Subject: Re: bytarr type conversion/structures
Posted by Martin Schultz on Mon, 28 Sep 1998 07:00:00 GMT
View Forum Message <> Reply to Message

Kevin Ivory wrote:
>
> Jacobus Koster wrote:
> [...]

> Looks like even "pro DF" is going to learn something today.
>
> It is almost always possible to do type conversions without writing
> into a dummy file and reading it again. The equivalent of Fortran
> internal files are IDL strings. So you will have to look into the READS
> procedure. Start off with the following lines:
>
> header_bytes = bytarr(2048, /nozero)
> openr, lun, /get_lun, image_file
> readu, lun, header_bytes
> free_lun, lun ; "forget forever about the file"
> header_string = string(header_bytes)
> ; read 100 descriptor structures from header_string
> header_structures = ({type:0,length:0,offset:0})[100]
> reads, header_string, header_structures
> ; now read the data with formatted reads
>
> I don't know about the pointer part (deleted from original message).
>
> Cheers,
> Kevin
> --

This is very nice, Kevin. Maybe I can contribute another 2 cents to this problem: As DF (in my definition about the only pro around) pointed out, there is a contradiction between "forget forever about the file" and retrieve the data later. Yet, I have recently written a few routines that achieve almost that. What I needed to do sounds somewhat similar: "parse" a huge file for its header structures *once* (there are several spread out over the file in our case), then store data pointers (for point_lun) and access the data as soon as it is needed. The whole thing is too complicated and long to "give away" on this newsgroup, but I will eventually place it on my web page (after some more testing), and I will give a few hints here:

* I use the logical file unit as a relational link between the data files and some "datainfo" structures that describe the contents and dimensions of the data as well as "where to find it" (the file pointer). For this, you need to avoid get_lun, because you must make sure that you

can re-open your files with the same unit numbers if they have been closed by some unaware user. I attach a little function GET_FREELUN that serves this purpose.
* When the user openes another file, I must first check whether this file had been opened before, so that it is not necessary to open and parse it again.
* Since I want access to all previously opened files at some point, I must use a global common block that contains a pointer to my array of datainfo and fileinfo structures (fileinfo stores filenames, logical units and some more stuff).
* As soon as I want to access some data that I select from the information stored in datainfo (in our case of a global atmospheric model, this can be a certain species at a certain timestep for example), I use the LUN field of datainfo to get the associated file name, make sure the file is open, set the file pointer, and read the data. The data is stored within the datainfo structure (referenced from a pointer), so I only have to read it once.

Hope this shows a way and helps to see a little clearer,
Martin.

PS: Note the difference between GET_LUN and GET_FREELUN is that GET_FREELUN first tests the information that IDL has available on opened files (help,/files) before it returns a free unit number. Hence, you can mix "explicit" assignments with "automatic" ones. In order to avoid conflicts with GET_LUN, unit numbers must be smaller than 100 (but that leaves you with 100 units instead of the standard 28!

--
-- -------
Dr. Martin Schultz
Department for Engineering&Applied Sciences, Harvard University
109 Pierce Hall, 29 Oxford St., Cambridge, MA-02138, USA

phone: (617)-496-8318
fax : (617)-495-4551

e-mail: mgs@io.harvard.edu
Internet-homepage: http://www-as.harvard.edu/people/staff/mgs/
-- -------

```
;------------------------------------------------------------ --
;+
; NAME:
;       GET_FREELUN (function)
;
; PURPOSE:
```

```
;       Return next available logical unit number. Unlike
;       the internal GET_LUN procedure, this function is not
;       restricted to unit numbers above 100, and it will
;       detect any blocked unit number.
;
; CATEGORY:
;       I/O tools
;
; CALLING SEQUENCE:
;       lun = GET_FREELUN([LUN])
;
; INPUTS:
;       none
;
; KEYWORD PARAMETERS:
;       none
;
; OUTPUTS:
;       The lowest available logical unit number. This number is
;       also returned in the LUN parameter for later use.
;
; SUBROUTINES:
;
; REQUIREMENTS:
;
; NOTES:
;
; EXAMPLE:
;       openw,get_freelun(lun),filename
;
; MODIFICATION HISTORY:
;       mgs, 17 Sep 1998: VERSION 1.00
;
;-
```

```
function get_freelun,lun

   help,/files,output=list

   lun = 1

   ; at least one file open
   ; find lowest available unit number
   if (n_elements(list) gt 1) then begin

      ; maximum allowed number of open files exceeded?
      if (n_elements(list) gt  99) then $
         message,'Cannot handle any more open files'

      ; extract numbers and compare to expectation
      for i=1,n_elements(list)-1 do begin
         usedlun = fix(strmid(list[i],0,3))
         if (usedlun gt i) then begin
            lun = i
            return,lun   ; this one's free
         endif
      endfor
      ; next free unit is greater than all used ones
      lun = i
      return,lun

   endif else $    ; no file opened
     return,lun


end
```

File Attachments

1) get_freelun.pro, downloaded 101 times