Subject: Returning A Variable Length struct to IDL from C
Posted by K Banerjee on Mon, 05 Nov 2001 19:05:09 GMT
View Forum Message <> Reply to Message

Folks,

I wrote a DLM, following the examples in "Calling C From IDL"
by Mr. Ronn Kling. Here's the simplified layout of what I
am doing:

In the file idl_vbio.h, I have the following struct:

```
typedef struct
{
  IDL_STRING vers;
  IDL_STRING *userHeader;
  IDL_LONG byteOffSet;
} vbHeader;
```

I need to read the text header of a data file and then populate
the vbHeader struct. However, there will be a varying number of
text lines that comprise the userHeader, i.e., 2 data files
may not have the same number of "user header" text lines.

In the function that reads the text header, named
idlvbio_get_cube_header(), I have:

```
int userHeaderArrayLength = some_function_that_returns_int();
```

Later on in this function, I have:

```
IDL_STRUCT_TAG_DEF vbHeaderTags[] =
{
   {"VERS", 0, (void *) IDL_TYP_STRING},
   {"USERHEADER", dims_user_header, (void *) IDL_TYP_STRING},
   {"BYTEOFFSET", 0, (void *) IDL_TYP_LONG},
   {0}
};
```

where

```
static IDL_LONG dims_user_header[] = {1, userHeaderArrayLength};
```

Continuing in this function, I have:

```
  typedef struct
  {
    IDL_STRING vers;
```

```
    IDL_STRING userHeader[userHeaderArrayLength];
    IDL_LONG byteOffSet;
  } vbHeaderActual;
```

The difference between the above struct and the first struct
(found in the C include file) is that the second field in the
first struct is a pointer to IDL_STRING while in the
above struct, the second field is an array of type IDL_STRING of
length userHeaderArrayLength.

An instance of the vbHeader struct is created, called theHeader
and then the following line is executed:

```
theHeader->userHeader = new IDL_STRING[userHeaderArrayLength];
```

The function then continutes to populate the fields of theHeader.

The function then instantiates a struct of type vbHeaderActual,
called theHeaderActual.

The function then copies the fields from theHeader
to theHeaderActual. So far, so good.

The next two lines are used to create the return value to IDL:

```
void *psDef = IDL_MakeStruct(NULL, vbHeaderTags);
IDL_VPTR ivReturn = IDL_ImportArray(1, ilDims, IDL_TYP_STRUCT,
(UCHAR *) theHeaderActual, releaseMemory, psDef);
```

where releaseMemory is the function:

```
extern "C" void releaseMemory(UCHAR *ptr)
{
  deleteMem(ptr);
} // extern "C" void releaseMemory(UCHAR *ptr)
```

and ilDims is:

```
ilDims[0] = 1;
```

The return line is:

```
return ivReturn;
```

Now assume that I need to read the header of 2 data files,
file1 and file2. Further assume that file1 has 10 user header
lines and file2 has 15 user header lines. So from IDL,
header's what it looks like:

IDL> h = idlvbio_get_cube_header('file1')
IDL> h = idlvbio_get_cube_header('file2')

The second IDL line causes a core dump due to a sementation
fault occurring in IDL_MemFree():

#0 0x4008cf58 in IDL_MemFree () at ../../gcc-2.95.2/gcc/cp/exception.cc:343
343   ../../gcc-2.95.2/gcc/cp/exception.cc: No such file or directory.

What I think happens is that upon the first call to the function
idlvbio_get_cube_header(), a certain amount of memory is allocated
to the IDL variable h. Now the second call to idlvbio_get_cube_header()
does not cause IDL to delete the memory already allocated to h
and then reallocate memory to h. More memory is now needed by h
since there are 15 user header lines in file2.

However, using a second IDL variable, h2, overcomes this
problem:

IDL> h = idlvbio_get_cube_header('file1')
IDL> h2 = idlvbio_get_cube_header('file2')

What I'd like to be able to do is reuse the IDL variable h.

