ADAM™ SmartBASIC™

ADVANCED REFERENCE SECTION

TABLE OF CONTENTS

APPEND

APPEND allows you to append more data to an existing sequential file. This is used only for sequential files. It is also important to bear in mind that a sequential file should be closed after you finish writing. Otherwise ADAM will use all the remaining space on the data pack for your file and will not be able to APPEND. It's syntax is:

APPEND <filename>,[D#]

where D#, or Device Number.

APPEND is similar to OPEN in that both commands open your file; but it differs from OPEN in three important respects:

1. To use APPEND, the file must already exist. You cannot use APPEND to create a file.
2. APPEND will place the pointer at the end of the file, where data will be added. OPEN places the pointer at the beginning of the file.
3. There is no need to do a WRITE after the APPEND.

Test Program:

```
10 D$=CHR$(4)
20 PRINT D$;"APPEND FACE,D1"
30 PRINT "ADDING TO FACE"
40 PRINT D$;"CLOSE FACE"
]MON C
```

Sample Run:

```
APPEND FACE
CLOSE FACE
```

Function

ASC

For every character you use there is, programmed into your computer, a corresponding ASCII decimal number value for it. ASC is the function that converts upper and lower case characters and symbols to their ASCII values.

Test Program:

```
10 PRINT "THE ASCII CODE FOR LETTER H IS
";
20 PRINT ASC("H")
30 IF ASC ("H")=72 THEN 60
40 PRINT "ASC BLEW IT"
50 GOTO 70
60 PRINT "ASC IS CORRECT"
70 END
```

Sample Run:

```
THE ASCII CODE FOR LETTER H IS 72
ASC IS CORRECT
```

ATN, or arctangent, is a trigonometric function. It is defined as the angle in a right triangle required for a particular ratio of the length of the side opposite it to the length of the side adjacent to it. ATN is the complement of TAN, and is expressed not in degrees, but in Radians. Its syntax is:

ATN(exprnm)

where (exprnm)=numerical expression.

NOTE: pi=4*ATN(1)

Test Program:

```
10 PRINT "ENTER A TANGENT VALUE:  ";
20 INPUT N
30 A=ATN(N)
40 PRINT "THE ARCTANGENT OF ";N;" IS
";A;" RADIANS"
50 END
```

Sample Run:

```
ENTER A TANGENT VALUE:  1
THE ARCTANGENT OF 1 IS .785398164
RADIANS
```

BLOAD

The command BLOAD retrieves images stored in binary files on your digital data pack and loads them into memory.
The syntax for BLOAD is:

BLOAD <filename>,A#,D#

A# is the memory location into which the binary file will be LOADed. Use of a memory address is optional for BLOAD, as is drive designation (D#). If these values are not input then ADAM assumes a preset default mode and adjusts for these values itself. BLOAD is useful for LOADing shape tables, among other things.

NOTE: To use BLOAD, you first need to do a BSAVE.

Test Program:

```
BLOAD shape, A51456
```

Sample Run:

Loads a previously BSAVE'd binary file named "shape" into memory starting at memory location 51456.

BRUN

BRUN is used in binary files and is much the same as BLOAD, except that after you load the file, BRUN will execute a machine language "jump" to the starting address automatically. It's syntax is:

BRUN <filename>,A#,D#

The A and D values are optional. If you don't specify an address or drive, then ADAM will go into the preset default for that information, will go directly to the address under which your image was stored, and will assume Drive 1.

Test Program:

NONE

Sample Run:

NONE

BSAVE

BSAVE is used to save binary information on your digital data pack. The syntax of a BSAVE command would be:

BSAVE <filename>, A#,L#,D#

A = specifies the starting address of the memory portion to be stored on the digital drive. Its use is NOT optional. You may input your memory address as either a decimal or a hexadecimal. If hexadecimal is used, the values must be preceded by a dollar sign ($). The range acceptable for decimal values is 0 to 65535. Be careful about using space reserved for SmartBASIC or the operating system. (See the memory map in Appendix C.)

L = the length of the image or information to be saved. Its use is NOT optional. Length specifies the number of bytes to be stored. Again, you may use decimals or hexadecimals to specify this.

