Subject: nested structures in dlm
Posted by lbusoni on Fri, 12 Jan 2007 12:55:50 GMT
View Forum Message <> Reply to Message

HI Guru's of DLMs,

I need to convert a rather complex C++ structure to something easy to manage in IDL.
The data I have to convert are a list of dictionaries of (time,force) pairs:
For instance to object V000 belong 2 array ("time and "force") of n0 elements,
to object V001 belong 2 array ("time and "force") of n1 elements,
to object V00M belong 2 array ("time and "force") of nm elements.

I know the value of M and of the varius n0, n1, ..., nm only at run-time, i.e. I don't know in advance how many object I will get from the query, nor how many elements each one of these objects will contain.

Hope the description is clear enough.

I tested with a small function that assign to its argument a structure FOO of structures V000 to V004.
Each substructure contains a pair of array of different length:
v000.time and v000.force are dblarr[3], v001.time and v001.force are dblarr[4] and so on.

Here is the code:

```
static IDL_MEMINT times_dims[] = { 1, 1 };
static IDL_MEMINT force_dims[]  = { 1, 1 };

static IDL_STRUCT_TAG_DEF substruct_tags[] = {
   {"TIME",   times_dims, (void *) IDL_TYP_DOUBLE},
   {"FORCE",  force_dims, (void *) IDL_TYP_DOUBLE},
   {0}
  };

IDL_VPTR IDLTestS(int Argc, IDL_VPTR Argv[])
{

   IDL_VPTR  variabile = Argv[0];

   void *s;
   void *struct_s;
   IDL_MEMINT n_ele=1;
   // Dummy data
   int size=100000;
   double *s_data = (double*)malloc(size*sizeof(double));
   for (int i=0; i<size; i++){
      s_data[i]=i;
   }


   // I need to create the IDL_STRUCT_TAG_DEF [] at run time
   // because I don't know a priori the number of objects
   int n_of_objects=5;
   IDL_STRUCT_TAG_DEF *struct_tags = (IDL_STRUCT_TAG_DEF*)
malloc(sizeof(IDL_STRUCT_TAG_DEF) * (n_of_objects+1) );
   IDL_STRUCT_TAG_DEF *tag;
   for (int i=0; i<n_of_objects; i++){
      tag =  &struct_tags[i];
      tag->name=(char*)malloc(5);
      snprintf(tag->name,5,"V%03d",i);
      tag->dims=(IDL_MEMINT*) malloc(2*sizeof(IDL_MEMINT));
      tag->dims[0]=1;
      tag->dims[1]=1;
      tag->type=NULL;
   }
   // terminating the array of IDL_STRUCT_TAG_DEF
   tag =  &struct_tags[n_of_objects];
   tag->name=0;

   // create substructs
   for (int i=0; i<n_of_objects; i++){
      char nome[5];
      snprintf(nome,5,"V%03d",i);

      times_dims[1]  = 3+i;
      force_dims[1]  = 3+i;
      s = IDL_MakeStruct(nome, substruct_tags);
      struct_tags[i].type =s ;
   }
   // create  main struct
   struct_s = IDL_MakeStruct("FOO", struct_tags);

   // see if IDL_STRUCT_TAG_DEF [] is correct
   printf("printing struct_tags\n");
   {
      int itag=0;
      IDL_STRUCT_TAG_DEF *tag = &struct_tags[itag];
      while ( ((char*)tag)[0] != 0) {
         printf("%s - (%ld %ld) - %p\n",tag->name, tag->dims[0],
tag->dims[1], tag->type);
```

```
            tag = &struct_tags[++itag];
        }
    }

    // attach data to the created structure
    IDL_VPTR vv = IDL_ImportArray( 1, &n_ele, IDL_TYP_STRUCT, (UCHAR
*)s_data, idl_free_cb, struct_s);
    IDL_VarCopy(vv, variabile);

    return IDL_GettmpLong(1);
}
```

The main point is that, instead of having the IDL_STRUCT_TAG_DEF array
defined at compile time as usual, I need to create this structure at
run time.

That's the output of the test:

```
IDL> print, tests(a)
% Loaded DLM: TESTS.
printing struct_tags
V000 - (1 1) - 0x823dec4
V001 - (1 1) - 0x823dfbc
V002 - (1 1) - 0x823e0b4
V003 - (1 1) - 0x823e1ac
V004 - (1 1) - 0x823e2a4
           1
IDL> help ,a , /str
** Structure FOO, 7 tags, length=400, data length=400:
   TIME            DOUBLE    Array[3]
   FORCE           DOUBLE    Array[3]
   V001            STRUCT    -> V001 Array[1]
   V002            STRUCT    -> V002 Array[1]
   TIME            DOUBLE    Array[6]
   FORCE           DOUBLE    Array[6]
   V004            STRUCT    -> V004 Array[1]
IDL> print, a
{       0.0000000       1.0000000       2.0000000
       3.0000000       4.0000000       5.0000000
{
       6.0000000       7.0000000       8.0000000       9.0000000
       10.000000       11.000000       12.000000       13.000000
}{
       14.000000       15.000000       16.000000       17.000000
18.000000
       19.000000       20.000000       21.000000       22.000000
23.000000
}
      24.000000      25.000000      26.000000      27.000000
28.000000      29.000000
      30.000000      31.000000      32.000000      33.000000
34.000000      35.000000
{
      36.000000      37.000000      38.000000      39.000000
40.000000      41.000000      42.000000
      43.000000      44.000000      45.000000      46.000000
47.000000      48.000000      49.000000
}}
IDL> help, a.v001, /str
** Structure V001, 2 tags, length=64, data length=64:
   TIME            DOUBLE    Array[4]
   FORCE           DOUBLE    Array[4]
IDL>
```

