Subject: Re: GUI interface update issues
Posted by Karl Schultz on Wed, 22 Jun 2005 16:22:13 GMT
View Forum Message <> Reply to Message

On Tue, 21 Jun 2005 15:16:19 -0700, Rick Towler wrote:

>
>
> Karl Schultz wrote:
>> On Tue, 21 Jun 2005 09:23:39 -0700, Rick Towler wrote:
>>
>>
>>> Hello group,
>>>
>>> I have an interesting problem with gui redraw on WinXP which maybe
>>> someone can shed some light on.
>>
>>
>> snip
>>
>>
>>> Why does IDL stop updating the gui? Any ideas? While the functions in
>>> the dlm are compute intensive, they aren't particularly complicated and
>>> only rely on some simple macros in an .h file. Just a *lot* of looping
>>> over a moderate amount of data.
>>
>>
>> You might try calling the (documented) IDL_BailOut function. One purpose
>> for this function is to let users have the opportunity to interrupt IDL
>> during a long-running calculation. Many of the IDL internal system
>> routines call IDL_BailOut as they make progress through a calculation. It
>> is probably good form to code your C DLM's to call this function every so
>> often so that users can break out of it.
>>
>> One side effect of IDL_BailOut is to flush window events, which depending
>> on how your app is designed, might help keep the GUI's and progress bars
>> fresh during the long-running operation.
>>
>> I'm not completely certain this will help you, but it is worth a try. It
>> is very easy to give it a shot. If it helps, then spend a little more
>> time to figure out a good policy for calling BailOut so that you don't
>> call it too often or too infrequently. You don't want to slow down your
>> function TOO much.
>
> Thanks for the tip.
>
> It didn't help with the GUI updating issue but it does allow the user to
> "bail out" if needed. Which raises an interesting question... How

> does IDL *really* handle interrupts.
>
> The application structure is pretty basic. There is a main procedure
> which sets up the gui (buttons, menus, windows). The buttons trigger
> the main event handler which calls a function to display a modal dialog
> where params are gathered. Clicking O.K. will return these params to
> the main event procedure where they are passed to the appropriate model
> procedure where the actual calculations are performed. After the model
> procedure runs it returns to the event handler which returns to the main
> routine to wait for the next event. There are no timers. The only
> events are button clicks.
>
> main ->
> event_handler ->
> gather_params modal dialog
> model procedure ->
> some stuff
> for loop
> main calculation here
> lots of IDL code + dlm calls
> endfor
> some more stuff
> event_handler
> main
>
>
> Reading the IDL_BailOut docs leads me to believe that IDL faithfully
> captures interrupts and will break before the next IDL command is
> executed and that the reason for IDL_BailOut is to provide an exit from
> external routines.
>
> But it seems that this is overly simplified. If I try to break my (pure
> IDL) program's execution once it has entered the model procedure it will
> continue to run thru the loop and thru 50 or so more lines of IDL code
> after the loop before it stops at a call to SAVE.
>
> So IDL is capturing the interrupt signal, but there seems to be limits
> on when it will stop execution based on where IDL is in the call stack.
> I would guess that SAVE has a call to IDL_BailOut to which is why my
> program breaks on SAVE. If I didn't have that call to SAVE execution
> would continue back up the call stack to some point where the interrupt
> would be handled?
>
> What's the logic? Is this an issue of how I have structured my program
> or is this how IDL always handles interrupts and I haven't noticed until
> now?
>

IDL doesn't check for pending interrupts like this one when running sequences of non-interactive code or when not doing I/O. I think that the rationale is that if you interrupt the execution of a sequence of non-interactive code, you don't really care about the exact place that sequence gets interrupted, mainly because you have no control at all over it. You have no way of reliably keeping any particular statement from executing when you hit the break key. So, there's no checking during this sequence, which may help interpreter efficiency a bit. And that explains why it didn't stop until you tried to do some I/O.

Back to the original topic: I was probably wrong about BailOut processing widget events. One thing you might want to try is a call to WIDGET_EVENT() with no widget id and /NOWAIT. You can add this call to your for loop in the model procedure. It will cause pending events to be processed and is sort of like waking up XMANAGER for a moment explicitly while the application is too busy to return to XMANAGER widget processing.

Karl