I can not resort to using the IDL procedure delvar since
delvar is only available at the IDL prompt and can not be used
from IDL functions and procedures (I need to wrap
idlvbio_get_cube_header() inside IDL functions and procedures).

I will need to return a struct with a variable length array as
its second field (the userHeader[] array).

Any suggestions on how I can do what I need to do?

My environment is:

IDL 5.3
gcc 2.95.2
Red Hat Linux 6.2
Kernel 2.2.19
I am using linkimage to load the DLM.

Thanks.

K. Banerjee

Subject: Re: Returning A Variable Length struct to IDL from C
Posted by Craig Markwardt on Tue, 06 Nov 2001 02:16:31 GMT
View Forum Message <> Reply to Message

Greetings--

K Banerjee <kbanerje2@home.com> writes:

> The next two lines are used to create the return value to IDL:
>
> void *psDef = IDL_MakeStruct(NULL, vbHeaderTags);
> IDL_VPTR ivReturn = IDL_ImportArray(1, ilDims, IDL_TYP_STRUCT,
> (UCHAR *) theHeaderActual, releaseMemory, psDef);

Here is your first problem. IDL_ImportArray only works when you import static data, not dynamically allocated. Basically ImportArray only works once per piece of memory per IDL session.

> IDL_STRUCT_TAG_DEF vbHeaderTags[] =
> {
> {"VERS", 0, (void *) IDL_TYP_STRING},
> {"USERHEADER", dims_user_header, (void *) IDL_TYP_STRING},
> {"BYTEOFFSET", 0, (void *) IDL_TYP_LONG},
> {0}
> };

Here (in my opinion) is your second problem: trying to do something too complicated within a DLM function. I personally think that while it's not impossible, trying to manipulate complex data structures within a DLM are very *close* to impossible and are really unnecessary.

In my opinion, the proper approach is to have your DLM function do the dirty work, and construct only the simplest of data, and then have a wrapper function, written in IDL itself, compose it into something more complex. What that means in this case, is to use positional or keyword arguments to return three variables (VERS, USERHEADER [an array of IDL strings], and BYTEOFFSET). Then in your wrapper function you can easily compose these variables into a structure.

Even returning simple data values is not particularly simple, at least not in my experience. Be sure you test your program for memory leaks afterwards.

Craig

--
---------- ----------

Craig B. Markwardt, Ph.D. EMAIL: craigmnet@cow.physics.wisc.edu
Astrophysics, IDL, Finance, Derivatives | Remove "net" for better response
---

Subject: Re: Returning A Variable Length struct to IDL from C
Posted by Nigel Wade on Tue, 06 Nov 2001 12:22:31 GMT
View Forum Message <> Reply to Message

K Banerjee wrote:

>
> Folks,
>
> I wrote a DLM, following the examples in "Calling C From IDL"
> by Mr. Ronn Kling. Here's the simplified layout of what I
> am doing:
>

It appears that you are using C++, and I don't really know C++ so I won't attempt to answer in C++. I do this in C so it should be perfectly possible to do it in C++.

> In the file idl_vbio.h, I have the following struct:
>
> typedef struct
> {
> IDL_STRING vers;
> IDL_STRING *userHeader;
> IDL_LONG byteOffSet;
> } vbHeader;
>
> I need to read the text header of a data file and then populate
> the vbHeader struct. However, there will be a varying number of
> text lines that comprise the userHeader, i.e., 2 data files
> may not have the same number of "user header" text lines.
>
> In the function that reads the text header, called
> idlvbio_get_cube_header(), I have:
>
> int userHeaderArrayLength = some_function_that_returns_int();
>
> Later on in this function, I have:
>
> IDL_STRUCT_TAG_DEF vbHeaderTags[] =
> {
> {"VERS", 0, (void *) IDL_TYP_STRING},