As you can see, the variable "a" now contains a structure named "FOO"
containing the correct data, but instead of having a.v000, a.v001, ...,
a.v004, V000 and V003 (in this example) have not been created
correctly.
Of course there's nothing special in V000 and v0003 and indeed this
behaviour changes from time to time, (sometimes all the substructures
are OK).

It seems that me and IDL_MakeStruct got confused :)
Any idea of what's happening? My code is completely crazy?
Thanks
Lorenzo

Subject: Re: nested structures
Posted by Phillip Bitzer on Mon, 27 May 2013 15:03:01 GMT
View Forum Message <> Reply to Message

You can certainly do what you're after. In fact, I do this sort of thing when building arrays of radar data, which may have different lengths, sizes, etc.

First, some basic pointer stuff:

Consider:
```
IDL> s1 = {tag1:0L, tag2:PTR_NEW(/ALLOCATE)}
```

Then,
```
IDL> help, s1
** Structure <314b91d8>, 2 tags, length=8, data length=8, refs=1:
   TAG1            LONG                 0
   TAG2            POINTER   <PtrHeapVar14>
```

So, we see tag2 is a pointer. Fine, let's assign the pointer to a (new) structure:
IDL> *s1.tag2 = {ntag1:0L, nTag2:0L}

Okey doke. So, s1.tag2 is the pointer, and when we dereference this:

```
IDL> help, *s1.tag2
** Structure <1dc84338>, 2 tags, length=8, data length=8, refs=1:
   NTAG1           LONG                 0
   NTAG2           LONG                 0
```

we see our (new) structure.

What about getting to one of these tags? Notice this doesn't work:

```
IDL> help, *s1.tag2.ntag2
% Expression must be a structure in this context: <No name>.
% Execution halted at: $MAIN$
```

But this does:

```
IDL> help, (*s1.tag2).ntag2
<Expression>    LONG      =            0
```

Remember, *s1.tag2 is the pointer, and that's what what we want to dereference. That's why the parentheses are where they are.

Arrays of structures with pointers can be a little more tricky, because you'll be throwing brackets in there too. Just keep in mind where the pointer is.

Further, you'll want to take a look a this for the initialization:

http://www.idlcoyote.com/code_tips/structptrinit.html

Subject: Re: nested structures
Posted by hannah_ue on Tue, 28 May 2013 07:28:17 GMT
View Forum Message <> Reply to Message

Thank you Phillip, that helped. I finally figured out (I hope) how to reference to the array of structures in the array of structures and how to replicate those independently. I think I'm getting on now.

Am Montag, 27. Mai 2013 17:03:01 UTC+2 schrieb Phillip Bitzer:
> You can certainly do what you're after. In fact, I do this sort of thing when building arrays of radar data, which may have different lengths, sizes, etc.
>
>
>

```
>  First, some basic pointer stuff:
>
>
>
>  Consider:
>
>  IDL> s1 = {tag1:0L, tag2:PTR_NEW(/ALLOCATE)}
>
>
>
>  Then,
>
>  IDL> help, s1
>
>  ** Structure <314b91d8>, 2 tags, length=8, data length=8, refs=1:
>
>    TAG1           LONG                0
>
>    TAG2           POINTER   <PtrHeapVar14>
>
>
>
>  So, we see tag2 is a pointer. Fine, let's assign the pointer to a (new) structure:
>
>  IDL> *s1.tag2 = {ntag1:0L, nTag2:0L}
>
>
>
>  Okey doke. So, s1.tag2 is the pointer, and when we dereference this:
>
>  IDL> help, *s1.tag2
>
>  ** Structure <1dc84338>, 2 tags, length=8, data length=8, refs=1:
>
>    NTAG1          LONG                0
>
>    NTAG2          LONG                0
>
>
>
>  we see our (new) structure.
>
>
>
>  What about getting to one of these tags? Notice this doesn't work:
>
>  IDL> help, *s1.tag2.ntag2
>
```

> % Expression must be a structure in this context: <No name>.
>
> % Execution halted at: $MAIN$
>
>
>
> But this does:
>
> IDL> help, (*s1.tag2).ntag2
>
> <Expression>   LONG   =   0
>
>
>
> Remember, *s1.tag2 is the pointer, and that's what what we want to dereference. That's why the parentheses are where they are.
>
>
>
> Arrays of structures with pointers can be a little more tricky, because you'll be throwing brackets in there too. Just keep in mind where the pointer is.
>
>
>
> Further, you'll want to take a look a this for the initialization:
>
>
>
> http://www.idlcoyote.com/code_tips/structptrinit.html