ACEReference.doc

COLLABORATORS

	TITLE : ACEReference.doc		
ACTION	NAME	DATE	SIGNATURE
WRITTEN BY		March 1, 2022	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

ACEReference.doc

1.1 Main Menu

```
                    +---------------------------------------+
                    | ACE v2.3 Command and Function Reference |
                    +---------------------------------------+


                    Introduction

                    FILES

                    MESSAGE WRITE

                    SGN

                    ABS

                    FIX

                    MID$

                    SHARED

                    ADDRESS

                    FONT

                    MOD

                    SHL

                    ALLOC

                    FOR..NEXT

                    MOUSE

                    SHR

                    AND
```

FORWARD

MOUSE ON

SHORTINT

ARG$

FRE

MSGBOX

SINGLE

ARGCOUNT

GADGET

NAME

SIZEOF

AREA

GADGET CLOSE

NOT

SIN

AREAFILL

GADGET MOD

OCT$

SLEEP

ASC

GADGET ON

ON..GOTO/GOSUB

SLEEP FOR

ASSEM

GADGET WAIT

OPEN

SOUND

ATN

GOSUB..RETURN

OPTION

SPACE$

BACK

GOTO

OR

SPC

BEEP

HANDLE

PAINT

SQR

BEVELBOX

HEADING

PALETTE

STICK

BIN$

HEX$

PATTERN

STOP

BREAK

HOME

PEEKx

STR$

CALL

IF

PENDOWN

STRIG

CASE

IFF

PENUP

STRING

CHDIR

IFF CLOSE

POINT

STRING$

CHR$

IFF OPEN

POKEx

STRUCT

CINT

IFF READ

POS

STYLE

CIRCLE

IMP

POTX

SUB..END SUB

CLEAR ALLOC

INKEY$

POTY

SWAP

CLNG

INPUTBOX

PRINT

SYSTEM

CLOSE

INPUTBOX$

```
PRINT #

TAB

CLS

INPUT

PRINTS

TAN

COLOR

INPUT #

PSET

TIME$

CONST

INPUT$

PTAB

TIMER

COS

INSTR

RANDOMIZE

TIMER ON

CSNG

INT

READ

TRANSLATE$

CSRLIN

KILL

REM

TURN

CSTR

LEFT$

REPEAT..UNTIL
```

```
TURNLEFT

DATA

LEN

RESTORE

TURNRIGHT

DATE$

LET

RIGHT$

UCASE$

DAY

LIBRARY

RND

VAL

DECLARE

LINE

SADD

VARPTR

DEF FN

LINE INPUT

SAY

WAVE

DEFxxx

LOCATE

SCREEN

WHILE..WEND

DIM

LOF

SCREEN BACK
```

WINDOW

EOF

LOG

SCREEN CLOSE

WINDOW CLOSE

END

LONGINT

SCREEN FORWARD

WINDOW ON

ERR

MENU

SCROLL

WINDOW OUTPUT

ERROR

MENU CLEAR

SERIAL

WRITE

EQV

MENU ON

SERIAL CLOSE

XCOR

EXIT FOR

MENU WAIT

SERIAL OPEN

YCOR

EXIT SUB

MESSAGE CLOSE

SERIAL READ

XOR

```
            EXP

            MESSAGE OPEN

            SERIAL WRITE
                ACE Guide

            EXTERNAL

            MESSAGE READ

            SETHEADING
                AIDE Guide

            FILEBOX$

            MESSAGE WAIT

            SETXY
```

1.2 Introduction

```
                            +----------+
                            | ACE v2.3 |
                            +----------+


            +------------------------------+
            |Command and Function Reference|
            +------------------------------+


                        ------------
                        Introduction
                        ------------
```

This document consists of a description of currently implemented commands
and functions.

As with AmigaBASIC, the case of commands and functions is of no
consequence.

NOTES: - [] means that a parameter or command component is optional.
        - <> surround literals, names and expressions.
        - .. implies that statements are expected to follow.

        - Commands and functions marked with an asterix are found only in
          ACE, not AmigaBASIC.

        - Standard trigonometric functions take their arguments in radians.

        - EOS = end-of-string character (ASCII 0).
        - MAXSTRINGLEN currently equals 1024. The last character in a
          string is EOS, so if you want a string which holds 1024
          characters, you need a 1025 byte string (see

```
                    STRING
                     command).
```

> – For boolean operators such as AND,OR,IMP etc the values T and F
>   (TRUE and FALSE) refer to –1 and 0 respectively.

1.3 ABS

```
ABS              – syntax: ABS(n)
                 – Returns the absolute value of n.
```

1.4 ADDRESS

```
ADDRESS *        – syntax: ADDRESS <identifier>[,..]
                 – Declares and initialises one or more variables of type
                   address. In fact, this data type is synonymous with the
                   long integer (see
                 LONGINT
                 ) data type. Its main purpose
                   is to make clear just what sort of data is going to be
                   used. This is especially useful when passing addresses
                   as parameters to subprograms.
                 – See also
                 SUB
                 ,
                 STRUCT
                 .
```

1.5 ALLOC

```
ALLOC *          – syntax: ALLOC(<bytes>[,<memory-type>])
                 – This is ACE's hassle-free memory allocator.
                 – You can call this function to get the start address of a
        block of memory at least <bytes> bytes in size.
                 – The <memory-type> argument can be one of the following:

                        0 = CHIP memory
                        1 = FAST memory
                        2 = PUBLIC memory

                        3 = CLEARED CHIP memory
                        4 = CLEARED FAST memory
                        5 = CLEARED PUBLIC memory

        6 = ANY suitable memory (MEMF_ANY)
        7 = ANY suitable cleared memory

                        – If a value outside this range is specified or this
```

parameter is omitted, the result is identical to having
specified a <memory-type> of 7. Note that in ACE v2.0 the
default was CLEARED PUBLIC memory.
- Specifying ANY (6,7) allows the operating system to select
  the best available memory, so specify a <memory-type> of 6
  or 7 for general use and CHIP memory for sound samples or
  other data which must be accessible by the co-processors.
                    - If the requested block of memory can't be allocated
                      for whatever reason (eg: memory is too fragmented)
                      ALLOC returns zero.
                    - CLEARED memory is filled with zeros.
                    - The main benefit of ALLOC is that it keeps a record of
                      memory allocations, freeing all memory allocated via it
  at the end of a program run.
                    - ALLOC will free allocated memory even if a program aborts
                      due to a ctrl-c break or an error (except where a GURU
                      results).
                    - Use of ALLOC assumes that you know what you're doing with
                      memory and why you want a chunk of it.
- For more information about memory allocation on the Amiga,
  see the Exec/Intuition autodocs re: AllocMem()/FreeMem()
  and AllocRemember()/FreeRemember(). See also the manual
  "RKM: Libraries".
- See also CLEAR ALLOC.

1.6 AND

AND   - Boolean operator: X AND Y.

                     X Y   Out

                     T T   T
                     T F   F
                     F T   F
                     F F   F

1.7 ARG$

           ARG$ *   - syntax: ARG$(n) where n=0..ARGCOUNT.
           - Returns the nth command line argument as a string.
           - If n=0 the name of the command is returned.
           - Note that ARG$ only works for CLI/Shell launched
             programs. See ace.guide for details about how to access
             Workbench arguments.
           - See also
           ARGCOUNT
           .

1.8 ARGCOUNT

ARGCOUNT *   - Returns the number of command line arguments.
                - See also ace.guide re: Workbench arguments.

1.9 AREA

AREA   - syntax: AREA [STEP](x,y)
                - Functions indentically to AmigaBASIC's AREA command.
                - Defines a set of up to 20 points to be joined
                  into a polygon and filled by
AREAFILL
                .

1.10 AREAFILL

AREAFILL   - syntax: AREAFILL [mode]
                - Same as AmigaBASIC's AREAFILL command.
                - The optional mode can be 0 or 1:

                    0 = fill polygon with current pattern and
                        foreground color.

                    1 = fill polygon with current pattern
                        but inverse of foreground color
                        (max-color-id - fdgnd-color-id).

                - See also
PATTERN
 command.

1.11 ASC

ASC   - syntax: ASC(X$)
                - Returns the ASCII code of the first character in X$.

1.12 ASSEM

ASSEM *   - syntax: ASSEM
                        <line of assembly code>
                        <line of assembly code>
                        .
                        .
                        END ASSEM
                - This allows for the inline inclusion of assembly source
                  code into the A68K file generated by ACE.
                - ACE does not attempt to check the correctness of the

```
                       inline code, leaving the task of assembly up to A68K.
                     - If you use this facility, it is assumed that you know
                       what you are doing.
                     - For correct handling of the assembly source lines,
                       do not place ASSEM or END ASSEM on the same line as
                       any of the code you wish to include.
```

1.13 ATN

```
ATN              - syntax: ATN(n)
                 - Returns the arctangent of n.
```

1.14 BACK

```
BACK *           - syntax: BACK n
                 - Moves the turtle back n steps.
```

1.15 BEEP

```
BEEP             - Issues a brief pulse from the speaker.
                 - BEEP doesn't flash the screen as it does in AmigaBASIC.
                 - This command is useful for alerting the user to an error
                   or other important event.
```

1.16 BEVELBOX

```
BEVELBOX *       - syntax: BEVELBOX (x1,y1)-(x2,y2),style
                 - This command renders a Wb 2.x/3.0 style 3D bevel-box
                   according to the specified rectangle and style.
                 - The style parameter can take on the following values:

                   Style        Bevel-Box
                   -----        ---------
                   1            RAISED
                   2            RECESSED
                   3            STRING-GADGET STYLE

                 - The style parameter will have different results
                   depending upon the combination of foreground and
                   background colours. The above styles hold true for
                   the standard Workbench 2.x colours.
```

1.17 BIN$

```
BIN$ *          - syntax: BIN$(n)
                - Returns a string containing the binary equivalent of n.
                - If n is a single-precision value, ACE coerces it to
                  integer.
```

1.18 BREAK

```
BREAK           - syntax: BREAK ON|OFF|STOP
                - These commands are used for enabling, disabling and
                  suspending ON BREAK event trapping.
                - See the Event Trapping section in ace.guide.
```

1.19 CALL

```
CALL            - Passes control to a user-defined subprogram,
                  shared library function, external function,
                  or user-defined machine code routine.
                - Subprogram CALLs can be recursive in ACE.
                - See also sections on subprograms, shared library access,
                  external functions and machine code calls in ace.guide.
```

1.20 CASE

```
CASE *          - This is ACE's version of the CASE statement and is
                  different from the Pascal CASE and C switch statements.
                - The syntax is:

                      CASE
                        <expression> : <statement>
                         .
                         .
                        [<expression> : <statement>]
                      END CASE

                   where <expression> can be any legal expression ranging
                   from a constant to a relational or mathematical
                   expression.

                - The expression is used as a boolean such that 0 is false
                  and any non-zero value is true.
                - Each expression is evaluated until one is found to be
                  true. The corresponding statement is then executed.
                - The statement can consist of a single legal ACE statement
                  (including block IF and loops) or a multi-statement.
```

1.21 CHDIR

```
CHDIR             - syntax: CHDIR <dirname>

                    where <dirname> is a string corresponding to the name of
                    a directory.

                  - If <dirname> is a legitimate directory and is accessible
                    from the current directory, it will become the new
                    current directory.
                  - In short, this is ACE's equivalent of the AmigaDOS "cd"
                    command, the only difference being that the path change
                    is not reflected in the shell prompt (if the program is
                    run from the shell).
```

1.22 CHR$

```
CHR$              - syntax: CHR$(n)
                  - Returns a string consisting of a single character with
                    the ASCII value n.
```

1.23 CINT

```
CINT              - syntax: CINT(n)
                  - Converts n to a signed short integer by rounding the
                    fractional portion.
                  - When the fractional portion is exactly .5, CINT *always*
                    rounds up in ACE, whereas in AmigaBASIC if the integer
                    portion is even, CINT rounds down, and up if the integer
                    portion is odd.
```

1.24 CIRCLE

```
CIRCLE            - syntax: CIRCLE (x,y),radius[,color-id,start,end,aspect]
                  - Start and end angles are specified in DEGREES *not*
                    radians because this is probably more useful when
                    thinking about circles.
                    (Note: this may be changed to radians in future).
                  - If a whole ellipse is to be drawn, the graphics library
                    DrawEllipse() function is used. However, if the start
                    angle is not 0 or the end angle is not 359 (these are
                    the defaults when not specified), a different routine
                    is used. The latter is quite slow and may well be changed
                    in a future release of ACE.
                  - The default ASPECT is .44 as in AmigaBASIC.
```

