Subject: Re: Returning a struct containing variable-length arrays
Posted by franzpx125 on Wed, 17 Sep 2008 21:01:24 GMT
View Forum Message <> Reply to Message

On 17 Set, 10:46, Nigel Wade <n...@ion.le.ac.uk> wrote:
> franzpx125 wrote:
>> Hi!
>
>> Within my DLM, I need to return a struct containing two arrays but the
>> length of the arrays is known only at run-time. Which is the correct
>> way to write the DLM code? I think I have to use IDL_MAX_ARRAY_DIM and
>> I defined:
>
>> static IDL_MEMINT my_tags_dims[] = { 1, IDL_MAX_ARRAY_DIM };
>
>> static IDL_STRUCT_TAG_DEF my_tags[] = {
>> { "X", my_tags_dims, (void *) IDL_TYP_DOUBLE },
>> { "Y", my_tags_dims, (void *) IDL_TYP_DOUBLE },
>> { 0 }
>> };
>
>> but I don't know how to fill the fields of this struct. Any help?
>
>> Thanks,
>> Brun Francesco
>
> You basically have two choices depending on whether you want to use your own
> memory storage allocated by your DLM code to hold the structure data, or have
> IDL allocate its own memory and copy the values into that. Each method has its
> own set of advantages and caveats/gotchas.
>
> If you use your own memory allocation then that memory has to persist after the
> DLM routine returns. This either means it has to be declared static (but since
> you want to use variable length arrays this isn't possible) or allocated by
> malloc. Using this method means that you are responsible for freeing the memory
> when IDL has finished with it. If it is simple throw away code which will only
> be used occasionally, and the function isn't called many times, and doesn't
> allocate much memory, you can ignore this but you *will* have a memory leak. To
> cure this you need to implement the callback routine, a function pointer to
> this is passed into the IDL_ImportArray function so that it can free whatever
> memory you allocate.
>
> If you create an IDL struct using your own memory then you don't have to worry
> about copying the data into the structure. If you use IDL_MakeStruct so that
> IDL allocates the storage (and handles de-allocation) then you do have to copy
> the data into that memory. This is a little tricky.
>
> In the first case, allocating your own storage, the sequence is to first

> determine how much storage is required. Then you set your dimensions arrays
> appropriately. Next you create the structure with IDL_MakeStruct and finally
> import your data into the structure using IDL_ImportArray. You are responsible
> for writing a callback function to free this memory when IDL has finished with
> the variable.
>
> To get IDL to handle the allocation the first two steps are the same. Then,
> rather than importing your own data you use IDL_MakeTempStruct to create a IDL
> temporary variable (which you can pass back to IDL). The tricky part is
> determining where to put your data. To do this you can use the function
> IDL_StructTagInfoByName, which returns the byte offset into the struct where
> the data for that tag is located. Then it's a case of some hairy pointer
> manipulation to write the data into the structure.
>
> --
> Nigel Wade

Thank you so much for your exhaustive explanation. As you can notice from a previous post I adopted the second solution and although results in slower code with greater memory occupation, I think it's cleaner without a callback function and more coherent with other routines included in my DLM. Thanks again.

Brun Francesco
