Subject: Re: read a C written binary file with IDL
Posted by manodeep@gmail.com on Fri, 10 Feb 2012 04:56:23 GMT
View Forum Message <> Reply to Message

Hi,

This is because C pads the structure to produce alignments. Under 'normal' operations, you would expect MyStruct to be 20 bytes, however, if you do a sizeof(struct MyStruct), you will probably see that the size is 24. (And you can enable the warning for gcc by using the compile time option -Wpadded).

In general, the padding is compiler specific -- so there is no standard way of reading in those binary files into IDL/other codes. The best bet would be to write out the individual fields of the structure and then read them back into IDL.

HTH,
Manodeep

On Feb 9, 9:44 pm, bing999 <thibaultga...@gmail.com> wrote:
> Hi,
>
> I am having a problem with reading a C written binary file with IDL.
> It may come from differences of type definitions between C and IDL but
> I could not really figure out from Google...
>
> In C, it writes a structure containing the following variable types:
>
> struct MyStruct
> {
> int a;
> long long b;
> int c;
> float d;
>
> };
>
> Then, in IDL, I read this with:
>
> MyStruct = {$
> a : 0L, $
> b : 0LL, $
> c : 0L, $
> d : 0.0 $
> }
>
> openr, 1, filename, /SWAP_IF_BIG_ENDIAN

> readu, 1, MyStruct
> close, 1
>
> but this gives me wrong values.
>
> Did I miss something about the type conversion??
>
> If someone could please clarify this, it would really help!
> Thanks !

Subject: Re: read a C written binary file with IDL
Posted by Thibault Garel on Fri, 10 Feb 2012 05:08:01 GMT
View Forum Message <> Reply to Message

Hi Manodeep,

Thanks for your reply.
It turns out that a Python script can read the file correctly:

```
import numpy as np

 MyStruct = [                              # define
MyStruct
        ('a'                ,
np.int32),
        ('b'                ,
np.int64),
        ('c'                , np.int32),
        ('d'                , np.float32)
        ]

f1 = open(filename, 'rb')  # Open the file
S1 = np.fromfile(f1, MyStruct)  # read MyStruct
```

However, I have to use IDL (and I do not really know much about
Python...)

Do you have any idea why Python can handle it and not IDL?

cheers

Subject: Re: read a C written binary file with IDL
Posted by Paul Van Delst[1] on Fri, 10 Feb 2012 14:09:14 GMT

Have you tried ASSOC? It has a PACKED keyword you can play with:

<quote>
PACKED
When ASSOC is applied to structures, the default action is to map the actual definition of the structure for the current
machine, including any holes required to properly align the fields. (IDL uses the same rules for laying out structures
as the C language). If the PACKED keyword is specified, I/O using the resulting variable instead works in the same
manner as READU and WRITEU, and data is moved one field at a time and there are no alignment gaps between the fields.
</quote>

So you might want to experiment with ASSOC and /PACKED, e.g.

```
mystruct={a:0L, b:0LL, c:0L,d:0.0}
openr, 1, filename, /swap_if_big_endian
a = assoc(1, {a:0L, b:0LL, c:0L,d:0.0})
  or
a = assoc(1, {a:0L, b:0LL, c:0L,d:0.0},/packed)
help, a[0]
```

cheers,

paulv

bing999 wrote:
> Hi,
>
> I am having a problem with reading a C written binary file with IDL.
> It may come from differences of type definitions between C and IDL but
> I could not really figure out from Google...
>
> In C, it writes a structure containing the following variable types:
>
> struct MyStruct
> {
> int a;
> long long b;
> int c;
> float d;
> };
>
> Then, in IDL, I read this with:
>

> MyStruct = {$
> a : 0L, $
> b : 0LL, $
> c : 0L, $
> d : 0.0 $
> }
>
> openr, 1, filename, /SWAP_IF_BIG_ENDIAN
> readu, 1, MyStruct
> close, 1
>
>
> but this gives me wrong values.
>
> Did I miss something about the type conversion??
>
> If someone could please clarify this, it would really help!
> Thanks !
>
>
>
>
>

