Subject: C Alignment/IDL structures
Posted by joey on Wed, 16 Mar 2005 16:13:03 GMT
View Forum Message <> Reply to Message

Hi! I have an C library which I link into IDL using IDL_MakeStruct. The structure I link in with IDL is not a true C structure, but a dynamic structure I create on the fly depending on the data I wish to have within IDL.

It works quite well; however, one of the elements incorporated in my structure is another structure which I would like to add a double value.

When I add this double, my structure size seems to get the wrong size for IDL to handle so nothing will work when accessing the structure. I compute the size to a value which is the sum of all bytes in the structure which obviously does not take into account the packing/aligning that C does.

My question is: is there a way I can figure out how many bytes to malloc such that IDL and I will always be in agreement?

Even if we are in agreement, I need to know where the padding will occur so I would like to have the same algorithm that IDL uses to compute its structure allocation when given a list of tags.

If I stick to floats/long/int/etc/strings/etc. everything seems to work great. Its only when adding the double that everything goes bad.

Thanks for any advice!

Joey

Subject: Re: C Alignment/IDL structures
Posted by Randall Skelton on Wed, 16 Mar 2005 18:07:33 GMT
View Forum Message <> Reply to Message

I'm not sure I completely understand what you are doing... can you post a snipit of code? I trust that you are passing back an unnamed structure from C as named structures are persistent in a given session. Hence, you should call as:

ptr = IDL_MakeStruct(NULL, mytags);

This is the reason the following fails:

IDL> a = {foo, a: 0, b: ''}
IDL> b = {foo, c: 0}
% Wrong number of tags defined for structure: FOO.

Once 'foo' is defined, you cannot change its fields.

In general, when I have to pass C structures from existing code back to IDL I do it by creating a shadow structure (in C) that uses all the defined IDL types and copying the data. You really cannot rely on generic C variables having the same size as thier IDL counterparts (take a look at the definition of IDL_ALLTYPES in idl_export.h). You can cheat a little when initializing arrays as you can point to the memory in your existing structures if the sizes align, but everything in the C structure should be defined with one of the macros given in the EDG. I don't think you should be seeing any structure alignment issues...

Cheers,
Randall

Subject: Re: C Alignment/IDL structures
Posted by joey on Mon, 21 Mar 2005 16:56:20 GMT
View Forum Message <> Reply to Message

Randall Skelton <randall.skelton@gmail.com> wrote:
> I'm not sure I completely understand what you are doing... can you post
> a snipit of code? I trust that you are passing back an unnamed

Here's my code that does the IDL/C interaction. I have a vector (_dataIDL) which has all the values I want into IDL within it as a malloc'ed sets of memory.

```
// Copy the real data

   unsigned long pos = 0;
   unsigned char *myStructureThatLooksLikeTags = malloc (_totalSpaceNeeded
                                          * _dataIDL.size ());
   for (unsigned int i = 0; i < _dataIDL.size (); i++) {
 memcpy (&(myStructureThatLooksLikeTags [pos]), _dataIDL [i],
            _totalSpaceNeeded);
 pos += _totalSpaceNeeded;
   }

   char idl_struct_name [100];
   sprintf (idl_struct_name, "%s_K%ld_V%d", dbVirtualName (data_key),
            data_key, version);

   void *idl_struct = IDL_MakeStruct (idl_struct_name, tags);
   IDL_LONG dims;
   dims = _dataIDL.size (); // number of elment in the array of structures
```

```
IDL_VPTR ivReturn = IDL_ImportArray (1, &dims, IDL_TYP_STRUCT,
                                     myStructureThatLooksLikeTags,
                                     cleanUpIDL, idl_struct);
```

> In general, when I have to pass C structures from existing code back to
> IDL I do it by creating a shadow structure (in C) that uses all the
> defined IDL types and copying the data. You really cannot rely on
> generic C variables having the same size as thier IDL counterparts
> (take a look at the definition of IDL_ALLTYPES in idl_export.h).

Ok, this probably answers my question. I was hoping I could create an array of structs, but this is maybe not so memory efficient so I might try to create one structure with multiple arrays.

Joey
