launch
it out of AMOS. To run an external program, use:

_EXECUTE["RUN >NIL: <NIL: Program_Name.AMOS"]

You can launch other CLI commands (like "Assign"). Of course
if you want to see the display, you have to perform an
                AMOS TO BACK
                instruction.

As the Amiga is a multitask machine, your AMOS program will go on running
as well as the launched program. Of course the speed will be bit reduced,
depending on the other program.
```

1.495 no icon mask

```
                NO Icon MASK [number] (Remove the mask from an icon).

This instruction has simply been forgotten in the manual. It simply
does the same job as
                NO MASK
                , but for icons.
```

1.496 rainbow del

```
Rainbow Del [Number] (Delete on or all rainbows).

Another instruction forgotten in the manual, but very useful when you
want to get rid of a rainbow!
"Number" specify the number of the rainbow to remove, or all if omitted.
```

1.497 multi wait

```
Multi Wait  (Force a multi-task wait vbl)

To make effective multi-tasking programs, you must not grab most of the
processor time, leaving only a limited amount of power for other tasks.
MULTI WAIT does a MULTI-TASK wait vbl. You should use it in your
programs main loop, when you wait for example, for a menu item to be
selected.

   Note that you should not use this instruction to make accurate
screen synchronisation as it is designed to multi-task. This instruction
is not consistent at all! It may skip many VBLs, depending on the number
of running tasks at the time.

   If you missed it elsewhere in the manual, Multitasking can be
activated by pressing Amiga+A to flick between AMOS and the CLI or
Workbench environments. This allows systems with at least 1 meg to run
AMOS and programs like DPaint III at the same time!
```

1.498 amos to back

```
Amos To Back  (Hide AMOS from view and show the Workbench)

This will bring forward the Workbench display, allowing you to access
other programs.
```

1.499 amos to front

```
Amos To Front  (Switch AMOS to the display)

AMOS is forced back onto the display with this command, leaving the
Workbench hidden.
```

1.500 amos here

```
                x=Amos Here  (Report which task is on display)

This returns
                TRUE
                 if AMOS is currently displayed and
                FALSE
                 if the
Workbench is in view.
```

1.501 amos lock

```
                Amos Lock (Locks AMOS in front position)

This instruction first does an "
                AMOS TO FRONT
                ", and then disable
the AMIGA-A system. Use this instruction if you do not want people
to know your program was written in AMOS.
```

1.502 amos unlock

```
Amos Unlock (make AMIGA-A active)

Just restores the AMIGA-A Workbench/AMOS flipping. You may want people to
stay under AMOS during certain parts of your program for example, to
see your name (!) and then free them.
```

1.503 bank swap

```
Bank Swap number1,number2

This instruction will swap the pointers of the two banks. Useful if you
want to turn an icon bank into a sprite bank. Example:

   Bank Swap 1,2

or have more than one music bank at the same time, for example:

   Bank Swap 3,5
```

1.504 laced

```
                Laced (Function to open an interlaced screen)

To open an interlaced screen use the following syntax:


                Screen Open
                 0,320,200,16,LACED [+HIRES] [+LOWRES]

LACED is a function that returns 4.

Important: As soon as one screen is opened with Interlace, all the other
screens turn to interlace. The interlacing will only truly effect the
screen actually opened with LACED. All the others will just have their
vertical lines doubled on the screen to adjust to the special mode.

   Interlaced mode is perfect for displaying pictures, but
animation runs at half normal speed. Games should not be written in
Interlace!

   As soon as the last interlaced screen is closed the whole
display returns to normal mode. Your TV monitor might not like lots of
fast switching from normal mode to Interlace, so you are advised not to
do this excessively.

   All normal operations are available in interlaced screens:

                SCREEN OFFSET
                ,
                SCREEN DISPLAY
                 and so on. The only problem that arises is
due to interlacing being software driven in AMOS. The bitplanes are
changed during the vertical blank and this particular interlace process
is forbidden during copper list calculation.

   So if you have a large copper list (i.e.. four screens, one
interlaced, and a rainbow), and have a copper calculation to do, the
interlaced screen will display only half of the picture during the
calculation. Nothing can be done to solve this, it is simply a
limitation of the whole system.

