Subject: DDREAD (was Re: Getting memory overflow...)
Posted by ryba on Wed, 25 Jan 1995 20:19:52 GMT
View Forum Message <> Reply to Message

By request, and since it wasn't in meteo, here (again?) is DDREAD.pro,
(stands for Data Dump READ) a nice tabular file reader. The credit
goes to Fred Knight for this gem....

```
;+
; Name:
; ddread
; Purpose:
; This routine reads data in formatted (or unformatted) rows and columns.
; The name stands for Data Dump Read. By default, comments are
; skipped and the number of columns is sensed. Many options
; exist, e.g., selecting rows and columns, reading binary data,
; and selecting non-default data type and delimiters.
;
; Examples:
; junk = ddread(/help)   ; get information only
; array = ddread(file)   ; read ASCII data
; array = ddread(file,/formatted)  ; ditto
; array = ddread(file,object=object) ; read binary data
; array = ddread(file,columns=[0,3]) ; get only 1st & 4th columns
; array = ddread(file,rows=lindgen(10)+10); get only 2nd 10 rows
; array = ddread(file,offset=10,last=19) ; get rows (10,19)
; array = ddread(file,/countall)  ; count comment lines
; array = ddread(file,/verbose)  ; echo comment lines
; array = ddread(file,type=1)  ; return bytes, not floats or longs
; array = ddread(file,range=['start text','stop text']) ; text delimiters
;
; ; Place the detailed output from a Lowtran run in a 2-D array---wow!
; output = ddread('lowtran.out',range=['(CM-1) (MICRN)','0INTEGRATED ABSORPTION'])
; % DDREAD: Read 69 data lines selecting 14 of 14 columns; skipped 395 comment lines.
; Usage:
; array = ddread([file][,options][,/help])
; Optional Inputs:
; file = file with data; if omitted, then call pickfile.
; Keywords:
; /formatted, /unformatted = flags to tell IDL whether data format is
;  binary or ASCII.  ddread tries to determine the type
;  of data but it's not foolproof.
; object = a string containing the IDL declaration for one instance
;  of the object in an unformatted file, e.g.,
;   'fltarr(4)'
;  or
;   '{struct,dwell:0.,pitch:0.,yaw:0.,roll:0.}'
; rows = an array to select a subset of the rows in a formatted file
;  Does not count comment lines, unless /countallrows is set!
```

```
; columns = likewise for columns
; type = data type of the output D=float (if '.' appears) or long
; delimiter = column separater, D=whitespace
; /help = flag to print header
; range = start and stop row or strings,
;  e.g. range = ['substring in 1st line','substring in last line']
; offset = start row (read to end of file, unless last set)
; last = stop row (read from start of file, unless offset set)
; /countallrows = flag to count comment rows as well as data rows (D=0)
; /verbose = flag to echo comments to screen
; Outputs:
; array = array of data from the lines (ASCII) or objects (binary)
; Common blocks:
; none
; Procedure:
; After deciding on ASCII or binary, read file and return array.
;
; Restrictions:
; - Comments can be either an entire line or else an end of a line, e.g.,
;  /* C comment. */
;  ; IDL comment
;  Arbitrary text as a comment
;  Comment in Fortran
;  The next line establishes # of columns (4) & data type (float):
;  6. 7 8 9
;  This line and the next are both considered comments.
;  6 comment because only one of 4 columns appears
;  1 2 3 4 but this line has valid data and will be read as data
; - Known problem: If the file has all comments, ddread is an infinite
;   loop. (9 Nov 94)
;
; - Even if a range of lines is selected with offset, range or last, all
;   lines are read.  This could be avoided.
