Subject: Re: Passing Image Data :)
Posted by David Fanning on Fri, 19 Oct 2001 19:56:27 GMT
View Forum Message <> Reply to Message

Logan Lindquist (llindqusit@mrdoc.cc) writes:

> Since everyone was so helpful last time, I figured I'd give this
> another shot. I read in the [Mr. Fanning's IDL PT 1st ed.] that
> it is better to use Struct's (info = { imageData:imageData})
> to pass common program information between pro/functions rather
> than the IDL 'common' keyword. Well I am having problems doing
> that. I would like to make it so it doesn't matter what
> type[2, 3, or 4 dimensional] of image I pass between pro/functions.
> I might try making a image data variable for each type, but
> that seems redundant. My original thinking was to make a
> dummy ByteArr and then resize it, if need be, but that didn't work.
>
> I tried several different variable initializations, even making
> it so that it was the same as the returned image and it still
> gives me an error saying that the expressions are not the same.
> I think I am just going to rewrite it so that the image data is
> passed using the common keyword. Does the common keyword make a
> pointer? Do I have to release this from memory, or does IDL
> handle that? Now that I think of it, that might be better,
> cause it would be faster if I could just create one instance
> of the image data in memory rather than copying and pasting
> it between parts of the program. So I guess what the question
> really is, What is the quickest [best] way to pass image data
> of varying dimensions between program components?

Oh, oh. I'd better send you the 2nd Edition of the book. :-(

What you want in your info structure image field is a pointer
to the image:

```
info= { image:Ptr_New(myimage), ...}
```

Then, you don't have to worry about the size or dimensionality
of the image. When you want a new image placed there, you just
do this:

```
*info.image = newimage
```

IDL takes care of all the memory management for you. You don't
have to worry about it.

If you display your image with TVIMAGE or IMDISP, then
you also don't have to worry about size and dimensionality:

TVImage, *info.image

You also learn in the 2nd edition how to make the
info structure a pointer, if you are brave enough
for that. :-)

Cheers,

David

P.S. Let's just say if you want a book, give me a call.
Anyone who calls me Mr. Fanning gets a *huge* discount. :-)

--
David W. Fanning, Ph.D.
Fanning Software Consulting
Phone: 970-221-0438, E-mail: david@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155

Subject: Re: Passing Image Data :)
Posted by Noam R. Izenberg on Fri, 19 Oct 2001 20:01:17 GMT
View Forum Message <> Reply to Message

David Fanning wrote:...
....

> Anyone who calls me Mr. Fanning gets a *huge* discount. :-)
>
> --
> David W. Fanning, Ph.D.

How about _Dr._ Fanning? :-)

Doesn't matter anyway. I think my order is already in thru my office.

Noam

Subject: Re: Passing Image Data :)
Posted by Pavel A. Romashkin on Fri, 19 Oct 2001 20:13:05 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
>
> You also learn in the 2nd edition how to make the
> *info* structure a pointer, if you are brave enough
> for that. :-)

First, Davis is implying that he is to be called Dr. Fanning. Anybody
who does otherwise gets buggy code from his web page, of which he keeps
two copies - one for good folk, one for those to be mislead.
Secondly, It is obviously the time to write that object (not OG!) book,
David! Who wants nowdays the Info structure as a pointer? The whole
widget program should be an object, and the GUI needs to be its property
- then everything will be right there when you need it :-)

Cheers,
Pavel

Subject: Re: Passing Image Data :)
Posted by David Fanning on Fri, 19 Oct 2001 20:19:46 GMT
View Forum Message <> Reply to Message

Noam R. Izenberg (noam.izenberg@jhuapl.edu) writes:

> David Fanning wrote:...
> ....
>
>> Anyone who calls me Mr. Fanning gets a *huge* discount. :-)
>>
>> --
>> David W. Fanning, Ph.D.
>
> How about _Dr._ Fanning? :-)
>
> Doesn't matter anyway. I think my order is already in thru my office.

Alas, Noam, your book went out about 10 minutes
before flattery got the best of me and I changed
my discount structure. :-(

The bad news is that my wife is apparently monitoring
this newsgroup now. I just got a phone call letting me
know in no uncertain terms *EXACTLY* what my discount
policy will be from here on out. Let's just say I won't
be announcing any huge discounts on the newsgroup any
time soon. :-(

Cheers,

David

--
David W. Fanning, Ph.D.
Fanning Software Consulting
Phone: 970-221-0438, E-mail: david@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155

Subject: Re: Passing Image Data :)
Posted by Logan Lindquist on Fri, 19 Oct 2001 21:41:00 GMT
View Forum Message <> Reply to Message

> P.S. Let's just say if you want a book, give me a calls
>Anyone who calls me Mr. Fanning gets a *huge* discount. :-)

Dr. Fanning,

Of course by no way did I attempt to dimean your honorific. I am sure you worked hard [paid very little so you could practice writing long papers while hopefully not having to grade a lot of tests or teach a lot of entry level courses] to obtain the title of Dr., instead of recieving an honory [free] one from a university that is more concerned with marketing itself. :)

I would have the company order your new book in addition to the book by Liam E. Gumley, but I would rather wait a bit and buy it/them for myself. That way I get to keep it/them. You could however set aside one of those spiral bound student versions. I am in class until Dec. of this year, even if it doesn't relate to IDL. Guess I better put that in my budget before then, if I want the discount. :) That way, everyone, including your wife will satisfied with the transaction.