Subject: Re: read a C written binary file with IDL
Posted by Bill Nel on Fri, 10 Feb 2012 15:12:35 GMT
View Forum Message <> Reply to Message

On Feb 9, 11:56 pm, Manodeep Sinha <manod...@gmail.com> wrote:
> Hi,
>
> This is because C pads the structure to produce alignments. Under
> 'normal' operations, you would expect MyStruct to be 20 bytes,
> however, if you do a sizeof(struct MyStruct), you will probably see
> that the size is 24. (And you can enable the warning for gcc by using
> the compile time option -Wpadded).

The N_TAGS() function has LENGTH and DATA_LENGTH keywords:

```
IDL> print, n_tags(mystruct, /length)
       24
IDL> print, n_tags(mystruct, /data_length)
       20
```

I mention it because one might not think to look at the N_TAGS
function

for this sort of information.

--Wayne

Subject: Re: read a C written binary file with IDL
Posted by Paul Van Delst[1] on Fri, 10 Feb 2012 16:02:07 GMT
View Forum Message <> Reply to Message

Cool! I did not know that.

cheers,

paulv

Bill Nel wrote:
> On Feb 9, 11:56 pm, Manodeep Sinha <manod...@gmail.com> wrote:
>> Hi,
>>
>> This is because C pads the structure to produce alignments. Under
>> 'normal' operations, you would expect MyStruct to be 20 bytes,
>> however, if you do a sizeof(struct MyStruct), you will probably see
>> that the size is 24. (And you can enable the warning for gcc by using
>> the compile time option -Wpadded).
>
>
> The N_TAGS() function has LENGTH and DATA_LENGTH keywords:
>
> IDL> print, n_tags(mystruct, /length)
> 24
> IDL> print, n_tags(mystruct, /data_length)
> 20
>
> I mention it because one might not think to look at the N_TAGS
> function
> for this sort of information.
>
> --Wayne

Subject: Re: read a C written binary file with IDL
Posted by Thibault Garel on Tue, 21 Feb 2012 01:49:32 GMT
View Forum Message <> Reply to Message

Hi,

sorry to reply so late, but I was away recently.

First, as the issue seems to be more complicated than i first thought,
i have to say that the structure i have to read is bigger than what I
wrote previously just to make an example.

It actually contains (in the right order):
int x 1
long long x 1
int x 5
float x 9
int x 1
float x 24

that is, 7 int, 33 float and 1 long long (written in C).

When I look at the IDL commands you suggest, i get:

  print,n_tags(mystruct,/length) 176

  print,n_tags(mystruct,/data_length) 172

So you were right.

Now, the question is "How to handle that?? " :)

I inserted an extra int (0L) in the structure of my IDL reading
routine in second position of the structure (just before the long
long), and it gives results coherent with what one gets with a correct
Python routine. However, I am not quite satisfied with this
solution...

Does anyone can think of a better way to proceed?

Thanks !!

>
>
>
> Bill Nel wrote:
>> On Feb 9, 11:56 pm, Manodeep Sinha <manod...@gmail.com> wrote:
>>> Hi,
>
>>> This is because C pads the structure to produce alignments. Under
>>> 'normal' operations, you would expect MyStruct to be 20 bytes,
>>> however, if you do a sizeof(struct MyStruct), you will probably see

>>> that the size is 24. (And you can enable the warning for gcc by using
>>> the compile time option -Wpadded).
>
>> The N_TAGS() function has LENGTH and DATA_LENGTH keywords:
>
>> IDL> print, n_tags(mystruct, /length)
>> 24
>> IDL> print, n_tags(mystruct, /data_length)
>> 20
>
>> I mention it because one might not think to look at the N_TAGS
>> function
>> for this sort of information.
>
>> --Wayne