;
; - Other routines needed:
;   pickfile.pro - to choose file if none is given
;   nlines.pro - to count lines in a file
;   nbytes.pro - to count bytes in a variable
;   replicas.pro - to replicate arrays (not scalars as in replicate.pro)
;   typeof.pro - to obtain the type of a variable
;
; Modification history:
; write, 22-26 Feb 92, F.K.Knight (knight@ll.mit.edu)
; allow reading with arbitrary delimiter using reads, 23 Mar 92, FKK
; add countallrows keyword and modify loop to read as little
;   data as possible, 20 May 92, FKK
; correct bug if /formatted set, 6 Jul 92, FKK
; add verbose keyword to print comments, 6 July 92, FKK
```

```
; correct bug if /rows=...,/countall set, 6 July 92, FKK & EJA
; add a guard against a blank line being converted to a
;   number, 21 Aug 92, FKK
; allow parital line just before the EOF.  Possibly this isn't the
;   right thing to do, but I decided to allow it.  If the final line
;   is incomplete, the values are still read and the remainder of
;   the line is filled with zeroes. 26 Oct 92, FKK
; allow range keyword to be a string array, 2 Dec 92, FKK
; make default for countallrows be true if range is present, 2 Dec 92, FKK
; add new function (typeof); called in a few places, 2 Dec 92, FKK
; make all row counters longs to avoid overflow at 32768, 23 Jun 93, FKK
; remove the 26 Oct 92 mod because, as it stands, it places a line of
;   zeroes at the end when the last line is a comment, 8 Dec 94, FKK
;-
function  ddread,help=help,file,formatted=formatted,unformatted=unform atted $
   ,rows=rows,columns=columns,object=object,type=type,delimiter =delimiter $
   ,offset=offset,last=last,range=range,countallrows=countallro ws $
  ,verbose=verbose
;
; =====>> HELP
;
on_error,2
if keyword_set(help) then begin & doc_library,'ddread' & return,0 & endif
;
; =====>> SETUP
;
on_ioerror,IOERROR
if n_elements(file) eq 0 then file = pickfile()
if n_elements(countallrows) eq 0 then countallrows = n_elements(range) gt 0
gotobject = n_elements(object) gt 0
if keyword_set(formatted) then begin
  if gotobject then message,/inform,'You should not specify an object for formatted data.  Try
tmp=ddread(/help).  Ignoring object.'
  goto,ASCII
  endif
if gotobject then begin
  status = execute('obj = '+object)
  if not status then message,'Your object '+object+' is incorrect.'
  goto,BINARY
  endif
if keyword_set(unformatted) and (not gotobject) then message,'For unformatted dump, you need
to supply an object. Try tmp=ddread(/help).'
goto,ASCII     ; The default
;
; =====>> READ UNFORMATTED DATA USING OBJ
;
BINARY:
openr,lun,file,/get_lun
```

```
stat = fstat(lun)
nb = nbytes(obj)    ; # of bytes in one object
all = stat.size/nb    ; the entire file
if n_elements(offset) eq 0 then offset = 0L ; user-defined offset
if n_elements(last) eq 0 then last = all - 1L ; user-defined end
if n_elements(range) eq 2 then begin  ; range takes precedence
  if typeof(range) ge 7 then message,'For binary data, range keyword must have 2 numbers.'
  offset = long(range(0))
  last = long(range(1))
  endif
array = replicas(obj,last-offset+1)  ; output array
point_lun,lun,nb*offset    ; locate file ptr at offset
readu,lun,array
close,lun
free_lun,lun
return,array
;
; =====>> OPEN FILE FOR FORMATTED DATA
;
ASCII:
nlines = nlines(file)    ; COUNT LINES IN FILE
openr,lun,file,/get_lun
;
; =====>> SET THE LIMITS OF THE READ IF POSSIBLE (countallrows=1)
;
lbegin = 0L     ; DEFAULT START
lend = nlines-1L    ; DEFAULT END
if countallrows then begin
  if n_elements(range) eq 2 then begin
    if typeof(range) eq 7 then begin  ; IF STRING, SEARCH FILE.
      point_lun,lun,0    ; GO TO START OF FILE
      count = 0L    ; LINE COUNTER
      rdbuf = ' '
      while not eof(lun) do begin  ; LOOP & SET lbegin & lend
        readf,lun,rdbuf
        if (strpos(rdbuf,range(0)) ge 0) and (lbegin eq 0) then begin
          lbegin = count
          stat = fstat(lun)
          data_start = stat.cur_ptr  ; SAVE PTR TO START OF DATA
          endif
        if (strpos(rdbuf,range(1)) ge 0) then lend = count
        count = count + 1L
        endwhile
    endif else begin
      lbegin = long(range(0))
      lend = long(range(1))
    endelse
  endif
```

```
  if n_elements(rows) gt 0 then begin
    lbegin = long(min(rows))
    lend = long(max(rows))
  endif
  if n_elements(offset) eq 1 then lbegin = long(offset)
  if n_elements(last) eq 1 then lend = long(last)
endif
;
; =====>> SKIP TO START OF DATA
;
if typeof(data_start) eq 0 then begin  ; IF UNDEFINED, SKIP LINES.
