Subject: Re: Putting bytes into structures
Posted by Marty Ryba on Thu, 23 Oct 1997 07:00:00 GMT
View Forum Message <> Reply to Message

This is a multi-part message in MIME format.

--------------2BE345064975
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit

Marty Ryba wrote:
> Brian Jackel wrote:
>> What's the best way to load a given number of bytes into a structure?
> Here's a routine called READB.PRO that does exactly what you want.
> It recursively descends structures and places the bytes into them.

Oops, I noticed it uses a routine called NBYTES.PRO; here it is.
Also, both routines need updating to handle type DCOMPLEX for IDL 5.0

--
Dr. Marty Ryba | Of course nothing I say here is official
MIT Lincoln Laboratory | policy, and Laboratory affililaton is
ryba@ll.mit.edu | for identification purposes only,
| blah, blah, blah, ...

--------------2BE345064975
Content-Type: text/plain; charset=us-ascii; name="nbytes.pro"
Content-Transfer-Encoding: 7bit
Content-Disposition: inline; filename="nbytes.pro"

```
;+
; 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

--------------2BE345064975--
```

Subject: Re: Putting bytes into structures
Posted by Marty Ryba on Thu, 23 Oct 1997 07:00:00 GMT
View Forum Message <> Reply to Message

This is a multi-part message in MIME format.

--------------4C447A4A12D4

Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit

Brian Jackel wrote:
> What's the best way to load a given number of bytes into a structure?

Here's a routine called READB.PRO that does exactly what you want.
It recursively descends structures and places the bytes into them.

--
Dr. Marty Ryba | Of course nothing I say here is official
MIT Lincoln Laboratory | policy, and Laboratory affililaton is
ryba@ll.mit.edu | for identification purposes only,
| blah, blah, blah, ...

----------4C447A4A12D4
Content-Type: text/plain; charset=us-ascii; name="readb.pro"
Content-Transfer-Encoding: 7bit
Content-Disposition: inline; filename="readb.pro"

```
;+
; Name:
; readb
; Purpose:
; Procedure to copy the bits of an array to another IDL variable.
; Good for internal conversion, but limited in scope.
; A more general approach would allow other than array input.
; Examples:
;    #1
; a = replicate(byte('ff'x),4) ; MAKE A SOURCE ARRAY
; t = intarr(2)   ; MAKE A DESTINATION ARRAY
; readb,a,t   ; NEW ROUTINE
; print,t    ; EQUIVALENT TO: t = fix(a,0,2)
;      -1      -1
;    #2
; print, 'test of nested structures'
; print, '========================='
; st1 = {st1, aa:0L, bb:55.}
; a=bindgen(216)   ; JUST ENOUGH BYTES!
; b={t1:0L, t2:0.0, t3:lonarr(5,5), t4:fltarr(5,5), t5:st1}
; readb, a, b
; Usage:
; readb,source,destination[,offset][,/help]
; Inputs:
; source = IDL source array, not a structure
; Optional Inputs:
; offset = offset of starting point measured in elements of source
;   Offset is set if this is a recursive call.  It might also be
```

```
;   set if you want to start reading from the middle of the source array.
;   Note that offset is just like offset in a = fix(source,offset).
; Keywords:
; help = flag to print header
; Outputs:
; destination = IDL destination variable, any type
; Common blocks:
; none
; Procedure:
; If keyword help is set, call doc_library to print header.
; The recursive portion is from Alan Youngblood (support@rsinc.com).
; For a structure, there is a loop through the tags and readb calls
; itself.  For each tag, you reach a conversion line using one of
; byte(), fix(), long(), etc., after which offset is incremented by
; the length of the tag.
; Modification history:
; start with Alan's ideas, 8-10 Sep 92, FKKnight (knight@ll.mit.edu)
;-
pro readb,source,dest,offset,help=help
;
; =====>> HELP
;
on_error,2
if keyword_set(help) then begin & doc_library,'readb' & return & endif
;
; =====>> SET DEFAULTS
;
if (n_elements(offset) eq 0) then offset = 0
szs = size(source)
stype = szs(szs(0)+1)   ; SOURCE TYPE
szd = size(dest)
dcount = szd(szd(0)+2)   ; NUMBER OF ELEMENTS TO CONVERT
dtype = szd(szd(0)+1)   ; DESTINATION TYPE
bp = [1,1,2,4,4,8,8,1,1]  ; BYTES PER ELEMENT, except struct
if bp(stype) gt bp(dtype) then message,'Can''t handle source element longer than destination
element.'
; print,'source:',szs
; print,'dest  :',szd
;
; =====>> ACT BASED ON THE TYPE OF DESTINATION.
;
if dtype eq 0 then begin  ; UNDEFINED
  dest = source(offset:*)  ; COPY THE REMAINDER OF SOURCE
  return
  endif
if dtype eq 8 then goto,STRUCTURE ; NEEDS RECURSION
if dcount eq 1 then goto,SCALAR
goto, ARRAY
```

```
;
; =====>> DESTINATION IS STRUCTURE OR ARRAY OF STRUCTURES.
;
STRUCTURE:
for c = 0,dcount-1 do begin
  for i=0,n_tags(dest)-1 do begin
    sz2 = size(dest(c).(i))
;    count = sz2(sz2(0)+2)
  ; Use temporary so that temp is passed by reference,
  ; so it can be changed by the recursive readb.
  ; If it is passed as the expression dest(c).(i), dest(c).(i)
  ; will be unchanged after the call
    temp = dest(c).(i)
  ; call readb recursively for each simple type or nested struct
  ; use optional parameter for offset in recursive call
    readb, source, temp, offset
  ; use temporary here to avoid making a copy
    dest(c).(i) = temporary(temp)
;    offset = offset + count
    endfor
  endfor
return
;
; =====>> DESTINATION IS SCALAR, so grab a value at offset.
;
SCALAR:
case dtype of
  1:begin
    dest(0) = byte(source,offset)  ; BYTE
    offset = offset + 1
    end
  2:begin
    dest(0) = fix(source,offset)  ; INTEGER
    offset = offset + 2/bp(stype)
    end
  3:begin
    dest(0) = long(source,offset)  ; LONG
    offset = offset + 4/bp(stype)
    end
  4:begin
    dest(0) = float(source,offset)  ; FLOAT
    offset = offset + 4/bp(stype)
    end
  5:begin
    dest(0) = double(source,offset)  ; DOUBLE
    offset = offset + 8/bp(stype)
    end
  6:begin
```

```
    dest(0) = complex(source,offset)  ; COMPLEX
    offset = offset + 8/bp(stype)
    end
  7:begin
    dest(0) = string(source,offset)  ; STRING
    offset = offset + strlen(source(offset))
    end
  else: message,'Invalid type for destination.'
  endcase
return
;
; =====>> DESTINATION IS ARRAY: MAKE A STRING TO EXECUTE.
;
ARRAY:
dims = ''
for i=1,szd(0) do dims = dims + ','+strtrim(szd(i),2)
conv = ['byte','fix','long','float','double','complex','string']
exestring = 'dest(*)='+conv(dtype-1)+'(source,offset'+dims+')'
;print,exestring
if not execute(exestring) then $
  message,'Error filling destination array.'
offset = offset + nbytes(dest)/bp(stype)
return
end

--------------4C447A4A12D4--
```