Subject: DLM heap variable access
Posted by penteado on Sat, 27 Jun 2009 19:34:21 GMT
View Forum Message <> Reply to Message

I got tired of waiting for the ITTVIS folks to implement some more data structures in IDL. Coding a bunch of them (lists, maps, stacks) in IDL would be a fair amount of rewriting the wheel, and also inefficient, because of the way IDL' s pointers and scalars work. So I decided that the nicest solution would be to have IDL objects as wrappers to C++ containers. It is simple enough to do it writing a DLM in the way Ronn Kling' s book suggests, with the IDL object containing a (real) pointer to the C++ object, and wrapper methods to call the C+ + methods.

However, I was unhappy with having to make a method in IDL that passes the object pointer and the arguments to a C++ wrapper, that then does the job with the C++ object. It would be much nicer to write the IDL method directly in C++. The trouble is how to get access to the IDL object's self from the C++ routine, to retrieve the C++ object pointer in it. As Ronn mentions, the IDL object reference gets passed to the method in argv[0], but nowehre I could find a reference to how to use it, except for this very unsatisfying sentence

"Direct access to pointer and object reference heap variables (types IDL_TYP_PTR and IDL_TYP._OBJREF, respectively) is not allowed."

from IDL' s documentation. I figured that the IDL object reference is passed in argv[0] for some use, and it appears that some objects written by ITTVIS do exactly that. So after some experimenting and browsing through idl_export.h, I eventually figured out how to do it.

In the description below, the IDL object was defined with a single structure member, self.obj, that is a pointer to a byte array where the C++ pointer is stored (as suggested in Ronn's book).

1) argv[0] has a type IDL_TYP_OBJREF. Therefore, its value contains the heap variable identifier (IDL_HVID hvid). Of course that is just IDL' s id number for the heap variable, not an actual pointer.

2) idl_exports.h contains the prototype:
IDL_HEAP_VPTR IDL_CDECL IDL_HeapVarHashFind(IDL_HVID hash_id)
I found that this function returns a pointer to the heap variable given its identifier.

3) What heap variable is pointed to by argv[0]->value.hvid? The IDL object's self!

4) It is now necessary to retrieve the heap variable pointed to by

self.obj. This is done with IDL_HeapVarHashFind on the heap variable
id in self.obj:

```
//Get a pointer to self (self is {pp_stl,obj:ptr_new()}):
IDL_HEAP_VPTR ohvptr=IDL_HeapVarHashFind(argv[0]->value.hvid);
//Get the identifier of the heap variable of self.obj:
IDL_HVID *pind=(IDL_HVID *) ohvptr->var.value.arr->data;
//Get a pointer to *(self.obj):
IDL_HEAP_VPTR hvptr=IDL_HeapVarHashFind(*pind);
//Get the real pointer from *(self.obj):
memcpy(&object,hvptr->var.value.arr->data,sizeof(object));
```

Subject: Re: DLM heap variable access
Posted by penteado on Thu, 02 Jul 2009 20:44:11 GMT
View Forum Message <> Reply to Message

On Jul 2, 1:52 pm, pp <pp.pente...@gmail.com> wrote:
> On Jun 29, 3:39 pm, rtk <oneelkr...@hotmail.com> wrote:
>
>> Clever. This might be of interest to you as well:
>
>> http://www.ittvis.com/info/hof/
>

Are there other files besides those in that link? The documentation
mentions a list.sav file with a list class, but it is not in either
linux32.tar.gz or win32.zip. I noticed that there is a
list__define.pro, but I could not find (neither could IDL) the method
Join. Am I missing something?

Subject: Re: DLM heap variable access
Posted by Michael Galloy on Thu, 02 Jul 2009 22:57:46 GMT
View Forum Message <> Reply to Message

pp wrote:
> On Jul 2, 1:52 pm, pp <pp.pente...@gmail.com> wrote:
>> On Jun 29, 3:39 pm, rtk <oneelkr...@hotmail.com> wrote:
>>
>>> Clever. This might be of interest to you as well:
>>> http://www.ittvis.com/info/hof/
>
> Are there other files besides those in that link? The documentation
> mentions a list.sav file with a list class, but it is not in either
> linux32.tar.gz or win32.zip. I noticed that there is a

> list__define.pro, but I could not find (neither could IDL) the method
> Join. Am I missing something?
>

Are you looking for list_join.pro? It is not a method of the List class,
but a routine that takes a list and creates a standard IDL array out
of it.

Mike
--
www.michaelgalloy.com
Associate Research Scientist
Tech-X Corporation

Subject: Re: DLM heap variable access
Posted by penteado on Fri, 03 Jul 2009 02:41:54 GMT
View Forum Message <> Reply to Message

> Are you looking for list_join.pro? It is not a method of the List class,
> but a routine that takes a list and creates a standard IDL array out
> of it.
>
No. I noticed that there is a list_join. But the documentation says
there is also a corresponding method function:

"ans = obj->Join([ERROR=err])
Return an IDL array built from the list. If the array cannot be made,
return 1
and set ERROR to
a text message explaining the reason for the error. If no error, ERROR
will be set to the empty
string."