There are two extra screen commands in AMOS now. These allow a program
to work out what type of display it is being run on:
```

1.505 display height

```
=Display Height  (Report how tall a screen can be)

This command returns 311 in PAL and 263 in NTSC.
```

1.506 ntsc

```
                =Ntsc (Flags the type of display in operation)

This returns
                TRUE
                 if the system is in NTSC mode or
                FALSE
                 if in PAL.
Ideal for international software development!

   NTSC refreshes the screen at 60 times a second whereas PAL
screens refresh at only 50 times a second. However, AMOS1.3 compensates
for this and now music runs at exactly the same speed in PAL and NTSC


                AMAL
                 also relies on the interrupt routine but is not slowed down
to comply with PAL speeds. You must therefore be careful not to
synchronise music and animations by just relying on the speed they run
at. Check that an animation frame has been reached or the music has
played a certain note. Using this technique you'll ensure the software
executes at the right points on all systems.
```

1.507 request on

```
                Request On (Generate a requester routine)

This will make AMOS use its own requester routine and is the
default.

See
                REQUEST WB
                 for important usage notes.
```

1.508 request off

```
                Request Off

AMOS will always select the CANCEL button of the requester if this
command is used. The actual requester will not be displayed, so this is
ideal for error trapping within a program.
See
                REQUEST WB
                 for important usage notes.
```

1.509 request wb

```
Request Wb

This tells AMOS to switch back to Workbench's system requester. You'll
```

come back to AMOS as soon as you have chosen one of the options.

Note: If you don't load up the Requester (by deleting it from the extension list using the config file), the normal Workbench requester will be used for displaying messages.

This does have a bad side-effect though, AMOS will seem to have crashed when a requester appears. If this happens you must simply press Amiga+A to return to the Workbench, answer the question and press Amiga+A once again to return to AMOS. It's only best to avoid loading the requester when memory is very low!

1.510 bob-sprite flipping

Bob and sprite flipping commands.

In a great number of games, the main character needs to animate left to right, and up and down. Up to now, you were obliged to keep in the sprite bank reversed copies of small animation sequences for the main character. As the main character usually has the best animation, you lose an enormous amount of space!

For the game RanXerox, for which AMOS author François Lionet wrote the sprite routines, a flipping routine was developed which allowed just one copy of the main character to be kept in the bank. This routine has been enhanced and placed into AMOS.

How does it work? Imagine your character is walking to the left and then back to the right. You would only have in your bank the image of him walking to the right. To display this right image, you simply refer to the image number in the bank as usual.

To display the image reversed in the X axis (left walking image), you set bit number 15 of the image number to 1. Don't panic, you can simply do it with:

```
$8000+Image number
```

So:

```
Bob
 1,160,100,1
```

will display your character walking right, and:

```
Bob
 1,160,100,$8000+1
```

will display it walking left. The same principle is used for vertical reversing. For this, bit number 14 is used - add $4000 to the image number. To have a vertical and horizontal reversing, use $C000.

The symmetry is a full symmetry: The hot spot of the bob is also

reversed. For example, if we had put the hot spot in X under the feet of our character, the reversed version would also have it under his feet. So be careful if you set the hot spot on the top left corner on a bob, the reversed image will be displayed at the top left!

You might say that $8000 and $C000 are a bit weird to use. We have provided special functions to give a better AMOS interface:

```
=Hrev(image)      adds $8000 to the image
=Vrev(image)      adds $4000
=Rev(image)       adds $C000
```

Use them in place of the hex values:

```
Bob 1,160,100,10
Bob 1,160,100,Hrev(10)
Bob 1,160,100,Vrev(10)
Bob 1,160,100,Rev(10)
```

To allow easy use of the bob flipper in AMAL, we have implemented Hexadecimal evaluation. So you can use hex notation to refer easily to reversed bobs. If hex frightens you, just add $8000, $4000 or $C000 before all references in your AMAL strings. Example:

Old

AMAL string:

```
"Anim 0,(1,2)(2,2)(3,2)(4,2)"
```

New reversed string:

```
"Anim 0,($8000+1,2)($8000+2)($8000+3)($8000+4)"
```

or

```
"Anim 0,($8001,2)($8002,2)($8003,2)($8004,2)"
```

If you use a register to calculate the image number, do not try to modify the calculation itself, but only when you assign the register to the image.

Old

AMAL string:

