Subject: Re: IDLVM and retall
Posted by Foldy Lajos on Tue, 05 Sep 2006 15:31:04 GMT
View Forum Message <> Reply to Message

On Mon, 4 Sep 2006, JD Smith wrote:

>
>
> I have a large widget program which runs (optionally) under the IDLVM.
> Around IDL version 6.0, when the VM was introduced, there was an
> inconsistency in how non-blocking widgets were dealt with in the VM
> vs. a normal IDL session, which was fixed in IDL 6.1:
>
> Beginning with IDL 6.1, the XMANAGER procedure honors the value of
> the NO_BLOCK keyword in Runtime and Virtual Machine modes.
>
> Because XMANAGER did not honor the NO_BLOCK keyword in previous
> releases, widget applications that worked properly (that is, not
> blocking) when run in a licensed full version of IDL behaved
> differently when run in Runtime or Virtual Machine mode. This
> difference in behavior has been removed; widget applications should
> behave identically (with regard to blocking behavior) in all IDL
> licensing modes.
>
> Other differences between IDL's Virtual Machine mode and full
> licensed mode, such as the fact that programs that call the IDL
> EXECUTE function will not run in the IDL Virtual Machine, are still
> in effect. If you have modified your widget application to work
> around the old blocking behavior by removing the NO_BLOCK keyword
> from calls to XMANAGER or by substituting the JUST_REG keyword, you
> may need to reinsert the NO_BLOCK keyword to achieve the desired
> behavior.
>
> My program has an error catching system that differentiates between
> two modes: running with a GUI, or running "lights-out" as a script.
> In the former case, the error is caught, displayed for the user with
> DIALOG_MESSAGE, and the RETALL command is then issued to return all
> the way back to the active command line, letting the program continue
> to run. When running as a script, instead MESSAGE is called, and the
> error halts the program.
>
> What I have just discovered, to my dismay, is that RETALL completely
> closes down an IDLVM session, rather than returning to its
> "non-blocking" widget event processing shell, as the above 6.1 update
> note would imply.
>
> Here's an example
>

```
> ;;; file test_vm.pro
> pro test_vm_event, ev
>   a=dialog_message('Testing RETALL',/ERROR)
>   retall
>   return
> end
>
> pro test_vm
>   b=widget_base(/ROW)
>   but=widget_button(b,VALUE='Do It!')
>   widget_control, b,/realize
>   XManager,'test_vm',b,/NO_BLOCK
>   return
> end
>
>
```

> Run TEST_VM in a fresh IDL session, and you'll see a button. Click
> it, get an error message 'Testing RETALL', but the tiny app will
> continue to run (the desired behavior).
>
> Now compile it to a .sav file:
>
> IDL> save,FILE='test_vm.sav',/ROUTINES
>
> quit IDL, and then:
>
> % idl -vm=test_vm.sav
>
> You'll see the same button. Click it, get the 'Testing RETALL'
> message, but notice that now when the message is dismissed and RETALL
> is called, the entire IDLVM session is shut down! I'm not sure why
> this is the case, but I need a way to work around it. Since there are
> many simultaneous event components all injecting events into the
> program suite, there is no convenient place to put a top level CATCH.
> I was relying on the background event processing inside of IDL to
> continue to run, i.e. to serve as an absolute top-level catch, but it
> doesn't do so in the VM.
>
> Suggestions appreciated.
>
> JD
>

probably a CATCH in main helps. It works in IDL 6.2 (linux).

regards,
lajos

```
 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; file test_vm.pro
pro test_vm_event, ev
on_error, 1
  a=dialog_message('Testing RETALL',/ERROR)
  message, /noprint
  return
end

pro test_vm
  b=widget_base(/ROW)
  but=widget_button(b,VALUE='Do It!')
  widget_control, b,/realize
  XManager,'test_vm',b,/NO_BLOCK
  return
end


while 1 do
    catch, status
    if status ne 0 then catch, /cancel
    wait, 1
endwhile

end
```

Subject: Re: IDLVM and retall
Posted by JD Smith on Wed, 06 Sep 2006 18:53:02 GMT
View Forum Message <> Reply to Message

On Tue, 05 Sep 2006 17:31:04 +0200, Fï¿½LDY Lajos wrote:

>
> On Mon, 4 Sep 2006, JD Smith wrote:
>
>> [quoted text muted]
>
> probably a CATCH in main helps. It works in IDL 6.2 (linux).

But that blocks my active command line.

Thanks,

JD

Subject: Re: IDLVM and retall
Posted by Foldy Lajos on Wed, 06 Sep 2006 19:30:08 GMT
View Forum Message <> Reply to Message

On Wed, 6 Sep 2006, JD Smith wrote:

> On Tue, 05 Sep 2006 17:31:04 +0200, FÖLDY Lajos wrote:
>
>>
>> On Mon, 4 Sep 2006, JD Smith wrote:
>>
>>> [quoted text muted]
>>
>> probably a CATCH in main helps. It works in IDL 6.2 (linux).
>
> But that blocks my active command line.
>
> Thanks,
>
> JD
>

your original problem was RETALL not working in the IDL VM, where you have
no command line. You can use RETALL in real IDL, CATCH in the VM. There is
a /VM test in LMGR. I know, the code will be ugly, but will work.

