Inter-Office Memorandum

To	NOVA Users	Date	May 31, 1973
From	Ben Laws	Location	Palo Alto Coyote Hill
Subject	BCPL I/O and Runtime Routines	Organization	PARC

This document is a description of a number of routines which have been written to provide limited but useful runtime support for BCPL programs. In most cases, the routines are very similar to the ALGOL and FORTRAN counterparts or to the actual assembly language DOS system call. Routines have been written to do many I/O functions and a few string functions. Limited formatted I/O functions have been implemented using general string integer conversion routines.

Before calling any of the I/O runtime routines, the routine initbcplio(1) must be called to set up several global variables. The I/O errors are handled by the routine whose address is in syserror. This routine is normally ioerror, a routine which corrects some inadequacies of the DOS error-handling facility, and optionally prints procedure level information. Input routines do not consider end of file to be an error and return this information through a byte count indicating how many bytes were actually read, or a special ASCII character. Errors may be captured by changing the routine in syserror to one of the user's routines or by setting syserror-trap to "false". If this is done, after an I/O routine is called, the location syserrorflag will be false if no error has occurred, but otherwise will be true; syserrorvalue will have the error value from AC2 after the DOS system call. End of file will be shown as an error when this facility is used. For doing routine tasks, the default error routine will usually be adequate.

DOS strings are not compatible with BCPO strings. All the I/O routines accept BCPL strings and convert them to DOS strings when necessary, with the exception of readline and writeline as described for those two procedures. Again, for routine tasks, string incompatibility is of no consequence.

This document is intended to be updatable and is organized in a way to make this process easier; all global variables are described in section II, all procedures are described in the following section III, and an index will be attached listing all names in sections II and III. When updates are made, sheets belonging to section III will be issued along with a new index. The index will carry names in alphabetical order with mnemonic arguments shown, so that in many cases the index will answer questions about a given procedure. The procedure descriptions will, in many cases, carry a cross-reference note to the DOS manual of the form DOS:ch-pp. In general, all procedure arguments must be specified but in a few specific cases, missing arguments will cause default assignments as noted by specific procedure descriptions -- arguments which are optional are delineated by brackets [].

SECTION II

sysac
The accumulators used for system calls to DOS. Not generally useful except inside the runtime routines.

syserrorflag
If set after a system call, an error has occured. This will be true independent of the state of syserrortrap. The value of the error will be in syserrorvalue until another error occurs.

syserrorvalue
If syserrorflag is set after a system call, this static contains the value of the error. The value is constant until another error occurs.

syserrortrap
If this static is set to true, the routine ioerror will print an appropriate error message and return to DOS CLI. If set to false, ioerror will simply return. If ioerror is called by the user program with a single parameter, ioerror behaves as if syserrortrap were set to true. For more information see ioerror(syserrorvalue).

sysprintpc
If set to true, ioerror will print the addresses of the system procedure from the runtime I/O and the user procedure which caused the error. This is the variable which is set to true by initbcplio(2).

filenamelength
The maximum length of DOS filenames--manifest constant which may be used for allocating vectors to receive DOS file names.

nbytes = readcomcm(chno, string [, switches])

Purpose:
To read arguments and switches from the DOS command file, COM.CM
Parameters:
chno
DOS channel number, previously opened to file COM.CM
string
A BCPL vector for the name read from COM.CM (may be allocated with vec filenamelength).
switches
A 26 element boolean vector in which each element corresponds to the alphabetic character for the switch.
Function Results:
nbytes
The number of bytes actually read is returned.

initbcplio(mode)

Purpose:
To initialize various constants needed by the runtime I/O routines. Failure to invoke this routine will lead to system crashes at undefined times!
Parameters:
mode
1 - normal mode. Error messages will be given normally.
2 - diagnostic mode. Stack information will be printed if this mode is set. Mode 2 mayalso be invoked by setting sysprintpc to true.

char = readch(chno)

Purpose:
To read one 8 bit character from channel chno previously opened to a DOS file.
Parameters:
chno
A DOS channel number 0-7.
Function Results:
char
The 8 bit character read from the channel.

writech(chno,char)

Purpose:
To write one 8 bit character from channel chno previously opened to a DOS file.
Parameters:
chno
A DOS channel number 0-7.
char
The 8 bit character to be written.

nbytes = readseq(chno, bytepointer, nbytes) DOS:4-14

Purpose:
Read a number of bytes using the DOS .RDS command.
Parameters:
chno
A DOS channel number 0-7.
bytepointer
DOS byte pointer to the first byte involved in the transfer.
nbytes
Number of bytes to be read.
Function Results:
nbytes
Number of bytes actually read--must be used to detect end of file.

writeseq(chno, bytepointer, nbytes) DOS:4-18

Purpose:
Write a number of bytes using the DOS .WRS command.
Parameters:
chno
A DOS channel number 0-7.
bytepointer
DOS byte pointer to the first byte involved in the transfer.
nbytes
Number of bytes to be written.

