Subject: Re: Destroying objects
Posted by JD Smith on Tue, 17 Feb 2004 20:30:20 GMT
View Forum Message <> Reply to Message

On Mon, 16 Feb 2004 15:06:35 -0700, David Fanning wrote:

> JD Smith writes:
>
>> I wonder if you can expand on this error handling technique a bit.
>> One feature I like to have is the ability for errors to "just work",
>> regardless of if the object is being used on the command-line, or via
>> its GUI. In the latter case, errors should be trapped and displayed
>> in a pop-up. In the former case, they should call MESSAGE to print to
>> the command line and exit. I've managed it so far by having a
>> super-class which implements Error, Warning, and other methods, which
>> checks to see if you are in a command-line invocation or not, and uses
>> message or dialog_message as appropriate. Suitable use of CATCH
>> allows deep errors in other routines further down the stack (ones you
>> didn't write, for instance) to be handled in the same way, but that
>> requires some care to "set the trap" when necessary. This is somewhat
>> similar to the way XManager can catch and disregard errors (as it does
>> by default), but with reporting, and valid even when XManager isn't
>> running.
>
> I would have thought you knew me well enough by now, JD,
> to realize that if I don't immediately provide the IDL
> code to solve a problem I'm either trying to get rich (ha!)
> or the solution is so simpleminded I prefer to not risk
> certain embarrassment. Alas, it is the latter in this case. :-(
>
> We don't do anything fancy. All our methods contain standard
> error handler code in them, added with a @error_handler
> sort of call. (We originally had one code fragment because I figured
> out a really slick way to tell if I was in a procedure or function
> and I could either return something or not, but the method
> used an EXECUTE function. When we wanted to run on the VM,
> we had to go to two error handlers: one for procedures
> and one for functions. What a shame!) The error handler
> simply sets up a CATCH for the method. If an error is caught,
> it is dispatched to our ERROR method, which is attached to
> the "atom" object all objects inherit.
>
> When we get an error we can handle in our error handler,
> we add a "handled" to the front of the error message.
> (I told you this was simpleminded.) As the error propagates
> back up the object chain, if the error handler sees a "handled"
> in front of the message it returns immediately.
>

> I can't think of a way to tell if the object is running
> at the IDL command line or not (I've shown you mine, you
> should show me yours), but our error handler can write to
> a log file (if there is one) or to the display. Typically,
> we issue a pop-up dialog (DIALOG_MESSAGE) with a short
> error message, and write a better formatted traceback of
> the error in the command output log of the display.

Interesting method. I would worry that putting an explicit CATCH in each method might bog things down, if your events typically make deeply nested excursions through dozens of routines (as mine often do). On the other hand IDL's native routine-based error-propagation is fast, but inflexible. I have a few methods for getting the best of both worlds by putting a top-level CATCH-based handler at some high-level, but this is complicated by non-blocking apps.

Since I'm not planning to get rich off of it, I don't mind showing you my simple method for discriminating between command-line and GUI. I require inheriting classes to override a "ReportWidget" method, like this:

```
 ;=============================================================
==================
;  ReportWidget - Return the widget to test the validity of to decide
;                 between graphical and text-based errors/warnings.
;                 Should be overridden.  By default, looks for a class
;                 tag "wInfo" which is a ptr to an info structure with
;                 a "Base" tag.
 ;=============================================================
==================
function ObjReport::ReportWidget
  catch, err
  if err ne 0 then begin
     catch,/cancel
     message, "No class tag pointer `wInfo' with tag `Base' found.  " + $
             "Override the ReportWidget Method?"
  endif
  if ptr_valid(self.wInfo) then return,((*self.wInfo).Base)
  return,-1L
end
```

As it turns out, I often use a detachable self.wInfo field for all widget info, and the top level base is often attached to a "Base" tag inside of that, so I don't typically need to override this, and the default works fine. Then testing for the presence of the GUI is a simple matter of:

```
 ;===========================================================
==================
;  isWidget - Do we use a widget for popup messages?
 ;===========================================================
==================
function ObjReport::IsWidget
  self.or_widget=self->ReportWidget() ;re-get the widget, it could be changing
  return,widget_info(self.or_widget,/VALID_ID)
end
```

JD