Subject: Re: IDL Object for creating a Singleton
Posted by mallors on Tue, 23 Jun 1998 07:00:00 GMT
View Forum Message <> Reply to Message

In article <358E06D3.AE9D606F@earthling.net>,
Phillip & Suzanne David <pdavid@earthling.net> writes:
> I've been doing a fair amount of reading on OO design and methodology. One of
> the books I've had pointed out to me was 'Design Patterns', which deals with
> OO algorithm design quite nicely. One of the patterns the authors describe is
> a Singleton -- A class for which only one instance exists. If the sole
> instance exists, the constructor function (object's init function in IDL) will
> return a reference to the existing object. If none exists, a new object is
> created and returned.
>
> This structure is particularly useful when there is no reason to have multiple
> objects around. I am trying to create a user preferences object that fits
> this criteria very nicely. I'd like to be able to store user preferences for
> screen size in an object. If that object exists, other screens can size
> themselves appropriately. The book gives sample C++ code that relies on class
> variables and class methods (i.e., not associated with a specific instance of
> the object, but with the object itself). IDL doesn't seem to have any way to
> do something similar. Does anyone know of a way to do this?
>
> My first attempt uses a common block called preferencesclassvars to store an
> object reference. This common block is called from a function that, while not
> part of the object, is stored in the same file. If the reference contained in
> the common block is not an object reference, then I know the object hasn't
> been initialized, and I set it to a new instance of the object. Otherwise, I
> return the object reference from the common block.
>

The implementation of the seemingly simple Singleton
pattern is not so easy in IDL since there is no such
thing as a C static variable, as J.D. has pointed out.
To implement the Singleton in IDL, one requires global
access to the Singleton object reference. Common blocks are
one obvious solution, but here is another bare-bones
implementation of the Singleton that does not use a
common block. Instead, it stores the object reference in
a read-only system variable. The system variable acts as
a "static" reference.

```
; Initalize the singleton
;
FUNCTION singleton::init

    ; Store the object reference
```

```
   ;
   READ_ONLY = 1
   DEFSYSV, '!objSingleton', self, READ_ONLY

   RETURN, 1

END ; init


; Implementation of a Singleton object
;
PRO singleton__define

   obj = { singleton, $

      ; Some instance data required
      ;
      instanceData: 0 $
   }

END ; define


FUNCTION SINGLETON

   DEFSYSV, '!objSingleton', EXISTS = exists
   IF (exists EQ 0) THEN BEGIN

     o = OBJ_NEW ('singleton')

   ENDIF ELSE BEGIN

     IF (OBJ_VALID (!objSingleton)) THEN BEGIN

       o = !objSingleton

     ENDIF ELSE BEGIN

       o = OBJ_NEW ('singleton')

     ENDELSE

   ENDELSE


   RETURN, o
```

```
END
```

This code has some limitations:

1. You have to create the singleton with

   ```
   mySingleton = SINGLETON ()
   ```

   instead of the usual OBJ_NEW ()

2. If you want several Singletons, the SINGLETON function will have to be modified to use a unique system variable for each. Even instantiating, destroying, then instantiating again will not work as it is now.

If you are just starting out using the Gamma book with IDL, don't get discouraged by the difficulty implementing the Singleton in IDL. I have implemented several other patterns from that book in IDL with good results. The implementations shown in C in that book are just that - implementations. The focus of the book in on *design techniques*, not implementations. All you need is to reproduce the functionality of the design pattern, and this can be done in IDL for many of the patterns. Concentrate on object delegation, which is easily handled in IDL.

For anyone unfamiliar with the Gamma book:

Design Patterns, Elements of Reusable Object-Oriented Software
Eirch Gamma, et al.
Addison-Wesley
ISBN 0-201-63361-2

By the way, I have not used the DEFSYSV function too much in any code yet, but it seems like an ideal place to store a structure of read-only widget IDs. You then have access to all widget IDs anywhere in your code, and the read-only nature makes it safe, unlike the COMMON block.

> This structure is particularly useful when there is no reason to have multiple
> objects around. I am trying to create a user preferences object that fits
> this criteria very nicely. I'd like to be able to store user preferences for

I have written a Preferences object that manipulates user preferences. It
is available from my web page

   http://cspar.uah.edu/~mallozzir/idl/idl.html

Regards,

-bob

--
Robert S. Mallozzi
http://cspar.uah.edu/~mallozzir/