nbytes = readline(chno, string[, true/false]) DOS:4-13

Purpose:
To read a string terminated by a carraige return from a DOS file.
Parameters:
chno
A DOS channel number 0-7.
string
A BCPL vector with enough space to receive the input string.
true/false
If true, the TRUE DOS readline function is executed. The .RDL function terminates on NULL as well as form feed, carraige return and end of file. One usually does not want to deal with this function. If false or absent, the NULL termination is removed.
Function Results:
nbytes
If 1, a terminator has been received. The last character in the string received is either carraige return or form feed (or NULL if the true .RDL) or carraige return followed by #377 if end of file.

writeline(chno, string) DOS:4-17

Purpose:
Write a string which MUST be terminated by a carraige return, null or form feed to the DOS channel previously opened. DOS interprets tabs, form feeds for certain devices.
Parameters:
chno

A DOS channel number 0-7.
string
A BCPL string or vector which must be terminated as specified above.

writestr(chno, string)

Purpose:
Write any BCPL string. A line feed is unconditionally issued following every carraige return character.
Parameters:
chno
A DOS channel number 0-7.
string
A BCPL string or vector which must be terminated as specified above.

writezoct(chno, number)

Purpose:
Write a six digit unsigned octal number with leading zeroes.
Parameters:
chno
A DOS channel number 0-7.
number
16 bit quantity.

writedec(chno, number[, space])

Purpose:
Write a signed decimal number with fixed or variable spacing.
Parameters:
chno
A DOS channel number 0-7.
number
16 bit quantity.
space
Number of spaces to be used. If missing or zero, a variable number of spaces are used.

writeoct(chno, number[, space])

Purpose:
Write a signed octal number with fixed or variable spacing.
Parameters:
chno
A DOS channel number 0-7.
number
16 bit quantity.
space
number of spaces to be used. If missing or zero, a variable number of spaces are used.

writeform(chno, formatcode, data[, formatcode, data ...])

Purpose:
Write a group of string or 16 bit data to the channel as specified by the formatcodes.

Parameters:

chno
A DOS channel number 0-7.

formatcode
0 - data following is string data.
2-10 - data following is a 16 bit quantity to be displayed in that radix.

writevalue(chno, number, rdx[, space])

Purpose:
Write a 16 bit signed number in arbitrary radix (2-10) using fixed or variable spacing.

Parameters:

chno
A DOS channel number 0-7.

number
A 16 bit signed quantity.

rdx
An arbitrary radix 2-10.

space
The number of spaces to be used. If the argument is missing or 0, a variable number of spaces will be used.

word = readbin(chno)

Purpose:
Read a 16 bit quantity from the DOS channel. No end of file detection is provided except by capturing the error with syserrortrap.

Parameters:

chno
A DOS channel number 0-7.

Function Results:

word
A 16 bit quantity read from the file.

writebin(chno, word)

Purpose:
Write a 16 bit quantity to the specified channel.

Parameters:

chno
A DOS channel number 0-7.

word
A 16 bit quantity to be written.

chno = open(name)   DOS:4-10

Purpose:
  Open a DOS file to a channel selected by the runtime routines.
Parameters:
  name
    Any BCPL string which is a legal DOS file name. Device specifier must be upper case, e.g., DP0--all other characters are translated to upper case.
Function Results:
  chno
    A DOS channel number 0-7 returned specifying the channel number to be used.

chno = append(name)   DOS:4-11

Purpose:
  Re-open a DOS file to a channel selected by the runtime routines. Writing will begin following the last character in the existing file.
Parameters:
  name
    Any BCPL string which is a legal DOS file name. Device specifier must be upper case, e.g., DP0--all other characters are translated to upper case.
Function Results:
  chno
    A DOS channel number 0-7 returned specifying the channel number to be used.

nbytes = curpos(chno)

Purpose:
  Return the current byte position of a DOS file.
Parameters:
  chno
    A DOS channel 0-7.
Function Results:
  nbytes
    Current byte pointer for the file.

setpos(chno, nbytes)

Purpose:
  Set the current byte position of a DOS file.
Parameters:
  chno
    DOS channel 0-7.
  nbytes
    Current byte pointer for the file.

curposdw(chno, doublewordvector)

Purpose:
Return the current block and byte number of a DOS file in a BCPL vector to overcome the lack of double precision integers in BCPL.
Parameters:
chno
A DOS channel 0-7.
doublewordvector
A 2 word BCPL vector giving the block number in word 0 and the byte number in word 1.

setposdw(chno, doublewordvector)

Purpose:
Set the current block and byte number of a DOS file in a BCPL vector to overcome the lack of double precision integers in BCPL.
Parameters:
chno
A DOS channel 0-7.
doublewordvector
A 2 word BCPL vector giving the block number in word 0 and the byte number in word 1.