1.25 CLEAR ALLOC

```
CLEAR ALLOC *   – syntax: CLEAR ALLOC
                – Frees all memory allocated by calls to ALLOC.
                – Subsequent use of ALLOC is permitted.
                – This allows for a more intelligent use of memory
                  allocation, especially when memory is tight.
```

1.26 CLNG

```
CLNG            – syntax: CLNG(n)
                – Converts n to a signed long integer by rounding the
                  fractional portion.
                – When the fractional portion is exactly .5, CLNG *always*
                  rounds up in ACE, whereas in AmigaBASIC if the integer
                  portion is even, CLNG rounds down, and up if the integer
                  portion is odd.
```

1.27 CLOSE

```
CLOSE           – syntax: CLOSE [#]filenumber[,[#]filenumber..]
                  where filenumber represents an open file.
                – This command closes at least one open file.
                – Note that in ACE, CLOSE must be followed by at least one
                  filenumber, unlike AmigaBASIC.
                – See section on files in ace.guide.
                – See also
                ERR
                .
```

1.28 CLS

```
CLS             – Clears the current output window or screen and sets the
                  pen position to the upper left corner.
                – CLS does not affect any other screens or windows except
                  the one which is currently active.
```

1.29 COLOR

```
COLOR           – syntax: color fgnd-id[,bgnd-id]
                – Changes the foreground and/or background color to
                  fgnd-id and bgnd-id respectively.
                – Note that in ACE, you can change just the foreground
                  color, both the foreground and background colors,
                  but not the background color alone. This may be changed
                  in a future revision.
                – The
                PALETTE
```

```
                         command is used to change the colors
                          corresponding to given color-ids.
```

1.30 CONST

```
CONST *              – syntax: CONST <ident> = [+|-]<constant>[,..]
  where <constant> is a signed numeric constant.
– Defines a named numeric constant or constants, the type
  being *unaffected* by the the
DEFxxx
 directives or type
  (%&!#$) suffixes. All constant definitions are GLOBAL.
– A number of definitions can be separated by commas.
```

1.31 COS

```
COS                 – syntax: COS(n)
                    – Returns the cosine of n.
```

1.32 CSNG

```
CSNG                – syntax: CSNG(n)
                    – Converts n to a single-precision value.
```

1.33 CSRLIN

```
CSRLIN              – Returns the print line in the current user-defined screen
                       or window.
                    – CSRLIN and POS have no meaning in a CLI/shell and will
                       return 0 if used when a CLI/shell is the current output
                       window.
```

1.34 CSTR

```
CSTR *              – syntax: CSTR(<address>)
                    – Coerces a long integer address into a string.
                    – This is intended for taking an allocated area of memory
                       and using it as a string of characters. Be aware that
                       this memory block must be NULL terminated.
                    – A typical use for CSTR is something like this:

                            x$=CSTR(addr&)
```

– The maximum string length of MAXSTRINGLEN bytes in some
  functions still applies.

1.35   DATA

DATA   – syntax: DATA [numeric-constant | string-constant ↩
   ][,..]
– Stores numeric and/or string constants into a global
  data list to be accessed by the
READ
 statement.
– DATA statements may be located anywhere in a program and
  are non-executable.
– Strings need only be enclosed in quotes if they contain
  commas, spaces or colons or other non-identifier
  characters.
– In ACE, all numbers from DATA statements are currently
  stored as single-precision values with a possible loss of
  accuracy if LARGE long integers are originally specified.
  This may be rectified in a future revision. Thus far
  however, I have not had problems because of it. In order
  to overcome this, do the following:

```
READ X$
X&=LONGINT(X$)
DATA "123456789"
```

– In the above example, the BASIC function
VAL
 is
  substituted with
LONGINT
 because the former always
  returns a single precision value which is what we
  are trying to avoid, while the latter extracts a
  long integer from a string.

1.36   DATE$

DATE$ – Returns the current system date as a ten-character string
                 of the format: mm-dd-yyyy.

1.37   DAY

DAY *   – Returns the day of the week as an integer from ↩
   0..6,
  where 0=Sunday and 6=Saturday.
– The value returned by DAY reflects the last call to
DATE$

and is otherwise undefined.

1.38 DECLARE

DECLARE   - This has four uses in ACE:

1. DECLARE FUNCTION [<type>] <func-name>[%&!#$][(param-list)]
                                   LIBRARY [<lib-name>]

   (see section on shared library functions in ace.guide)

2. DECLARE FUNCTION [<type>] <func-name>[%&!#$][(param-list)]
                                   EXTERNAL

   which declares an external function. See also

EXTERNAL
 command.

   (see section on External References in ace.guide)

3. DECLARE SUB [<type>] subprogram-name[(parameter-list)]
                                   [EXTERNAL]

   which is used for forward SUB declarations. If
   the EXTERNAL keyword is used the subprogram is
   expected to be defined in another ACE module.
   The reference will be resolved at link time.

   (see "Creating & using ACE subprogram modules"
    in ace.guide)

In 1,2 and 3 above, <type> may be one of the following:

ADDRESS,LONGINT,SHORTINT,SINGLE,STRING

while param-list consists of comma-separated identifiers
each optionally preceded by one of the above type
specifiers.

4. DECLARE STRUCT <type> [*] <ident1> [,[*] <identN>..]

   where a structure variable of type <struct-type> is
   created. If "*" precedes the variable identifier,
   a pointer to the structure is created, otherwise
   memory is allocated. In both cases, "identN"
   holds the start address of the structure. In the
   latter case, the address is resolved at load time
   while in the former, the address is allocated at
   run time (eg: with ALLOC).

   - Only the first usage is supported by AmigaBASIC (but
without type specifier keywords).

1.39 DEF FN

```
DEF FN        - syntax:
DEF [FN]funcname[!#%&$] [(param-list)] [EXTERNAL] = <expr>
```

- As an extension to this syntax, in ACE it is also
  possible to follow the DEF keyword with one of the
  following:

        ADDRESS,LONGINT,SHORTINT,SINGLE,STRING

- These keywords may also precede each item in the
  parameter list.
- This command provides the simple defined function
  capability found in many BASICs.
- The parameters are passed by value and are combined
  in the expression on the right hand side of the "=" to
  yield a function return value.
- Like a subprogram, a defined function in ACE doesn't have
  access to global variables. Unlike the former, DEF FNs
  cannot use SHARED to get around this. In other words, if
  the function needs to use a particular value, you must
  pass it to the function via the parameter list. If a
  variable is defined in the expression (just by being used)
  its value will be local to the function (and unknown).
- The function may only be invoked as part of an expression,
  eg:

        DEF SEC(x)=1/COS(x)
        PRINT SEC(12)

  defines and invokes the secant function which can then be
  used in the same way as other built-in functions (eg:
  COS
).

- Note from the above that the "FN" prefix is optional
  in ACE. If used, there must be no spaces between "FN"
  and the function name.
- The fact that subprograms (SUBs) in ACE have return
  values and so can be treated as functions obviates the
  need for DEF FN to some extent, but the shorter definition
  may be considered better in some cases. Contrast the
  above definition with the following:

        SUB SEC(x)
          SEC=1/COS(x)
        END SUB

- A slightly different example is:

        DEF ADDRESS chipmem(bytes&) = ALLOC(bytes&,0)
```

                    which when invoked would return the start address of
                    a block of CHIP memory.
              – Once a function has been defined, you cannot redefine
                it (AmigaBASIC allows this) in the same program.
              – If the optional EXTERNAL keyword is used, the function
                will be externally visible to other modules. See ace.guide
                section "Creating & using ACE subprogram modules".
              – See the file ACEinclude:MathFunc.h for examples of
                defined functions (taken from Appendix E of the
                AmigaBASIC Manual).

## 1.40  DEFxxx

DEFxxx              – syntax: DEFxxx <letter> | _ [-<letter> | _] [, ..]

                    – The DEFxxx commands (DEFINT,DEFLNG,DEFSNG,DEFDBL,DEFSTR)
                      are global data type directives which affect data objects
                      in both the main program and subprograms.
                    – For example:

                          DEFLNG  a-z,_

                    declares all data objects to be of type LONGINT unless
                    overridden by another DEFxxx directive, variable
                    declaration or trailing character (%&!#$).
                    – DEFDBL currently defaults to single-precision since
                      double-precision floating-point is not yet supported by
                      ACE.

## 1.41  DIM

                DIM          – syntax:
          DIM [<type>]<name>(<i>[,..]) [SIZE <n>] [ADDRESS <addr>][,..]

          where <type> may be one of the following:

     ADDRESS,LONGINT,SHORTINT,SINGLE,STRING

              – ACE requires that _all_ arrays be dimensioned before use.
              – For a subscript range of 0..n, you must DIMension
                an array with an index of n.
              – Up to 255 dimensions can be specified with up to
                32767 elements per dimension. On a 3 Mb machine, around
                11 dimensions is the practical limit.
              – Each dimension must be specified as a short integer constant
          (literal or defined).
              – The SIZE option is for the specification of string element
                length other than the default MAXSTRINGLEN value.
              – The ADDRESS option allows you to specify some arbitrarily
                allocated area of memory for the array space.
              – Both options (SIZE and ADDRESS) may be used together in

DIM. This is not so for simple (string) variables where
only one or the other may be used (see
  STRING
    command).
When used in DIM, the SIZE option specifies how big each
string element is to be.
- SHARED is not an option and ACE arrays are shared in the
  same way as variables. See "Subprograms" in ace.guide.
- Arrays may be dynamically allocated in ACE, eg:

```
CONST STRSIZE=80
myStrArrayAddr& = ALLOC(numlines*STRSIZE)
IF myStrArrayAddr& = 0& THEN STOP
DIM wds$(1) SIZE STRSIZE ADDRESS myStrArrayAddr&
```

This will allocate space for an array of numlines strings,
each 80 bytes in length. A single array element is
specified just to keep ACE happy, but since there is no
array range checking, and the ADDRESS option has been
used, the number of elements in the array is in reality
numlines (a variable containing say, the number of lines
in a file).

Note that this means that you will be able to access
elements from 0..numlines-1. If you want 0..numlines
- or even 1..numlines - then the ALLOC line must read:

```
myStrArrayAddr& = ALLOC((numlines+1)*STRSIZE)
```

Here's a more complex example, showing how to
dynamically allocate space for a 2D array:

```
rangeArray& = ALLOC((N+1) * (SIZEOF(SHORTINT)*(3+1)))
IF rangeArray& = 0 THEN STOP
DIM range%(1,3) ADDRESS rangeArray&
```

The first index is just to keep ACE happy. Space is
allocated via ALLOC and the really critical thing here
is the "3" indicating how many columns in the table (as
it were) - 0 to 3 - to ensure correct array element
calculations.

Since ACE does no run-time array bounds checking, you can
specify range%(N,M) where N>=0 and M>=0 and M<=3. The zeroth
index is the reason why we need the +1 in two places in the
above ALLOC call.

See also ACEinclude:array_size.h for a subprogram which
returns the correct size to be passed to ALLOC for 2D and 3D
arrays, thus making such calculations unnecessary.

1.42   EOF

```
EOF              - syntax: EOF(n)
  where n is the filenumber of an open file.
- EOF is a function which returns either -1 or 0 depending
  upon whether the file pointer has reached the end-of-file
  or not.
- If the file doesn't exist or hasn't been opened, EOF
  returns -1.
- See also
ERR
.
```

1.43 END

```
END              - Closes standard libraries, performs other  ←
    cleanup
  operations and passes control back to parent process
  (CLI/Shell or Wb).
- Don't use END within an
IF..THEN..END IF
 block. Use
STOP
                  instead which is functionally equivalent in ACE.
```

1.44 ERR

```
ERR              - syntax: ERR
- This parameterless function returns the error code
  corresponding to a failed operation (or zero if no
  error has occurred) and then *immediately* clears the
  error code (sets it to zero).
- It is important to realise that the error code is
  cleared before the function returns its value, since
  unless this value is stored, it will be lost.
- The most typical usage is as part of a conditional test,
  eg: IF ERR<>0 THEN PRINT "Error!":STOP
- ERR may also be called after an error has been trapped
  by the ON ERROR event trapping mechanism. See ace.guide
  for more details about event trapping in ACE.
- Here are the current codes:

        -- AmigaBASIC codes --
      52  - Bad File Number
      54  - Bad File Mode

      -- AmigaDOS codes --
      103
      to
      233 - See The AmigaDOS Manual (Bantam),
            Error Codes and Messages.
```

```
                         -- ACE codes --
                         300 - Error opening serial port
                         301 - Error closing serial port
                         302 - Error reading from/querying serial port
                         303 - Error writing to serial port
                         304 - Bad channel number/serial port not open

                         400 - Error opening message channel
                         401 - Error closing message channel
                         402 - Error reading message channel
                         403 - Error writing to message channel
                         404 - Error waiting on message channel
                         405 - Bad message channel

                         500 - Error opening IFF file
                         501 - Error closing IFF file
                         502 - Error reading IFF file
                         503 - Bad IFF channel

        600 - Error opening screen

        700 - Error opening window
```

1.45 ERROR

```
ERROR              - syntax: ERROR ON|OFF|STOP
                   - These commands are used for enabling, disabling and
                     suspending ON ERROR event trapping.
                   - See the Event Trapping section in ace.guide.
```

1.46 EQV

```
EQV                - Boolean operator: X EQV Y.

