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