D = the drive used. Its use in the program is optional.

Test Program:

```
BSAVE shape, A51456,L14
```

Sample Run:

This will save the shape table created in the SHAPE TABLE example to a binary file named "shape."

WARNING: FOR EXPERTS ONLY! Statement

CALL

CALL lets you execute a machine-language subroutine at a specified memory location, the decimal value of which may be 0 through 65535. Its syntax is:

CALL <location of machine-language sub>

Test Program:

NONE

Sample Run:

NONE

COS

COS is a trigonometric function which will compute the cosine of an angle, expressed in radians rather than degrees. Cosine is defined for a right triangle as the ratio of the length of the side adjacent to the angle to the length of the hypotenuse. Its syntax is:

```
COS(exprnm)
```

where exprnm is a numeric expression.

Test Program:

```
10 PRINT "ENTER ANGLE (IN RADIANS):  ";
20 INPUT T
30 Y=COS(T)
40 PRINT "'COS' OF ";T;" IS ";Y
50 END
```

Sample Run:

```
ENTER ANGLE (IN RADIANS):  ?1
'COS' OF 1 IS .540302307
```

DRAW

DRAW is used in high resolution graphics to draw a pre-defined shape at a specific location. DRAW assumes that a shape table has already been loaded into memory. You must follow the statement DRAW with the shape table index number of the shape you wish to have ADAM take from your shape table and draw on the screen. The screen location at which you want your shape drawn follows the word AT and is a two-expression term, separated by a comma. If the screen location is omitted, the shape will be drawn at the last point plotted; or at 0,0 if this is the first plot.

Test Program:

```
5 HIMEM:51455
10 DATA 01,00,04,00
20 DATA 54,63,36,36
30 DATA 45,45,54,54
40 DATA 63,00
50 FOR i=0 TO 13
60 READ a
70 POKE 51456+i,a
80 NEXT
90 POKE 16766,0
100 POKE 16767,201
110 HGR: HCOLOR=3
120 SCALE=10
130 FOR i=1 TO 64
140 ROT=i
150 DRAW 1 AT 125,85
160 XDRAW 1 AT 125,85
170 NEXT
```

Sample Run:

A white square with a straight line from the midpoint downward.

EXP

EXP raises e to the indicated power, exprnm.
EXP's syntax is:

```
EXP(exprnm)
```

where exprnm is a numeric expression.

Test Program :

```
10 PRINT "ENTER 'EXP' VALUE: ";
20 INPUT N
30 E=EXP(N)
40 PRINT "'EXP' OF ";N;" IS ";E
50 END
```

Sample Run:

```
ENTER 'EXP' VALUE: ?1
'EXP' VALUE OF 1 IS 2.71828183
```

NOTE: You can also raise numbers to various powers by using the CARET (^) key. For example, 10^2 would equal 100.

FRE(expr) allows you to keep a running tally on exactly how many bytes of allocated space you have available to you in memory.

Test Program:

```
PRINT FRE(0)
```

Sample Run:

```
25820
```

Test Program:

```
10 B=FRE(1)
20 A$="HELLO"
30 C=FRE(1)
40 PRINT B-C
```

Sample Run:

```
13
```

It takes 13 bytes to store HELLO.

HIMEM:

HIMEM: or highest program memory location, is automatically preset, but you may change the settings if you wish to. HIMEM: is used to protect the area of memory above a designated location, since HIMEM: will create a boundary to allow you to safely input your data without writing over important data above your boundary. But, since HIMEM: is pre-set to allow you the maximum free memory space, we DO NOT recommend that you reset it unless you are working with assembly language. It is imperative that you investigate reserved memory before going ahead and changing the HIMEM: boundary.The HIMEM: syntax is:

HIMEM:<memory location>

where memory location has a range of 0-65535 as well as -32767 down to -1. Positive or negative values, if equivalent, may be used interchangeably (-32767 equivalent to 65535).