createfile(name) DOS:4-6

Purpose:
Create a DOS file.
Parameters:
name
A legal DOS file name.

deletefile(name) DOS:4-7

Purpose:
Delete a DOS file.
Parameters:
name
A legal DOS file name.

initdev(name) DOS:4-4

Purpose:
Initialize a DOS device.
Parameters:
name
A legal DOS device name.

directorydev(name) DOS:4-4

Purpose:
Change the default directory to the indicated device.
Parameters:
name
A legal DOS device name.

releasedev(name)   DOS:4-5

Purpose:
Parameters:
name
A legal DOS device name.

renamefile(name,newname)   DOS:4-7

Purpose:
Change the name of an existing DOS file.
Parameters:
name
A legal DOS file name.

close(chno)   DOS:4-12

Purpose:
Close an I/O channel to further use until re-opened.
Parameters:
A legal DOS channel number (0-7).

resetfiles()   DOS:4-13

Purpose:
Close all I/O channels to further use until re-opened.
Parameters:
A legal DOS channel number (0-7).

errvalue = systemcall(ac0, ac1, ac2, syscallname, err) DOS:4-1

Purpose:
Generate a DOS system call directly.
Parameters:
ac0
NOVA ac 0 to be passed as part of the system call.
ac1
Nova ac 1.
ac2
Nova ac 2.
syscallname
a name from the list of system calls contained in iox,
generally, the DOS mnenmonic preceded by "sys"--e.g., sysrdl for
.RDL.
err
The BCPL procedure to be called in the event of an error return
from the system call.
Function Results:
err
The error value if an error occurs, otherwise -1. The error
code is returned in global vector SYSAC!2 and in the global
variables syserrorflag and syserrorvalue. If syserrorflag is
set, syserrorvalue contains the value of the error.
syserrorvalue will not be changed. If there is no error but

sysac!2 will be changed with every system call.

ioerror(syscallname, sysac) or (syserrorvalue)

Purpose:
Writes an error message to the teletype output device. Most messages are generated by DOS, but in a few cases, ioerror generates the correct message. If called with 2 parameters, the error value is taken from the vector specified by sysac and in some cases the name specified by sysac. If called with 1 parameter, the error value is taken to be the value of that parameter and if needed syserrorname will be used. If syserrortrap is set to false, this routine will simply return when called with TWO parameters. The routine is executed unconditionally if called with only one parameter.

Parameters:
- syscallname
  - The DOS system call used to generate the error.
- syac
  - The system call accumulator vector.
- syserrorvalue
  - The error value which may be given directly in lieu of the two above.

install(chno) DOS:4-5

Purpose:
Install a DOS on the default directory device.

Parameters:
- chno
  - The DOS channel previously opened to the DOS to be installed.

chatr(chno, ac0) DOS:4-8

Purpose:
Change the attributes of a DOS file.

Parameters:
- chno
  - A DOS channel previously opened to the file to be changed.
- ac0
  - The value for ac0 as specified in the DOS manual for file attributes.
    - R=#100000
    - S=#020000
    - P=#000002
    - W=#000001

    WARNING!!!!! if #040000 (bit 1) is set and the file is permanent, it cannot be removed except by a full initialization of the directory!!!!!!!!

ac0 = getfileatr(chno) DOS:4-9

Purpose:
Returns the attributes of a DOS file.

Parameters:
- chno

A DOS channel previously opened to the file in question.

Function Results:

ac0

The word returned with meanings defined by the DOS manual.

incr = memavail()   DOS:4-21

Purpose:

Returns the amount of available memory for the user program.

Function Results:

incr

The increment of available memory.

memincr(incr)   DOS:4-21

Purpose:

Change the amount of user available memory.

Parameters:

incr

The increment of memory to be claimed.

dosexec(name, acl)   DOS:4-23

Purpose:

Execute a DOS save file.

Parameters:

name

The name of a DOS save file to be executed.

acl

The value for acl as specified by the DOS manual. If missing, 0 will be used so that the current execution level is pushed to the disk and the next save file will be started at its normal starting address.

dosreturn()   DOS:4-24

Purpose:

Return control to DOS CLI.

dosereturn(ac2) DOS:4-24

Purpose:

Return control to DOS giving an error code. The common error messages will be misprinted due to DOS assumptions about file names.