```
For R0=1 To 10; Let A=R0; Next R0
```

New one:

```
For R0=1 To 10; Let A=$C000+R0; Next R0
```

How does the flip routine work?

It is really important for you to understand how it works internally, so that you do not ask this system to do things it is not designed to do.

The reversing system is designed to free memory before trying to be fast (although we would not mind if it was actually fast, would we?). Concessions had to be made to have it fast, and at the same time easy and powerful.

The routine actually works right in the middle of the bank, and does not use any extra memory. The bobs are flipped during the update process, just before a bob is redrawn on the screen. AMOS looks to see if the image needs to be flipped in the bank. If it does, it is flipped and a flag is set within the bank. On the next update, if the bob image has not changed, it will not be flipped, thus saving a lot of time.

If you understand the above, you will also realise one big limitation. It is not wise to use more than one flipped bob pointing to the same image. Let's see the next example:

```
Bob
 1,160,100,1

Bob
 2,160,150,$8001

Bob
 3,20,20,$4001

Bob
 4,20,100,$C001

Update
During the
UPDATE
 process, AMOS will first draw bob number 1. No
```

problem, it is in the right position. Then bob number 2 - AMOS needs to reverse it in X. Bob number 3 needs a Y and an X reversing (to put the bob back to normal in X!). Then bob number 4 needs an X flipping.

On the next update, providing the bob's image has not changed, to display bob 1, AMOS will have to flip it in X and Y, then bob 2...

As you can see, for each

```
UPDATE
```

, that is, every 50th of second, if the bobs move they need to be reversed! This will work, but will take a lot of processor time, and the animation will be disastrous.

So the golden rule is, use the reversed bobs for objects alone on a screen (or be sure that normal and reversed images are not displayed at the same time on the screen). If you want, you can have two bobs like this - experiment!

We told you before that this system was for use with bobs. Yes, it is totally automatic with bobs. But as it directly affects the sprite bank, you can also use it with sprites.

When a hardware computed sprite is calculated, AMOS looks into

the sprite bank and gets the image from it. If the image is reversed at that moment, the hardware sprite will display a reversed image. You can therefore have reversed hardware sprites using this method. But you cannot do this for example:

```
Sprite
 1,200,200,$8001
```

Pasting flipped bobs

```
PASTE BOB
```
also accepts reversed images. A simple trick to reverse an image in the bank without having to display a bob, is to PASTE the reversed image outside of the screen. Example:

```
Paste Bob
 500,500,$C000
```

This will reverse image 4 in the bank, without any output (and quite fast).

Collision detection

This is an important point, and you have to be very careful when you detect collisions with reversed bobs!

The collision detection uses the shapes in the bank at the very moment it is called. Let's see an example that will never work:

```
Bob
 1,160,100,1
Do

Bob
 2,XScreen(XMouse),YScreen(YMouse),$8001

Wait Vbl

Exit
 if
Bob Col
(1)
Loop
```

Why doesn't it work? We have two reversed images of the same definition in the bank. After the updating process, the image in the bank is left reversed. So the

```
Bob Col
```
instruction will take bob shape 1, the reversed image, and this will not work!

So remember: Thou shalt never use collision detection with more

than one reversed image on the screen!

How is it coded into the sprite bank?

Two bits of each images X Hot Spot are used to flag the flipping (at

SPRITE BASE
+6).

Bit number 15 for X 0 if normal, 1 if reversed
Bit number 14 for Y 0 if normal, 1 if reversed

Before RUN and SAVE, the bank is restored to its normal state, so that it is still compatible with version 1.1.

1.511 hrev block

Hrev Block (Flip a block horizontally)

Hrev Block image

Flips block number image horizontally.

1.512 vrev block

Vrev Block (Flip a block vertically)

Vrev Block image

Flips block number image vertically.

1.513 (bob) priority reverse on-off

Priority Reverse On-Off (Change the order in which Bobs are printed to the screen)

Priority Reverse On
Priority Reverse Off

Priority Reverse On, reverses the entire bob's priority table. This means that bob number 1 will be the first one drawn in front of all other bobs, 2 will come in second etc... This priority list is compatible with STOS.

   This instruction has another feature when used in conjunction with the PRIORITY ON command. The bobs are not printed from TOP to BOTTOM any more, but from BOTTOM to TOP! The highest bob on the screen will be displayed in front of the others.

1.514 serial open

```
                Serial Open (Opens a channel for Serial I/O)

Serial Open channel, port_no [,shared, xdisabled, 7wires]

Opens a communication channel to a serial device.

