Subject: object newbie
Posted by John D. Sample on Thu, 10 Aug 2000 07:00:00 GMT
View Forum Message <> Reply to Message

I must admit that based on comments from this list, I have experimented with the object features of IDL for the past week or so, and have implemented them in a few places in a fairly big widget code I have written.

My initial observations are that there seems to be less of a temptation to use common blocks when objects are used. On the other hand my first impression was that an object is a structure whose fields can not be accessed until you write additional "methods" to get at each and every damn one of them. So my object was littered with about 25 "methods" just so I could pry the data out of the object.

I eventually came on a work around to write a "proto_object" with a method allowing you to pass a string containing a tag name which returns the contents of the field with that tag name. This "proto_object" is inherited by all other objects I create just so I can use this method. Along the way I found that the TAG_NAMES function in IDL doesn't work for objects so I had to create one. It basically copies the object structure into a regular structure so the TAG_NAMES can be used.

Am I making this too hard?

Chip

Subject: Re: object newbie
Posted by marc schellens[1] on Sun, 13 Aug 2000 07:00:00 GMT
View Forum Message <> Reply to Message

Chip Sample wrote:
>
> I must admit that based on comments from this list, I have experimented with
> the object features of IDL for the past week or so, and have implemented
> them in a few places in a fairly big widget code I have written.
>
> My initial observations are that there seems to be less of a temptation to
> use common blocks when objects are used. On the other hand my first
> impression was that an object is a structure whose fields can not be
> accessed until you write additional "methods" to get at each and every damn
> one of them. So my object was littered with about 25 "methods" just so I
> could pry the data out of the object.
>
> I eventually came on a work around to write a "proto_object" with a method
> allowing you to pass a string containing a tag name which returns the
> contents of the field with that tag name. This "proto_object" is inherited

> by all other objects I create just so I can use this method. Along the way
> I found that the TAG_NAMES function in IDL doesn't work for objects so I had
> to create one. It basically copies the object structure into a regular
> structure so the TAG_NAMES can be used.
>
> Am I making this too hard?
>
> Chip

Check out
http://gammaray.msfc.nasa.gov/~mallozzi/home/software/idl/sr c/generic__get.pro

There are a lot of other useful routines Robert Mallozzi offers on his
webpage also.
Seems to me you re-invented the wheel, eh?

But generally objects are more than structures which only hold data.
So if you need every tag returned from your object, perhaps you should
consider to write a method instead, condensing the data to the values
you really need (or take the appropriate actions).
Probably this would made your program more the way object oriented
programs are meant to be.
But of course the problem you had arises typicaly, when using objects
only in some places, together with non-object oriented parts.

cheers,
:-) marc

Subject: Re: object newbie
Posted by promashkin on Mon, 14 Aug 2000 07:00:00 GMT
View Forum Message <> Reply to Message

I have a feeling that Chip's question is not quite answered by the dump
truck full of technicalities the group readily poured out. It looks to
me that the question is, "my object is just a structure but it is hell
of a lot harder for me to use it than a regular structure".
I would say that if all you need is the data and there is one instance
of it, you do not need an object. The whole idea is that object is a *reuseable*.
For example, you have one data processing task. You have a vector map
and an overlayed image. You want to be able to access the data for that
image only. Might as well write plain code for it.
Now, you have the same map, but 50 images that will be processed
similarly. In this case, if you write an object, then you can write the
"25 methods" only once, and very elegantly call those methods in your
widget event processing code. This way, you have intact data and a set
of actions (methods) you can perform on that data. And, if you happened
to aquire more bands of the image(s), you do not need to alter the code.

You just add more objects that use the same methods. On the other hand,
if you create a structure (or array of such), then you can't as easily
add another elements to it. Actions (code that modifies the data) tend
to become less organized (because they are not linked to data), and
event handlers become very complex.
This is not to mention inheritance, which simplifies dramatically
operations on data that are derivative or daughter to the main object class.
Cheers,
Pavel