                        X Y     Out
                        -----------
                        T T     T
                        T F     F
                        F T     F
                        F F     T
```

1.47 EXIT FOR

```
             EXIT FOR *      - This command allows for the premature,  ←
                conditional
              termination of a
            FOR..NEXT
             loop.
            - Since ACE uses the stack for FOR..NEXT loop counter &
              step values, issuing a
```

```
RETURN
 inside a FOR loop is
  dangerous because the top item on the stack is something
  other than the expected return address.
```
– In short, leaving a FOR loop before it has finished and
  never returning (
```
CALL
 and
GOSUB
 are okay since they will
  return to the loop) is unsafe in ACE, which is why EXIT
  FOR has been provided because it properly cleans up the
  stack before prematurely exiting the loop.
```
– When nesting one FOR loop inside another, be aware that
  the inner FOR loop's EXIT FOR will override any previous
  EXIT FOR directives in the enclosing outer FOR loop.
  As a consequence of this:

```
     FOR I=1 TO 10
       PRINT I
       FOR J=1 TO 5
         PRINT J
         IF MOUSE(0) THEN EXIT FOR
       NEXT
       IF MOUSE(0) THEN EXIT FOR
     NEXT
```

  will have the desired effect, whereas:

```
     FOR I=1 TO 10
       PRINT I
       IF MOUSE(0) THEN EXIT FOR  '..overridden below!
       FOR J=1 TO 5
         PRINT J
         IF MOUSE(0) THEN EXIT FOR
       NEXT
     NEXT
```

  will not. Observe the effect of running these two
  code fragments in order to see what's going on here.

1.48 EXIT SUB

EXIT SUB   – This command can only be used inside a subprogram and
                when encountered, has the effect of passing control back
                to the caller of the subprogram in which it appears.
              – If the current instantiation of the subprogram is the
                result of a recursive call, control will be returned
                to the previous instantiation of the same subprogram.

1.49 EXP

```
EXP                 - syntax: EXP(n)
                    - Returns e to the power n, where e is the base of
                      natural logarithms or 2.7182818284590.
```

1.50 EXTERNAL

```
                    EXTERNAL *      - syntax: EXTERNAL [FUNCTION] [<type>] <identifier ←
                      >[%&!$]
                    - Used to declare an external function or variable.
              - To declare the data type of an external object, either
                qualifier characters or one of the following type keywords
                may be used:

                ADDRESS,LONGINT,SHORTINT,SINGLE,STRING

                    - See the section on External References in ace.guide.
                    - See also the
                    DECLARE
                     command for an alternative
                       (and better) external function declaration syntax.
```

1.51 FILEBOX$

```
FILEBOX$ *          - syntax: FILEBOX$(title-string[,default-directory])
                    - This function invokes a file requester and returns
                      the user's selection as a fully qualified path.
                    - The title-string is displayed in the title bar of
                      the file requester (eg: "Open", "Select a file").
                    - If the (optional) default-directory is specified,
                      the file requester's initial "view" will be in that
                      directory.
                    - If the program is running under Wb 2.04 or higher,
                      an ASL file requester appears. If not, an Arp
                      requester is invoked which means that if you are
                      running Wb 1.3 or lower, you'll need arp.library
                      in your LIBS: directory.
                    - If you are using FileBox$ under Wb 1.3 make sure
                      you have a stack (in the shell/CLI or Tool) which
                      is at least 5000 bytes in size.
```

1.52 FILES

```
FILES               - syntax: FILES [TO <storefile>] [,<target>]
                    - Gives an unsorted directory listing ala AmigaBASIC
                      except that ACE's version takes two optional arguments
                      while AmigaBASIC's takes one (<target>).
                    - If <storefile> is specified, the listing will be
                      captured by that file.
```

       – If <storefile> is omitted, it is assumed that the
         program containing the FILES command was invoked
         from a shell or CLI (since the listing will be
         displayed).
       – The <target> argument can be a file, directory or
         AmigaDOS device name which is to be the subject
         of the directory listing.

1.53 FIX

FIX   – syntax: FIX(n)
– The function returns the truncated integer portion of n.
– FIX(n) is equivalent to SGN(n)*INT(ABS(n)).
– Whereas INT(n) rounds off a negative number
  to the next lowest whole number, FIX does not.

or

– syntax: FIX n
– The command which is found only in ACE is intended to
  have a similar effect to the FIX button found on some
  calculators that is, to change the number of decimal
  places ACE rounds a single-precision number to.
– FIX utilises the ami.lib function arnd(). When the value
  of n is anything other than 8, arnd() is invoked. This
  affects the commands:
PRINT
,
PRINTS
,
WRITE#
,
PRINT#
 and

STR$
.
– FIX should be considered experimental since I have not
  completely figured out what all the values of n (as used
  directly by arnd()) do yet.
– In a future release, a given value for n may have
  different results than it does now. Currently, n may be
  positive or negative.

     Examples

     FIX -3
     PRINT 12.3456

     would display: 12.35

– PRINT USING will obviate the need for this command in
  a future release in any case.

1.54 FONT

```
        FONT *           - syntax: FONT <name>,<size>
                         - Changes the font for the current output window.
   - <font> is a string such as "opal" or "opal.font"
     and <size> is an integer point size.
                         - Currently only works for windows created with the

        WINDOW
         command, not for shells.
   - It is best to follow a FONT statement with a LOCATE
     command to "notify" the window of the font change (eg.
     LOCATE 1,1). This ensures correct line-feed height for
     future PRINT statements.
                         - See also the
        STYLE
         command (which works
     in ALL windows).
```

1.55 FOR..NEXT

```
FOR..NEXT        - syntax: FOR <variable>=x TO y [STEP z]
                 - The statements between FOR and NEXT are iterated
                   the number of times it takes for <variable> to become
                   equal to or greater than y (or less than y if z is
                   negative) starting from x. The loop index <variable>
                   is incremented by z, or 1 if STEP is not specified.
                 - NEXT can only be followed by a variable, colon or
                   comment and must appear on a separate line or in a
                   multi-statement (not after THEN for example).
                 - Any attempt to use a shared variable as a FOR loop index
                   will result in an (intentional) compilation error.
                 - If you want to branch out of a FOR loop never to return,
                   use EXIT FOR. See also the further discussion of this
                   issue (including RETURNing from within a FOR loop) in the
                   "Limitations" section of ace.guide.
```

1.56 FORWARD

```
FORWARD *        - syntax: FORWARD n
                 - Move the turtle forward n steps.
```

1.57 FRE

```
FRE                 - syntax: FRE(n)
                      where n is -1,0,1,2 or 3.
                    - Since ACE's run-time environment is different to
                      AmigaBASIC's, FRE returns different values and
                      takes different arguments than in AmigaBASIC.
                    - FRE returns the amount of free system memory according
                      to n:

                         n = -1  ->  total CHIP + FAST memory free.
                         n =  0  ->  total CHIP memory free.
                         n =  1  ->  total FAST memory free.
                         n =  2  ->  largest contiguous CHIP memory available.
                         n =  3  ->  largest contiguous FAST memory available.
```

1.58 GADGET

```
        GADGET *        - syntax: GADGET id,status[,gadval,rectangle,type ↩
          [,style]]

        where id is a unique gadget ID from 1 to 255 and status
        is 1 or 0 to enable or disable the gadget, respectively.
        The keywords ON and OFF can be used instead of 1 and 0.

      - The remainder of the parameters are optional, but all
        except style must be specified when creating a new
        gadget. If a string or longint gadget has no style
        specification, the default is left-justification of text.

      - The first of these, gadval, is either a string or long
        integer (see below); rectangle defines the border of
        the gadget as (x1,y1)-(x2,y2).

      - The GADGET command creates a new gadget or alters the
        status of an existing gadget according to the above
        and in accordance with the final two parameters: type
        and style, as follows (gadval meaning is also shown):

      - Type may either be a numeric value from 1 to 5 or one
        of the following keywords: BUTTON, STRING, LONGINT,
        POTX or POTY, correspondingly.
```

Type	Gadget	Style	Effect	GadVal
1	Boolean	1	All points inside the gadget are complemented when it is clicked (this is the default).	Gadget text
		2	A box is drawn around the gadget when clicked.	Gadget text
		3	Borderless.	Gadget text

```
        2     String    1     Center justifies text.   Default text

                        2      Right justifies text.

                       (The default is left justification).


        3     LongInt   1     Center justifies number. Default number
                                                        (as string)
                        2      Right justifies number.

                       (The default is left justification).


        4     Horiz.
              Slider    1     Borderless.              Maximum
                                                       slider value
                                                       (0..gadval)

        5     Vertical
              Slider    1     Borderless.              Maximum
                                                       slider value
                                                       (0..gadval)

    OR

       - syntax: GADGET(n)

         where n is a number from 0 to 3.

       - The GADGET function returns information about the
         last gadget event according to the following:

         N      Returns
         -      -------
         0      -1 if a gadget event has occurred since the last
                call to GADGET(0), 0 otherwise.

         1      The number of the last gadget selected. If the
                window's close gadget was clicked after doing a
                GADGET WAIT 0, 256 will be returned. This is not
                the case for event trapping of gadgets, where
                ON WINDOW should be used instead.

         2      Returns the address of the string from the most
                recently selected string gadget or the long integer
                value from the most recently selected LongInt
                gadget.
                In the former case, use ACE's
       CSTR
        function to
                convert the address into an ACE string.

         3      Returns the slider position of the most recently
                selected (horizontal or vertical) proportional
                gadget.
```

1.59   GADGET CLOSE

```
GADGET CLOSE *  - syntax: GADGET CLOSE id
- This command removes the specified gadget from the
  current output window and should always be called
  when you are finished with a gadget.
- Make sure that the window belonging to the gadget you
  wish to close is the current output window (see
WINDOW

OUTPUT
 command).
```

1.60   GADGET MOD

```
GADGET MOD *    - syntax: GADGET MOD id,knob-position[,max- ↩
   positions]
- This command modifies the specified proportional
  gadget.
- The new knob position (within the gadget's body) must
  be specified.
- The optional max-positions parameter if specified changes
  the number of discrete positions in which the knob may be
  found. A significant change from the previous value given
  (eg. see the gadval parameter in the
GADGET
 command) may
   result in a change to the knob size.
```

1.61   GADGET ON

```
GADGET ON .. *  - syntax: GADGET ON|OFF|STOP
                - These commands are used for enabling, disabling and
                  suspending ON GADGET event trapping.
                - See the Event Trapping section in ace.guide.
```

1.62   GADGET WAIT

```
GADGET WAIT *  - syntax: GADGET WAIT id
- This command puts the program to sleep until it receives
  a message that the specified gadget has been selected.
- If id=0 the program will wake up when ANY gadget is
  selected. A call to
```

```
                    GADGET
                    (1) can then be used to
                      determine the number of the gadget.
```

1.63 GOSUB..RETURN

```
GOSUB..RETURN    - syntax: GOSUB <label> | <line>
                 - GOSUB transfers control to the specified label or line.
                 - RETURN passes control back to the statement following the
                   most recent GOSUB command.
      - Issuing a RETURN without a matching GOSUB will generally
        invoke the GURU.
```

1.64 GOTO

```
GOTO             - syntax: GOTO <label> | <line>
                 - Transfers control to the specified label or line.
```

1.65 HANDLE

```
HANDLE *         - syntax: HANDLE(n)
                   where n is the file number of an OPENed file (1..255).
                 - This function returns a long integer which is a pointer
                   to a dos file handle suitable for use with dos.library
                   functions such as Read (xRead when declared in
                   ACE/AmigaBASIC).
                 - If HANDLE returns 0 the file does not exist or can't be
                   opened as requested.
```

1.66 HEADING

```
HEADING *        - Returns the turtle's current heading in degrees (0..359).
```

1.67 HEX$

```
HEX$             - syntax: HEX$(n)
                 - Returns a string which represents the hexadecimal value
                   of the decimal argument n.
```

1.68 HOME

```
HOME *           - Move the turtle to its home position.
```

1.69 IF

```
IF              - syntax: IF..THEN..[ELSE..]
        IF..GOTO..[ELSE..]
        IF..THEN
        .
        [ELSE]
        .
        END IF
```
- ELSEIF is not yet implemented.
- IF..[ELSE]..END IF blocks can be nested.
- Use
STOP
 rather than
END
 before an END IF
  otherwise the compiler will become confused.
- There must be _something_ between IF..THEN
  and END IF, even if only a blank line or comment,
  eg.
```
                IF x=2 THEN
                  '..do something or maybe nothing
                END IF
```


1.70 IFF

```
IFF *           - syntax: IFF(channel,n)
```
- This function returns information about the IFF graphics
  file associated with the specified channel.
- The channel parameter must be in the range 1..255.
- The values returned are dictated by N thus:

```
  N     Return value
  -     ------------
  0     Address of name of IFF picture form (eg: ILBM).
        Use ACE's
CSTR
 function to retrieve the string.