  point_lun,lun,0    ; GO TO START OF FILE
  rdbuf = ' '
  if lbegin gt 0 then begin   ; READ TO START OF DATA
    for i=0L,lbegin-1 do readf,lun,rdbuf
    endif
  stat = fstat(lun)
  data_start = stat.cur_ptr   ; SAVE PTR TO START OF DATA
  endif
point_lun,lun,data_start   ; GO TO START OF DATA
;
; =====>> SETUP FOR READING FORMATTED DATA
;
on_ioerror,COMMENT   ; READY TO SKIP COMMENTS
ncols = 0    ; ASSUME NO COLUMNS
tmp = ''
COMMENT:    ; FIND A VALID LINE-ONE STARTING WITH
  readf,lun,tmp    ;   A VALID FLOAT
  if n_elements(delimiter) eq 1 then $ ; CHANGE DELIMITER TO SPACE
    while (((i = strpos(tmp,delimiter))) ne -1) do strput,tmp,' ',i
  if strlen(tmp) eq 0 then goto,COMMENT
  line = float(tmp)
tmp = ' ' + strcompress(strtrim(tmp,2)) ; KILL EXCESS WHITESPACE
for i = 0,strlen(tmp)-1 do $  ; LOOP OVER CHARACTERS
  ncols = ncols + (strpos(tmp,' ',i) eq i) ; COUNT SPACES: ONE/COLUMN
if ncols eq 0 then message,'No valid lines in file.'
if n_elements(columns) eq 0 then columns = lindgen(ncols); SELECT COLUMNS
ncolumns = n_elements(columns)
if n_elements(type) eq 0 then $
  if strpos(tmp,'.') ge 0 then type = 4 else type = 3 ; USE FLOATS OR LONGS.
;
; =====>> DEFINE DATA ARRAYS FOR READING AND OUTPUT
;
nrows = lend - lbegin + 1L
array = make_array(size=[2,ncolumns,nrows,type,ncolumns*nrows])
tmp = make_array(size=[1,ncols,type,ncols]) ; READ BUFFER
linectr = lonarr(nrows)    ; 1(0) = DATA(COMMENT) LINE
;
```

```
; =====>> READ FORMATTED DATA.
; =====>> IF DELIMITER SPECIFIED, THEN CHANGE EACH DELIMITER TO BLANK.
; =====>> TO AVOID OVERHEAD OF TRANSLATING DELIMITER, MAKE TWO LOOPS: ONE
; =====>> FOR WHITESPACE AND THE OTHER FOR USER-SPECIFIED DELIMITER
;
point_lun,lun,data_start   ; RETURN TO START OF DATA
if n_elements(delimiter) eq 1 then begin ; HERE FOR USER-SPECIFIED DELIMITER
  on_ioerror, SKIP1
  for l = 0L,nrows-1 do begin   ; LOOP OVER LINES
    readf,lun,rdbuf    ; READ AS ASCII
    while (((i = strpos(rdbuf,delimiter))) ne -1) do strput,rdbuf,' ',i
    rdbuf = strcompress(strtrim(rdbuf,2)) ; KILL EXCESS WHITESPACE
    reads,rdbuf,tmp    ; CONVERT FROM STRING TO NUMBER
    array(0,l) = tmp(columns)
    linectr(l) = 1
    goto,ENDLOOP1
    SKIP1:
      if keyword_set(verbose) then message,/inform,rdbuf
;     if eof(lun) and (strlen(rdbuf) gt 0) then begin ; HANDLE PARTIAL LINE BEFORE EOF
      if 0 then begin ; omit PARTIAL LINE BEFORE EOF
        tmp(*) = 0    ; FILL FIRST
        on_ioerror, GOTEOF1
        reads,rdbuf,tmp
        GOTEOF1:
        array(0,l) = tmp(columns)
        linectr(l) = 1
        endif
    ENDLOOP1:
    endfor
endif else begin
  on_ioerror, SKIP0
  for l = 0L,nrows-1 do begin   ; LOOP OVER LINES
    readf,lun,rdbuf