Parameters:

ac2

The error value to be returned.

dosbreak()   DOS:4-25

Purpose:

Create the file BREAK.SV. WARNING!!!! All I/O channels must be closed with a resetfiles command if the file is to be re-executed.

word = strtovalue(string[, radix])

Purpose:
Convert a signed string to a 16 bit integer in the specified radix.
Parameters:
string
The BCPL string to be converted.
radix
The radix of the conversion. If unspecified, 8 is assumed.
Function Results:
word
A 16 bit word having the converted value.

valuetostr(word, string, radix[, space])

Purpose:
Convert a 16 bit signed value to a signed string with no leading zeros having either fixed or variable spcing.
Parameters:
word
The 16 bit value to be converted.
string
A vector with enough space to hold the converted value. If fixed spacing is specified, overflow will cause more spaces to be used in this vector. The maximum number of spaces used depends on the radix and is 16 for radix 2, 6 for radices 8 and 10.
radix
The conversion radix.
space
The number of string spaces to be used. If zero or missing, variable space is assumed.

packstr(ustring, pstring)

Purpose:
Change a BCPL string from unpacked format (one byte per word) to packed format (two bytes per word).
Parameters:
ustring
A vector containing a BCPL unpacked string, one character per word, the first word specifying the length.
pstring
A vector with enough room to receive the packed string.

unpackstr(pstring, ustring)

Purpose:
Change a BCPL string from packed format (two bytes per word) to unpacked format (one byte per word).
Parameters:
pstring
A BCPL string.
ustring
A vector with enough room for the BCPL unpacked string, one character per word, the first word specifying the length.

movestr(stringsrc, stringdest)

Purpose:
  Move a BCPL string which may be in either packed or unpacked format.
Parameters:
  stringsrc
    A BCPL string to be moved.
  stringdest
    A vector with sufficient room to receive the source string.

byteptr = dostr(bcplstring, dosstring)

Purpose:
  Convert a BCPL string to a DOS string.
Parameters:
  bcplstring
    A BCPL string to be converted.
  dosstring
    A vector with sufficient space to receive the converted string. The only difference in the two formats is that DOS requires a null character at the end of many strings.
Function Results:
  byteptr
    A DOS byte pointer to the first character of the DOS string.

word = lengthstr(string)

Purpose:
  Return the length of a BCPL string.
Parameters:
  string
    A BCPL string.
Function Results:
  word
    The length of the string.

char = extractchar(string, index)

Purpose:
  Extract a single character from a string at a specified index.
Parameters:
  string
    A BCPL string.
  index
    The index for the character. If out of range, garbage is returned.
Function Results:
  char
    A 16 bit word containing the value of the character.

lengthstring1 = extractstr(string1, string2, index, lengthstring1)

Purpose:
  Extract string 1 from string 2 beginning at the specified index.

Parameters:
  string1
    A vector of sufficient size to receive the extracted string.
  string2
    The string from which the extraction is to be made.
  index
    The beginning index for extraction; if the index goes out of the range of string2 at any time, the length of the extracted string will be adjusted accordingly.
  lengthstr1
    The length of the string to be extracted.
Function Results:
  lengthstr1
    The actual length of the extracted string.

lastbyteindex = imbedchar(char, string[, index])

Purpose:
  Imbed a character into a vector containing a BCPL string. The existing character at that index is destroyed. If the index for the imbedded character is greater than the length of the string, the second string is filled with blanks up to the imbedded character. If no index is specified, the character will be appended.
Parameters:
  char
    The character to be imbedded.
  string2
    A vector or BCPL string in which the character is to be imbedded. If index extends the length of string2, string2 must be a vector large enough to hold the results.
  index
    The index in string2 at which the character is to be imbedded.
Function Results:
  lastbyteindex
    The last position of string2 which was modified.

lastbyteindex = imbedstr(string1, string2[, index])

Purpose:
  Imbed string1 in string2. The existing sub-string at that index is destroyed. If the index for the imbedded string1 is greater than the length of the string2, string2 is filled with blanks up to the imbedded character. If no index is specified, string1 will be appended to string 2.
Parameters:
  string1
    The string to be imbedded.
  string2
    A vector or BCPL string in which the first string is to be imbedded. If string1 extends the length of string2, string2 must be a vector large enough to hold the results.
  index
    The index in string2 at which string1 is to be imbedded.
  lastbyteindex
    The index of the last byte imbedded in string2.
Function Results:
  lastbyteindex

The last position of string2 which was modified.

index = searchstr(string1, string2[, startindex])

Purpose:
Search string1 for string2 at the specified starting index or at the start of string1.
Parameters:
string1
The string to be searched.
string2
The string to be found.
startindex
The index in string1 at which to begin the search.
Function Results:
index
The index of the string if it is found; if not, then -1.