> {"USERHEADER", dims_user_header, (void *) IDL_TYP_STRING},
> {"BYTEOFFSET", 0, (void *) IDL_TYP_LONG},
> {0}
> };
>
> where
>
> static IDL_LONG dims_user_header[] = {1, userHeaderArrayLength};
>
> Continuing in this function, I have:
>
> typedef struct
> {
> IDL_STRING vers;
> IDL_STRING userHeader[userHeaderArrayLength];
> IDL_LONG byteOffSet;
> } vbHeaderActual;
>
> The difference between the above struct and the first struct
> (found in the C include file) is that the second field in the
> first struct is a pointer to IDL_STRING while in the
> above struct, the second field is an array of type IDL_STRING of
> length userHeaderArrayLength.
>
> An instance of the vbHeader struct is created, called theHeader,
> and then the following line is executed:
>
> theHeader->userHeader = new IDL_STRING[userHeaderArrayLength];
>
> The function then continutes to populate the fields of theHeader.
>

How do you do that? The IDL_STRING entries need to use IDL_StrStore to
make sure a copy of the string is stored into the IDL variable.

> The function then instantiates a struct of type vbHeaderActual,
> called theHeaderActual.
>
> The function then copies the fields from theHeader
> to theHeaderActual. So far, so good.
>
> The next two lines are used to create the return value to IDL:
>
> void *psDef = IDL_MakeStruct(NULL, vbHeaderTags);
> IDL_VPTR ivReturn = IDL_ImportArray(1, ilDims, IDL_TYP_STRUCT,
> (UCHAR *) theHeaderActual, releaseMemory, psDef);
>

I've never actually tried using ImportArray for variables containing
strings.
I don't know what happens to the memory allocated by IDL_StrStore. This is
allocated by IDL, so IDL should free it when the variable disappears. But
IDL_ImportArray tells IDL not to free the memory associated with the
variable but call your routine instead.

> where releaseMemory is the function:
>
> extern "C" void releaseMemory(UCHAR *ptr)
> {
> deleteMem(ptr);
> } // extern "C" void releaseMemory(UCHAR *ptr)

What does deleteMem do?

>
> and ilDims is:
>
> ilDims[0] = 1;
>
> The return line is:
>
> return ivReturn;
>
> Now assume that I need to read the header of 2 data files,
> file1 and file2. Further assume that file1 has 10 user header
> lines and file2 has 15 user header lines. So from IDL,
> here's what it looks like:
>
> IDL> h = idlvbio_get_cube_header('file1')
> IDL> h = idlvbio_get_cube_header('file2')
>
> The second IDL line causes a core dump due to a segmentation
> fault occurring in IDL_MemFree():
>
> #0 0x4008cf58 in IDL_MemFree () at
> #../../gcc-2.95.2/gcc/cp/exception.cc:343
> 343   ../../gcc-2.95.2/gcc/cp/exception.cc: No such file or directory.
>
> What I think happens is that with the first call to the function
> idlvbio_get_cube_header(), a certain amount of memory is allocated
> to the IDL variable h. Now the second call to idlvbio_get_cube_header()
> does not cause IDL to delete the memory already allocated to h
> and then reallocate memory to h.

It should do. As soon as IDL stores a new value into a variable it cleans
up the memory held by that variable. In your case it will call your cleanup

function releaseMemory.

> More memory is now needed by h
> since there are 15 user header lines in file2.

That should be allocated by your DLM.

>
> However, using a second IDL variable, h2, overcomes this
> problem:
>
> IDL> h = idlvbio_get_cube_header('file1')
> IDL> h2 = idlvbio_get_cube_header('file2')

What's more likely is that somehow your code written outside the area of memory it allocated and corrupted the malloc arena. When you come to free the data you allocated it has some garbage in the arena resulting in a core dump.

>
> Any suggestions on how I can do what I need to do?
>
>

Here's some code which I use to return a structure containing a variable length array of strings.

