Subject: Re: Surprising Odds and Ends
Posted by davidf on Mon, 14 Aug 2000 07:00:00 GMT
View Forum Message <> Reply to Message

Mark Hadfield (m.hadfield@niwa.cri.nz) writes:

> This has been discussed on the group before, but I'm not keen on excessively
> enthusiastic error handling in code. If in doubt, stop where the error
> occurred and let the user sort it out!

Spoken like a true programmer!

But I've seen some of your code, Mark, and I know
you are a softie. *Lots* of comments in there. Shame
on you! :-)

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting
Phone: 970-221-0438 E-Mail: davidf@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155

Subject: Re: Surprising Odds and Ends
Posted by promashkin on Mon, 14 Aug 2000 07:00:00 GMT
View Forum Message <> Reply to Message

I agree on how widespread Heap_gc needs to be in your code. To me, it is
a command line thing that is useful to see if your program is "leaking",
which may happen. I run it after quitting my programs while in
development, to see if it finds something (I guess, help, /heap would do
it but then I'd have to clean up anyway). I have, however, a related
note. When my program crashes or freezes (or I had a bug that caused
"leaking"), I call "Clear IDL" from the "RUN" menu. In essence, it is
the same as resetting Xmanager (thus killing all widgets) and running
Heap_gc, /Verbose:

```
widget_control, /reset & close, /all & heap_gc, /verbose & retall
```

I noticed that sometimes the first "Clear IDL" call does not print out
any pointer (or object) information. Immediate second call, however,
does find lost heap variables. I guess this is exactly what David
explained. Somewhere a managed widget reflection still exists, and the

first call resets Xmanager and kills that widget, releasing the State
structure. You'd think Heap_gc would wipe the pointers in it. However,
RETALL is the last command, and Heap_gc does not find any leaks when it
is called from a last routine's level. Second "Clear IDL" finishes the
pointers off.
It seems that moving RETALL to the third position in "Clear IDL" call
would make sense. I know, I know, mucho guys don't use them wimpy menus.
They have RETALL and HEAP_GC programmed into their keyboards :-)
However, I found one nice thing about Run menu: it is available even
when your event handler gets into an infinite loop by (your) mistake,
and does not even respond to keyboard break event.
Cheers,
Pavel

Subject: Re: Surprising Odds and Ends
Posted by davidf on Mon, 14 Aug 2000 07:00:00 GMT
View Forum Message <> Reply to Message

David Fanning (davidf@dfanning.com) writes:

> 3. The HEAP_GC command (which, heaven forbid, you *really*
> shouldn't be using anyway) is dependent on program
> level. For example, in Cleanup routines I want to
> destroy pointers in my info structure. But if the
> program crashes in an event handler, the info structure
> is undefined in the Cleanup routine. In such a case
> I might want to clean up the pointers by issuing a HEAP_GC
> command. But I could never get this to work. Today I
> found out why. The heap apparently exists *only*
> at the main IDL level. If you try to call Heap_GC from
> some other level (e.g., inside a Cleanup routine) the
> command appears to work, but nothing really happens. This
> command can *only* be used from the IDL command line.

Oh, dear. While my explanation is consistent with
the facts, it is too fanciful by far. (I always
did prefer William Blake's burning tiger's eyes
to Occum's Razor.) In fact, HEAP_GC can be called
from any program level. And it really does search
every variable for a heap reference.

Here is what was happening to make me think differently.
(If you really care--and I told you not to be using
HEAP_GC anyway!--I have a test program you can run.)
I crashed my widget program in an event handler with
the info structure checked out with NO_COPY. I stop,
as I am suppose to, in the event handler module where

the info structure *is* defined, which has the pointer
reference inside it.

I now destroy the widget. My cleanup routine gets called
and the code is executed, including the HEAP_GC which
is in there if the info structure is undefined inside
the cleanup routine, which it is. My pointer does not
get cleaned up. The question is, why not?

The reason appears to be that the Cleanup routine is
called from *within* the stopped event handler module.
But when the Cleanup routine exits, I am back in the
event handler module. Of course, the info structure
exists here, so the previous HEAP_GC didn't clean
up the pointer.

If I type HEAP_GC at the IDL command line, the pointer
is still there, because I am *still* in the event
handler. If I type RETALL & HEAP_GC the pointer
does get cleaned up because now I have exited the
event handler module. The RETALL is what made me
think HEAP_GC was program level dependent.

I makes sense to me, sort of, although how and when modules
are getting called has always been the big MYSTERY to me,
and is one of the reasons I think of widget programming
as magical.

The bottom line is that HEAP_GC doesn't belong in
your code (which is what I have been saying ever
since it first appeared on the scene). What you should
do is CATCH errors, and handle things in such a way that
you never have a need for HEAP_GC. And that is what
this chapter in my book is going to be about. :-)

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting
Phone: 970-221-0438 E-Mail: davidf@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155

Subject: Re: Surprising Odds and Ends
Posted by Mark Hadfield on Mon, 14 Aug 2000 23:12:13 GMT
View Forum Message <> Reply to Message

> The bottom line is that HEAP_GC doesn't belong in
> your code

Agreed.

> ..What you should
> do is CATCH errors, and handle things in such a way that
> you never have a need for HEAP_GC. And that is what
> this chapter in my book is going to be about. :-)

I'd better buy your book when it comes out :-) Until then, HEAP_GC and its
friends RETALL, WIDGET_CONTROL,/RESET and CLOSE,/ALL are mighty handy from
the command line when cleaning up after an error.