  1     Width of picture.
  2     Height of picture.
  3     Depth of picture.
  4     Screen Mode to use in SCREEN command. Note: if
        IFF(channel,3) returns a depth of 6, HAM mode
        is currently assumed even though it might be
        extra-halfbrite. If the picture doesn't render
        correctly, use screen-mode 6 rather than 5 (see
```
SCREEN
 command). Alternatively, don't specify
        the screen-id when using the
IFF READ
 command.

This issue may be resolved in a future revision.

- Information returned by values to this function when
  N is in the range 1..4 can be used directly in a SCREEN
  command.
- See also
IFF OPEN
,
IFF READ
 and
ERR
.

1.71 IFF CLOSE

IFF CLOSE *   - syntax: IFF CLOSE [#]channel
- Closes the specified IFF channel.
- If a screen was opened by IFF READ, IFF CLOSE will
  close this.
- See also
ERR
.

1.72 IFF OPEN

IFF OPEN *   - syntax: IFF OPEN [#]channel,file-name
- This command associates an IFF picture file with the
  specified channel.
- All subsequent IFF command/function calls use this
  channel number.
- The IFF OPEN command also stores important information
  about the picture file for
IFF READ
 and
IFF
(channel,n).
- See also
ERR
.

1.73 IFF READ

IFF READ *   - syntax: IFF READ [#]channel[,screen-id]
- This command loads the IFF picture from the file
  associated with the specified channel.
- The screen-id is optional. If not supplied, a non-ACE
  screen and window will be used to display the picture,

                    which is closed later by a call to
            IFF CLOSE
            .
            – Otherwise, the screen should be opened in accordance with
    the information returned via the
            IFF
             function.
            – See also
            ERR
             and ace.guide.

1.74 IMP

IMP   – Boolean operator: X IMP Y.

                    X Y   Out

                    T T   T
                    T F   F
                    F T   T
                    F F   T

1.75 INKEY$

INKEY$ – syntax: INKEY$
                    – Returns a single character string when a keystroke
                      is pending, otherwise the NULL string is returned.
                    – INKEY$ works fine in user-defined windows, but since
                      a normal CON: window intercepts all keystrokes, INKEY$
                      is not very useful in a shell/CLI.

1.76 INPUTBOX

            INPUTBOX *   – syntax: INPUTBOX(prompt[,title][,default][,xpos ←
              ][,ypos])
            – This function returns a long integer value after invoking
              a requester which prompts the user to enter a value. If
              you need to get a single-precision value, apply VAL to
              the result of the
            INPUTBOX$
             function (see next entry).
            – An OK and Cancel gadget allow the user to accept or
              reject the entered value. Zero is returned if the Cancel
              gadget is selected.
            – The prompt string must be specified but all other
              parameters are optional: title goes into the requester's
              title bar; default is a string containing a default
              integer value which becomes the return value if nothing
              is entered; xpos and ypos specify where to place the

```
                       requester on the screen.
                     - Example: num& = INPUTBOX("Enter a number:",,"12345")
```

1.77   INPUTBOX$

```
INPUTBOX$ *       - syntax: INPUTBOX$(prompt[,title][,default][,xpos][,ypos])
                  - This function returns a string value after invoking
                    a requester which prompts the user to enter a value.
                  - An OK and Cancel gadget allow the user to accept or
                    reject the entered string. If Cancel is selected the NULL
                    string is returned.
                  - The prompt string must be specified but all other
                    parameters are optional: title goes into the requester's
                    title bar; default is a string return value to be used if
                    no new value is entered; xpos and ypos specify where to
                    place the requester on the screen.
                  - Example: command$ = INPUTBOX$("Enter a command:")
```

1.78   INPUT

```
INPUT             - syntax: INPUT [<prompt-string>] [;|,] var1 [[;|,] varN..]
                  - Strings, integers and fixed-point or exponential format
                    single-precision values can be input from the keyboard.
                  - Each value must appear on a separate line even when
                    a single INPUT statement contains multiple variables.
                  - If a semicolon precedes a variable "? " will appear,
                    while if a comma is used no "? " will appear.
                  - As of ACE v2.0 INPUT works with any screen or window
                    mode.
```

1.79   INPUT

```
                  INPUT #         - syntax: INPUT #filenumber,<variable-list>
                  - Reads data items from a sequential file.
                  - The variables in <variable-list> must each match the type
                    of item being read.
                  - If unambiguous data format is required, it is best to
                    use
                  WRITE#
                   to store the values that INPUT# will read
                    since WRITE# separates each item with commas and delimits
                    strings with double quotes allowing for spaces. WRITE#
                    will also result in more efficient use of disk space and
                    faster reading by INPUT#.
                  - ACE accepts white space (line feeds, spaces, tabs),
                    commas and quotes as delimiters for each field in a
                    sequential file.
                  - AmigaBASIC and ACE sequential file formats are virtually
                    identical.
```

```
        - See also "Files" section in ace.guide.
        - See also
        ERR
        .
```

1.80 INPUT$

```
INPUT$          - syntax: INPUT$(X,[#]filenumber)
- Returns a string of X characters from the filenumber'th
  file.
- There is a 32K upper limit for X in ACE, but if you
  want to read a whole file for example, and the file
  length (determined by the
LOF
 function) is greater than
  MAXSTRINGLEN you should do the following:

        STRING myString SIZE N
        OPEN "I",#1,filename$
         myString = INPUT$(LOF(1),#1)
        CLOSE #1

  or if you want to allocate space at run-time according
  to the exact file size:

        bytes& = LOF(1) + 1      '..need "+1" for EOS marker
        addr& = ALLOC(bytes&)
        STRING myString ADDRESS addr&
        OPEN "I",#1,filename$
         myString = INPUT$(bytes&,#1)
        CLOSE #1

- This method should only be used for small text files as
  it is slow, and text is really the only useful thing to
  put in a string if you wish to manipulate it. Some string
  functions will react unexpectedly to non-text characters
  in strings.
- If you wish to read a large file rapidly, it's best to
  use the dos.library function Read (declared as xRead in
  BASIC). The sound player play.b gives an example of this.
- In general INPUT$ is most useful for reading a few
  characters at a time from a file. If you wish to read a
  line at a time, use
LINE INPUT#
. Use
INPUT#
 if you want
  to read numbers or delimited strings.
- INPUT$ in ACE is only used for sequential file input, so
  the filenumber is not optional. In AmigaBASIC, if the
  latter is omitted, input is taken from the keyboard.
  Not so in ACE.
- See also section on files in ace.guide.
```

1.81 INSTR

```
INSTR              - syntax: INSTR([I,]X$,Y$)
                   - INSTR searches for the first occurrence of Y$ in X$ and
                     returns the character position from 1..N in X$.
                   - If the optional offset I is specified, the search starts
                     from that position, otherwise the search starts from the
                     first character in X$.
                   - If I is greater than len(X$) or X$="" or Y$ is not found
                     in X$ or len(Y$) > len(X$), INSTR returns 0.
                   - If Y$="", INSTR returns I or 1.
                   - X$ and Y$ can be string expressions, variables or
                     literals or any combination thereof.
```

1.82 INT

```
INT                - syntax: INT(n)
                   - Returns the largest integer less than or equal to n.
```

1.83 KILL

```
KILL               - syntax: KILL <filespec>
                   - Deletes a file or directory.
```

1.84 LEFT$

```
LEFT$              - syntax: LEFT$(X$,I)
                   - Returns a string which contains the leftmost I characters
                     of X$.
                   - If I > len(X$), the whole string (X$) is returned.
                   - If I = 0, the NULL string is returned.
```

1.85 LEN

```
LEN                - syntax: LEN(X$)
                   - Returns the number of characters in X$.
```

1.86 LET

```
LET                - syntax: [LET] <variable> = <expression>
                   - LET assigns a value to a variable.
                   - Its use is optional so that LET X=1 is equivalent
                     to X=1.
```

1.87 LIBRARY

```
LIBRARY              - syntax: LIBRARY [CLOSE] [<libname>]
                     - Opens or closes one or more Amiga shared libraries.
                     - Note that <libname> may be with or without quotes
                       and can either end in ".library", ".bmap" or have no
                       file extension whatever in ACE.
                     - For example, to open the graphics library, two legal
                       syntaxes are:

                             LIBRARY graphics
                     and
                             LIBRARY "graphics.library"

                     - LIBRARY CLOSE closes all open libraries or a single
                       library can be specified instead.
                     - See "Shared library function calls" section in ace.guide.
```

1.88 LINE

```
LINE                 - The syntax of this command - apart from the simple
                       case of LINE (x1,y1)-(x2,y2)[,color,b[f]] - is a little
                       unclear from the AmigaBASIC manual.

                     - The syntax of the LINE command in ACE is currently as
                       follows:

                             LINE [STEP](x1,y1)[-(x2,y2)[,[color],[b[f]]]]

                     - The second STEP directive has been omitted, but may be
                       added in a future revision.
                     - A statement such as LINE STEP (100,90) will cause a line
                       to be drawn from the last referenced coordinate to
                       100,90.  In addition, this use of LINE does *not* allow
                       for colour setting as can be seen from the ACE syntax
                       specification whereas LINE (100,90)-(200,150),color does.
                       The same is true for the "b" and "bf" options. A future
                       version may correct this problem.
                     - Note: When using "b" or "bf", x2 must be >= x1 and y2
                       must be >= y1 otherwise display weirdness will result!
```

1.89 LINE INPUT

```
                LINE INPUT      - syntax: LINE INPUT #filenumber,<string-variable>
                     - Reads a line from the filenumber'th sequential file and
                       stores it in <string-variable> (simple variable or array
        element).
                     - If <string-variable> does not exist, ACE creates it.
                     - Lines are delimited by a line-feed character (ASCII 10)
                       and the string which is returned consists of the
                       characters up to but not including the line-feed.
                     - Note that the AmigaBASIC manual (8-72) shows a semicolon
```

```
                    instead of a comma in the above syntax which is incorrect
                    since AmigaBASIC itself accepts only a comma.
                 - The alternative form of LINE INPUT for keyboard input is
                    not currently implemented in ACE.
                 - LINE INPUT will not read more than MAXSTRINGLEN
                    characters.
                 - See also
                 INPUT$
                  (which will read up to 32K of
                   characters),
                 INPUT#
                  and ace.guide's section on files.
                 - See also
                 ERR
                 .
```

1.90 LOCATE

```
LOCATE            - syntax: LOCATE line[,column].
                 - LOCATE changes the printing position for the current
                    screen or window.
                 - Note that the use of LOCATE on a screen or user-defined
                    window currently also changes the next graphics drawing
                    coordinates.
```

1.91 LOF

```
LOF               - syntax: LOF(n)
                  where n is the file number of an open file.
                 - LOF returns the length of the file in bytes.
                 - If the file is not open or is non-existent, LOF returns 0.
                 - See also
                 ERR
                 .
```

1.92 LOG

```
LOG               - syntax: LOG(n)
                 - Returns the natural logarithm of n (log base e of n).
                 - The argument n should be greater than zero.
```

1.93 LONGINT

```
                    LONGINT *        - syntax: LONGINT <identifier>[,..]
                    - Declares and initialises (to zero) one or more long
                      integer variables.

