Subject: Re: Importing data from C/C++ to IDL when type is only known at runtime
Posted by Nigel Wade on Thu, 27 Apr 2006 12:09:50 GMT
View Forum Message <> Reply to Message

kathryn.ksm@gmail.com wrote:

> Hi Folks,
>
> I'm trying to write a DLM that uses existing C/C++ libaries to read
> data in from files that have a custom (and fairly complex) format.
> This is my first attempt at doing anything serious with IDL and I'm
> having a hard time figuring out what's possible and what isn't. So far
> most of what I've tried doesn't seem to be! So I could use some help.
>
> My existing C code reads selected portions of the data files into a set
> of arrays, where the data type is specified at run-time by the user.
> So, on the C side of things, I have an array of void pointers that
> point to dynamically allocated arrays with various types and sizes
> depending on what the user asked for.
>
> What I need to do is get this data into variables of appropriate types
> in an interactive session in IDL. Ideally, what I would like to do is
> to define and fill a set of nested structures in the DLM code at
> run-time, and return a single structure variable to the IDL session.
> This doesn't seem to be possible though. I can see how I could define
> such a structure at run-time using an array of IDL_STRUCT_TAG_DEFs and
> IDL_MakeStruct, but I don't see any way to fill that structure without
> creating an analogous structure in C in advance (which doesn't seem
> doable, but maybe I'm wrong about that).
>

It's possible, but tiresome and error prone.

If we take one of the examples from the External Ref Guide:

```
static IDL_MEMINT one = 1;
static IDL_MEMINT tag2_dims[] = { 3, 2, 3, 4};
static IDL_MEMINT tag3_dims[] = { 1, 10 };
static IDL_STRUCT_TAG_DEF s_tags[] = {
  { "TAG1", 0, (void *) IDL_TYP_LONG},
  { "TAG2", tag2_dims, (void *) IDL_TYP_FLOAT},
  { "TAG3", tag3_dims, (void *) IDL_TYP_STRING},
  { 0 }
};
typedef struct data_struct {
  IDL_LONG tag1_data;
  float tag2_data [4] [3] [2];
  IDL_STRING tag_3_data [10];
} DATA_STRUCT;
static DATA_STRUCT s_data;
void *s;
IDL_VPTR v;

/* Create the structure definition */
s = IDL_MakeStruct(0, s_tags);
/* Import the data area s_data into an IDL structure,
   note that no data are moved. */
v = IDL_ImportArray(1, &one, IDL_TYP_STRUCT,
              (UCHAR *) &s_data, 0, s);
```

As you've pointed out, it's fairly easy to create the s_tags array dynamically, and so create the IDL structure tags dynamically. The problem is the s_data structure, which would need to be defined in advance, at compile time.

However, the s_data structure is only a convenience. All that gets passed to IDL_ImportArray is a pointer to an area of static memory. The IDL_ImportArray doesn't care how you fill that area of memory, all it requires is that the right data is in each "slot". So s_data can be a pointer to an area of memory, allocated by malloc, which is then populated by writing the appropriate data values into it via some pointer. This is the bit that's messy and error prone.

The IDL variable, v, returned by IDL_ImportArray is actually using the area of memory which is passed into it (pointed to by s_data) so you can write whatever you want into that memory at any time. The trick is knowing where to write what. With a C structure which maps the IDL structure this is easy. With dynamic data you need to use a pointer and offset to determine where to write. You can handle this manually, by determining for yourself where within the data you need to write a particular element (highly error prone, but quick), or you can use the function IDL_StructTagInfoByName which will tell you the offset of particular tag name within the structure.

The big problem which now remains is to determine how much data needs to be allocated to store the dynamic structure. The only way I can think how to do this, in the generic case, is to first determine the number of tags using IDL_StructNumTags. Then get the location and type of the last structure member using IDL_StructTagInfoByIndex. Use the offset returned by that function, plus the size of the structure member returned in the var parameter, to calculate the total size of memory required.

--
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 2523548, Fax : +44 (0)116 2523555