Subject: Re: bytarr type conversion/structures
Posted by davidf on Sat, 26 Sep 1998 07:00:00 GMT
View Forum Message <> Reply to Message

Jacobus Koster (jkdj@ix.netcom.com) writes:

> Here's one for the pro's (and DF :-), at least, I think, but I'm no
> (big) expert.

Being separated from the "pro's" only serves to accentuate
my decreasing stature as a programmer to be reckoned with.
I am simply going to have to *stop* writing those damn FOR
loops in example programs I publish on this newsgroup. :-(

> Suppose my image files have a standard format with a standard 2048 byte
> header.
> This header consists of 10 bytes of extraneous information followed by a
> descriptor part and a data part.
> The descriptor part consists of about 100 descriptor fields, each of
> which is a record/structure consisting of 3 short ints : key.type,
> key.length, and key.offset.
> [...]
> I would like to read these headers as byte arrays of 2048 bytes, and
> then forget forever about the file I got them from. From this byte
> array, I want to read the 100 descriptor structures into a 100-element
> structure array, with the structure elements described by :
> {type:0,length:0,offset:0}. And then, I would like to access the actual
> data itself, of course.

Does it strike you that forgetting "forever about the file" and
accessing "the actual data itself" might be mutually exclusive
goals in this case? :-)

> Is it possible in IDL to do this kind of type conversion, WITHOUT first
> writing the byte array out again into a dummy file and using an - albeit
> very beautiful - ASSOC variable or something like that ?

No. Given the constraints you are putting on the problem (I suspect
needlessly, but that is up to you to determine), I would say it is
impossible to do this without writing a FOR loop (Oh boy!). What
you want to do, it seems, is read a structure array from a byte
array. You can certainly read integers or long integer, or even
floats, from a byte array, but you can't read structures without
doing it in some kind of loop.

> All I know is : it should involve pointers (if only because the type of
> the actual data varies). In addition to this, I would like to avoid
> FOR-loops, and explicitly calculating integers as 256*byte1 + byte2

> strikes me as kinda crude.

I suspect, although I can't prove it, that your thinking about
this problem is not crystal clear yet. Tell me again *why* you
want to do this? And why you want to do it in this "throw it out"
sort of way?

Cheers,

David

---
David Fanning, Ph.D.
Fanning Software Consulting
E-Mail: davidf@dfanning.com
Phone: 970-221-0438, Toll-Free Book Orders: 1-888-461-0155
Coyote's Guide to IDL Programming: http://www.dfanning.com/

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
;
;-
; Copyright (C) 1998, Martin Schultz, Harvard University
; This software is provided as is without any warranty
; whatsoever. It may be freely used, copied or distributed
; for non-commercial purposes. This copyright notice must be
; kept with any copy of this software. If this software shall
; be used commercially or sold as part of a larger package,
; please contact the author to arrange payment.
; The copyright is granted if this program becomes part of the
; IDL distribution.
; Bugs and comments should be directed to mgs@io.harvard.edu
; with subject "IDL routine get_freelun"
 ;------------------------------------------------------------ --


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

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

Kevin Ivory (Kevin.Ivory@linmpi.mpg.de) writes:

> Looks like even "pro DF" is going to learn something today.

Indeed. In fact, hardly a day goes by that I don't learn
something from this newsgroup. That's why I like it so
much.

And I would have learned this tip too if I had taken 10
minutes to read that IDL Programming Techniques book by
ol' whatshisname. :-(

Cheers,

David

David Fanning, Ph.D.
Fanning Software Consulting
E-Mail: davidf@dfanning.com
Phone: 970-221-0438, Toll-Free Book Orders: 1-888-461-0155
Coyote's Guide to IDL Programming: http://www.dfanning.com/

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

Jacobus Koster wrote:
> I would like to read these headers as byte arrays of 2048 bytes, and
> then forget forever about the file I got them from. From this byte
> array, I want to read the 100 descriptor structures into a 100-element
> structure array, with the structure elements described by :
> {type:0,length:0,offset:0}. And then, I would like to access the actual
> data itself, of course.
> Is it possible in IDL to do this kind of type conversion, WITHOUT first

> writing the byte array out again into a dummy file and using an - albeit
> very beautiful - ASSOC variable or something like that ?

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
--

```
  Kevin Ivory                          Tel: +49 5556 979 434
  Max-Planck-Institut fuer Aeronomie    Fax: +49 5556 979 240
  Max-Planck-Str. 2                     mailto:Kevin.Ivory@linmpi.mpg.de
D-37191 Katlenburg-Lindau, GERMANY      http://www.gwdg.de/~kivory2/
```