Unfortunately I don't have a simple example so I'll provide a potted version of what I do. This is a stipped down version of a DLM to return various info about an X display - I've removed all but the code to handle the array of strings in a structure for simplicity.

```
    /* dims for the structure containing an array of strings */
    static IDL_LONG mask_dims[IDL_MAX_ARRAY_DIM];
    /* dims for the returned structure */
    static IDL_LONG screen_dims[IDL_MAX_ARRAY_DIM];

    /* the structure tags which contains an array of strings */
    IDL_STRUCT_TAG_DEF screen_tags[] = {
        { "EVENTS_MASKED", mask_dims, (void *)IDL_TYP_STRING },
        { 0 },
    };


    void *screen_s;
    char *screen_data;
    IDL_LONG tag_offset;
    IDL_STRING *event_masks;
```

```
    IDL_VPTR screen_var;

    int event_mask;

    /* don't worry about this, it's an X call to get the even mask */
    event_mask = EventMaskOfScreen(s);

    /* set the number of strings to be stored in the structure */
    mask_dims[0] = 1;
    mask_dims[1] = nbits(event_mask);

    /* create the structure with the strings in it */
    screen_s = IDL_MakeStruct(0, screen_tags);

    screen_dims[0] = 1;

    /*
     * create the final structure and allocate memory
     * I always use temp structures so that they are deallocated if
     *  not used
     */
    screen_data = (char *)IDL_MakeTempStruct(screen_s, 1, screen_dims,
                        &screen_var, 0);


    /* find where the array of strings is stored in the structure */
    tag_offset = IDL_StructTagInfoByName(screen_s, "EVENTS_MASKED",
                     IDL_MSG_LONGJMP, NULL);
    /* set an IDL_STRING pointer to that location. */
    event_masks = (IDL_STRING *)(screen_data+tag_offset);
    /* store a string into each IDL_STRING in the array */
    for ( i = 0; i<mask_dims[1]; i++)
       IDL_StrStore(event_masks++, some_string_to_store );

    return screen_var;
}
```

--

Nigel Wade, System Administrator, Space Plasma Physics Group,
University of Leicester, Leicester, LE1 7RH, UK
E-mail : nmw@ion.le.ac.uk
Phone : +44 (0)116 2523568, Fax : +44 (0)116 2523555

Subject: Re: Returning A Variable Length struct to IDL from C
Posted by Nigel Wade on Tue, 06 Nov 2001 13:36:21 GMT

Craig Markwardt wrote:

>
> Greetings--
>
> K Banerjee <kbanerje2@home.com> writes:
>
>> The next two lines are used to create the return value to IDL:
>>
>> void *psDef = IDL_MakeStruct(NULL, vbHeaderTags);
>> IDL_VPTR ivReturn = IDL_ImportArray(1, ilDims, IDL_TYP_STRUCT,
>> (UCHAR *) theHeaderActual, releaseMemory, psDef);
>
> Here is your first problem. IDL_ImportArray only works when you
> import static data, not dynamically allocated. Basically ImportArray
> only works once per piece of memory per IDL session.

Really?. I run code which calls IDL_ImportArray many 1000s of times within
a session, all with malloc'd data and have not had any problems.

Why do you think it can only be used once per session, with static data?

>
>> IDL_STRUCT_TAG_DEF vbHeaderTags[] =
>> {
>> {"VERS", 0, (void *) IDL_TYP_STRING},
>> {"USERHEADER", dims_user_header, (void *) IDL_TYP_STRING},
>> {"BYTEOFFSET", 0, (void *) IDL_TYP_LONG},
>> {0}
>> };
>
> Here (in my opinion) is your second problem: trying to do something
> too complicated within a DLM function. I personally think that while
> it's not impossible, trying to manipulate complex data structures
> within a DLM are very *close* to impossible and are really
> unnecessary.