SOURCE CODE

```
//  BCPL runtime -- global definitions

// DOS system definitions

manifest [

sysgchar = #67400
syspchar = #70000
sysopen = #74077
sysappend = #72477
sysclose = #74477
syscreate = #60000
sysdelete = #60400
sysrds = #75077
syswrs = #76477
sysrdl = #75477
syswrl = #77077
sysinit = #64000
sysdir = #63000
sysrlse = #62400
sysinst = #71477
sysrename = #61000
syschatr = #73077
sysgtatr = #73477
sysreset = #65000
sysmem = #61400
sysmemi = #71000
sysexec = #63400
sysrtn = #64400
sysertn = #66400
sysbreak = #62000

]


//  various constants

manifest [
filenamelength = 20

]

external [

//  static variables

syscall
syserror
sysac
syserrorflag
syserrortrap
syserrorvalue
sysprintpc

//      procedures

readcomcm
initbcplio
noargs
readch
writech
readseq
writeseq
readline
writeline
writestr
writezoct
readbin
writebin
createfile
open
append
close
curpos
curposdw
setpos
setposdw
systemcall
ioerror
deletefile
initdev
directorydev
releasedev
renamefile
chatr
getfileatr
getdevatr
resetfiles
memavail
memincr
dosexec
dosreturn
dosereturn
dosbreak

]

//  string procedures

external [

lengthstr
extractchar
searchstr
extractstr
imbedstr
imbedchar
packstr
movestr
unpackstr
strtovalue
valuetostr
writedec
writeoct
writeform
writevalue
]
```

```
//      BCPL I/O and Runtime

get "iox"

static [
syscall = nil   //dos system call procedure
syserror = nil  //dos system error procedursysac = nil
sysac = nil     //dos system call acs
sysprintpc = nil        //determines runtime error procedure address printout
syserrorpc = nil        //system address for print routine
usererrorpc = nil       //user address for print routine
syserrorflag = nil      //user error response flag
syserrortrap = nil      //user error control flag
syserrorvalue= nil      //error value
syserrorname = nil      //error name for ioerror
]

let readcomcm(chno, name, sw) be
[       //read the next name and switch list from com.cm
        //switches are returned in a 26 element boolean vector
        //iff sw is present.
  let i = readline(chno,name, true);  name!0 = name!0 - #400
  if i eq 0 then [ name!0 = 0; return ]

  let s,j,three = nil,0,noargs() eq 3
  for k = 1 to 4 do
    [   s = readch(chno)
        if three then for l = 1 to 8 do
          [    sw!j = (s & #200) ne 0
               if j ge 25 then break
               j=j+1; s = s lshift 1
            ]
       ]
 ]


and initbcplio(arg) be
[  syscall = rv #360
sysac = rv #362         //init system ac pointer for dos system calls
syserror = ioerror              //new error processor
sysprintpc = arg eq 2   //set procedure address print to true
                        //if argument of init call is 2
syserrortrap = true             //execute ioerror if true
]


and readch(chno) = valof
[
  if chno eq -1 do
    [   systemcall(nil, nil, nil, sysgchar, syserror)
        resultis sysac!0 & #377
      ]

  let c = 0
  let err = systemcall((lv c lshift 1) + 1, 1, chno, sysrds, 0)
  test err eq 6 then c = #377           //end-of-file error
  or   unless err eq -1 do syserror(sysrds, sysac)
  resultis c
 ]

and writech(chno,c) be
[
  if chno eq -1 do
    [   systemcall(c, nil, nil, syspchar, syserror)
        return
      ]

  systemcall((lv c lshift 1) + 1, 1, chno, syswrs, syserror)
 ]

and readseq(chno, bptr, nbts) = valof
[  let err = systemcall(bptr, nbts, chno, sysrds, 0)
  unless err eq 6 % err eq -1 do syserror(sysrds,sysac)
  resultis sysac!1
.
]

and writeseq(chno, bptr, nbts) be
[  systemcall(bptr, nbts, chno, syswrs, syserror)
]

and readline(chno, string, rdl) = valof
[  if noargs() ls 3 then rdl = false
   let bptr = (string lshift 1) + 1
   let n, err = 0, nil
   [  err = systemcall(bptr+n, nil, chno, sysrdl, 0)
         unless err eq 6 % err eq -1 do syserror(sysrdl, sysac)
         n = n + sysac!1 - (rdl? 0, 1)
   ]  repeatwhile (extractchar(string, n+1) & #177) eq 0 & not rdl
   n = n + (rdl ? 0, 1)
   string!0 = (n lshift 8) + (string!0 & #377)
   if err eq 6 & not rdl then n = imbedstr("*n$377",string)
   resultis n
]

and writeline(chno,string) be
[ if ((string!0 & #177400) eq 0) then return
  systemcall((string lshift 1) + 1, nil, chno, syswrl, syserror)
]

and writestr(chno,s) be
  for i = 1 to lengthstr(s) do [  let ch = extractchar(s,i)
        writech(chno,ch)
        if ch eq $*n then writech(chno,$*l)
        ]

and writezoct(chno,n) be
[ let zsw = false
  for i = 15 to 3 by -3 do
    [   let d = (n rshift i) & #7
        test zsw & (d eq 0)
        then [ writech(chno,$*s) ]
        or   [ writech(chno,d+$0); zsw = false ]
     ]
  writech(chno,(n & #7) + $0)
 ]

and readbin(chno) = valof
[
  let w = nil
  systemcall(lv w lshift 1, 2, chno, sysrds, syserror)
  resultis w
 ]

and writebin(chno,w) be
[
  systemcall(lv w lshift 1, 2, chno, syswrs, syserror)
 ]

and open(bcplname) = valof
[ if bcplname eq 0 resultis -1
  if bcplname!0 eq 0 resultis -1

  let channel = findchno()
      //if no free channels, system call will give error