    reads,rdbuf,tmp    ; CONVERT FROM STRING TO NUMBER
    array(0,l) = tmp(columns)
    linectr(l) = 1
    goto,ENDLOOP0
    SKIP0:
      if keyword_set(verbose) then message,/inform,rdbuf
;     if eof(lun) and (strlen(rdbuf) gt 0) then begin ; HANDLE PARTIAL LINE BEFORE EOF
      if 0 then begin ; omit PARTIAL LINE BEFORE EOF
        tmp(*) = 0    ; FILL FIRST
        on_ioerror, GOTEOF0
        reads,rdbuf,tmp
        GOTEOF0:
        array(0,l) = tmp(columns)
        linectr(l) = 1
        endif
```

```
    ENDLOOP0:
    endfor
endelse
nl = long(total(linectr))   ; # OF DATA LINES
ns = long(nrows - nl)    ; # OF COMMENT LINES
message,/inform,'Read '+strtrim(nl,2)+' data lines selecting '+strtrim(n_elements(columns),2) +' of
'+strtrim(ncols,2) $
 + ' columns; skipped '+strtrim(ns,2)+' comment lines.'
close,lun
free_lun,lun
;
; =====>> SELECT LINES FROM FORMATTED FILE
;
if nl eq 0 then return,0
if keyword_set(countallrows) then begin
  if n_elements(rows) eq 0 then begin   ; SELECT LINES
    return,array(*,where(linectr))
  endif else begin
    return,array(*,where(linectr(rows-lbegin)))
  endelse
endif else begin
  if n_elements(rows) eq 0 then rows = lindgen(nl) ; SELECT DATA LINES
  array = array(*,where(linectr))
  return,array(*,rows)
endelse
;
; =====>> I/O error
;
IOERROR:
message,/inform,!err_string
if n_elements(lun) eq 1 then begin
  close,lun
  free_lun,lun
  endif
return,0
end

 ----------------------snip---------------------------------
;+
; Name:
; nlines
; Purpose:
; Return the number of lines in a file
; Usage:
; nl = nlines(file)
; Inputs:
; file = file to scan
; Optional Inputs or Keywords:
```

```
; help = flag to print header
; Outputs:
; nl = number of lines in the file.
; Common blocks:
; none
; Procedure:
; Assume ASCII data and read through file.
; Modification history:
; write, 24 Feb 92, F.K.Knight (knight@ll.mit.edu)
; make nl a long to avoid overflow at 32768 lines, 23 Jun 93, FKK
;-
function nlines,file,help=help
;
; =====>> HELP
;
on_error,2
if keyword_set(help) then begin & doc_library,'nlines' & return,0 & endif
;
; =====>> LOOP THROUGH FILE COUNTING LINES
;
tmp = ' '
nl = 0L
on_ioerror,NOASCII
if n_elements(file) eq 0 then file = pickfile()
openr,lun,file,/get_lun
while not eof(lun) do begin
  readf,lun,tmp
  nl = nl + 1
  endwhile
close,lun
free_lun,lun
NOASCII:
return,nl
end
 ----------------------snip---------------------------------
;+
; Name:
; nbytes
; Purpose:
; Return the number of bytes in the variable
; Usage:
; nb = nbytes(variable)
; Inputs:
; variable = any IDL variable
; Optional Inputs or Keywords:
; help = flag to print header
; Outputs:
; nb = number of bytes in variable
```

```
; Common blocks:
; none
; Procedure:
; Idea from David Stern.