Chip Sample wrote:
>
> I must admit that based on comments from this list, I have experimented with
> the object features of IDL for the past week or so, and have implemented
> them in a few places in a fairly big widget code I have written.
>
> My initial observations are that there seems to be less of a temptation to
> use common blocks when objects are used. On the other hand my first
> impression was that an object is a structure whose fields can not be
> accessed until you write additional "methods" to get at each and every damn
> one of them. So my object was littered with about 25 "methods" just so I
> could pry the data out of the object.
>
> I eventually came on a work around to write a "proto_object" with a method
> allowing you to pass a string containing a tag name which returns the
> contents of the field with that tag name. This "proto_object" is inherited
> by all other objects I create just so I can use this method. Along the way
> I found that the TAG_NAMES function in IDL doesn't work for objects so I had
> to create one. It basically copies the object structure into a regular
> structure so the TAG_NAMES can be used.
>
> Am I making this too hard?
>
> Chip

Subject: Re: object newbie
Posted by Martin Schultz on Tue, 15 Aug 2000 07:00:00 GMT
View Forum Message <> Reply to Message

Pavel,

these are valuable comments. However, I disagree that objects are
only useful for
N >= 50 as you indicate. Especially when using inheritance, they have
tremendous power
even for N = 1 if you like some uniform access to "things" that belong
to the same category

but may all be a little different. Certainly you are right that
structures still have their
value, and you don't need an object for EVERYTHING in IDL. Yet, I have
the feeling that
your comment misses the point in answering Chip's question, and that
this is probably
because the question itself is flaky. I haven't pondered over this day
and night, but
I dare to say that you made something wrong when you need to know the
fields that are
stored in the object. Object oriented programming does require
substantial twists of
the sequential programmer's mind in terms of program structure. As
always, the design of a
program is the most important step in programming, but for objects it is
even more
important. Every object field (structure element) that you want to
access from outside
must be explicitely interfaced for a reason (encapsulation). All other
fields should
be regarded as private properties of the object for use only by the
object itself.
If you need access to object fields, the GetProperty, SetProperty
convention also
allows you to distinguish between Readonly and Writeonly access, simply
by including
or emitting respective keywords. Sure, OOP requires more typing than a
classical program,
but so far I still think I am rewarded in the end by a product that is
much more powerful
than a classical program of similar complexity. And if I had wanted to
include all that
power into a classical program, I probably would have had typed at least
as much.

Cheers,
Martin

Pavel Romashkin wrote:
>
> I have a feeling that Chip's question is not quite answered by the dump
> truck full of technicalities the group readily poured out. It looks to
> me that the question is, "my object is just a structure but it is hell
> of a lot harder for me to use it than a regular structure".
> I would say that if all you need is the data and there is one instance
> of it, you do not need an object. The whole idea is that object is a *reuseable*.
> For example, you have one data processing task. You have a vector map

> and an overlayed image. You want to be able to access the data for that
> image only. Might as well write plain code for it.
> Now, you have the same map, but 50 images that will be processed
> similarly. In this case, if you write an object, then you can write the
> "25 methods" only once, and very elegantly call those methods in your
> widget event processing code. This way, you have intact data and a set
> of actions (methods) you can perform on that data. And, if you happened
> to aquire more bands of the image(s), you do not need to alter the code.
> You just add more objects that use the same methods. On the other hand,
> if you create a structure (or array of such), then you can't as easily
> add another elements to it. Actions (code that modifies the data) tend
> to become less organized (because they are not linked to data), and
> event handlers become very complex.
> This is not to mention inheritance, which simplifies dramatically
> operations on data that are derivative or daughter to the main object class.
> Cheers,
> Pavel
>
> Chip Sample wrote:
>>
>> I must admit that based on comments from this list, I have experimented with
>> the object features of IDL for the past week or so, and have implemented
>> them in a few places in a fairly big widget code I have written.
>>
>> My initial observations are that there seems to be less of a temptation to
>> use common blocks when objects are used. On the other hand my first
>> impression was that an object is a structure whose fields can not be
>> accessed until you write additional "methods" to get at each and every damn
>> one of them. So my object was littered with about 25 "methods" just so I
>> could pry the data out of the object.
>>
>> I eventually came on a work around to write a "proto_object" with a method
>> allowing you to pass a string containing a tag name which returns the
>> contents of the field with that tag name. This "proto_object" is inherited
>> by all other objects I create just so I can use this method. Along the way
>> I found that the TAG_NAMES function in IDL doesn't work for objects so I had
>> to create one. It basically copies the object structure into a regular
>> structure so the TAG_NAMES can be used.
>>
>> Am I making this too hard?
>>
>> Chip

--
[[[[[[[[[
[[Dr. Martin Schultz   Max-Planck-Institut fuer Meteorologie   [[
[[Bundesstr. 55, 20146 Hamburg   [[
[[phone: +49 40 41173-308   [[

[[fax: +49 40 41173-298   [[
[[martin.schultz@dkrz.de   [[
[[[[[[[[

Subject: Re: object newbie
Posted by John D. Sample on Tue, 15 Aug 2000 07:00:00 GMT
View Forum Message <> Reply to Message

Maybe if I provided a few more details, it would be apparent whether I've missed the boat.