Logan

Subject: Re: Passing Image Data :)
Posted by Logan Lindquist on Tue, 23 Oct 2001 19:31:10 GMT
View Forum Message <> Reply to Message

Dr(.) Fanning,
[I thought Dr. had an period after it because it is an abbrevation of doctor? I do not know what Andrew Cool is talking about.]

> What you want in your info structure image field is a pointer

> to the image:
>
> info= { image:Ptr_New(myimage), ...}

I am wondering if you could clear up a couple of things about pointers in IDL. How come myimage does not have to be defined during initalization? Does the statement above create space in memory for a variable of indefinite size? It seems to operate this way., where the data in memory is allocated once the data has to be stored to the pointer array. Maybe I am understanding pointers incorrectly.

1.. The Pointer is created - a variable that 'points' to space in RAM reserved for a variable of indefinate size.
2.. The data is read into RAM during the read_image.pro.
3.. The Pointer then needs to store the image data for future reference. This is done by '*info.image = newimage'. Where newimage is the image data in RAM.
4.. Is the data then copied into the space originally allocated for it or does it simply change it's reference so as to point to the location in RAM where the image data was read into?

> *info.image = newimage
>
> IDL takes care of all the memory management for you. You don't
> have to worry about it.

I went back and reviewed how pointers are treated in C++. I was wondering if I made my Struct a pointer, could I access memebers of Struct's using the '->'?

Thank You,

Logan Lindquist

Below is what I found on pointers in C++.
****

Pointers to Objects

Pointers can point to objects as well as to simple data types and arrays. We've seen many examples of objects defined and given a name, in statements like

Distance dist;

where an object called dist is defined to be of the Distance class.

Sometimes, however, we don't know, at the time that we write the program, how many objects we want to create. When this is the case we can use new to create objects while the program is running. As we've seen, new returns a pointer to an unnamed object. Let's look at a short example program, ENGLPTR, that compares the two approaches to creating objects.

```
// englptr.cpp
// accessing member functions by pointer
 #include <iostream>
 using namespace std;
  ////////////////////////////////////////////////////////// ////
 class Distance              //English Distance class
   {
  private:
      int feet;
      float inches;
  public:
       void getdist()        //get length from user
         {
         cout << "\nEnter feet: ";  cin >> feet;
         cout << "Enter inches: ";  cin >> inches;
        }
       void showdist()       //display distance
         { cout << feet << "\'-" << inches << '\"'; }
   };
  ////////////////////////////////////////////////////////// ////

 int main()

   {
    Distance dist;           //define a named Distance object
   dist.getdist();          //access object members
   dist.showdist();         //   with dot operator
   Distance* distptr;       //pointer to Distance
   distptr = new Distance;  //points to new Distance object
    distptr->getdist();      //access object members
    distptr->showdist();     //   with -> operator
    cout << endl;
   return 0;

   }
```

This program uses a variation of the English Distance class seen in previous chapters. The main() function defines dist, uses the Distance member function getdist() to get a distance from the user, and then uses showdist() to display it.

Referring to Members

ENGLPTR then creates another object of type Distance using the new operator, and returns a pointer to it called distptr.

The question is, how do we refer to the member functions in the object pointed to by distptr? You might guess that we would use the dot (.) membership-access operator, as in

```
distptr.getdist();   // won't work; distptr is not a variable
```

but this won't work. The dot operator requires the identifier on its left to be a variable. Since distptr is a pointer to a variable, we need another syntax. One approach is to dereference (get the contents of the variable pointed to by) the pointer:

```
(*distptr).getdist();  // ok but inelegant
```

However, this is slightly cumbersome because of the parentheses. (The parentheses are necessary because the dot operator (.) has higher precedence than the indirection operator (*). An equivalent but more concise approach is furnished by the membership-access operator ->, which consists of a hyphen and a greater-than sign:

```
distptr->getdist();   // better approach
```

As you can see in ENGLPTR, the -> operator works with pointers to objects in just the same way that the . operator works with objects. Here's the output of the program:

```
Enter feet: 10   ;this object uses the dot operator
Enter inches: 6.25
10'-6.25"
Enter feet: 6    ; this object uses the -> operator
Enter inches: 4.75
6'-4.75"
```