Atari BASIC  
And  
OSS BASIC A+

This card provides a complete syntax summary of all statements and functions in both Atari BASIC and OSS BASIC A+. The various keywords of the languages are grouped as follows:

  First: by category, with a heading for each group. A keyword may appear in more than one category.

  Second: within the category group, those keywords found in both BASICs precede those found only 
          OSS BASIC A+.

  Third: within each language partition, all statements 
          precede all functions. Functions are denoted by 
          an 'f' in front of the keyword.

  Finally: within each list of statements and list of 
          functions, keywords are placed alphabetically.

NOTE: All capabilities found in OSS BASIC A+ are shown 
      shaded, as in this sentence.

DEFINITION OF TERMS

KEYWORDS are shown in bold face type, and should be 
typed as shown. The following syntax for each keyword is 
shown in normal type and generally consists of zero or 
mores of the syntax items shown below. Explanations are 
shown in italics.

Items enclosed [in square brackets] are optional.
Enclosed items [followed by ellipses ...] may be repeated 
any number of times.

SYNTAX ITEMS

<stmt> any valid statement  <stmts> any number of 
var any VARiable  valid statements 
avar an Arithmetic var placed on any 
svar a String var  number of lines 
mvar a Matrix var (or 
      matrix element) 
asvar avar or svar, but 
      never mvar 
file name asexp used as a 
      file specifier 
      pm aexp used as P/M # 
      addr aexp used as a 
      memory address 

exp any expression 
exp an Arithmetic exp 
exsp a String exp 
line aexp used as a 
      line # 
      fn aexp used as a 
      file # 

Legal forms of file specifiers: <device>:<file>.<ext> where <device> consists of a single letter optionally followed by 
      a single digit. When the device is the disk, <file> is any 
      name consisting of 1 to 8 alphanumeric characters, the 
      first of which is a letter. <ext> is an optional 1 to 3 alpha-
      numeric characters. Here are some examples:
      E: (the screen editor)  D2:MENU, SAV (a disk file
      P: (the printer)        on drive 2 with the name
      R2: (RS-232 port number 2)  "MENU" and the name
                                 extension "SAV")
COMMAND & CONTROL
BYE
  goes to memo pad
CLR
  zeroes simple variables, changes all DIMs to 0
LOAD
  load a program from cassette
SAVE
  save a program to cassette
ENTER filename
  only works with ATASCII version of a program (see LIST); actually a merge unless NEW is used first
LIST filename
  lists program to file in ATASCII just as it appears on the screen for LIST alone
LOAD filename
  load a previously SAVED program
RUN filename
  load and run a SAVED program
SAVE filename
  save a program to a file using internal format

PROGRAM DEVELOPMENT STATEMENTS
CLOAD
  load a program from cassette
CONT
  continue a program after a STOP or BREAK
SAVE
  save a program to cassette
END
  close all files, stop the program
ENTER file name
  merges an ATASCII (LISTed) program into that already in memory
LIST [filename]
  list program in ATASCII to screen or file
LIST [filename], line[,line]
  list only a portion of a program
LOAD filename
  load a previously SAVED program
NEW
  remove all programs and variables from memory
REM <any remark>
  allows commenting of program listings
RUN
  begin executing program in memory at lowest line number
RUN [filename]
  load a SAVED program and start executing it
SAVE filename
  save a program in memory to a file in internal format
STOP
  halt execution of program
FREE(0)
  returns amount of memory still available
DEL line[,line]
  delete all lines in range specified
LOMEM addr
  can reserve memory; does a NEW
LVAR filename
  list all variables in use by program in memory to given file
RENUM [start][, increment]
  renumbers entire program
TRACE
  begin displaying each line's number as it is executed
TRACEOFF
  cease displaying line numbers
PROGRAM CONTROL

END
close files, stop program
FOR avar = aexp TO aexp
[STEP aexp] <stmts>:
NEXT avar
traditional loop control
GOSUB line
call a subroutine
GOTO line
transfer control to new line
IF aexp THEN
<stmt>[:<stmt>...]
statements after THEN are executed only if the aexp
is non-zero
IF aexp THEN line
control is transferred to new line only if the aexp is
non-zero
NEXT {see FOR}
ON aexp GOTO line
[,line ...]
ON aexp GOSUB line
[,line ...]
if aexp = 1, control moves to first line given; if aexp =
2, then to a second line; etc.