        OR

                    - syntax: LONGINT(X$)
                    - This function returns the numeric value of X$ as a long
                      integer number.
                    - The hexadecimal and octal directives (&H and &O) may
                      prefix the string in order to allow the handling of these
                      bases.
                    - LONGINT strips off leading whitespace (eg: spaces, tabs).
                    - The main use for this function is to overcome the loss of
                      accuracy which results when VAL is used to extract a
                      _large_ long integer value from a string.
                    - See also
                    VAL
                    .
```

1.94 MENU

```
MENU                - syntax: MENU menu-id,item-id,state[,title[,command-key]]
                    - This command creates or modifies the state of a menu or
                      menu item as per AmigaBASIC.
                    - The final optional parameter is peculiar to ACE and if
                      used, specifies the Amiga-<key> sequence which if issued
                      results in the selection of the corresponding menu
                      option. The command key option is displayed along with
                      the menu item when the menu is rendered.
                    - The state parameter can have the following values:

                          State           Effect
                          -----           ------

                          0               Menu or item is disabled (shadowed).

                          1               Menu or item is enabled.


                          2               Menu item is checkmarked.
                                          There must be at least 2 spaces
                                          preceding the item for the tick
                                          to be rendered properly.

                    - The most advisable method of creating menus is to start
                      from the first menu and first item in each menu, and code
                      them in sequence thereafter.

        OR

                    - syntax: MENU(n)
                    - This function returns information about the most recently
```

                    selected menu and item. If n=0 the number of the menu is
                    returned. If n=1 the number of the menu item is returned.
                  – MENU(0) returns 0 between menu events after being called
                    once for a particular menu selection.
                  – This function must be used in conjunction with MENU event
                    trapping or WAITing.

1.95 MENU CLEAR

              MENU CLEAR *   – syntax: MENU CLEAR
              – This command is the equivalent of MENU RESET in
                AmigaBASIC.
              – The result of calling this is to clear the menu strip for
                the current output window. In AmigaBASIC the initial menu
                for the interpreter's window is restored if a new menu is
                set up in that window. This does not apply in ACE.
              –
              WINDOW CLOSE
               performs a menu clear in case you don't.

1.96 MENU ON

MENU ON ..   – syntax: MENU ON|OFF|STOP
                  – These commands are used for enabling, disabling and
                    suspending ON MENU event trapping.
                  – See the Event Trapping section in ace.guide.

1.97 MENU WAIT

MENU WAIT *   – syntax: MENU WAIT
                  – This command puts the program to sleep until menu
                    activity is detected.

1.98 MESSAGE CLEAR

              MESSAGE CLEAR * – syntax: MESSAGE CLEAR [#]channel
              – Clears the message port associated with the specified
                channel.
              – See also
              ERR
               .

1.99   MESSAGE CLOSE

```
MESSAGE CLOSE * - syntax: MESSAGE CLOSE [#]channel
- Closes the specified message channel.
- See also
ERR
.
```

1.100   MESSAGE OPEN

```
MESSAGE OPEN *  - syntax: MESSAGE OPEN [#]channel,port-name,mode
- Creates a message channel for reading (mode="R")
  or writing (mode="W").
- If the channel is for writing, the port-name is
  the name of a message port which is assumed to
  exist. If it does not exist an error will result
  (see ERR).
  You can therefore poll a remote port to determine
  when it has been created.
- See also
ERR
.
```

1.101   MESSAGE READ

```
MESSAGE READ *  - syntax: MESSAGE READ [#]channel,buffer
- Reads a message into buffer from the specified message
  channel.
- See also
ERR
.
```

1.102   MESSAGE WAIT

```
MESSAGE WAIT *  - syntax: MESSAGE WAIT [#]channel
- Waits for a message to appear on the specified channel.
- Please note that if no message is forthcoming, this
  command will wait forever.
- Waiting on a port opened for writing (mode = "W") has the
  effect of waiting for the remote task to signal that it
  has accepted a message written to its port. This allows
  for synchronisation between processes, ie. A writes to B,
  B accepts message from A, A continues processing.
- See also
ERR
.
```

1.103 MESSAGE WRITE

MESSAGE WRITE * – syntax: MESSAGE WRITE [#]channel,buffer
– Writes a message to the specified message channel from
  the buffer.
– See also
ERR
.

1.104 MID$

MID$ – syntax: MID$(X$,I[,J])
                  – Only the MID$ _function_ is currently implemented in ACE.
                  – Returns a string containing J characters from X$ starting
                    from the Ith character.
                  – If J is omitted or there are fewer than J characters
                    to the right of (and including) the Ith character, all
                    characters from the Ith position to the end of the string
                    are returned.
                  – If I > len(X$), MID$ returns the NULL string.

1.105 MOD

MOD   – Modulo arithmetic operator: X MOD Y.

                        eg: 101 MOD 10 = 1

1.106 MOUSE

MOUSE   – syntax: MOUSE(n)
                  – Returns information about the current status of the
                    mouse.
                  – Values of n ranging from 0..2 are presently meaningful
        in ACE.
                  – MOUSE(0) returns –1 or 0 to indicate whether the
                    left mouse button is currently being pressed or not.
                  – MOUSE(1) returns the X location of the mouse pointer
                    in the current output window or screen.
                  – MOUSE(2) returns the Y location of the mouse pointer
                    in the current output window or screen.
                  – Future revisions of ACE will add more functionality to
                    MOUSE(n).

1.107 MOUSE ON

```
MOUSE ON ..       - syntax: MOUSE ON|OFF|STOP
                  - These commands are used for enabling, disabling and
                    suspending ON MOUSE event trapping.
                  - See the Event Trapping section in ace.guide.
```

1.108 MSGBOX

```
MSGBOX *          - syntax: MSGBOX(message,button-text1[,button-text2])
                  - This function invokes a system requester having one or
                    two buttons (boolean gadgets) with the specified text
                    in each, plus a message in the requester's main body
                    as specified by the message parameter.
                  - If only button-text1 is given, a single button is
                    rendered, otherwise two buttons appear.
                  - The function's return value is -1 or 0 depending
                    upon whether the first or second button is selected by
                    the user. With only one button present, the return
                    value is always -1.

                  - Example: result = MsgBox("Really Quit?","Yes","No")

  OR

    - syntax: MSGBOX message,button-text
    - This statement can be used to display a simple system
      requester. Since no value is returned via this statement,
      only a single button is permitted.

    - Example: MsgBox "File Deleted!","Continue"

                  - Note that the message may only consist of a single line
                    but a future revision will allow for multiple lines.
                  - Note also that under Wb 1.3 the "message" text is used
                    to determine the width of the requester. Under Workbench
                    2.x/3.0, the operating system proportions the requester
                    appropriately.
```

1.109 NAME

```
NAME              - syntax: NAME <filespec1> AS <filespec2>
                  - Renames a file or directory.
```

1.110 NOT

```
NOT               - Boolean operator: NOT X.

                        X         Out
                        -----------
                        T         F
                        F         T
```

1.111 OCT$

```
OCT$            - syntax: OCT$(n)
                - Returns the octal string representation of the long
                  integer value n.
```

1.112 ON..GOTO/GOSUB

```
ON..GOTO/GOSUB  - syntax 1: ON <integer-expr> GOTO | GOSUB <label> | <line>

                  eg: ON n GOTO one,two,three,four,five

                  such that if n=1 the program will branch to the label
                  "one" and if n=4 the branch will be to "four".

                - syntax 2: ON <event-spec> GOTO | GOSUB <label> | <line>
                - See "Event Trapping" section in ace.guide.
```

1.113 OPEN

```
                OPEN            - syntax: OPEN mode,[#]filenumber,<filespec>
                  which is the same as syntax 1 in AmigaBASIC
                  except that no file-buffer size can be specified.
                - Mode is an upper or lower case character where:

                        - "I" = open file for input

                        - "O" = open file for output

                        - "A" = open file for appending;
                                creates new file if <filespec>
                                doesn't exist.

                - Filenumber is a value from 1..255 and <filespec>
                  is a string containing the file name (eg: "test.doc",
                  "df1:letters/santa").
                - Multiple files can be open simultaneously.
                - See also
                ERR
                .
```

1.114 OPTION

```
OPTION *        - syntax: OPTION <switch>+|-[,<switch>+|-..]
                - Compiler directives (switches) can be issued via this
                  command instead of from the command line. The latter
                  only allows for compiler directives to be *activated*.
                - Each switch must be followed by a "+" or "-" with
```

the former activating the directive and the latter
neutralising it.
– Switches currently implemented are: b,c,E,i,l,m,O,w
– See ace.guide, "Compiler options" for details of each
  switch. Notice that for switches i and O, activation
  or deactivation takes effect at the end of compilation.

1.115 OR

OR   – Boolean operator: X OR Y.

                      X Y   Out

                      T T   T
                      T F   T
                      F T   T
                      F F   F

1.116 PAINT

PAINT   – syntax: PAINT (x,y)[[,color-id][,border-id]]
                  – PAINT flood-fills an enclosed region with the
                    color specified by color-id and if the latter
                    is omitted, the current foreground pen is used.
                  – If border-id is not specified, color-id is used
                    to determine when to stop the filling process by
                    looking for a border of that color. The use of
                    border-id allows a region to be filled with one
                    color and be bordered by another.
                  – x and y can be anywhere within the enclosed region.
                  – Note that the ACE version of PAINT has no STEP
                    option so x and y constitute an absolute coordinate.
                  – STEP may be added in a future revision.

1.117 PALETTE

PALETTE   – syntax: PALETTE color-id,R,G,B
                    where R,G,B are the red, green and blue color
                    components of color-id, each in the range 0..1.
                  – Palette changes colors in the current screen
                    (including the Workbench!).

1.118 PATTERN

PATTERN   – syntax: PATTERN [line-pattern][,area-pattern] | RESTORE
                  – Same as in AmigaBASIC with the addition of a RESTORE
                    option. PATTERN RESTORE resets the line and area patterns

                              to their default values.
                            – The line-pattern is a short integer value.
                            – The area-pattern is a DIM'd short integer array.
                            – The number of elements in area-pattern must be a power
                              of 2.

1.119 PEEKx

PEEKx   – syntax: PEEKx(<address>)
                            – The functions PEEK,PEEKW and PEEKL return an 8-bit,
                              16-bit and 32-bit value from memory, respectively.

1.120 PENDOWN

PENDOWN *   – Lowers the turtle's "pen". This enables drawing by the
                              turtle graphics commands.

1.121 PENUP

PENUP *   – Raises the turtle's "pen". This disables drawing by the
                              turtle graphics commands.

1.122 POINT

POINT   – syntax: POINT(x,y)
                            – Returns the color-id of a point in the current output
                              window or screen.

1.123 POKEx

POKEx   – syntax: POKEx <address>,<numeric-value>
                            – The commands POKE,POKEW and POKEL change the contents of
                              <address> to <numeric-value>.
                      – The number of bits affected is 8, 16 and 32 respectively.
                            – Unless you know what you are POKEing and why, don't (!!)
                              or you can expect a visit from the Guru.

1.124 POS

POS   - Returns the print column in the current user- ←
     defined
   screen or window.
- Note that the syntax is different from AmigaBASIC where a
  dummy argument of zero is used: POS(0).
- POS and
CSRLIN
 have no meaning in a CLI/shell and will
  return 0 if used when a CLI/shell is the current output
  window.

1.125 POTX

POTX *   - syntax: POTX(n)
                  where n=0 or 1 (game port 1 or 2).
                - Returns a short integer value corresponding to the
                  current potentiometer reading on pin 5 of the game port.
      - POTX(0) returns 0 currently.

1.126 POTY

POTY *   - syntax: POTY(n)
                  where n=0 or 1 (game port 1 or 2).
                - Returns a short integer value corresponding to the
                  current potentiometer reading on pin 9 of the game port.
      - POTY(0) returns 0 currently.

1.127 PRINT

PRINT   - syntax: PRINT [<expression>][,|;| ..]
                  where <expression> is a string or numeric value to
                  be printed at the current print location of the current
                  (DOS or Intuition) output window.
                -
                LOCATE
                 can be used to set the location for the next PRINT
                  command. So can
                SETXY
                 for printing in a non-shell window.
                - PRINT can be abbreviated to '?' as in AmigaBASIC.
                - If <expression> is followed by a semi-colon, a line-feed
                  will not occur before the next PRINT.
                - If <expression> is followed by a comma, the effect is
                  the same except that first, a horizontal tab (CHR$(9))
                  is sent to the output window.
                - Note that ASCII 9 does not have exactly the same effect
                  as an AmigaBASIC tab, but the result is similar.
                  If spacing is critical, you should use TAB or SPC.

1.128 PRINT

