

---

Subject: Re: [Object IDL] routines that require user-supplied functions ...  
Posted by [Phillip & Suzanne\[2\]](#) on Tue, 11 Aug 1998 07:00:00 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

I've run across similar problems. In particular, XManager expects to have functions/procedures for event handlers. The way I've been dealing with the situation is to write a function/procedure that's defined in the same file as my class, but uses the IDL notation for defining a class. So, in your case, I would call the function test4\_\_poly instead of test4::poly (replace the colons with underscores). This naming convention establishes test4\_\_poly as a "friend" of the class for anyone reading the code, while still allowing it to be accessed as a standard procedure/function. In IDL 5.1, RSI realized that this was a problem, and added the Call\_Method routine which allows an object's method to be specified as a string. However, this won't work with canned routines.

You might be able to cobble together a way of invoking the object's poly routine if you can obtain a reference to the object from within another routine. If you can get an object reference, you could then do something like this:

```
function test4__poly, x
; Add some code to find the object being referred to
; I'm not sure how to accomplish this, but if you can
; obtain a reference to the object, this would work in
; a roundabout way.
return, obj->poly, x
end
```

In either case, good luck.

Phillip David

dEdmundson@Bigfoot.com wrote:

```
>
> Here is a *demo* object-IDL code I wrote to illustrate a problem.
> This object integrates x^n over the interval (a,b) using the
> QROMB function. QROMB requires a user-defined function but I
> cannot manage to pass the 'poly' method.
```

Note that the requirement for QROMB is a user-defined FUNCTION, not METHOD. While the two seem like they're interchangeable, they aren't. A function doesn't have its own data, so it doesn't need an object reference. Methods (can) use data internal to the object as well as arguments passed at the command line. Therefore, they need not only to be invoked, but provided with an object reference as well. The arrow notation for objects provides the object in question.

```
> While this is a contrived example, one often wants to pass
> object method functions/procedures to other IDL routines.  Is
```

```

> there a generic way of passing object methods to such intrinsic
> routines?
>
> Cheers,
> Darran.
>
> ;;-----
> ;; save the following as test4__define.pro somewhere in your IDL path
> ;; uncomment one of the return statements in test4::integral
> ;; invoke the object with t = obj_new('test4',3.0,0.0,1.0)
>
> function test4::init, degree, a, b
>   self.degree = degree
>   self.a = a
>   self.b = b
>   print, 'Integral = ', self->integral()
>   return, 0
> end
>
> function test4::integral
> ; try both of these ...
> ; return, qromb('test4::poly', self.a, self.b)
> ; return, qromb('self->poly', self.a, self.b)
> end
>
> function test4::poly,x
>   return, x^(self.degree)
> end
>
> pro test4__define
>   struct = {test4, degree:0.0, a:0.0, b:0.0}
> end

```

---

Subject: Re: [Object IDL] routines that require user-supplied functions ...

Posted by [davidf](#) on Tue, 11 Aug 1998 07:00:00 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

Darran Edmundson (dEdmundson@Bigfoot.com) writes:

```

> Here is a *demo* object-IDL code I wrote to illustrate a problem.
> This object integrates  $x^n$  over the interval (a,b) using the
> QROMB function. QROMB requires a user-defined function but I
> cannot manage to pass the 'poly' method.
>
> While this is a contrived example, one often wants to pass
> object method functions/procedures to other IDL routines. Is
> there a generic way of passing object methods to such intrinsic

```

> routines?

I don't believe there is a generic way of passing object methods to intrinsic routines unless (perhaps) they are written in such a way as to expect them. It is clear QROMB has not been.

Nor have I found in the 15 minutes or so that I fooled around with this a completely satisfactory "object-like" way to solve this problem, although I did find a contrived solution that uses a common block.

The idea is this: Put the self object in a common block that can be declared in the POLY function. (It is a restraint of the QROMB routine that POLY be defined with one and only one parameter and that it be a vector of values for which the function is solved.) This gets the self object into the POLY function. BUT...not as a structure, as an object reference.

This means that the equation to be solved can't look like this, as it does in Darran's code:

```
x^(self.degree)
```

because the data of an object is hidden and is accessible only by its methods. So I had to write a Get\_Degree method for the object that returns the degree of the function.

My solution looks like this:

```
function test4::init, degree, a, b
common polycommon, xx
; ~~~~~~
self.degree = degree
self.a = a
self.b = b
xx = self
; ~~~~~~
print, 'Integral = ', self->integral()
return, 1
end

function test4::integral
return, qromb('poly', self.a, self.b)
end

function test4::get_degree
```

```
return, self.degree
end
;~~~~~
```

```
function poly, x
common polycommon, obj
;~~~~~
return, x^(obj->get_degree())
; ~~~~~
end
```

```
pro test4__define
  struct = {test4, degree:0.0, a:0.0, b:0.0}
end
```

This object can be created and called like this:

```
t = obj_new('test4',3.0,0.0,1.0)
```

Cheers,

David

--

David Fanning, Ph.D.

Fanning Software Consulting

E-Mail: [davidf@dfanning.com](mailto:davidf@dfanning.com)

Phone: 970-221-0438, Toll Free Book Orders: 1-888-461-0155

Coyote's Guide to IDL Programming: <http://www.dfanning.com/>

---