; Modification history:
; write, 22 Feb 92, F.K.Knight
; increase speed by writing to disk only for structures, 10 Sep 92, FKK
; eliminate Unix-specific file (from ali@rsinc.com), 11 Sep 92, FKK
;-
function nbytes,variable,help=help
;
; =====>> HELP
;
on_error,2
if keyword_set(help) then begin & doc_library,'nbytes' & return,-1 & endif
;
; =====>> CHOOSE OPTION BASED ON TYPE OF VARIABLE
;
sz = size(variable)
type = sz(sz(0)+1)
nelements = sz(sz(0)+2)
case type of
0:return,0    ; UNDEFINED
1:return,nelements   ; BYTE
2:return,nelements*2   ; INT
3:return,nelements*4   ; LONG
4:return,nelements*4   ; FLOAT
5:return,nelements*8   ; DOUBLE
6:return,nelements*8   ; COMPLEX
7:return,long(total(strlen(variable))) ; STRING: DOESN'T COUNT NULLS
8:begin     ; STRUCTURE: COMPILER-SPECIFIC LENGTH
  file = filepath('IDL',/tmp)
  openw,lun,file,/get_lun,/delete
  writeu,lun,variable(0)  ; SO WRITE 1ST ELEMENT TO FILE
  stat = fstat(lun)
  close,lun
  free_lun,lun
  return,stat.size*nelements  ; AND RETURN TOTAL LENGTH
  end
else: message, 'unknown type = '+strtrim(type,2)
endcase
end
 ----------------------snip---------------------------------
;+
; Name:
; replicas
; Purpose:
; Replicate an array, as replicate.pro does for scalars or structures.
```

```
; Replicas calls replicate if input is a scalar or structure.
; Usage:
; array = replicas(/help)
; copies = replicas(array,ncopies)
; Inputs:
; array = array to replicate
; ncopies = number of replicas
; Optional Inputs or Keywords:
; help = flag to print header
; Outputs:
; copies = an array of one dimension more than array and filled
;  with copies of array
; Common blocks:
; none
; Procedure:
; If keyword help is set, call doc_library to print header.
; This routine could be incorporated into a more general replicate.
; However, the use of the parameter ncopies is not as general as
; in replicate.
; Modification history:
; write, 22 Feb 92, F.K.Knight
; guard against calling replicate with an array of structures, 18 Nov 92, FKK
;-
function replicas,array,ncopies,help=help
;
; =====>> HELP
;
on_error,2
if keyword_set(help) then begin & doc_library,'replicas' & return,-1 & endif
;
; =====>> SETUP
;
if ncopies le 0 then message,'Number of replicas ('+strtrim(ncopies,2)+') must be positive.'
sa = size(array)
type = sa(sa(0)+1)
;
; =====>> IF SCALAR OR STRUCTURE, THEN USE REPLICATE.
;
if sa(0) eq 0 or ((sa(1) eq 1) and (type eq 8)) then $
  return, replicate(array,ncopies)
;
; =====>> TREAT ARRAY OF STRUCTURES: CAN'T FIGURE OUT HOW TO DO IT
;
if type eq 8 then message,'Cannot do array of structures.'
;
; =====>> OTHERWISE, REPLICATE USING make_array.
;
na = n_elements(array)
```

```
sc = [sa(0)+1,sa(1:sa(0)),ncopies,type,ncopies*na]
copies = make_array(size=sc)
for i = 0,ncopies-1 do copies(na*i) = array(*)
return,copies
end
 ----------------------snip--------------------------------
;+
; Name:
; typeof
; Purpose:
; Function to return the type of the variable---a shorthand
; for extracting the type from the array returned by size.
; Usage:
; if typeof(variable) eq 7 then message,'Variable is a string.'
; Inputs:
; variable = any IDL variable
; Optional Inputs or Keywords:
; help = flag to print header
; Outputs:
; typeof = type code from the size array
; Common blocks:
; none
; Procedure:
; Just get the type code from the size array.
; Modification history:
; write, 2 Dec 92, F.K.Knight (knight@ll.mit.edu)
;-
function typeof,variable,help=help
;
; =====>> HELP
;
on_error,2
if keyword_set(help) then begin & doc_library,'typeof' & return,0 & endif
;
; =====>> RETURN THE TYPE CODE
;
szv = size(variable)
return,szv(szv(0)+1)
end
```

--

Dr. Marty Ryba | Generation X:
MIT Lincoln Laboratory | Too young to be cynical,
ryba@ll.mit.edu | too old to be optimistic.