```
PRINT #         - syntax: PRINT #filenumber,<expression>[,|;| ..]
  where <expression> is a string or numeric value to
  be printed at the current print location in the
  filenumber'th file.
- PRINT can be abbreviated to '?' as in AmigaBASIC.
- This version of PRINT # writes values to a file in the
  same format as they would appear in a window.
- One oddity is that since ACE strings are NULL-terminated,
  and this NULL (ASCII 0) is normally not displayed, any
  attempt to send this character to a file, eg:

        PRINT #filenumber,CHR$(0)

  should by all rights be ignored. However, since some
  programs write NULLs to files as delimiters, ACE does NOT
  ignore a lone CHR$(0). A consequence of this is that if
  you send an empty - LEN(<string>) = 0 - string to a file,
  an ASCII 0 will be written. This also holds true for
  WRITE #filenumber,<string>. Just check the length of a
  string before sending it to a file if in doubt.
- Given the above behaviour, use:

        PRINT #filenumber,CHR$(10)
  or
        PRINT #filenumber," "    '..at least 1 space

  to cause a line-feed to be sent to the file.
- See also
ERR
.
```

1.129 PRINTS

```
PRINTS *        - syntax: PRINTS [<expression>][,|;| ..]
  where <expression> is a string or numeric value to
  be printed at the current x,y location of an open
  screen or window.
SETXY
 or
LOCATE
 can be used
  to set the X,Y coordinates for the next PRINTS command.
- This command is now redundant since as of ACE v2.0
PRINT
                  handles DOS and Intuition windows/screens  ←
                       transparently.
- However since PRINTS doesn't have to make a decision
  about whether to print to a DOS or Intuition window,
  it is faster than PRINT. It is not intended for use in
a CLI/shell however.
```

1.130 **PSET**

```
PSET            - syntax: PSET [STEP] (x,y)[,color-id]
                - Plots a point in the current output window or
                  screen.
                - If color-id is not specified, the current
                  foreground color is used.
                - If STEP is specified, the point is relative to
                  the current x,y location as set by the last
                  graphics command.
```

1.131 **PTAB**

```
PTAB            - syntax: PTAB(n)
  where n is in the range: 0..32767
- This function is used in conjunction with
PRINT
 to
  move the horizontal print position for the current
  output window to the nth pixel.
- Subsequent graphics commands are also affected by
  PTAB.
```

1.132 **RANDOMIZE**

```
RANDOMIZE       - syntax: RANDOMIZE <expression>
- Seeds the random number generator.
- In ACE, RANDOMIZE *requires* an argument.
TIMER
 and
  all other arguments will be coerced to long integers.
- RANDOMIZE TIMER is the most commonly used syntax.
```

1.133 **READ**

```
READ            - syntax: READ <variable>[,<variableN>..]
- Assigns <variable> the value of the next item in the
  global data list as created by DATA statements in
  the current program.
- The <variable> must be of the same type as the data
  item to be read otherwise an unexpected value will be
  assigned to <variable>.
- See also
DATA
  (especially re: READing long values).
```

1.134 REM

```
REM              - syntax: REM <comment>
                 - A single-line comment.
                 - All characters after REM until the end of line are
                   ignored.
                 - REM can be substituted by an apostrophe as in AmigaBASIC.
                 - While REM is treated as a true statement, and must
                   either appear on a separate line or after a ":" in a
                   multi-statement, an apostrophe followed by a comment
                   can appear anywhere in the text of a program.
                 - Note that ACE also supports block comments: {..}.
                 - The ACE compiler can handle the three types of comments
                   while the pre-processor APP can only handle the ' and
                   {..} forms. Some form of commenting is required by APP
                   so that pre-processor commands can be commented out.
```

1.135 REPEAT..UNTIL

```
REPEAT..UNTIL *  - syntax: REPEAT
                            .
                            .
                            UNTIL <condition>

                   where <condition> is an expression which reduces
                   to a boolean (true/false) value.

                 - Statements between the REPEAT and UNTIL are executed
                   until the <condition> is true (ie: non-zero).
                 - Styled after the Pascal REPEAT..UNTIL construct.
                 - The loop is always executed at least once.
```

1.136 RESTORE

```
                 RESTORE          - syntax: RESTORE
                 - Resets the DATA pointer to the first
                 DATA
                  statement
                   in the program.
                 - Note that there is no optional label in the ACE version
                   of RESTORE. This may be added in a future revision.
```

1.137 RIGHT$

```
RIGHT$           - syntax: RIGHT$(X$,I)
                 - Returns a string which contains the rightmost I
                   characters of X$.
                 - If I > len(X$), the whole string (X$) is returned.
                 - If I = 0, the NULL string is returned.
```

1.138 RND

```
RND              – syntax: RND[(X)]
                 – The RND function takes an optional parameter and always
                   returns a single-precision pseudo-random value between 0
                   and 1.
                 – At present if it is supplied, X is ignored in ACE.
```

1.139 SADD

```
          SADD          – syntax: SADD(<string-expression>)
      – Returns the address of <string-expression> which can be
        a string literal, variable or expression.
      – Unlike AmigaBASIC, string allocations after a call to
        SADD have no impact upon the address of <string-expression>.
      –
        VARPTR
          can also safely be used to find the address of
        a string variable.
```

1.140 SAY

```
SAY              – In ACE, there is a SAY command and a SAY function.

                 SAY command
                 -----------
                 – syntax: SAY <phoneme-string>[,mode-array]
                 – Same as AmigaBASIC's SAY command: speak a phoneme string.
                 – The <phoneme-string> can be a string literal, expression
                   or variable, while the optional mode-array is a 9-element
                   (0..8) DIM'd short integer array.
                 – The mode-array is allowed, and the following parameters
                   are supported:
```

Argument	Element	Values	Default
pitch	0	65..320	110
inflection	1	0 or 1	0
rate	2	40..400	150
voice	3	0 or 1	0
tuning	4	5000..28000	22200 (Hz)
volume	5	0..64	64
channel	6	0..11	10
mode	7	0 or 1	0
control	8	0,1 or 2	0

```
                 – Inflection=0 allows inflections and emphasis of syllables
                   while inflection=1 gives a monotone voice.
                 – The voice parameter specifies gender: 0=male; 1=female.
                 – Audio channel values have the same meaning as in
                   AmigaBASIC:
```

```
          Value    Channel(s)
          -----    ----------
          0        0
          1        1
          2        2
          3        3
          4        0 and 1
          5        0 and 2
          6        3 and 1
          7        3 and 2
          8        either available left channel
          9        either available right channel
          10       either available left/right pair of channels
          11       any available single channel
```

- Mode is used to specify synchronous or asynchronous
  speech (0 and 1 respectively).
- Control is used when mode=1 to determine what action is
  to be taken when asynchronous speech is active. If
  control is set to 0, the current SAY command waits until
  the last SAY is finished before executing. When control=1
  the last SAY statement is cancelled and speech processing
  stops until the next call to SAY. When control=2 ACE
  interrupts the last SAY command and initiates the current
  one.
- The defaults are the same as in AmigaBASIC.

SAY function   (only works properly under 2.04 or higher)

- syntax: SAY(n)

  where n equals 0, 1 or 2.

  SAY(0) - returns true or false (-1 or 0) to indicate
           whether there is currently active asynchronous
           speech.

  SAY(1) - returns the width of the "mouth" corresponding
           to the phoneme being spoken.

  SAY(2) - returns the height of the "mouth" corresponding
           to the phoneme being spoken.

- SAY(0) allows for monitoring of the asynchronous
  speech process (see details of mode-array above).
- Use of SAY(1) and SAY(2) allows an animated mouth
  to be drawn.
- SAY(1)'s and SAY(2)'s values reflect the last call
  to SAY(0) and so must be used in conjunction with
  the latter.
- Usage of the SAY function is typically like this:

        SAY ...   '..start asynchronous speech

        WHILE SAY(0)
          x = SAY(1)
```

```
                            y = SAY(2)
                            .
                            .
                        WEND
```

1.141 SCREEN

SCREEN   – The SCREEN statement syntax is the same as in AmigaBASIC:

                      SCREEN screen-id,width,height,depth,mode

              where mode is one of the following:

              1 = lores
              2 = hires
              3 = lores,interlaced
              4 = hires,interlaced.
              5 = HAM (hold-and-modify)   [ACE only]
              6 = extra-halfbrite   [ACE only]

          – The SCREEN function (ACE only) syntax is SCREEN(n),
            where:

            SCREEN(0) – Returns a pointer to the Intuition window,
                        that is, the current output window or default
                        window for the screen.

            SCREEN(1) – Returns a pointer to the Intuition screen.

            SCREEN(2) – Returns a pointer to the rastport of
                        the default window or current output
                        window for the screen.

            SCREEN(3) – Returns a pointer to the screen's viewport.

            SCREEN(4) – Returns a pointer to the screen's bitmap.

            SCREEN(5) – Returns the width of the screen's font.

            SCREEN(6) – Returns the height of the screen's font.

          – A future revision of ACE's SCREEN command will support
            AGA screen modes.

1.142 SCREEN BACK

SCREEN BACK   – syntax: SCREEN BACK screen-id
                 – Sends the specified screen to the back of the display.

1.143 SCREEN CLOSE

```
SCREEN CLOSE      - syntax: SCREEN CLOSE screen-id
                  - Closes a single screen.
```

1.144 SCREEN FORWARD

```
SCREEN FORWARD    - syntax: SCREEN FORWARD screen-id
                  - Makes the specified screen frontmost.
```

1.145 SCROLL

```
SCROLL            - syntax: SCROLL (xmin,ymin)-(xmax,ymax),delta-x,delta-y
                  - Scrolls bits inside the specified rectangle.
                  - Delta-x and delta-y specify motion right and down
                    respectively.
                  - Negative delta values produce motion to the left and up.
```

1.146 SERIAL

```
       SERIAL *          - syntax: SERIAL(channel,n)

         where channel is a serial channel identifier from 1..255
         and n is a function number from 0..12 (see below).

       - This function returns information about an open serial
         channel.

         n value
         -------

         0      - Returns the number of characters in the serial
                  read buffer. Use this value to determine how many
                  bytes to read from the buffer (see
       SERIAL READ
       ).

         1      - Unit number of serial device in use by this
                  channel (see
       SERIAL OPEN
       ).

         2      - Baud rate.

         3      - Parity. Actually the ASCII value of the character
                  representing the selected parity (N,E,O,M,S). Use

       CHR$
        function to recover the character.

         4      - Number of data bits.
```

```
            5       - Number of stop bits.

            6       - Number of wires for handshaking: 3 or 7.

            7       - XON/XOFF feature: 0=disabled; 1=enabled.

            8       - Shared access mode: 0=disabled; 1=enabled.

            9       - Fast mode: 0=disabled; 1=enabled.

           10       - Serial (read) buffer size in bytes.

           11       - Name of serial device.  Actually, the value
                      returned is the address in memory of the name
                      string.  Use ACE's
        CSTR
         function to convert it
                      to a string.

12 - A 16-bit word representing the status of the serial
   port lines and registers.

   Bit Active  Symbol  Function
   --- ------  ------  --------
   0 -   Reserved
   1 -   Reserved
   2 high  (RI)  Parallel Select on A1000
      + Ring-indicator on A500/A2000
   3 low (DSR) Data Set Ready
   4 low (CTS) Clear To Send
   5 low (CD)  Carrier Detect
   6 low (RTS) Ready To Send
   7 low (DTR) Data Terminal Ready
   8 high     Read overrun
   9 high     Break sent
  10 high     Break received
  11 high     Transmit x-OFFed
  12 high     Receive x-OFFed
  13 -   Reserved
  14 -   Reserved
  15 -   Reserved

   If you wanted to test for Carrier Detect, code
   such as:

    carrier_detect = SERIAL(1,12) AND 32

   would store 32 in carrier_detect if CD was high
   (ie. no carrier) and 0 if CD was low (ie. carrier
   detected). The value 32 is used here since CD is
   associated with bit 5 and 2^5 is 32. The 1 here
   means serial channel 1.

   Note that the above status word is taken directly
   from querying the serial device associated with a
   particular channel and the above table is taken
```

directly from the ROM Kernel Ref. Manual: Devices,
(1991), pg 278.

- For more information about the serial device modes etc,
  see SERIAL OPEN command below and Commodore's ROM Kernel
  Reference Manual: Devices.
            - See also
            ERR
            .

1.147   SERIAL CLOSE

            SERIAL CLOSE * - syntax: SERIAL CLOSE [#] channel
            - Closes a logical channel to a serial device.
            - See also
            ERR
            .