No, no, no! You're missing the entire purpose of DLMs.
It's the realm of DLMs to do those things which are too complicated to do
in IDL. ;-)

--

Nigel Wade, System Administrator, Space Plasma Physics Group,
University of Leicester, Leicester, LE1 7RH, UK

E-mail : nmw@ion.le.ac.uk
Phone : +44 (0)116 2523568, Fax : +44 (0)116 2523555

Subject: Re: Returning A Variable Length struct to IDL from C
Posted by Richard Younger on Tue, 06 Nov 2001 15:58:25 GMT
View Forum Message <> Reply to Message

K Banerjee wrote:
>
> Folks,
>
> I wrote a DLM, following the examples in "Calling C From IDL"
> by Mr. Ronn Kling. Here's the simplified layout of what I
> am doing:

[...]

> theHeader->userHeader = new IDL_STRING[userHeaderArrayLength];

[...]

> The next two lines are used to create the return value to IDL:
>
> void *psDef = IDL_MakeStruct(NULL, vbHeaderTags);
> IDL_VPTR ivReturn = IDL_ImportArray(1, ilDims, IDL_TYP_STRUCT,
> (UCHAR *) theHeaderActual, releaseMemory, psDef);
>
> where releaseMemory is the function:
>
> extern "C" void releaseMemory(UCHAR *ptr)
> {
> deleteMem(ptr);
> } // extern "C" void releaseMemory(UCHAR *ptr)
>

[...]

> The second IDL line causes a core dump due to a sementation
> fault occurring in IDL_MemFree():
>
> #0 0x4008cf58 in IDL_MemFree () at ../../gcc-2.95.2/gcc/cp/exception.cc:343
> 343 ../../gcc-2.95.2/gcc/cp/exception.cc: No such file or directory.
>
> What I think happens is that upon the first call to the function
> idlvbio_get_cube_header(), a certain amount of memory is allocated
> to the IDL variable h. Now the second call to idlvbio_get_cube_header()
> does not cause IDL to delete the memory already allocated to h

> and then reallocate memory to h. More memory is now needed by h
> since there are 15 user header lines in file2.

[...]

Hi, K.

I've had troubles in the past allocating and deallocating memory with IDL, so I wouldn't be too surprised if it's a memory issue. It looks to me like you're right, and the problems happen when you try to clean up. Does reassigning h (h=0) cause the same problem?

But I'm a bit puzzled by the 'No such file or directory' error. I haven't used gcc for a couple years, so I'm not confident about the nature of the error message you've included. One could interpret it as IDL_MemFree() is trying to call deleteMem() (or possibly releaseMemory()) and not finding it. I'm not familiar with deleteMem() as a library function. Where is it defined?

But maybe I'm reading the error message too literally, and the problem is the cleanup isn't being called properly or in the correct context. I never give IDL memory allocated by new or delete, as new and delete have been a source of troubles in the past. I never successfully got IDL_ImportArray() to work with memory from new, though I didn't try too hard.

In any case, a workaround, if deleteMem() is being called and working correctly, would be to use IDL_GetScratch() and go through the bother of traversing the IDL variable to get to your scratch memory. Since this particular snippet of code that creates the memory is IDL aware (i.e. export.h defined), it's not too hard.

Best,
Rich

--
Richard Younger

Subject: Re: Returning A Variable Length struct to IDL from C
Posted by Richard Younger on Tue, 06 Nov 2001 16:27:21 GMT
View Forum Message <> Reply to Message

Nigel Wade wrote:
>
> Really?. I run code which calls IDL_ImportArray many 1000s of times
> within a session, all with malloc'd data and have not had any
> problems.

>
> Why do you think it can only be used once per session, with static
> data?

It could be that IDL handles malloc() and free() just fine, but not new[] and delete[], which is what K. seems to be using (though I'm not sure about delete[]). I am not at all familiar with how new[] is implemented, but it could use malloc() or include garbage collection, or be radically different. From what I remember, I don't think it's in the C++ spec and so would depend on your compiler.