  let dosname = vec filenamelength
  systemcall(dostr(bcplname, dosname), 0, channel, sysopen, syserror)
  resultis channel
]

and append(bcplname) = valof
[ if bcplname eq 0 resultis -1
  if bcplname!0 eq 0 resultis -1

  let channel = findchno()
  let dosname = vec filenamelength
  systemcall(dostr(bcplname, dosname), 0, channel, sysappend, syserror)
  resultis channel
]
```

```
and curposdw(channel,dw) be
[ unless 0 le channel & channel le 7 then [ dw!0 = 0; dw!1 = -1; return  ]
  let v = #430           //DOS channel table in page one
  let t = v!channel      //DOS descriptor for the channel
  dw!1 = t!#25  //word 25 is byte number in current block
  dw!0 = t!#24  //word 24 is current block number
]

and curpos(channel) = valof
[ let dw = vec 2;  curposdw(channel, dw)
  resultis ((dw!0 * 255) lshift 1) + dw!1
]

and setpos(channel, pos) be
[   let dw = vec 2
  dw!0 = (pos rshift 1) / 255   //file block number
  dw!1 = pos - ((dw!0 * 255) lshift 1)  //file bytenumber in last block
  setposdw(channel, dw)
]

and setposdw(channel,dw) be
[ unless 0 le channel & channel le 7 return
  let v = #430
  let t = v!channel
  t!#25 = dw!1  //dos byte count in last block
  t!#24 = dw!0  //dos block count in file
  t!#17 = t!#17 % #4    //set "first write" bit in status word
]


// now the dos system calls--


and systemcall(ac0,ac1,ac2,call,err) = valof
[       //generalized dos system call routine.
        //system acs returned in sysac vector, error value through function.
  sysac!0 = ac0; sysac!1 = ac1; sysac!2 = ac2
  let errsw = syscall(call,sysac)
  test errsw eq 0
   ifso [  syserrorflag = false;  resultis -1  ]
   ifnot [
        seterrorpc();  syserrorflag = true
        syserrorname = ac0 rshift 1
        syserrorvalue = errsw
        unless (err eq 0) do err(call,sysac);  resultis errsw  ]
]

and seterrorpc(arg) be
[  arg = rv(rv((lv arg) - 6) - #200) - #200  //points to system routine stack
  syserrorpc = rv(arg+2) - 3
  usererrorpc = rv((rv arg) - #200 + 2) - 3
]

and ioerror(call,ac) be
[  let ierr, jerr = syserrorpc, usererrorpc
  let name, err = nil, vec 1
  test noargs() eq 1
   ifso [ ac = sysac; ac!2 = call; err = call; name = syserrorname ]
   ifnot [  name = ((ac!0) rshift 1);  err = ac!2 ]
  if (not syserrortrap) & noargs()eq 2 then return
  if sysprintpc then
        [ writestr(-1,"*nsystem proc="); writeoct(-1,ierr);
          writestr(-1," user proc ="); writeoct(-1,jerr);
          writestr(-1,"*n")
        ]
  if err eq 1 % err eq 3 % err eq 4 % err eq #36 then
        [  writestr(-1, name); writech(-1,$*s); dosereturn(err)  ]
  switchon err into
        [  case #11:[  writestr(-1,"file already exists, file: "); endcase ]
          case #12:[  writestr(-1,"file does not exist, file: "); endcase ]
          case #13:[  writestr(-1,"attempt to alter a permanent file: "); endcase ]
          default:[  dosereturn(err)  ]
          ]
  writestr(-1,name);  writestr(-1,"*n");  dosreturn()
]

and noargs(arg) = rv(rv((lv arg) - 6) - #200 + 5)
                // back to the last frame to the number of args

and findchno() = valof
[  let v = #430
  for i = 0 to 7 do if (v!i & #100000) ne 0 do resultis i
  resultis #100000
]

and createfile(name) be
[   let dosname = vec filenamelength
  systemcall(dostr(name, dosname), nil, nil, syscreate, syserror)
]

and deletefile(name) be
[   let dosname = vec filenamelength
  let err = systemcall(dostr(name, dosname), nil, nil, sysdelete, 0)
  unless (err eq #12) % (err eq -1) do syserror(sysdelete,sysac)
]

and initdev(name) be
[   let dosname = vec filenamelength
  systemcall(dostr(name, dosname), 0, nil, sysinit, syserror)
]

and directorydev(name) be
[   let dosname = vec filenamelength
  systemcall(dostr(name, dosname), nil, nil, sysdir, syserror)
]

and releasedev(name) be
[   let dosname = vec filenamelength
  systemcall(dostr(name, dosname), nil, nil, sysrlse, syserror)
]

and renamefile(name,newname) be
[ let newdosname = vec filenamelength
  let dosname = vec filenamelength
  systemcall(dostr(name, dosname), dostr(newname, newdosname), nil, sysrename, syserror)
]

and close(chno) be
[ systemcall(nil, nil, chno, sysclose, syserror)
]

and resetfiles() be
[ systemcall(nil, nil, nil, sysreset, syserror)
]

and install(channel) be
[  systemcall(channel, nil, nil, sysinst, syserror)
]

and chatr(chno,ac0) be
[
systemcall(ac0, nil, chno, syschatr, syserror)
]

and getfileatr(chno) = valof
[ systemcall(nil, nil, chno, sysgtatr, syserror)
  resultis sysac!0
]

and memavail() = valof
[ systemcall(nil, nil, nil, sysmem, syserror)
  resultis sysac!0 - sysac!1
]

and memincr(incr) = valof
[ systemcall(incr, nil, nil, sysmemi, syserror)
  resultis sysac!1
```

```
]

and dosexec(name, acl) be
[   let dosname = vec filenamelength
  systemcall(dostr(name, dosname), (noargs() eq 2? acl, 0), nil, sysexec, syserror)
]

and dosreturn() be
[ systemcall(nil, nil, nil, sysrtn, syserror)
]

and dosereturn(ac2) be