Which it says is in list.sav. But there is no list.sav in either file
at http://www.ittvis.com/info/hof/

Subject: Re: DLM heap variable access
Posted by penteado on Fri, 03 Jul 2009 15:46:38 GMT
View Forum Message <> Reply to Message

I was looking around the files, and found some odd things:

1) There is a method in list__define, which appears to do what the
documentation calls Join. But it is called Simp. However, it tries to
call a nonexisting function list_simp, which I thus I replaced by a

call to list_join.

2) 3 of the methods (function and procedure Show, plus Simp), when called, apparently destroyed the list. That is:

```
IDL> a=obj_new('list')
IDL> a->append,9d0
IDL> a->append,5d0
IDL> print,list_active()
           1
IDL> a->show
(9.0000000 5.0000000)
IDL> print,list_active()
           0
IDL> obj_destroy,a
% LIST_FREE: Argument not of list type.
```

Which I found could be solved (though I do not understand why), by replacing the argument in the calls to the list functions, to use a copy of self.d. That is, replacing

```
print, list_show(self.d)
```

by

```
d=self.d
print, list_show(d)
```

Subject: Re: DLM heap variable access
Posted by Jason Ferrara on Mon, 06 Jul 2009 12:16:12 GMT
View Forum Message <> Reply to Message

On Jun 27, 3:34 pm, pp <pp.pente...@gmail.com> wrote:
> I got tired of waiting for the ITTVIS folks to implement some more
> data structures in IDL. Coding a bunch of them (lists, maps, stacks)
> in IDL would be a fair amount of rewriting the wheel, and also
> inefficient, because of the way IDL' s pointers and scalars work. So I
> decided that the nicest solution would be to have IDL objects as
> wrappers to C++ containers. It is simple enough to do it writing a DLM
> in the way Ronn Kling' s book suggests, with the IDL object containing
> a (real) pointer to the C++ object, and wrapper methods to call the C+
> + methods.

With Slither you can use Python's container classes from IDL. I realize its a bit extreme to use a method that requires a Python install plus a Slither license just to get some containers, but it does work rather well.

```
IDL> pb=pyimport("__builtin__")
% Loaded DLM: SLITHER.
IDL> l=pb->list()
IDL> l->append, 6
IDL> l->append, 3
IDL> l->append, 23
IDL> print, pb->len(l)
          3
IDL> print, l->__getitem__(1)
       3
IDL> print, l->__getitem__(0)
       6
IDL> print, l->__getslice__(1,3)
       3      23
IDL> print, pytoidl(l)
       6       3      23
IDL> print, l->pop()
      23
IDL> print, l->pop()
       3
IDL>
IDL>
IDL>
IDL> d=pb->dict()
IDL> d->__setitem__, "something",2
IDL> d->__setitem__, "otherthing",3
IDL> d->__setitem__, "nothing",6
IDL> print, d->__getitem__("otherthing")
       3
IDL> print, d->keys()
nothing otherthing something
IDL> print, d->values()
       6       3       2
```

Subject: Re: DLM heap variable access
Posted by penteado on Sat, 11 Jul 2009 17:45:22 GMT
View Forum Message <> Reply to Message

> Which I found could be solved (though I do not understand why), by
> replacing the argument in the calls to the list functions, to use a
> copy of self.d. That is, replacing
>
> print, list_show(self.d)
>
> by
>

> d=self.d
> print, list_show(d)

Of course the answer to this was obvious. I just keep forgetting that
structure elements are passed by value, not by reference. Even though
list_show() does not change the value of its argument, if it is given
an identifier (to a non-empty list) by value, it erases the list.

Subject: Re: DLM heap variable access
Posted by rtk on Mon, 13 Jul 2009 16:05:56 GMT
View Forum Message <> Reply to Message

On Jul 11, 11:45 am, pp <pp.pente...@gmail.com> wrote:
> Of course the answer to this was obvious. I just keep forgetting that
> structure elements are passed by value, not by reference. Even though
> list_show() does not change the value of its argument, if it is given
> an identifier (to a non-empty list) by value, it erases the list.

Sorry, I missed following up on this thread. The list functions, and
the
higher-order functions, all destroy any list given as an argument if
that
list is not assigned to a variable already. This is necessary to
allow the
output of one function call to be immediately used by another without
wasting
memory all over the place.

It is entirely possible that I left list.sav out. The list object
calling
list_simp is probably left over from an earlier version. Personally,
I use
the bare list functions for speed, not the list object.

I'll look into building 64-bit versions of the DLMs but this week
seems like
it will be busy. The 32-bit versions should work on 64-bit machines.

Ron

Subject: Re: DLM heap variable access
Posted by rtk on Mon, 13 Jul 2009 21:40:03 GMT
View Forum Message <> Reply to Message

On Jul 13, 10:05 am, rtk <oneelkr...@hotmail.com> wrote:

> The 32-bit versions should work on 64-bit machines.

Of course, this is only true of 32-bit applications, not DLMs :)

Anyway, I just built 64-bit Linux versions, email me if you want them. I'll try to get them posted on the web site as well. 64-bit Windows I'll get to later in the week.

Ron
rkneusel@ittvis.com
