Subject: Re: Are user-defined private methods possible?
Posted by David Fanning on Wed, 08 Jul 2009 17:59:05 GMT
View Forum Message <> Reply to Message

Paul van Delst writes:

> As the subject states, are user-defined private methods possible in IDL?
>
> I have some object methods that I want to be private to the class (like Init and Cleanup)
> but I can't find any info on how to do that in the IDL help. Once the methods in question
> are invoked (and, thus, compiled) in a public method, they're accessible from outside the
> class.

There is nothing like a "private" method in IDL.
However, there are some things you can do to keep
neighborhood eyes from oggling the babes in the
backyard pool.

For example, you can give your "private" methods
ugly names (I usually append an underscore to the
method name), and you can set the HIDDEN compile
option, so the casual user doesn't know it is there
when the object code is compiled:

```
PRO myobject::_myprivateMethod
  compile_opt HIDDEN
  ...
END
```

But nothing is going to stop a determined 14-year-old
with a good set of field glasses. :-(

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: Are user-defined private methods possible?
Posted by Foldy Lajos on Wed, 08 Jul 2009 18:56:11 GMT
View Forum Message <> Reply to Message

On Wed, 8 Jul 2009, Paul van Delst wrote:

> As the subject states, are user-defined private methods possible in IDL?
>
> I have some object methods that I want to be private to the class (like Init
> and Cleanup) but I can't find any info on how to do that in the IDL help.
> Once the methods in question are invoked (and, thus, compiled) in a public
> method, they're accessible from outside the class.

You can not prevent calls to your "private" methods, but you can check
whether the caller is a method of your class or not:

```
; cut here, copy to test.pro and .ru test
;
pro o::private
catch, stranger
if stranger ne 0 then $
begin
    catch, /cancel
    goto, prn
endif

if not obj_isa((scope_varfetch('self', level=-1)), 'o') then stranger=1

prn:
if stranger ne 0 then print, 'private call failed' $
                 else print, 'private call succeeded'
end


pro o::public
self->private
end


s={o, i:0}
o=obj_new('o')
print, 'calling private from class method:'
o->public
print, 'calling private directly:'
o->private
end

; cut here
```

regards,
lajos

Subject: Re: Are user-defined private methods possible?
Posted by Paul Van Delst[1] on Wed, 08 Jul 2009 19:16:02 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
> Paul van Delst writes:
>
>> As the subject states, are user-defined private methods possible in IDL?
>>
>> I have some object methods that I want to be private to the class (like Init and Cleanup)
>> but I can't find any info on how to do that in the IDL help. Once the methods in question
>> are invoked (and, thus, compiled) in a public method, they're accessible from outside the
>> class.
>
> There is nothing like a "private" method in IDL.
> However, there are some things you can do to keep
> neighborhood eyes from oggling the babes in the
> backyard pool.
>
> For example, you can give your "private" methods
> ugly names (I usually append an underscore to the
> method name), and you can set the HIDDEN compile
> option, so the casual user doesn't know it is there
> when the object code is compiled:
>
>   PRO myobject::_myprivateMethod
>   compile_opt HIDDEN
>   ...
>   END
>
> But nothing is going to stop a determined 14-year-old
> with a good set of field glasses. :-(

Cool, thanks!

cheers,

paulv

Subject: Re: Are user-defined private methods possible?
Posted by Paul Van Delst[1] on Wed, 08 Jul 2009 19:17:47 GMT
View Forum Message <> Reply to Message

Fï¿½LDY Lajos wrote:
>
> On Wed, 8 Jul 2009, Paul van Delst wrote:
>

>> As the subject states, are user-defined private methods possible in IDL?
>>
>> I have some object methods that I want to be private to the class
>> (like Init and Cleanup) but I can't find any info on how to do that in
>> the IDL help. Once the methods in question are invoked (and, thus,
>> compiled) in a public method, they're accessible from outside the class.
>
> You can not prevent calls to your "private" methods, but you can check
> whether the caller is a method of your class or not:
>
> ; cut here, copy to test.pro and .ru test
> ;
> pro o::private
> catch, stranger
> if stranger ne 0 then $
> begin
>   catch, /cancel
>   goto, prn
> endif
>
> if not obj_isa((scope_varfetch('self', level=-1)), 'o') then stranger=1
>
> prn:
> if stranger ne 0 then print, 'private call failed' $
>   else print, 'private call succeeded'
> end
>
>
> pro o::public
> self->private
> end
>
>
> s={o, i:0}
> o=obj_new('o')
> print, 'calling private from class method:'
> o->public
> print, 'calling private directly:'
> o->private
> end
>
> ; cut here

schweet! I'll see if I can put this code in my error handler include file for use....

Thanks,

paulv

Subject: Re: Are user-defined private methods possible?
Posted by Oliver on Fri, 10 Jul 2009 08:33:48 GMT
View Forum Message <> Reply to Message

cool - I'm always amazed about the creativity of people when it comes
to circumvent all those IDL annoyances.

For my lib I used the much simpler approach of defining a 'magic'
number, something along the lines of

```
pro myClass::privateMethod, arg1, arg2, magic
  if( magic ne 148 ) then message, 'private calls only...'
  ; ...
end
```

This is obviously no protection at all against someone who on purpose
wants
to call this method, but then its his/her own fault and I would
consider this
bad style and a programming error.

Speaking about mimicking OO concepts in IDL: what also annoys me is
the absence of static methods and class variables.
For the former I use the convention

```
pro myClass_staticMethodName
```

i.e. by mixing "classic" IDL and object orientation. Ugly but it
works.
For the latter I use commons:

```
common myClassStaticVars initializedFlag, var1, var2, var3
```

The problem with this approach is that variables cannot be initialized
here.
So I define a method

```
pro myClass_initializeStaticVars
```

which I either call at program start or (if there're not too many)
in the constructor and all 'static methods' of the class.
InitializedFlag
is set to '1' during the first call so repeated calls have no effect.

Cheers
Oliver