1.148   SERIAL OPEN

            SERIAL OPEN *   - syntax:
              SERIAL OPEN [#] channel,unit,baud,params[,size][,dev]
            - This command opens a logical channel to a serial device.
            - The channel parameter must be in the range 1..255.
            - The unit parameter tells ACE which serial device unit to
              open (eg. for a multi-port serial card). Normally
              however, you should specify 0 for a single serial port.
            - The baud rate is specified by the baud parameter. This
              value can be in the range 110..292,000 on the Amiga.

            - The next parameter is a string consisting of at least
              three single character "switches":

                    parity   - N,E,O,M or S. Other = N.
                    data bits   - usually 7 or 8.
                    stop bits   - usually 1 or 2.

                    wires   - 3 or 7. Other = 7.
                    XON/XOFF   - X = enabled. Other = disabled.
                    Access   - S = shared. Other = exclusive.
                    Fast mode   - F = fast mode. Other = normal.

            - Parity, data bits and stop bits MUST be specified and
              in the order shown above, while the remaining switches
              are optional and can be given in any order.
    - Fast mode is intended for use in conjunction
      with peripherals which require high serial
      throughput, eg. a MIDI device. Higher throughput
      is achieved by certain internal serial device checks
      being skipped. Fast mode should be used only when:

parity checking has been disabled, XON/XOFF handling
is disabled and 8 bit characters are in use.
        – For a letter, upper or lower case can be used.
        – In the above description of switches "Other" means any
          other character (I suggest you use "?" or some other
          character consistently, to indicate "don't care").

        – The optional parameter "size" specifies the size of the
          serial *read* buffer. At high baud rates the buffer can
          fill up quickly. The default is 512 bytes.
        – The final parameter (dev) is also optional. This
          specifies the name of the serial device to be used. The
          device name defaults to "serial.device" if not specified.
          An alternate serial device can be used as long as the
          device's commands are compatible with the standard
          serial.device supplied with the Amiga. This device
          normally lives in the DEVS: directory.
        – If using another serial device, simply supply its name
          if it resides in the DEVS: directory, otherwise a full
          path must be specified.

        – Here's a typical example of SERIAL OPEN usage:

            SERIAL OPEN 1,0,2400,"N81",1024

          which opens a channel (#1) to the standard serial device
          with a baud rate of 2400, no parity, 8 data bits and 1
          stop bit. All 7 wires will be used for handshaking and
          the serial read buffer size will be set to 1K.
        – See also
        ERR
        .

1.149 SERIAL READ

        SERIAL READ *   – syntax: SERIAL READ [#] channel,buffer,length
        – Tells ACE to read length bytes from the serial buffer
          corresponding to the (open) logical channel into a string
          buffer.
        – The buffer can be a string variable or array.
        – Note that this command will wait for the serial port
          read to complete before returning control to your
          program, so use SERIAL(channel,0) to find out how many
          bytes are waiting on the port and make length equal to
          that value.
        – See also
        ERR
        .

1.150 SERIAL WRITE

```
                 SERIAL WRITE *  - syntax: SERIAL WRITE [#] channel,string,length
                 - Tells ACE to write length bytes to the serial port
                   corresponding to the (open) logical channel from a
                   string buffer.
                 - The string buffer can be any string expression.
                 - See also
                 ERR
                 .
```

1.151 SETHEADING

```
SETHEADING *    - syntax: SETHEADING n
                - Changes the turtle's heading to n degrees.
```

1.152 SETXY

```
                 SETXY *          - syntax: SETXY x,y
                 - Sets the x,y location for the next graphics command
                   in the current output window or open screen.
                 - Its primary use is for turtle graphics. To prevent the
                   turtle drawing a line when SETXY is used, the
                 PENUP
                                       command should first be issued.
```

1.153 SGN

```
SGN               - syntax: SGN(n)
                  - Returns the sign of the number n:

                         if n>0, SGN(n) returns  1
                         if n=0, SGN(n) returns  0
                         if n<0, SGN(n) returns -1
```

1.154 SHARED

```
                 SHARED          - syntax: SHARED <ident>[,<ident> ... ]
                 - Variables, arrays and structures must explicitly
                   be shared between the main program and subprograms.
                 - Only
                 EXTERNAL
                  variables are exempt from such sharing in
                   ACE since they are global (see "Identifiers" in ace.guide).
                 - One or more SHARED statements can appear in a subprogram
                   and are usually placed before all other code in that SUB.
                 - Declarations of objects to be shared must appear in the
```

```
                    main program before the subprogram is *declared*.
                  - See subprograms section in ace.guide and the entry for

                  DIM
                   above re: DIM SHARED.
```

1.155 SHL

```
SHL *            - syntax: SHL(n,m)
                    where n is the value to be shifted and m is the number
                    of bit positions to shift.
                  - Arithmetic shift left function. Returns a long integer.
                  - Shifting left by 1 bit (or more) is faster than multiplying
               by 2 (or powers thereof).
```

1.156 SHR

```
SHR *            - syntax: SHR(n,m)
                    where n is the value to be shifted and m is the number
                    of bit positions to shift.
                  - Arithmetic shift right function. Returns a long integer.
                  - Shifting right by 1 bit (or more) is faster than dividing
               by 2 (or powers thereof).
```

1.157 SHORTINT

```
SHORTINT *       - syntax: SHORTINT <identifier>[,..]
                  - Declares and initialises one or more short integer
                    variables.
```

1.158 SINGLE

```
SINGLE *         - syntax: SINGLE <identifier>[,..]
                  - Declares and initialises one or more single-precision
                    variables.
```

1.159 SIZEOF

```
SIZEOF *         - syntax:
                  SIZEOF(byte|shortint|longint|address|single|string|<ident>)
                  where <ident> is the name of a variable, array, structure
                  type or structure variable (not a SUB, function or external
                  variable).
                  - A size in bytes is returned.
```

- The intention is the same as that of C's sizeof() operator.
- SIZEOF is most useful when allocating memory for
  structures.

1.160 SIN

```
SIN              - syntax: SIN(n)
                 - Returns the sine of n.
```

1.161 SLEEP

```
SLEEP            - syntax: SLEEP
                 - This command puts a program to sleep until there is
                   mouse, menu or keyboard activity. The program will
                   also be woken up by IntuiTicks (timer signals from a
                   user-defined window or default screen window) at regular
                   intervals (every ~0.1 of a second) so your program can
                   perform other tasks.
                 - If SLEEP is called when the current output window is
                   a CLI/shell, SLEEP returns control to your program
                   immediately.
                 - Once a window loses the "focus" SLEEP waits indefinitely.
                   If this is likely to happen, you might want to use the

                 SLEEP FOR
                   command instead.
```

1.162 SLEEP FOR

```
SLEEP FOR *      - syntax: SLEEP FOR <seconds>
                 - Suspends execution of a program for the specified number
                   of seconds, which can be a single-precision floating
                   point value greater than 0 (including values between 0 and 1).
                 - This command does NOT use a busy waiting method. Instead
                   it relies upon the dos.library Delay() function to delay
                   execution in a system-friendly way, without hogging CPU
                   time.
                 - The smallest practical value for <seconds> is 0.02 since
                   there are 50 ticks per second and 50*0.02 = 1 tick. Any
                   value less than 0.02 will therefore cause SLEEP FOR to
                   return immediately. This would have the same effect as
                   busy waiting which hogs CPU time. To see the effect of
                   various values of <seconds> run the following program
                   with the system tool PerfMon running:

                       WHILE INKEY$=""
                         SLEEP FOR n            '..where n is <seconds>
                       WEND
```

```
            – You should notice that as <seconds> approaches zero,
              CPU time looks more like it would if you had used
              the above loop without SLEEP FOR at all.
```

1.163 SOUND

```
        SOUND           – syntax: SOUND period,duration[,volume][,voice]
        – Note that the syntax of this command is different
          from the equivalent statement in AmigaBASIC.
        – See the sound section in ace.guide for details.
        – See also the
        WAVE
         command. A combination of
          these two commands in ACE allows you to easily
          play sound samples (see example program play.b).
        – SOUND currently uses the audio hardware directly
          but a future revision will use the audio device.
```

1.164 SPACE$

```
SPACE$          – syntax: SPACE$(n)
                – Returns a string of n spaces.
```

1.165 SPC

```
        SPC             – syntax: SPC(n)
        – This function is generally used in conjunction with
        PRINT
                        and returns a string of n spaces, where n is a  ←
                                value from
          0 to 255.
```

1.166 SQR

```
SQR             – syntax: SQR(n)
                – Returns the square root of n.
                – The argument n must be >= 0.
```

1.167 STICK

```
STICK              - syntax: STICK(n)
                   - Returns information about joystick direction.
                   - At the moment, STICK(0) & STICK(1) always return 0,
                     while STICK(2) & STICK(3) return the state of
                     the joystick in port 2 (B), where:

                           STICK(2) is joystick B in X direction.
                           STICK(3) is joystick B in Y direction.

                   - Return values are:

                         0 = joystick is not engaged.
                         1 = movement is upward or to the right.
                        -1 = movement is downward or to the left.
                   - STICK currently goes straight to the hardware. A future
                     revision may use the gameport device.
```

1.168 STOP

```
STOP              - This is functionally equivalent to
END
 in ACE.
- See also
IF..[ELSE]..END IF
.
```

1.169 STR$

```
STR$               - syntax: STR$(n)
                   - Returns the string representation of the numeric value n.
                   - The string includes a leading space or "-" depending upon
                     the sign of the number.
```

1.170 STRIG

```
STRIG              - syntax: STRIG(n)
                   - Returns information about the state of a joystick button.
                   - At the moment, STRIG(0), STRIG(1) & STRIG(2) always
                     return 0.
                   - STRIG(3) returns -1 if the port 2 joystick's
                     fire button is *currently* pressed and 0 if it isn't.
                   - STRIG currently goes straight to the hardware. A future
                     revision may use the gameport device.
```

1.171 STRING

```
STRING *          - syntax:
                    STRING <ident> [[ADDRESS <addr>] | [SIZE <size>]][,..]
                  - Declares and initialises one or more string variables
                    with an optional size or address. If the size is not
                    specified, a length of MAXSTRINGLEN bytes is assumed.
                  - If an address is specified, the SIZE option can't be used
                    since the size of the area of memory pointed to by <addr>
                    has already been determined.
```

1.172 STRING$

```
STRING$           - syntax: STRING$(I,J) or STRING(I,X$).
                  - STRING$ returns a string of length I consisting of
                    characters with ASCII code J or ASC(MID$(X$,1,1)).
```

1.173 STRUCT

```
STRUCT *          - Defines a new structure data type, thus:

   STRUCT <ident>
     <type> <ident1>
     <type> <ident2>
     .
     .
     <type> <identN>
   END STRUCT

   where <type> can be BYTE,SHORTINT,LONGINT,ADDRESS,SINGLE,
   STRING and <ident1>..<identN> are structure members of
   one of these data types.
 - A structure member may also be another structure. In this
   case, <type> must be the name of a previously defined
   structure type. See ace.guide's "Structures" section for
   more about this.
 - Where a member is of type STRING, an optional size can be
   specified (STRING <ident> [SIZE <size>]).
 - See also:
 DECLARE
  and the section on structures in
   ace.guide.
 - Structures have been provided in ACE primarily to make
   communicating with the operating system a little nicer
   and to make dynamic data structures possible (see the
   example programs turtle/bst.b and misc/linkedlist.b).
 - ACE structures cannot currently be array elements
   although there is nothing to stop you from storing
   structure start addresses in array elements. For an
   example of this, see prgs/misc/array_of_structs.b.
 - See "Structures" section in ace.guide for more details.
```

1.174 STYLE

```
STYLE *          - syntax: STYLE n
                 - Changes the text style for the current output window
                   (user-defined window or shell).
                 - The single parameter can take on the following values:

                   n     Effect
                   -     ------
                   0     Plain
                   1     Underlined
                   2     Bold
                   4     Italic
                   8     Extended width  (non-shell/CLI window only)

                 - These values can be added to produce cumulative effects
                   (eg: n=3 gives bold and underlined text).
```

1.175 SUB..END SUB

```
         SUB..END SUB    - syntax:

            SUB [<type>] <ident> [([<type>] <param> [..])] [EXTERNAL]
                 <statement1>
                 <statement2>
                 .
                 .
                 <statementN>
            END SUB

            where the optional <type> is one of: LONGINT,ADDRESS,
            SHORTINT,SINGLE or STRING.
         - In ACE, subprograms are non-static, allow recursion, may
           have return values and have optional parameter lists.
         - Parameters are call-by-value but ACE does provide
           mechanisms for call-by-reference parameters.
         - SHARED variables are supported in ACE (see
         SHARED
                            command).
         - Note that since ACE SUBs are non-static, the STATIC
           keyword is not allowed.
         - The optional EXTERNAL keyword makes the subprogram
           visible to other ACE modules.
         - See "Subprograms" section in ace.guide for more details.
```

