Subject: Re: Returning C struct to IDL
Posted by Nigel Wade on Mon, 13 May 2002 10:05:30 GMT
View Forum Message <> Reply to Message

K Banerjee wrote:

```
>
>  Folks,
>
>  Here's the platform info:
>  IDL version 5.3
>  RedHat Linux
>  g++ 2.95.2 and g++ 2.96
>
>  I have to get IDL to use a C function to read a data
>  file. This C function returns a structure to IDL. I am using IDL's
>  linkimage facility.
>
>  Four of the fields of the structure are arrays of type IDL_STRING
>  whose lengths depend on the data file to be read, so these lengths
>  are not known until run time.
>
>  Using g++ 2.95.2, this is how I had things set up:
>
>    static IDL_LONG dims_user_header[IDL_MAX_ARRAY_DIM];
>    dims_user_header[0] = 1;
>    dims_user_header[1] = userHeaderArrayLength;
>
>  (userHeaderArrayLength has been determined previously from the data
>  file.)
>
>    IDL_STRUCT_TAG_DEF headerTags[] =
>    {
>      {"VERS", 0, (void *) IDL_TYP_STRING},
>      {"DATATYPE", 0, (void *) IDL_TYP_INT},
>      {"T", 0, (void *) IDL_TYP_LONG},
>      {"X", 0, (void *) IDL_TYP_LONG},
>      {"Y", 0, (void *) IDL_TYP_LONG},
>      {"Z", 0, (void *) IDL_TYP_LONG},
>      {"VOXDIMS", dims_4, (void *) IDL_TYP_LONG},
>      {"USERHEADER", dims_user_header, (void *) IDL_TYP_STRING},
>  // The first of the 4 variable length arrays --^
>                    .
>                    .
>                    .
>      {0}
>    };
>
```

>
> Next I define a C struct to match the headerTags array:
>
> typedef struct
> {
> IDL_STRING vers;
> short dataType;
> IDL_LONG t;
> IDL_LONG x;
> IDL_LONG y;
> IDL_LONG z;
> IDL_LONG voxDims[4];
> IDL_STRING userHeader[userHeaderArrayLength];
> .
> .
> .
> } vbHeader;
>
> I populate all the fields of the C struct and then have the lines:
>
> void *psDef = IDL_MakeStruct(NULL, headerTags);
> IDL_LONG ilDims[IDL_MAX_ARRAY_DIM];
> ilDims[0] = 1;
> IDL_VPTR ivReturn = IDL_ImportArray(1, ilDims, IDL_TYP_STRUCT,
> (UCHAR *) theHeaderActual, NULL, psDef);
> return ivReturn;
>
> (The impelmentation of my C function to read the data file is based on
> "Calling C From IDL" by Mr. Ronn Kling.)
>
> Everything works as I expect. The problem arises when I try to
> compile this function with g++ 2.96. g++ 2.96 has a problem with
> the line:
>
> IDL_STRING userHeader[userHeaderArrayLength];
>
> The specific error message is:
>
> size of member `userHeader' is not constant
>
> So I decided to dynamically allocate memory for the userHeader[]
> array and changed the troublesome struct field to:
>
> IDL_STRING *userHeader;

That won't work, it doesn't match the definition in the TAG_DEF array.
That field in the tags array is an array of IDL_STRING. In the data you are
trying to match it to an array of *pointers* to IDL_STRING, which is going

to fail in an unpredictable way.

The bad news is that there is no way to include a dynamically allocated array within a structure and include it into the structure with IDL_ImportArray. You have two totally separate areas of memory, which cannot be used to create a single IDL structure.

>
> Later on I have:
>
> theHeader->userHeader = new IDL_STRING[userHeaderArrayLength];

That looks like C++ to me.
All bets are off.

>
> where theHeader is a pointer to the C struct. I then go ahead
> and populate the struct (I am confident I am populating the
> C struct properly), go through the same steps to return the struct
> to IDL. However, garbage ends up in the userHeader array of the
> IDL structure. It seems to me that I can not properly "marry"
> a C struct to an IDL structure when the C struct has pointer
> fields. I have not come across any example of where
> a C struct with pointer fields are returned to IDL.
>
>

Now the good news. You can achieve what you want, but just not in the way you are currently trying to do it.

The area of memory you want to return to IDL as a structure must be contiguous. It can either be "statically" allocated as you are doing, or dynamically allocated in one lump - but you cannot mix the two.

The method I use to include dynamic arrays in structs is to define the struct tags in the same way you have. Next I populate the dims arrays which define the size of the dynamic arrays, and make a structure with IDL_MakeStruct. Finally I create a temp structure using IDL_MakeTempStruct which allocates sufficient memory to hold the entire structure and returns a pointer to the allocated memory. You return this temporary structure to IDL after populating it.

Populating the allocated memory can be a bit tricky. Static members are easy, as you define a struct to map the allocated memory to the members, just as you have done, and set the values. The dynamic arrays are more fun and require some interesting use of pointer arithmetic. It's not for the faint hearted but does work.

The main point to remember is that the allocated memory in the IDL structure in your case will be multiple arrays of IDL_STRING. I prefer to set each array entry using IDL_StrStore, but if you didn't want to create a copy of the string you could set the pointer in the IDL_STRING to where your string already exists (and set the other members of the IDL_STRING structure), but this is probably not the best way as it relies on you knowing the internals of the IDL_STRING structure, which might change (and already has in the past).

--

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