Test Program:

NONE

Sample Run:

NONE

INIT

INIT, or initialize, is a command that should only be used in the immediate mode. Use INIT when you wish to wipe out your old directory in your digital data pack. INIT will reinitialize the directory (delete all file name entries), so be very careful when using it! INIT will not initialize a SmartBASIC tape, so don't worry about that. INIT syntax is:

```
INIT <volumename>,D#
```

The D value is optional, and indicates a particular digital data drive.

> **NOTE:** You don't need to initialize your blank digital data packs before you use them. Also, it is not possible to turn an ordinary cassette into a blank digital data pack using this command.
>
> Be careful not to 'INIT' over a Super Game!

Test Program:

```
NEW
INIT HELLO,D1
---wait while ADAM initializes
]CATALOG
```

Sample Run:

```
VOLUME: HELLO

253 Blocks Free
```

This says that you're operating in Drive 1, that your volume name is HELLO with 253 blocks free.

LOG computes the natural logarithm of any number whose value is larger than 0.

Test Program:

```
10 PRINT "ENTER POSITIVE NUMBER  ";
20 INPUT Q
30 L=LOG(Q)
40 PRINT "'LOG' OF ";Q;" IS ";L
50 END
```

Sample Run:

```
ENTER POSITIVE NUMBER  ?10
'LOG' OF 10 IS 2.30258509
```

WARNING: FOR EXPERTS ONLY! Statement

LOMEM:

The opposite of HIMEM:, LOMEM: sets the lowest memory location available. It's syntax is:

LOMEM:<memory location>

where <memory location> is an arithmetic expression between -32767 and +65535. Note that either positive or negative addresses can be used if equivalent.

Again, it is inadvisable to change the value of LOMEM: unless you are a very experienced programmer. We do not recommend it unless you are involved with assembly language.

Test Program:

NONE

Sample Run:

NONE

WARNING: FOR EXPERTS ONLY! Statement

PEEK

PEEK allows you to look at whatever data is in the memory location you specify. No writing, just looking! You are able to PEEK into RAM (read/write memory). The syntax of PEEK is:

PEEK <memory location>

where <memory location> is the address of the memory location you wish to read. Range for PEEK is 0 - 65535; but PEEK may also be used with POKE to read what has been POKED in at whatever address is within POKE range, as well.

Test Program:

```
5 HIMEM: 51456
10 FOR X=51457 TO 51467
20 READ Y
30 POKE X,Y
40 NEXT X
50 FOR X=51457 TO 51467
60 Y=PEEK(X)
70 PRINT CHR$(Y);
80 NEXT X
90 DATA 80,69,69,75,45,65,45,66,79,79,33
100 END
```

Sample Run:

PEEK-A-BOO!

WARNING: FOR EXPERTS ONLY! Statement

POKE

POKE alters the contents of a memory location, in that it will store a designated value directly into a specific memory location. Syntax for POKE is:

POKE <memory location>,<value 0-255>

where value is the decimal number representing the eight-bit quantity of data to be stored in the memory location specified. Memory location range is -32767 through 65535 inclusive. You can POKE into RAM (read/write memory) only.

NOTE: The SmartBASIC POKE will not function when using values above 54160 or below 0.

Test Program:

```
5 HIMEM:51455
10 DATA 01,00,04,00
20 DATA 54,63,36,36
30 DATA 45,45,54,54
40 DATA 63,00
50 FOR i=0 TO 13
60 READ a
70 POKE 51456+i,a
80 NEXT
90 POKE 16766,0
100 POKE 16767,201
110 HGR: HCOLOR=3
120 SCALE=10
130 FOR i=1 TO 64
140 ROT=i
150 DRAW 1 AT 125,85
160 XDRAW 1 AT 125,85
170 NEXT
```

Sample Run:

This will give you the "default shape", which is a square with a straight line running from the mid point, downward. For a picture of it, see the Shape Table section of the Compendium at the back of this manual.

WARNING: FOR EXPERTS ONLY!

Statement

POP

The POP statement "pops" or discards the line number of the most recent GOSUB. So when a RETURN statement appears, it will return to the statement following the secondmost GOSUB.