CONT
after a TRAPped error, continue at line after error
ELSE {see IF below}
ENDIF {see IF below}
ENDWHILE {see WHILE}
IF aexp : <stmts>
[ELSE : <stmts>]
ENDIF
use when both 'true' and 'false' paths are needed; may be nested 127 deep

WHILE aexp:
<stmts>
ENDWHILE
loops between WHILE and ENDWHILE so long as aexp is non-zero
f ERR (aexp)
returns last run-time error code

CONSOLE & FILE I/O

CLOSE #fn
close I/O to file channel fn
GET #fn, avar
set a single byte from fn
INPUT [#fn,] asvar
[,asvar ...]
input ATASCII data
LPRINT [exp [:exp ...]
[, exp ...]]
output ATASCII to line
printer
OPEN #fn, mode, avar, filename
begin I/O with filename on
channel fn
NOTE #fn, avar, avar
find current position/disk file
POINT #fn, avar, avar
change current file position
PRINT [#fn]
output new line only
PRINT exp [:; exp ...
[, exp ...]] [:]
output data items in
ATASCII
PRINT #fn [:; exp ...]
[, exp ...]; [:]
output ATASCII items to a
file
PUT #fn, aexp
output a single byte to fn
STATUS #fn, avar
dynamic status check
XIO aexp, #fn, aexp, aexp, filename
extended I/O operation?
{same as PRINT}
usable wherever PRINT is legal
CONSOLE & FILE I/O (cont)

BGET #fn, addr, len
set binary block from file fn
BPUT #fn, addr, len
put a binary block to file fn
INPUT "...", var [,var ...]
al lows prompt to replace "?"
LPRINT [#fn,] USING exp, [exp[,exp ...]]
see special table:PRINT USING
PRINT [#fn,] USING exp, [exp[,exp ...]] [;]
see special table:PRINT USING

MACHINE CONTROL

MOVE fromaddr, toaddr, lenaexp
move any piece of memory to anywhere; moves "down" if lenaexp is positive (contracts); moves "up" if lenaexp is negative (expands)
POKE addr, aexp
change contents of memory location addr to aexp
DPOKE addr, aexp
change contents of WORD at location addr

OPERATOR PRECEDENCE TABLE

The operators of BASIC are listed in order precedence, from highest to lowest. Higher precedence implies the operator will be executed first. Example: 3*4*5 is seen as 3*(4*5) because '*' has a higher precedence than '+'.

( ) functions () parenthesized subexpressions

\(<=><><><><\) string comparisons [e.g., A$><"EXIT"]

NOT + - unary operators only [e.g., -3+Z]
\(\wedge\) exponentiation
\& ! binary "and", binary "or"
\(/\) multiply and divide
\(+ /\) add and subtract

\(<=><><><><\) numeric comparisons [e.g., TOTAL > 30]

AND logical "and" (always gives 1 or 0 result)
OR logical "or" (always gives 1 or 0 result), when used in array and function references [e.g., PRINT ARRAY (7,5)]

NOTE: In Atari BASIC, NOT was given a precedence just above AND, but it does not always execute properly unless it is followed by a sub-expression in parentheses [e.g., NOT (A>B) is safe].
ASSIGNMENT & MATHEMATICS

[LET] avar = aexp
[LET] mvar = aexp

arithmetic assignment; keyword is optional

DEG
selects degrees for trig functions

RAD
selects radians for trig functions

f ABS (aexp)
returns absolute value of argument aexp

f ATN (aexp)
returns arc tangent of argument; returns radians or degrees, as selected

f CLOG (aexp)
returns common log (base 10) of argument

f COS (aexp)
returns cosine of argument

f EXP (aexp)
returns 'e' to the power aexp, 'exponentiation'

f INT (aexp)
returns largest integer less than or equal to argument

f LOG (aexp)
returns natural logarithm of the argument

f RND (0)
returns a pseudo-random number between 0 (inclusive) and 1 (exclusive)

f SGN (aexp)
returns +1, 0, -1 according to the sign of the argument (0 only if argument is 0)

f SIN (aexp)
returns sine of argument

f SQR (aexp)
square root of argument

f VAL (aexp)
returns the 'value' of a number contained in a string

INITIALIZATION

CLR
zeros numeric variables, sets all DIMs to zero

DEG
selects degrees for trig functions

DIM svar (aexp)

DIM mvar (aexp[,aexp])
allocate space for either a string or array

DEG
selects radians for trig functions

f FRE (0)
returns amount of memory still available

LOMEM addr
can reserve memory; does a NEW

SET aexp, aexp
see separate chart

f SYS (aexp)
returns value SET before

DOS COMMANDS

DOS
exit to "DOS"

CP
same as DOS

DIR filename
list disk directory on screen

ERASE filename
remove file from disk

PROTECT filename
disallow writes and/or erases of given filename

RENAME filenames
changes name of a file—CAUTION: form must be "Dr: oldname, newname"