[ systemcall(nil, nil, nil, sysertn, syserror)
]

and dosbreak() be
[ systemcall(nil, nil, nil, sysbreak, syserror)
]


// now the string procedures necessary for io-runtime


and lengthstr(s) = s!0 rshift 8

and imbedstr(s1,s2,i) = valof
        //if i is larger than length of s2, s1 is still inserted
        //and blanks are filled in empty space.
        //if i is not specified, ch is appended.

[  let ls1, ls2 = s1!0 rshift 8, s2!0 rshift 8
  if noargs() eq 2 then i = ls2 + 1
  if (ls1 + i) gr 255 then ls1 = 255-ls2
  if (i le 0 ) % (i gr 255) then resultis 0

   [  let t = i+ls1-ls2-1;  if t gr 0 then s2!0 = s2!0 + (t lshift 8) ]
  let bcnt = i - ls2 - 1
  if bcnt gr 0 then [ let wls2 = ls2 rshift 1;
        if (ls2 & 1) eq 0 then [  s2!wls2 = (s2!wls2 & #177400)+#40
                         bcnt = bcnt - 1 ]
        for i = 1 to (bcnt+1) rshift 1 do s2!(i+wls2) = #20040
        ]
  let mfb = ((ls1+i)&1) eq 1
  let wi, wls1 = i rshift 1, ls1 rshift 1
  let bdry = true

//move first byte if i is odd to get on a word bdry of dest

  if (i&1) eq 1 then [  s2!wi = (s2!wi & #177400) + (s1!0 & #377)
                        i = i + 1; wi = wi + 1;
                        bdry = false  ]

//  now do the word moves

  for j = 1 to wls1 do [  s2!wi = bdry ?
                (s1!(j-1) lshift 8) + (s1!j rshift 8), s1!j
                wi=wi+1 ]

//  now check for the final byte

  if mfb then s2!wi = (s2!wi&#377) +
                ((ls1&1) eq 1 ? s1!wls1 lshift 8,
                               s1!wls1 & #177400)
  ls2 = s2!0 rshift 8;  let wls2 = ls2 rshift 1
  if (ls2 & 1) eq 0 then s2!wls2 = (s2!wls2)&#177400
  resultis (wi lshift 1) + (mfb ? 1, 0)
]

and imbedchar(ch,s1,i) = valof
        //if i is larger than length of s1, ch is still inserted
        //and blanks are filled in empty space.
        //if i is not specified, ch is appended.

[  let s = vec 1; s!0 = #400 + ch
```

```
  test noargs() ls 3 then resultis imbedstr(s,s1) or resultis imbedstr(s,s1,i)
]

and movestr(p1, p2) be
[
  if p1 eq p2 then return
  let n = p1!0 rshift 8
  test n eq 0
  then n = p1!0
  or   n = n/2
  for i = 0 to n do p2!i = p1!i
 ]

and dostr(bn,dn) = valof [
movestr(bn,dn)
imbedstr("*000",dn)
resultis (dn lshift 1) + 1
]

and extractchar(s,i) = ((i&1) eq 1) ?
        (s!(i rshift 1) & #377), (s!(i rshift 1) rshift 8)
```

```
get "iox"

let searchstr(s1,s2,ix) = valof

[
  let ls1,ls2 = s1!0 rshift 8, s2!0 rshift 8

  let ch2   = s2!0 & #377
  let streq = false
  let k,wls2,kbit = nil,(ls2-1) rshift 1,nil

  for i = ((noargs() eq 3)&(ix gr 0)? ix, 1) to ls1-ls2+1 do

  [1  if ch2 eq (((i&1) eq 1) ? (s1!(i rshift 1) &#377), (s1!(i rshift 1) rshift 8))
                then [2
                kbit = (i+1) & 1;  k = (i+1) rshift 1; streq = true
                for j = 1 to wls2 do
                [3
                        unless (s2!j eq ((kbit ?
                               ((s1!k lshift 8) + (s1!(k+1) rshift 8)),
                               (s1!k))))
                        do [ streq = false; break ]
                k = k + 1
                ]3

        if streq & ((ls2&1) eq 0) then
                        if (s2!(wls2+1) & #177400)
                               eq (kbit ? ((s1!k) lshift 8),
                                          ((s1!k) & #177400))
                        then resultis i
        ]2
  if streq then resultis i
]1

resultis 0      // exit here if no match is found.
]

and extractstr(s1, s2, i, ls1) = valof

[  let ls2 = s2!0 rshift 8
  if noargs() eq 3 then ls1 = s1!0 rshift 8
  if ls1 eq 0 then [  s1!0 = 0; resultis 0  ]
  if ls1 gr (ls2-i+1) then ls1 = ls2 - i + 1
  let k, kbit, wls1 = (i+1) rshift 1, (i+1) & 1, (ls1 - 1) rshift 1

  s1!0 = (ls1 lshift 8) +
        (((i&1) eq 1)?(s2!(i rshift 1) & #377), (s2!(i rshift 1) rshift 8))

  for j = 1 to wls1 do
        [1  s1!j = kbit ? (s2!k lshift 8) + (s2!(k+1) rshift 8), s2!k
          k = k + 1
        ]1
  if ((ls1 & 1) eq 0 ) then s1!(wls1 + 1) = kbit ?
                s2!k lshift 8, s2!k & #177400

  resultis ls1
]

and strtovalue(name,rdx) = valof
[       //get number from string in radix rdx, default is 8
  if noargs() eq 1 then rdx = 8
  let n,s,minus = 0,nil,false
  for i = 1 to lengthstr(name) do
    [     s = extractchar(name,i) & #177
        if s eq $- then [  minus = true; loop]
        s = s - $0
        if 0 le s & s le rdx-1 do
        n = n*rdx + s
     ]
  resultis minus?-n, n
 ]


and packstr(u, p) be
```

```
[
  let n = u!0
  let i, j = 0, 0
  [ p!j = u!i lshift 8
    i = i + 1; if i gr n return
    p!j = p!j + (u!i & #377)
    i = i + 1; if i gr n return
    j = j + 1
  ] repeat
 ]

and unpackstr(p, u) be
[
  let n = p!0 rshift 8
  let i, j = 0, 0
  [ u!i = p!j rshift 8
    i = i + 1; if i gr n return
    u!i = p!j & #377
    i = i + 1; if i gr n return
    j = j + 1
  ] repeat
 ]

and valuetostr(w, s, rdx, sp) be
[ let spc = (noargs() eq 4) & (sp gr 0)
  let min = w ls 0
  s!0 = 0
```