Test Program:

```
10 PRINT "ONE ";
20 GOSUB 100
30 PRINT "FOUR"
90 END
100 PRINT "TWO ";
110 GOSUB 200
120 PRINT "ONE TWENTY"
130 RETURN
140 STOP
200 PRINT "THREE ";
210 POP
220 RETURN
```

Sample Run:

```
ONE TWO THREE FOUR

At line 210 you "popped" past line 110
and returned after line 20 to print line
30.
```

POSITION moves the position-in-the-file pointer forward only. It can be used only in sequential files, and only to skip the number of records you select. You cannot write to your file using POSITION. Its syntax is:

```
POSITION <filename>,R#
```

where R#=record number (relative)

Test Program: (Assumes you have a sequential file named UP with at least 4 records).

```
10 D$=CHR$(4)
20 PRINT D$; "OPEN UP"
30 PRINT D$; "POSITION UP,R3"
40 PRINT D$; "READ UP"
50 INPUT A$: PRINT A$
60 PRINT D$; "CLOSE UP"
]MON C
```

Sample Run:

```
OPEN UP
POSITION UP,R3
READ UP
(prints 'record 4'as 4th record
 in file 'UP')
CLOSE UP
```

RANDOM/Fixed Length Files

Random access files will allow you to open a file at a particular record number, READ it and/or WRITE a new record into it. R#=record number and L#=record length. Please keep in mind that the file must already exist, and the READ or WRITE must be within the existing file. And always, **always** remember to CLOSE your file when you're through.

For more information on random files, refer to the Compendium at the back of this book. (Appendix C).

Test Program:

```
10 D$=CHR$(4)
20 PRINT D$;"OPEN DOOR,L20"
30 PRINT D$;"READ DOOR,R5"
40 INPUT A$:PRINT A$
50 PRINT D$;"CLOSE DOOR"
        or
30 FOR I=1 TO 10
40 PRINT D$;"WRITE DOOR,R";I
50 PRINT "RECORD NUMBER";I
60 NEXT
70 PRINT D$;"CLOSE DOOR"

]MON C
NOTE:R#=record number
     L#=record length
```

Sample Run:

```
OPEN DOOR,L20
READ DOOR,R5
CLOSE DOOR

        or

OPEN DOOR,L20
WRITE DOOR,R1
RECORD NUMBER 1
: (write ten records)
CLOSE DOOR
```

The files you access in this manner do not have to be sequential.

The RECOVER statement is used to access a back up file. Its syntax is:

RECOVER <filename>,D#

where D# means "device number" and is optional.

Test Program:

```
]CATALOG
Volume:  BOZO
A   1 FOO
a   1 FOO
253 BLOCKS FREE
]DELETE FOO
]CATALOG
Volume:  BOZO
a   1 FOO
254 BLOCKS FREE
]RECOVER FOO
]CATALOG
Volume:  BOZO
A   1 FOO
254 BLOCKS FREE
]LOAD FOO
```

Sample Run:

```
You will now have access to your back up
file of FOO.  The most recent version of
"FOO" has been deleted.  To keep both
versions, first RENAME "FOO".
```

ROT=

The ROT=, or ROTation statement sets the spatial orientation of a shape in high resolution graphics before your shape is drawn. The number following ROT= dictates the rotation of the shape in 5.625 degrees/unit. Therefore, ROT=0 orients the shape exactly as defined in the shape table; whereas ROT=16 turns the shape 90 degrees in a clockwise direction. ROT=32 turns it 180 degrees. ROT=64 turns it one complete revolution so that you're back where you started from. The number following the = may be a variable.

NOTE: Because the ratio of width to height of most TVs and monitors is not 1, the proportions of a shape may change as the shape is rotated.

Test Program:

```
10 HGR2
20 HCOLOR=6
30 SCALE=10
40 ROT=0
50 DRAW 1 AT 50,100
60 ROT=8
70 DRAW 1 AT 120,100
```

Sample Run:

You will end up with two blue shapes on your screen—one oriented as originally defined, and one rotated 45 degrees.

SCALE=

SCALE= is used in high resolution graphics to set the relative size at which a shape will be drawn. This scaling factor may be as low as 1 or as high as 255; 1 being an exact size reproduction from the one specified; 255 being the shape reproduced at 255 times its originally defined size. The number following = may be a variable.