1.176 SWAP

```
SWAP             - syntax: SWAP <object>,<object>
                   where <object> is a simple/external variable, parameter,
                   array element, structure or structure member.
                 - This command swaps the value of the specified data
```

```
                  objects.
                - SWAP is not intended to be used for exchanging two whole
                  arrays.
                - ACE currently assumes a maximum length of MAXSTRINGLEN
                  when swapping strings.
```

1.177 SYSTEM

```
        SYSTEM          - syntax 1: SYSTEM n
          where n is an integer exit value (return code).
        - SYSTEM causes an ACE program to exit with the specified
          return code. The latter can be tested in a shell script
          as WARN, ERROR etc. This value is hidden from a Workbench
          launched program.
        - Note that in AmigaBASIC, SYSTEM returns from the
          interpreter to the shell/CLI or Workbench. The same is
          true in ACE, except that
        END
         and
        STOP
         will also do this,
          so SYSTEM's intended purpose in ACE is different to that
          in AmigaBASIC.

   OR

        - syntax 2: SYSTEM command-string
        - This version of the SYSTEM command attempts to run a
          shell/CLI command. It is equivalent to the following
          dos.library command:
                Execute(command-string,stdin,stdout).
        - If the command writes to standard output, make sure you
          are running the program from a shell/CLI or at least
          that you have given the EXTERNAL stdout variable a valid
          value corresponding to an open file's handle, typically a
          CON: or RAW: window (see
        HANDLE
         function).
        - Also, make sure that "Run" is in your C: directory.
        - Examples:

            SYSTEM "list"       '..lists files in current directory

            SYSTEM "dir > fred" '..runs dir command and redirects
                                '..output to a file called fred.

   OR

        - syntax 3: SYSTEM
        - This *function* returns the Exec library version,
          enabling your program to do different things depending
          upon the version of the operating system under which it
          is running.
        - A value of 34 indicates Workbench 1.3 while 37 indicates
          Workbench 2.04.
```

1.178   TAB

```
 TAB            - syntax: TAB(n)
- Used in conjunction with
  PRINT
   to move the print position
 to the nth column.
- TAB(n) - where n=1..81.
- if n>81, wraparound will occur in a DOS window while
  a user-defined (Intuition) window/screen will clip any
  output past the last character position.
- if n<1, the next print position will be column 1 (leftmost).
```

1.179   TAN

```
TAN             - syntax: TAN(n)
                - Returns the tangent of n.
```

1.180   TIME$

```
TIME$           - syntax: TIME$
                - Returns the current time as a string of the format:

                        hh:mm:ss

                  where hh is hours, mm is minutes and ss is seconds.
```

1.181   TIMER

```
TIMER           - syntax: TIMER
                - Returns a single-precision value corresponding to
                  seconds elapsed since midnight.
```

1.182   TIMER ON

```
TIMER ON ..     - syntax: TIMER ON|OFF|STOP
                - These commands are used for enabling, disabling and
                  suspending ON TIMER(n) event trapping.
                - See the Event Trapping section in ace.guide.
```

1.183 TRANSLATE$

```
TRANSLATE$          - syntax: TRANSLATE$(<string-expression>)
                    - Returns the phoneme-string equivalent of
                      <string-expression> where the latter contains words.
```

1.184 TURN

```
TURN *              - syntax: TURN n
                    - Rotates the turtle by n degrees.
                    - If n is negative, the turtle will rotate
                      counter-clockwise while if it is positive,
                      the rotation will be clockwise.
```

1.185 TURNLEFT

```
TURNLEFT *          - syntax: TURNLEFT n
                    - Rotates the turtle counter-clockwise by n degrees.
                    - If n is negative, the result will be the same as
                      TURNRIGHT ABS(n).
```

1.186 TURNRIGHT

```
TURNRIGHT *         - syntax: TURNRIGHT n
                    - Rotates the turtle clockwise by n degrees.
                    - If n is negative, the result will be the same as
                      TURNLEFT ABS(n).
```

1.187 UCASE$

```
UCASE$              - syntax: UCASE$(<string-expression>)
                    - Returns <string-expression> with all alphabetic characters
          in upper case.
```

1.188 VAL

```
                    VAL              - syntax: VAL(X$)
                    - Returns the numeric value of X$ as a single-precision
                      number.
                    - The translation of integers plus fixed-point and
                      exponential format single-precision values is supported.
                    - The hexadecimal and octal prefixes (&H and &O) are also
                      recognised by VAL.
                    - VAL strips off leading whitespace (eg: spaces, tabs).
```

                – There may be a loss of accuracy if the string contains a
                  LARGE long integer value, due to the limitations of the
                  single-precision numeric format. To overcome this, use
                  the
                LONGINT
                (n) function.

1.189 VARPTR

                VARPTR   – syntax: VARPTR(<data-object>)
                – Returns the absolute address of a numeric variable,
                  string, array, array element, structure, structure
                  member, external function or subprogram.
                – You can safely use VARPTR to find a string variable's
                  address (
                SADD
                 has also been provided for string variables
                  and expressions).
                – Unlike AmigaBASIC, an object's address does *not* move
                  around in memory once allocated.
                – In ACE, the symbol "@" can be used instead of VARPTR,

                    eg: addr& = @n(2) '..finds address of an array element

                – When used in conjunction with a structure variable x,
                  @x will return the address of the variable itself, NOT
                  the start address of the structure (see "Structures" in
                  ace.guide for more).
                – See also section on indirection operators in ace.guide.

1.190 WAVE

                WAVE   – syntax: WAVE voice,SIN | [waveform-address,byte- ←
                  count]
                – Defines a waveform of any length to be used by the SOUND
                  statement for a specified audio channel (voice: 0..3).
                – If the SIN option is used, a sine waveform table is
                  allocated to the specified channel. This is the default
                  waveform for the
                SOUND
                 statement.
                – Unlike AmigaBASIC, the number of bytes in the waveform
                  table must be specified when SIN is not used.
                – See also the Sound section in ace.guide.

1.191 WHILE..WEND

```
WHILE..WEND        - syntax: WHILE <condition>
                             .
                             .
                           WEND

               where <condition> is an expression which reduces
               to a boolean (true/false) value.

             - Statements inside the WHILE and WEND are executed
               while the <condition> is true (ie: non-zero).
```

1.192 WINDOW

```
               WINDOW        - syntax:
               WINDOW id,[title-string],(x1,y1)-(x2,y2)[,type][,screen-id]

               where screen-id specifies the screen to which the window
               should be attached and type can be a combination of the
               following (31 is the default if type is not specified):

                       Type               Effect
                       ----               ------
                       1                  Window size can be changed via
                                          sizing gadget.

                       2                  Window can be moved about using
                                          the title bar.

                       4                  Window can be moved from front to
                                          back using the Back gadget.

                       5                  Under Release 2.x of the OS, when
                                          this Type value is specified alone
                                          or as a component of larger Type
                                          value (eg: 7,15,23) a zoom gadget
                                          is added to the window allowing it
                                          to be switched between the two most
                                          recent window sizes.

                       8                  Close gadget added to window.

                       16                 Contents of window reappear after
                                          it has been covered.

                       32                 Window will be borderless.

             - The window-id must be from 1 to 9.
             - Note that if the rectangle as specified in the WINDOW
               command is too large (according to screen mode), the
               window won't open.
        - See also
               ERR
                 .
```

```
        OR

                - syntax: WINDOW(n)
                - This function returns information related to ACE windows.

                  WINDOW(0)  - window-id of the selected output window.
                  WINDOW(1)  - window-id of current output window.
                  WINDOW(2)  - present width of current output window.
                  WINDOW(3)  - present height of current output window.
                  WINDOW(4)  - x-coordinate in current output window where
                               next pixel will be plotted.
                  WINDOW(5)  - y-coordinate in current output window where
                               next pixel will be plotted.
                  WINDOW(6)  - max legal colour-id for current output window.
                  WINDOW(7)  - pointer to Intuition Window for current output
                               window.
                  WINDOW(8)  - pointer to Rastport of current output window.
                  WINDOW(9)  - pointer to AmigaDOS file handle for current
                               output window (non-zero for shell/CLI only).
                  WINDOW(10) - foreground pen in current output window.
                  WINDOW(11) - background pen in current output window.
                  WINDOW(12) - font width for current output window.
                  WINDOW(13) - font height for current output window.

                - See the section on Windows in ace.guide for more details.
```

1.193 WINDOW CLOSE

```
WINDOW CLOSE    - syntax: WINDOW CLOSE id
                - Closes the id'th window if it is open.
```

1.194 WINDOW ON

```
WINDOW ON .. *  - syntax: WINDOW ON|OFF|STOP
                - These commands are used for enabling, disabling and
                  suspending ON WINDOW event trapping.
                - See the Event Trapping section in ace.guide.
```

1.195 WINDOW OUTPUT

```
WINDOW OUTPUT   - syntax: WINDOW OUTPUT id
                - Makes the id'th open window the current output window.
```

1.196 WRITE

```
WRITE           - syntax: WRITE #filenumber,expression-list
  where filenumber corresponds to an open file.
- The expression-list can contain any combination
  of data items (constants, variables) of any type
  separated by commas.
- Note that the form of WRITE allowing for screen
  output is not supported by ACE.
- See
PRINT#
 re: the treatment of CHR$(0) in file I/O by
  ACE.
- See also
INPUT#
 and the section on files in ace.guide.
- See also
ERR
.
```

1.197 XCOR

```
XCOR *           - Returns the turtle's current x-coordinate.
```

1.198 YCOR

```
YCOR *           - Returns the turtle's current y-coordinate.
```

1.199 XOR

```
XOR              - Boolean operator: X XOR Y.

                    X Y     Out
                    -----------
                    T T     F
                    T F     T
                    F T     T
                    F F     F
```

1.200 Index

```
A


ABS
ACE.Guide


ADDRESS
```

AIDE.Guide

ALLOC

AND

AREA

AREAFILL

ARG$

ARGCOUNT

ASC

ASSEM

ATN
B

BACK

BEEP

BEVELBOX

BIN$

BREAK
C

CALL

CASE

CHDIR

CHR$

CINT

CIRCLE

CLEAR ALLOC

CLNG

CLOSE

CLS

COLOR

CONST

COS

CSNG

CSRLIN

CSTR
D

DATA

DATE$

DAY

DECLARE

DEF FN

DEFxxx

DIM
E

END

EOF

EQV

ERR

ERROR
Example.Guide

EXIT FOR

EXIT SUB

EXP

EXTERNAL
F

FILEBOX$

FILES

FIX

FONT

FOR..NEXT

FORWARD

FRE
G

GADGET CLOSE

GADGET MOD

GADGET ON

GADGET WAIT

GADGET

GOSUB..RETURN

GOTO
H

HANDLE

HEADING

HEX$
History

HOME
I

IF

IFF CLOSE

IFF OPEN

IFF READ

IFF

IMP

INKEY$

INPUT #

INPUT

INPUT$

INPUTBOX

INPUTBOX$

MID$

MOD

MOUSE ON

MOUSE

MSGBOX
N

NAME

NOT
O

OCT$

ON..GOTO/GOSUB

OPEN

OPTION

OR
P

PAINT

PALETTE

PATTERN

PEEKx

PENDOWN

PENUP

POINT

POKEx

POS

POTX

POTY

PRINT #

PRINT

PRINTS

```
PSET

PTAB
R

RANDOMIZE

READ

REM

REPEAT..UNTIL

RESTORE

RIGHT$

RND
S

SADD

SAY

SCREEN BACK

SCREEN CLOSE

SCREEN FORWARD

SCREEN

SCROLL

SERIAL CLOSE

SERIAL OPEN

SERIAL READ

SERIAL WRITE

SERIAL

SETHEADING

SETXY

SGN

SHARED

SHL
```

SHORTINT

SHR

SIN

SINGLE

SIZEOF

SLEEP FOR

SLEEP

SOUND

SPACE$

SPC

SQR

STICK

STOP

STR$

STRIG

STRING

STRING$

STRUCT

STYLE

SUB..END SUB

SWAP

SYSTEM
T

TAB

TAN

TIME$

TIMER ON

TIMER

TRANSLATE$

```
TURN

TURNLEFT

TURNRIGHT
U


UCASE$
V


VAL

VARPTR
W


WAVE

WHILE..WEND

WINDOW CLOSE

WINDOW ON

WINDOW OUTPUT

WINDOW

WRITE
X


XCOR

XOR
Y


YCOR
```