UNPROTECT filename
remove file protection
STRING & CHARACTER HANDLING

[LET] svar = sexp
the destination string variable may be subscripted
f CHR$ (aexp)
returns a one byte string—character has a value of aexp
f ADR (svar)
returns the address of the given string
f LEN (sexp)
returns length of string
f STR$ (aexp)
returns a string equivalent to what would be visible if aexp were PRINTed

[LET] svar = sexp [,sexp ...]
allows concatenation of several strings
f FIND (sexp, sexp, aexp)
finds location of 2nd str within 1st string starting at given position plus one

GRAPHICS, SOUND, & PLAYER/MISSILE GRAPHICS

COLOR aexp
choose a color for subsequent PLOT and DRAWTO
DRAWTO aexp, aexp
draw a line from last point PLOTted or drawn to
GRAPHICS aexp
choose a graphics mode
LOCATE aexp, aexp, avar
find what color a given point on the screen is
PLOT aexp, aexp
plot a single point (pixel)
POSITION aexp, aexp
set screen location cursor

SETCOLOR aexp, aexp, aexp
change color register values; order is register number, hue, luminance
SOUND aexp, aexp, aexp, aexp
change sound register values; order is register number, frequency, waveform, volume
f PADDLE (aexp)
current paddle value
f PTRIG (aexp)
returns 0 if trigger pushed
f STICK (aexp)
current joystick position
f STRIG (aexp)
returns 0 if trigger pushed

MISSILE pm, aexp, aexp
"shoot" a missile
PMCLR pm
clear a player area
PMCOLOR pm, aexp, aexp
change a player color—same format as SETCOLOR
PMGRAPHICS aexp
select player/misille mode
PMMOVE pm[,aexp] [,aexp]
move a player or missile
PMWIDTH pm, aexp
change player/misille width

f BUMP (pmnum, aexp)
check for player/misille and/or playfield collisions
f HSTICK (aexp)
returns -1, 0, +1 if joystick is left, center, right
f PEN (aexp)
returns light pen values
f PMADR (pm)
gets address of a player or missile
f VSTICK (aexp)
returns -1, 0, +1 if joystick is down, center, up

OSS
Optimized Systems Software, Inc.
10379 Lansdale Avenue
Cupertino, CA 95014
Telephone (408) 446-3099
IN-MEMORY DATA HANDLING

DATA <ATASCII data>
data may contain any characters except a comma

READ asvar [.asvar ...]
evaluate next data from
DATA statement(s) and
place in specified variable

DATA ['"quoted data"']
[<ATASCII data>]
if data is quoted may
contain any characters
except another quote

READ var [,var ...]
may read directly into
subscripted array elements
or substrings

RESTORE [line]
move data pointer to given
line number, (or beginning
of program)

BASIC ERROR MESSAGES

<table>
<thead>
<tr>
<th>Number</th>
<th>Message</th>
<th>Number</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Break Key Abort</td>
<td>16</td>
<td>RETURN With No</td>
</tr>
<tr>
<td>2</td>
<td>Memory Full</td>
<td>17</td>
<td>Matching GOSUB</td>
</tr>
<tr>
<td>3</td>
<td>Value (usually num too big)</td>
<td>18</td>
<td>Bad Line (syntax error/line)</td>
</tr>
</tbody>
</table>
| 4      | Too Many Variables         | 19     | Not Numeric (VAL func.
|        |                             |        | error)                      |
| 5      | String Length               |        | Program Too Big             |
| 6      | No More Data Available      | 20     | To Load                     |
| 7      | For Read                    | 21     | File Number Invalid         |
|        | >32767                      |        | Not A SAVEd Program         |
| 8      | Input Or Read Data Error    | 22     | 'USING' Format              |
| 9      | Dimension Error             | 23     | 'USING' Too Big             |
| 10     | Expression Too Complex      | 24     | 'USING' Type                |
|        |                             | 25     | Dimension Mismatch          |
|        |                             |        | (RGET)                      |
| 11     | Floating Point Overflow     | 26     | Type Mismatch (RGET)        |
| 12     | No Such Line Number         | 27     | INPUT Abort                 |
| 13     | NEXT, With No               | 28     | Nesting                     |
| 14     | Matching FOR                | 29     | Player/ Missile Number      |
| 15     | Line Too Long               | 30     | PM Graphics Not Active      |
|        | (GOSUB, FOR or WHILE)       | 32     | End of 'ENTER'              |

CIO ERROR MESSAGES

<table>
<thead>
<tr>
<th>Number</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>Break Abort</td>
</tr>
<tr>
<td>129</td>
<td>File Number Already Open</td>
</tr>
<tr>
<td>130</td>
<td>Nonexistent Device</td>
</tr>
<tr>
<td>131</td>
<td>File Is Write Only</td>
</tr>
<tr>
<td>132</td>
<td>Invalid Command</td>
</tr>
<tr>
<td>133</td>
<td>File Not Open</td>
</tr>
<tr>
<td>134</td>
<td>Bad File Number</td>
</tr>
<tr>
<td>135</td>
<td>File Is Read Only</td>
</tr>
<tr>
<td>136</td>
<td>End Of File</td>
</tr>
<tr>
<td>137</td>
<td>Truncated Record</td>
</tr>
</tbody>
</table>