I have a widget that is used to read and graph data from a radar model written in C. I also use the widget to launch and control runs of this C program on many machines across the network (thereby insulating me from UNIX), as well as browsing and editing the input files and creating additional runs according t a trade space. This all was developed without objects.

Based on discussion here a curiosity I decided to begin to "objectify" this widget and began with the structure that contains "jobs", i.e. the path/name of the executable, the path/name of the target file, the path/name of a dump file containing what would normally go to the screen, a list of program options, the lun being used, and IDs of some control widgets associated with that job. These widgets can be used to kill the remote job, look at the dump and output files, and look at a status display containing most of the information on the job object. So to pass these fields into the display widgets, I need access to most of the field values.

Is this a case where the object is inappropriate, or perhaps a case where I simply haven't gone far enough, and more of the program should be "objectified" in order to see the benefits.

Chip

In article <39991735.7D7D9C6F@dkrz.de>, Martin Schultz
<martin.schultz@dkrz.de> wrote:

> Pavel,
>
>   these are valuable comments. However, I disagree that objects are
> only useful for
> N >= 50 as you indicate. Especially when using inheritance, they have
> tremendous power
> even for N = 1 if you like some uniform access to "things" that belong
> to the same category

> but may all be a little different. Certainly you are right that
> structures still have their
> value, and you don't need an object for EVERYTHING in IDL. Yet, I have
> the feeling that
> your comment misses the point in answering Chip's question, and that
> this is probably
> because the question itself is flaky. I haven't pondered over this day
> and night, but
> I dare to say that you made something wrong when you need to know the
> fields that are
> stored in the object. Object oriented programming does require
> substantial twists of
> the sequential programmer's mind in terms of program structure. As
> always, the design of a
> program is the most important step in programming, but for objects it is
> even more
> important. Every object field (structure element) that you want to
> access from outside
> must be explicitely interfaced for a reason (encapsulation). All other
> fields should
> be regarded as private properties of the object for use only by the
> object itself.
> If you need access to object fields, the GetProperty, SetProperty
> convention also
> allows you to distinguish between Readonly and Writeonly access, simply
> by including
> or emitting respective keywords. Sure, OOP requires more typing than a
> classical program,
> but so far I still think I am rewarded in the end by a product that is
> much more powerful
> than a classical program of similar complexity. And if I had wanted to
> include all that
> power into a classical program, I probably would have had typed at least
> as much.
>
> Cheers,
> Martin
>
>
> Pavel Romashkin wrote:
>>
>> I have a feeling that Chip's question is not quite answered by the dump
>> truck full of technicalities the group readily poured out. It looks to
>> me that the question is, "my object is just a structure but it is hell
>> of a lot harder for me to use it than a regular structure".
>> I would say that if all you need is the data and there is one instance
>> of it, you do not need an object. The whole idea is that object is a
reuseable.

