Subject: Re: Common block access in DLM
Posted by Haje Korth on Tue, 18 May 2004 19:53:24 GMT
View Forum Message <> Reply to Message

Rick
thanks for the example, this is more than I expected. Regarding my project, I ported a FORTRAN library for geophysical coordinate transformations and magnetic field calculation from FORTRAN to IDL usind a DLM, basically IDL <-> DLM <-> C <-> FORTRAN, taking Ronn's exercises a little further. Unfortunately I only have Ronn's first edition book, so I do not know if he expanded on this in the second edition. Anyway, everything works as expected, but having to supply year, day of year, hour minute second for every call is nerve wrecking, and blows up the call to so many parameters that the IDL code looks confusing. I will give the global variables a try. Here a very stupid question though: How do setup such a global variable in C? Is this just regular C language, or is there some fancy handling required with the IDL external API? (I am pretty much C newby, but once I now this, I think I can handle the interfacing with FORTRAN.)

Thanks,
Haje

"Rick Towler" <rtowler@u.washington.edu> wrote in message
news:c8dhm8$ooo$1@nntp6.u.washington.edu...
>
> "Haje Korth" wrote...
>
>> I have a routine that sets up coordinate transformations that are
>> subsequently used by calls to other procedures in the DLM. Rather than
>> calling this setup routine for every procedure call, it would be more
>> eficcient to establish the transformation matrices, store them in an IDL
>> common block, and access this common block over and over. However, this
>> scenario requires to be able to create IDL common block from C and have
> read
>> and write access to it. Therefore my question: Does anybody know whether
> it
>> is possible to access an IDL common block from C in a DLM? If anyone
could
>> point me to the right functions in the IDL external API, I would be very
>> thankful.
>
> Hi Haje,
>
> Sounds like you have something interesting brewing...
>
> I can't tell you how to do it with common blocks but if I am reading you
> correctly, you have a couple of other options:
>

> Use a global variable in your DLM. If you need to access this on the IDL
> side, then write your "init" function to return the transform back to IDL
> too and keep a copy there. Simple, if not very elegant. The downside is
> that you can only store one instance of your transform
>
>
> I used to do this quite a bit but have since started creating little C++
> classes to store this type of data so I could have multiple instances that
> didn't clash. And when I mean little, I *mean* little:
>
>

```
>  class HajeTransform
>  {
>
>  public:
>
>   float transform[4][4];
>
>  };
```

>
>
> Then in your init function you set up your transform and return the
pointer
> to your HajeTransform object back to IDL:
>
>

```
>  char   *cptr;
>  IDL_MEMINT  dims[1];
>  IDL_VPTR  result;
>  HajeTransform  *hTransform = new HajeTransform;
>
>  //  do what you do to set up your transform
>  (*hTransform).transform[0][0] = <insert stuff here>
>  .
>  .
>  .
>
>  //  Return ptr to HajeTrans object back to IDL
>  dims[0] = sizeof(hTransform);
>  cptr = IDL_MakeTempArray(IDL_TYP_BYTE,1,dims,IDL_ARR_INI_NOP, &result);
>  memcpy(cptr, &hTransform, dims[0]);
>
>  return result;
```

>
>
> This pointer will serve as your instance ID, all subsequent calls to your
> DLM functions will require this ID so you can dereference the pointer and
> access your object members.

```
>
>  Assuming your first argument is the pointer value:
>
>
>   HajeTransform  *hTransform;
>
>   //  obtain the pointer to the HajeTransform object
>   IDL_ENSURE_ARRAY(argv[0]);
>   memcpy(&hTransform, argv[0]->value.arr->data, sizeof(hTransform));
>
>   // Now you can access your HajeTransform object
>   (*hTransform).transform[0][0] = ....
>
>
>
>  You will need to add a cleanup routine to free your object when you are
>  done:
>
>  // The SAFE_DELETE macro used in the function below is defined as:
>  #define SAFE_DELETE(p)  { if(p) { delete (p);     (p)=NULL; } }):
>
>
>  void IDL_CDECL HajeTransform_Cleanup(int argc, IDL_VPTR *argv)
>  {
>
>     /*
>        Frees memory associated with the HajeTransform object.
>     */
>
>   IDL_MEMINT  dims[1];
>   HajeTransform  *hTransform;
>
>   IDL_ENSURE_ARRAY(argv[0]);
>
>   dims[0] = sizeof(hTransform);
>   memcpy(&hTransform, argv[0]->value.arr->data, dims[0]);
>
>   SAFE_DELETE(hTransform);
>
>  }
>
>
>  The details of passing pointers back and forth has been covered before
>  (Nigel just posted on this subject again today).  There have been other
>  posts regarding compiling C++ dlms and Ronn's new version of his "Calling
C
>  from IDL" now covers C++ too.
>
```

> Hope this helps!
>
> -Rick
>
>