Channel This is an identification number which will be used for all
subsequent communication commands. Allowable values range from 0 to 3.

Port_no Specifies the logical device number of the serial port.
Normally, this value should be set to zero. However, if you've plugged a
MULTI SERIAL card into your Amiga, you can access your additional ports
using the numbers from one onwards.

Shared (optional) This is a flag which informs AMOS that the device can
be shared with other tasks which are currently running on your Amiga.
It's used in multitasking. A value of
                FALSE
                 (zero) will grab the channel
for AMOS Basic, and will deny access to any other program. If it's is
set to
                TRUE
                 (-1), the serial port can be shared between several programs
in memory. Beware: This system must be used with extreme care or the
Amiga could easily crash!

Xdisabled (optional) Toggles XON/XOFF checking during transmission of
your data over the serial line. It's essential to set this flag when you
are first opening the device, even if it will only be required later.
The default value is
                FALSE
                 (0) and disables the system. If you want to
enable the checking, use a value of
                TRUE
                 (-1). After the port has been
opened, you'll then need to set the XON and XOFF characters using a
separate call to the Serial X command.

7Wires (optional) A value of
                TRUE
                 (-1) tells the device to use the 7
wires system as explained in the official Commodore documentation. The
default is
                FALSE
                 (0).

When you call the Serial Open command for the first time, it will
automatically load the SERIAL.DEVICE library from your system disc. So
make sure it's available from the current drive.

Default settings depends on the number in "Port_no":

-Port_no=0 refers to the default serial port, it will be opened using the
parameters set in the "Preference" workbench program. You should open this
```

port if you use the workbench.

-Port_no=1 refers to the built in serial port. Every Amiga has one. This is the port you should open. The default settings will be set for this port to use the French minitel: 1200 Baud, 7 bits, 1 stop bit, Even parity. This can be easily changed using the
Serial Speed
,
Serial Bits
or

Serial Parity
instructions if required.

-Port_no>1 can only be used if you have a multi-serial card.

1.515 serial close

Serial Close (Closes one or more serial channels)

Serial Close [channel]

If you don't include the channel number, Serial Close will close all currently opened serial channels with absolutely no error checking. The optional channel number allows you to close a single channel and uses all the normal error checks.

Note: Whenever a program is RUN from AMOS Basic, any opened channels will be automatically closed for you.

1.516 serial send

Serial Send (Output a string via a serial channel)

Serial Send channel, t$

Sends the string t$ straight to the specified serial channel. It does not wait for the data to be transmitted through the actual port. You'll therefore need to use the =
Serial Check
(Channel) function to detect when
the transmission has been completed.

1.517 serial out

Serial Out (Outputs a memory block via a serial channel)

Serial Out channel, address, length

This is identical to

```
                SERIAL SEND
                 except that it works with RAW data.

Address is the address of your data in memory.
Length is the number of bytes to be sent.
```

1.518 serial get

```
Serial Get  (Gets a byte from a serial device)

=Serial Get(channel)

Reads a single byte from the serial device. If nothing is available a
value of -1 will be returned.
```

1.519 serial input$

```
Serial Input$  (Gets a string from the serial port)

=Serial Input$(channel)

Reads an entire string of characters from the serial port. If there's no
data, the command will return an empty string "". Otherwise the string
will contain all the bytes which have been sent over the serial line up
to the present time.
   Be careful when using this command with high speed transfers
(such as MIDI). If you wait too long between each Serial Input$ command,
you can overload the system completely, and generate annoying errors
such as "string too long" or "serial device buffer over-run".
```

1.520 serial speed

```
Serial Speed  (Sets the transfer baud rate for a serial channel)

Serial Speed channel, baud rate

Sets the current transfer rate of the appropriate channel. The same
value will be used for both reading and writing operations. Note that
you can't set split baud rates for a single channel. If the baud rate
you have specified is not supported by the current device, it may be
rejected by the system.
```

1.521 serial bits

```
Serial Bits  (Sets the Nbit & Stopbit part of a protocol)

Serial Bits channel, nbits, stopbits
```

```
Assigns the number of bits which will be used for each character you
transmit.