Test Program:

```
10 HGR
20 HCOLOR=5
30 ROT=0
40 SCALE=1
50 DRAW 1 AT 50,50
60 SCALE=3
70 DRAW 1 AT 100,150
80 SCALE=2
90 DRAW 1 AT 200,100
```

Sample Run:

Color is set to medium red. You will draw one shape at original size and orientation at column 50, row 50. The same shape will be redrawn at column 100, row 150 at 3 times the original size. The same shape will be redrawn a third time at column 200, row 100 at twice its original size.

SIN

SIN, a trigonometric function, is used to compute the sine of an angle (expressed in radians). Sine is the ratio of the length of the side opposite the angle under examination to the length of the hypotenuse in a right triangle. Syntax of SIN is:

```
SIN(x)
```

where "x" is an arithmetic expression.

Test Program:

```
10 PRINT "ENTER ANGLE (IN RADIANS):  ";
20 INPUT M
30 Y=SIN(M)
40 PRINT "'SIN' OF ";M;" IS ";Y
50 END
```

Sample Run:

```
ENTER ANGLE (IN RADIANS):  1
'SIN' OF 1 IS .841470988
```

TAN

In trigonometry, TAN is used to find the tangent of an angle, expressed in radians. Tangent is the ratio of the length of the side opposite the angle in question to the length of the side adjacent to it in a right triangle. Its syntax is:

```
TAN(x)
```

where "x" can be an arithmetic expression.

Test Program:

```
10 PRINT "ENTER ANGLE (IN RADIANS):  ";
20 INPUT Z
30 Y=TAN(Z)
40 PRINT "'TAN' OF ";Z;" IS ";Y
50 END
```

Sample Run:

```
ENTER ANGLE (IN RADIANS):  1
'TAN' OF 1 IS 1.55740772
```

WARNING: FOR EXPERTS ONLY! Function

USR

USR executes a machine-language function routine and requires a full knowledge of machine-language programming. USR stands for User Supplied Routine and is usually used to perform a high-speed computation that can neither be done quickly nor expressed in SmartBASIC. Store a USR function in memory with a POKE statement.

Test Program:

NONE

Sample Run:

NONE

WARNING: FOR EXPERTS ONLY!!

Command Statement

WAIT

WAIT does what its name implies. It halts a program execution until a designated bit pattern appears at a particular port location. If you try to access a port device that is not receiving, a WAIT command will "hang up" your system. Syntax of WAIT is:

WAIT <port #(0-255)>,<value 1>,<value 2>

where value 1 is ANDed with data from <port number (0-255)>. Value 2 is optional, it is XORed with value 1. If value 2 is omitted, zero is assumed. WAIT will loop until the logical operation is zero.

Test Program:

NONE

Sample Run:

NONE

In high resolution graphics, XDRAW does the same thing DRAW does, except for the fact that the color you choose in drawing the shape is the complement of the color already existing at each plotted point. You may also use XDRAW to erase a shape (by plotting a new shape at the same location, SCALE, and ROT), and leave the surrounding graphics untouched.

IF YOUR COLOR IS:	THEN XDRAW COLOR IS:
MED.BLUE	MED.RED
MED.RED	MED.BLUE
DK.RED	GREEN
GREEN	DK.RED
BLACK	WHITE
WHITE	BLACK

Test Program:

```
5 HIMEM:51455
10 DATA 01,00,04,00
20 DATA 54,63,36,36
30 DATA 45,45,54,54
40 DATA 63,00
50 FOR i=0 TO 13
60 READ a
70 POKE 51456+i,a
80 NEXT
90 POKE 16766,0
100 POKE 16767,201
110 HGR: HCOLOR=3
120 SCALE=10
130 FOR i=1 TO 64
140 ROT=i
150 DRAW 1 AT 125,85
160 XDRAW 1 AT 125,85
170 NEXT
```

Sample Run:

A square with a straight line from the midpoint, downward. See the Compendium (APPENDIX C) for a picture of it.