This has been discussed on the group before, but I'm not keen on excessively
enthusiastic error handling in code. If in doubt, stop where the error
occurred and let the user sort it out!

Mark Hadfield
m.hadfield@niwa.cri.nz http://katipo.niwa.cri.nz/~hadfield/
National Institute for Water and Atmospheric Research
PO Box 14-901, Wellington, New Zealand

Subject: Re: Surprising Odds and Ends
Posted by Liam E. Gumley on Tue, 15 Aug 2000 07:00:00 GMT
View Forum Message <> Reply to Message

Mark Hadfield wrote:
>
>> The bottom line is that HEAP_GC doesn't belong in
>> your code
>
> Agreed.
>
>> ..What you should
>> do is CATCH errors, and handle things in such a way that
>> you never have a need for HEAP_GC. And that is what
>> this chapter in my book is going to be about. :-)
>
> I'd better buy your book when it comes out :-) Until then, HEAP_GC and its
> friends RETALL, WIDGET_CONTROL,/RESET and CLOSE,/ALL are mighty handy from
> the command line when cleaning up after an error.

>
> This has been discussed on the group before, but I'm not keen on excessively
> enthusiastic error handling in code. If in doubt, stop where the error
> occurred and let the user sort it out!

My colleague Paul van Delst recently pointed (ha!) me to some features
of the PTR_VALID function which help in identifying and reclaiming
dangling references (i.e. heap variables for which no valid pointer
exists):

(1) When no argument is specified, PTR_VALID returns a vector of
pointers to all existing heap variables, regardless of whether a valid
pointer exists for each heap variable,

(2) The PTR_VALID keyword CAST creates a new pointer to the heap
variable index identified in the first function argument.

These debugging methods are a little more subtle than HEAP_GC (which I
agree should *never* be used in IDL programs).

Cheers,
Liam.
http://cimss.ssec.wisc.edu/~gumley

Subject: Re: Surprising Odds and Ends
Posted by davidf on Tue, 15 Aug 2000 07:00:00 GMT

View Forum Message <> Reply to Message

Martin Schultz (martin.schultz@dkrz.de) writes:

```
> Therefore, the solution should be to free the pointer
> within your event handler routine BEFORE calling the
> cleanup routine.
>
> In plain words:
>
> CATCH, theError
> IF theError NE 0 THEN BEGIN
>    CATCH, /Cancel
>    Message,...
>    IF Ptr_Valid(info.thepointer) THEN Ptr_Free, info.thepointer
>    Cleanup
> ENDIF
```

Well, in plain words, this is right, although you can't
(well, "shouldn't" is a more accurate word) call the
Cleanup procedure yourself. The Cleanup procedure is

called when the widget associated with it dies. The
CATCH would more likely look like this:

```
 CATCH, theError
 IF theError NE 0 THEN BEGIN
    CATCH, /Cancel
    Message,...
    IF N_Elements(info) NE 0 THEN $
       Widget_control, event.top, Set_UValue=info, /No_Copy
 ENDIF
```

> If the crash happens in a routine which you never expected to crash,
> well then you
> apparently caught an oversight by the programmer, and he should be told
> to fix the
> bug. In this case, a .reset_session usually helps ;-)

I'm a big fan of .Reset_Session, but I still see far too
many people in my travels who exit IDL every time they
crash a widget program because they have never heard of
RETALL and they can't ever get a widget program running
after one crashes! Getting those folks hooked on .Reset
would be just as bad. :-(

My problem is that I'm trying to tell a story in
some logical order and the denouement doesn't happen
until the hero gets burned almost beyond recognition
by asking for user input to read a file. Error Handling
is a nurse who brings him back to life and gives him
reason to live again. But I'm several chapters away
from all of that.

Cheers,

David

P.S. Let's just say the working title of my new book
is The IDL Patient. Catchy, don't you think. :-)
--
David Fanning, Ph.D.
Fanning Software Consulting
Phone: 970-221-0438 E-Mail: davidf@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155

Subject: Re: Surprising Odds and Ends

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

David Fanning wrote:
>
> [far too many words about a routine that one should not use anyway ;-)]
> [and then ...]
> I crashed my widget program in an event handler with
> the info structure checked out with NO_COPY. I stop,
> as I am suppose to, in the event handler module where
> the info structure *is* defined, which has the pointer
> reference inside it.
> [...][/color]

Seems to me as if your problem is related to a somewhat sloppy
distinction
between "global" and "local" information. To my understanding, the
Cleanup routine
only exists to clean up "global" mess, i.e. stuff that can be accessed
via a
well-defined interface (may this be the UValue field of TLB or its
child, or
may these be object fields). From what I leardned from your programs ,
the
info structure which gets parsed by the event handler would instead
contain
a local pointer, i.e. something that has a finite lifetime and was
created specifically
to serve an event. Therefore, the solution should be to free the pointer
within
your event handler routine BEFORE calling the cleanup routine.

In plain words:

```
CATCH, theError
IF theError NE 0 THEN BEGIN
  CATCH, /Cancel
  Message,...
  IF Ptr_Valid(info.thepointer) THEN Ptr_Free, info.thepointer
  Cleanup
ENDIF
```

If the crash happens in a routine which you never expected to crash,
well then you
apparently caught an oversight by the programmer, and he should be told
to fix the
bug. In this case, a .reset_session usually helps ;-)

Cheers,

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   [[
[[[[[[[[[