SIO ERROR MESSAGES

<table>
<thead>
<tr>
<th>Number</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>138</td>
<td>Device Timeout</td>
</tr>
<tr>
<td>139</td>
<td>Device NAK (refuses command)</td>
</tr>
<tr>
<td>140</td>
<td>Serial Bus Frame Error</td>
</tr>
<tr>
<td>141</td>
<td>Cursor Out Of Range</td>
</tr>
<tr>
<td>142</td>
<td>Serial Bus Overrun</td>
</tr>
<tr>
<td>143</td>
<td>Serial Bus Checksum</td>
</tr>
</tbody>
</table>

S: (Screen) ERROR MESSAGE

HARDWARE ERROR MESSAGES

<table>
<thead>
<tr>
<th>Number</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>144</td>
<td>Device Error (usually write protected disk)</td>
</tr>
<tr>
<td>145</td>
<td>Read/Write Verify</td>
</tr>
<tr>
<td>146</td>
<td>Invalid Function</td>
</tr>
</tbody>
</table>
SET/SYS VALUES

SET is used to configure certain BASIC A+ system parameters. The companion function SYS( ) may be used to find out what the configuration is at any point in time. The format is: SET parameter number, value. A number in parentheses is the “power-on” default value.

<table>
<thead>
<tr>
<th>Parameter Number</th>
<th>Values</th>
<th>Meanings</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(0)</td>
<td>BREAK key functions normally.</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>BREAK causes trappable error.</td>
</tr>
<tr>
<td>128</td>
<td></td>
<td>BREAKs are ignored.</td>
</tr>
<tr>
<td>1</td>
<td>1 to 127</td>
<td>Tab size for comma in PRINT (10).</td>
</tr>
<tr>
<td>2</td>
<td>0 to 255</td>
<td>Prompt character for INPUT (63, &quot;?&quot; ).</td>
</tr>
<tr>
<td>3</td>
<td>(0)</td>
<td>FOR ... NEXT loops execute at least once.</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>FOR loops may execute zero times.</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>Reprompt user if too little INPUT data.</td>
</tr>
<tr>
<td></td>
<td>(1)</td>
<td>No reprompt, a TRAPpable error occurs.</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>Lower case/inverse video unchanged.</td>
</tr>
<tr>
<td></td>
<td>(1)</td>
<td>For program entry ONLY, lower case &amp; inverse video converted to upper case.</td>
</tr>
<tr>
<td>6</td>
<td>(0)</td>
<td>Print error messages and error numbers.</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Print only error numbers.</td>
</tr>
<tr>
<td>7</td>
<td>(0)</td>
<td>Player/misillies will NOT wrap around.</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Player/misillies wrap around from top to bottom and vice versa.</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>No parameter count push for USB calls.</td>
</tr>
<tr>
<td></td>
<td>(1)</td>
<td>DO push the count of parameters.</td>
</tr>
<tr>
<td>9</td>
<td>(0)</td>
<td>ENTER statements work like Atari BASIC.</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>End of an ENTER is treated as a trappable error.</td>
</tr>
</tbody>
</table>

PRINT USING TABLE

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>Blank Fill On Left</td>
</tr>
<tr>
<td>*</td>
<td>Asterisk Fill On Left</td>
</tr>
<tr>
<td>&amp;</td>
<td>Zero Fill On Left</td>
</tr>
<tr>
<td>,</td>
<td>Numeric Comma Placeholder</td>
</tr>
<tr>
<td>.</td>
<td>Numeric Decimal Point Placeholder</td>
</tr>
<tr>
<td>$</td>
<td>Fixed Dollar Sign</td>
</tr>
<tr>
<td>&amp;$</td>
<td>Floating Dollar Sign</td>
</tr>
<tr>
<td>+</td>
<td>Floating Forced Sign (+ or −)</td>
</tr>
<tr>
<td>−</td>
<td>Floating Minus Sign (Blank or −)</td>
</tr>
<tr>
<td>%</td>
<td>Right Justified String</td>
</tr>
<tr>
<td>!</td>
<td>Left Justified String</td>
</tr>
</tbody>
</table>
| \\
| Leading Or Trailing Fixed Forced Sign (+ or −) |
| −      | Leading Or Trailing Fixed Minus Sign (Blank or −) |
| /X     | Escape Sequence (X is ANY character and is forced whether in a format or not) |