> No, no, no! You're missing the entire purpose of DLMs.
> It's the realm of DLMs to do those things which are too complicated to do
> in IDL. ;-)

Well, complicated DLM's are part of the fun, but I think Craig's point is that it's a little easier to manipulate IDL structures in, um, IDL. ;-) Though that code you posted should give anyone a good head start on passing structures with arrays of strings.

Best,
Rich

--
Richard Younger

Subject: Re: Returning A Variable Length struct to IDL from C
Posted by Craig Markwardt on Tue, 06 Nov 2001 16:28:16 GMT
View Forum Message <> Reply to Message

Nigel Wade <nmw@ion.le.ac.uk> writes:

> Craig Markwardt wrote:
>
>>
>> Greetings--
>>
>> K Banerjee <kbanerje2@home.com> writes:
>>
>>> The next two lines are used to create the return value to IDL:
>>>
>>> void *psDef = IDL_MakeStruct(NULL, vbHeaderTags);
>>> IDL_VPTR ivReturn = IDL_ImportArray(1, ilDims, IDL_TYP_STRUCT,
>>> (UCHAR *) theHeaderActual, releaseMemory, psDef);
>>
>> Here is your first problem. IDL_ImportArray only works when you
>> import static data, not dynamically allocated. Basically ImportArray

>> only works once per piece of memory per IDL session.
>
> Really?. I run code which calls IDL_ImportArray many 1000s of times within
> a session, all with malloc'd data and have not had any problems.
>
> Why do you think it can only be used once per session, with static data?

I spoke too soon. I didn't remember ImportArray having a "release"
argument. I stand corrected.

>> Here (in my opinion) is your second problem: trying to do something
>> too complicated within a DLM function. I personally think that while
>> it's not impossible, trying to manipulate complex data structures
>> within a DLM are very *close* to impossible and are really
>> unnecessary.
>
> No, no, no! You're missing the entire purpose of DLMs.
> It's the realm of DLMs to do those things which are too complicated to do
> in IDL. ;-)

Humor noted :-). That may be true, but data manipulation is one thing
that is easy to do in IDL itself. From experience I know that
populating IDL structures within a DLM can be very tricky indeed,
especially, as you mentioned, when there are complex data (i.e.,
strings) encapsulated within the structure.

Craig

--
---------- ----------
Craig B. Markwardt, Ph.D. EMAIL: craigmnet@cow.physics.wisc.edu
Astrophysics, IDL, Finance, Derivatives | Remove "net" for better response
---------- ----------

Subject: Re: Returning A Variable Length struct to IDL from C
Posted by Nigel Wade on Wed, 07 Nov 2001 10:00:37 GMT
View Forum Message <> Reply to Message

Richard Younger wrote:

> Nigel Wade wrote:
>>
>> Really?. I run code which calls IDL_ImportArray many 1000s of times
>> within a session, all with malloc'd data and have not had any
>> problems.
>>

>> Why do you think it can only be used once per session, with static
>> data?
>
> It could be that IDL handles malloc() and free() just fine, but not
> new[] and delete[], which is what K. seems to be using (though I'm not
> sure about delete[]). I am not at all familiar with how new[] is
> implemented, but it could use malloc() or include garbage collection, or
> be radically different. From what I remember, I don't think it's in the
> C++ spec and so would depend on your compiler.
>

Could be. I'm a C programmer (mostly) and used to using malloc/free, so no garbage collection, constructors or destructors to worry about.

I quite happily malloc memory, and use ImportArray to add it to an IDL variable. Then free it in the callback when the IDL variable is destroyed. I'm more comfortable with doing the clean up for myself - I'm never convinced in Java that all the objects I create are going to get cleaned up by the GC.

--

Nigel Wade, System Administrator, Space Plasma Physics Group,
University of Leicester, Leicester, LE1 7RH, UK
E-mail : nmw@ion.le.ac.uk
Phone : +44 (0)116 2523568, Fax : +44 (0)116 2523555