>> For example, you have one data processing task. You have a vector map
>> and an overlayed image. You want to be able to access the data for that
>> image only. Might as well write plain code for it.
>> Now, you have the same map, but 50 images that will be processed
>> similarly. In this case, if you write an object, then you can write the
>> "25 methods" only once, and very elegantly call those methods in your
>> widget event processing code. This way, you have intact data and a set
>> of actions (methods) you can perform on that data. And, if you happened
>> to aquire more bands of the image(s), you do not need to alter the code.
>> You just add more objects that use the same methods. On the other hand,
>> if you create a structure (or array of such), then you can't as easily
>> add another elements to it. Actions (code that modifies the data) tend
>> to become less organized (because they are not linked to data), and
>> event handlers become very complex.
>> This is not to mention inheritance, which simplifies dramatically
>> operations on data that are derivative or daughter to the main object class.
>> Cheers,
>> Pavel
>>
>> Chip Sample wrote:
>>>
>>> I must admit that based on comments from this list, I have experimented
with
>>> the object features of IDL for the past week or so, and have implemented
>>> them in a few places in a fairly big widget code I have written.
>>>
>>> My initial observations are that there seems to be less of a temptation to
>>> use common blocks when objects are used. On the other hand my first
>>> impression was that an object is a structure whose fields can not be
>>> accessed until you write additional "methods" to get at each and every damn
>>> one of them. So my object was littered with about 25 "methods" just so I
>>> could pry the data out of the object.
>>>
>>> I eventually came on a work around to write a "proto_object" with a method
>>> allowing you to pass a string containing a tag name which returns the
>>> contents of the field with that tag name. This "proto_object" is inherited
>>> by all other objects I create just so I can use this method. Along the way
>>> I found that the TAG_NAMES function in IDL doesn't work for objects so I
had
>>> to create one. It basically copies the object structure into a regular
>>> structure so the TAG_NAMES can be used.
>>>
>>> Am I making this too hard?
>>>
>>> Chip
>
> --
> [[[[[[[[[[

> [[Dr. Martin Schultz   Max-Planck-Institut fuer Meteorologie   [[
> [[Bundesstr. 55, 20146 Hamburg   [[
> [[phone: +49 40 41173-308   [[
> [[fax: +49 40 41173-298   [[
> [[martin.schultz@dkrz.de   [[
> [[[[[[[[[[

Subject: Re: object newbie
Posted by Martin Schultz on Wed, 16 Aug 2000 07:00:00 GMT
View Forum Message <> Reply to Message

Chip Sample wrote:
>
> Maybe if I provided a few more details, it would be apparent whether I've
> missed the boat.
>
> I have a widget that is used to read and graph data from a radar model
> written in C. I also use the widget to launch and control runs of this C
> program on many machines across the network (thereby insulating me from
> UNIX), as well as browsing and editing the input files and creating
> additional runs according t a trade space. This all was developed without
> objects.
>
> Based on discussion here a curiosity I decided to begin to "objectify" this
> widget and began with the structure that contains "jobs", i.e. the path/name
> of the executable, the path/name of the target file, the path/name of a dump
> file containing what would normally go to the screen, a list of program
> options, the lun being used, and IDs of some control widgets associated with
> that job. These widgets can be used to kill the remote job, look at the
> dump and output files, and look at a status display containing most of the
> information on the job object. So to pass these fields into the display
> widgets, I need access to most of the field values.
>
> Is this a case where the object is inappropriate, or perhaps a case where I
> simply haven't gone far enough, and more of the program should be
> "objectified" in order to see the benefits.
>
> Chip
>

Hi Chip,

  oh yes, I think you are swimming about 50 yards next to the boat -
almost
there but still struggling ;-) As far as I can see from your
description, the
problem is as I said a conceptual one: In objectworld, you don't pass

data to
a widget, instead the object includes the widget. In your case you could have an
object structure for a single job which might vaguely resemble this:

```
   obj_class = { cs_job,  $
               exepath : '',  $    ; path name of executable
               targetpath : '',  $ ; target path
               dumppath : '',   $  ; path of screen dump file
               options : Ptr_New(), $  ; program options
               tlb : -1L, $   ; widget ID of top level base
               idexepath : -1L,  $  ; widget ID of exepath text field
            ...   etc.
             }
```

Then you write a method named cs_job::GUI or something similar which build the widget and
passes it to XManager (with /no_block). Take a look at David's web site how to best control
widget events (I particularily like the idea of generating messages which contain the object
reference and method name). If you destroy the widget, you set self.tlb to -1, and in your
cleanup method, you test IF self.tlb GE 0 THEN widget_control, self.tlb,/DESTROY. The widget
that this object would build, controls only one job - it could also be a compound widget
which would then be included in ... see below.

Now, if you want to manage several jobs, you probably want to use a container object which
you may derive from the original IDL_Container, or you can use my MGS_Container if you like
to access the jobs by name. You can either write an object that uses a container, or one that
inherits from a container (I don't have enough experience to favour one over the other).
This object would also include widget ID's and a GUI method, this time you build a GUI
which lists all jobs that are currently defined, or all jobs that are active, etc.