if rdx ls 2 % rdx gr 10 return

fixed: 6/14/73 [illegible]
Jun 14.2m

```
  let getdigt(w, s, rdx, sp, min, spc) = valof
        [  let j = w; w = w/rdx; sp = sp-1
        test w ne 0
          ifso imbedchar(getdigt(w, s, rdx, sp, min, spc), s)
          ifnot [  test min
                ifso imbedstr("-", s, (spc?sp,1))
                ifnot if spc then imbedstr(" ", s, sp)
                ]
        resultis $0 + (min? -j+w*rdx, j-w*rdx)
        ]
  imbedchar(getdigt(w, s, rdx, sp, min, spc),s)
]

and writevalue(chno, w, rdx, sp) be
[ if noargs() ls 4 then sp = 0
  let s = vec 10
  valuetostr(w, s, rdx, sp)
  writestr(chno, s)
]

and writedec(chno, w, sp) be
[  if noargs() ls 3 then sp = 0
  writevalue(chno, w, 10, sp)
]

and writeoct(chno, w, sp) be
[  if noargs() ls 3 then sp = 0
  writevalue(chno, w, 8, sp)
]

and writeform(chno, nil,nil, nil,nil, nil,nil, nil,nil, nil,nil, nil,nil,
nil,nil, nil,nil, nil,nil, nil,nil) be
[  let arg = lv chno
  for i = 1 to noargs()-1 by 2 do
        [  if arg!i ls 0 % arg!i gr 10 loop
        test (arg!i) eq 0
          ifso writestr(chno, arg!(i+1))
          ifnot writevalue(chno, arg!(i+1), arg!i)
        ]
]
```