Nbits is the number of bits
Stopbits is the number of STOP bits
```

1.522 serial parity

```
Serial Parity  (Sets the parity checking, correct version)

Serial Parity channel, parity

Sets the parity checking you are using for the current serial channel.
Here's a list of the available options.

Parity can have 5 different states:

   -1 : no parity
    0 : EVEN parity
    1 : ODD parity
    2 : SPACE parity
    3 : MARK parity

See the Commodore documentation for a full explanation of this system.
```

1.523 serial x

```
                Serial X  (Sets XON/XOFF)

Serial X channel, xmode (Activates/deactivates the XON/XOFF handshaking
system)

A value of
                TRUE
                 for Xmode disables handshaking. Any other value turns it
on. Xmode should be loaded with the correct control characters. These
must be specified in the following format:

   Xmode=XON*$10000000+XOFF*$10000

Check Commodore's documentation for more information.
```

1.524 serial buffer

```
Serial Buffer  (Sets the size of the serial buffer)

Serial Buffer channel, length

Allocates length bytes of buffer space for the required channel. Note
```

that the default value is 512 bytes and the minimum allocation is 64
bytes.

   It's a good idea to increase the buffer size for high speed
transfers.

1.525 serial fast

Serial Fast (Switches on FAST transfer mode)

Serial Fast channel

This sets a special fast flag in the current device and disables a lot
of internal checking which would otherwise slow down the communication
process. Use it for high speed transfers such as MIDI.

Warning: When you call this command, the protocol will be changed to:
PARITY EVEN,NO XON/XOFF and 8 bits.

1.526 serial slow

Serial Slow (Switches serial transfer back into SLOW mode)

Serial Slow channel

Slows the serial device back to normal speed and reactivates all the
error checks.

1.527 serial check

                Serial Check (Reports on current serial device activity)

=Serial Check(channel)

Asks the device for a read-out of its current status. You can use it to
check whether all the information you've transferred with a previous

                SERIAL SEND
                 command has been sent.

   CHECK=
                FALSE
                 (0) -> if the last serial command is still being
executed.

   CHECK=
                TRUE
                 (-1) -> All done!

1.528 serial error

```
Serial Error  (Reports success or failure of last transfer)

=Serial Error(channel)

Looks for the ERROR byte in the serial device. A value of zero indicates
that everything is fine. Otherwise, the last transmission was faulty.
```

1.529 serial sending tips

```
                Sending large strings

Transmitting a large string may take quite a long time, especially at
low baud rates. As AMOS is multitasking, your program will continue to
run AFTER a
                SERIAL SEND
                 instruction.

   It's essential to avoid provoking a garbage collection before
the transfer has been completed, otherwise your data will be corrupted.
So:

   Use the =
                SERIAL CHECK
                 function before doing a lot of string work.
   Perform a garage collection using X=FREE to ensure that your program
will not provoke one automatically.

   Use the
                SERIAL OUT
                 channel,address,length instruction
with 'address' containing the location of a previously reserved memory
bank.

More information about the Amiga's serial system can be found in the
Commodore ROM KERNEL Reference Manual, Library and Devices. This will
allow expert users to make maximum use of the serial device.
```

1.530 dev first$

```
                =Dev First$  (Get first device from the current device list)

dev$=Dev First$("filter")

Works the same as
                Dir First$
                 and
                Dir Next$
                , but reports back the device
list. Note that you should remove the spaces with -" " to get the right
name.
```

1.531 dev next$

```
                =Dev Next$  (Get the next device satisfying the filter)

dev$=Dev Next$

Gets the next device from the device list. A null string indicates the
end of the list has been reached. Example:


                Print

                Dev First$
                   Do
          A$=Dev Next$
          A$=A$-" "
          If A$="" Then
                End

                Print
                 A$
   Loop
```

1.532 set tempras

```
Set Tempras [address,size]

Warning, due to the nature of this instruction, it is suggested you
re-read the command in your manual and all of it's associated commands.

Set Tempras is in your manual on page 71.
```

1.533 rem

```
Rem Comment.
' Comment.

Note the ' shortform can only be used at the beginning of a line.
```