If you proceed this way, the only data that you need to pass from the individual job objects
to the job container object are the data that you want to display in the list widget. If you
like to see detailed information about all jobs at once, you can also have some keyword like

jobdescription in cs_job::GetProperty which would return a string that
is formatted ready
to go into the list widget.

Hope, this helps,
Martin

PS: please take RSI's advice seriously and name all your objects
beginning with your initials
(three letters are best). This tremendously decreases the likelihood of
name conflicts and may
one day lead to a nice collection of objects from various people that
can actually interact
with each other (dream, dream, ...). OK: at least all objects that have
any slight chance to
end up in some library at some point -- and I could conceive job control
being one of these things!

--
[[[[[[[[[[[
[[Dr. Martin Schultz   Max-Planck-Institut fuer Meteorologie   [[
[[Bundesstr. 55, 20146 Hamburg   [[
[[phone: +49 40 41173-308   [[
[[fax: +49 40 41173-298   [[
[[martin.schultz@dkrz.de   [[
[[[[[[[[[[[

Subject: Re: object newbie
Posted by promashkin on Thu, 17 Aug 2000 07:00:00 GMT
View Forum Message <> Reply to Message

marc wrote:
>
> ...
>> If you destroy the widget, you set self.tlb to -1, and in your
>> cleanup method, you test IF self.tlb GE 0 THEN widget_control,
>> self.tlb,/DESTROY.
> Nearly irrelevant, but just for completeness:
> Better put self.tlb to 0. Only 0 is garanteed to never be a widget ID.
> (Somewere I saw this in the documentation (al least implicit))
> and better check with widget_info(self.tlb,/VALID_ID).
> Then you did not even need to set it to 0.
> (But better do since a widget ID might be reused (perhaps same
> probability as that it might be -1 ;-))
> :-) marc

I thought that object definition statement can have any number as a field definition, and it won't matter, because when new object is created, they all will be set to zero anyway. Top base ID will have to be placed in that field once it is created, right?

Cheers,
Pavel

Subject: Re: object newbie
Posted by marc schellens[1] on Thu, 17 Aug 2000 07:00:00 GMT
View Forum Message <> Reply to Message

...
> If you destroy the widget, you set self.tlb to -1, and in your
> cleanup method, you test IF self.tlb GE 0 THEN widget_control,
> self.tlb,/DESTROY.
Nearly irrelevant, but just for completeness:
Better put self.tlb to 0. Only 0 is garanteed to never be a widget ID.
(Somewere I saw this in the documentation (al least implicit))
and better check with widget_info(self.tlb,/VALID_ID).
Then you did not even need to set it to 0.
(But better do since a widget ID might be reused (perhaps same probability as that it might be -1 ;-))
:-) marc

Subject: Re: object newbie
Posted by Martin Schultz on Mon, 21 Aug 2000 07:00:00 GMT
View Forum Message <> Reply to Message

Pavel Romashkin wrote:
>
> marc wrote:
>>
>> ...
>>> If you destroy the widget, you set self.tlb to -1, and in your
>>> cleanup method, you test IF self.tlb GE 0 THEN widget_control,
>>> self.tlb,/DESTROY.
>> Nearly irrelevant, but just for completeness:
>> Better put self.tlb to 0. Only 0 is garanteed to never be a widget ID.
>> (Somewere I saw this in the documentation (al least implicit))
>> and better check with widget_info(self.tlb,/VALID_ID).
>> Then you did not even need to set it to 0.
>> (But better do since a widget ID might be reused (perhaps same
>> probability as that it might be -1 ;-))

>> :-) marc
>
> I thought that object definition statement can have any number as a
> field definition, and it won't matter, because when new object is
> created, they all will be set to zero anyway. Top base ID will have to
> be placed in that field once it is created, right?
>
> Cheers,
> Pavel

well, no. At least I always like to be sure what is stored in my
fields, hence I rather initialize the ID in my Init method. But
using zero instead of -1 seems ok.

Martin

--
[[[[[[[[[
[[Dr. Martin Schultz   Max-Planck-Institut fuer Meteorologie
[[
[[Bundesstr. 55, 20146 Hamburg
[[
[[phone: +49 40 41173-308
[[
[[fax: +49 40 41173-298
[[
[[martin.schultz@dkrz.de
[[
[[[[[[[[[[