Maybe IDL is right to terminate the VM for RETALL. RETALL can be defined
as "return to the main program level interactive prompt", so I doubt
ITTVIS will fix it.

regards,
lajos

Subject: Re: IDLVM and retall
Posted by JD Smith on Thu, 07 Sep 2006 06:21:54 GMT
View Forum Message <> Reply to Message

On Wed, 06 Sep 2006 21:30:08 +0200, Fï¿½LDY Lajos wrote:

>
> On Wed, 6 Sep 2006, JD Smith wrote:
>
>> On Tue, 05 Sep 2006 17:31:04 +0200, Fï¿½LDY Lajos wrote:
>>
>>>
>>> On Mon, 4 Sep 2006, JD Smith wrote:
>>>

>>>> [quoted text muted]
>>>
>>> probably a CATCH in main helps. It works in IDL 6.2 (linux).
>>
>> But that blocks my active command line.
>>
>> Thanks,
>>
>> JD
>>
>
> your original problem was RETALL not working in the IDL VM, where you have
> no command line. You can use RETALL in real IDL, CATCH in the VM. There is
> a /VM test in LMGR. I know, the code will be ugly, but will work.
>

I thought of that after I responded, but (correct me if wrong), I don't
think you can even have a main level routine in the VM, since your entry
routine must be a named procedure.

JD

Subject: Re: IDLVM and retall
Posted by JD Smith on Thu, 07 Sep 2006 08:57:12 GMT
View Forum Message <> Reply to Message

On Wed, 06 Sep 2006 23:21:54 -0700, JD Smith wrote:

> On Wed, 06 Sep 2006 21:30:08 +0200, Fï¿½LDY Lajos wrote:
>
>>
>> On Wed, 6 Sep 2006, JD Smith wrote:
>>
>>> On Tue, 05 Sep 2006 17:31:04 +0200, Fï¿½LDY Lajos wrote:
>>>
>>>>
>>>> On Mon, 4 Sep 2006, JD Smith wrote:
>>>>
>>>> > [quoted text muted]
>>>>
>>>> probably a CATCH in main helps. It works in IDL 6.2 (linux).
>>>
>>> But that blocks my active command line.
>>>
>>> Thanks,
>>>
>>> JD

>>>
>>
>> your original problem was RETALL not working in the IDL VM, where you have
>> no command line. You can use RETALL in real IDL, CATCH in the VM. There is
>> a /VM test in LMGR. I know, the code will be ugly, but will work.
>>
>
> I thought of that after I responded, but (correct me if wrong), I don't
> think you can even have a main level routine in the VM, since your entry
> routine must be a named procedure.

I eventually resorted to something along these lines:

```
pro test_vm_event, ev
  a=dialog_message('Testing RETALL',/ERROR)
  if lmgr(/VM) || lmgr(/RUNTIME) then message,'TEST-VM-ERROR',/NOPRINT else $
     retall
end

pro test_vm
  command_line=~lmgr(/VM) && ~lmgr(/RUNTIME)
  b=widget_base(/ROW)
  but=widget_button(b,VALUE='Do It!')
  widget_control, b,/realize
  if ~command_line then begin
     XManager,CATCH=0
     catch,err
     ;; Just keep processing events
     if err ne 0b then begin
        XManager
        return
     endif
  endif
  XManager,'test_vm',b,NO_BLOCK=command_line
end
```

I.e. using a blocking XManager call if there is no command line. As
long as the first entry call to XManager in the VM program is
blocking, all subsequent calls (starting other related widgets, for
exmaple) will use the XManager event loop, by-passing the
(quasi-broken, in my opinion) "active command line" widget processing
equivalent in the VM.

Unhandled errors will be sent all the way out to this top level
routine, which can ignore the error and kick start event processing
again with an argument-less call to XMANAGER. If this (or any
subsequent) nested XMANAGER returns, return from the entire program to
avoid double calls to the original entry XMANAGER.

The only other ingredient is turning off XManager's default catch
mechanism, and using MESSAGE to throw an error instead of RETALL.
It's not elegant, but it does the job.

JD

P.S. BTW, Fï¿½ldy, in your original suggestion, the main level CATCH
part was never being compiled: when the compiler gets to TEST_VM, it
stops. The ON_ERROR,1 was what was doing the trick for you (only in
commmand-line enabled runtimes).

Subject: Re: IDLVM and retall
Posted by David Fanning on Thu, 07 Sep 2006 13:52:09 GMT
View Forum Message <> Reply to Message

JD Smith writes:

> The only other ingredient is turning off XManager's default catch
> mechanism, and using MESSAGE to throw an error instead of RETALL.
> It's not elegant, but it does the job.

I've been trying to follow this, but I confess, I am
confused. Why is RETALL needed at all? I've never used
anything but RETURN in event handler CATCH statements,
and I have never had any problem running programs both
from the command line and from the VM.

What usually screws people up is calling pop-up dialogs
from their widget programs that rely on blocking behavior,
rather than modal behavior. But that doesn't seem to be
the issue here.

In fact, I can't really tell WHAT the issue is here.
I'm in need of enlightenment. :-(

Confused,

David
--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Sepore ma de ni thui.
(Opata Indian saying, meaning "Perhaps thou speakest truth.")

Subject: Re: IDLVM and retall
Posted by JD Smith on Thu, 07 Sep 2006 18:54:51 GMT
View Forum Message <> Reply to Message

On Thu, 07 Sep 2006 07:52:09 -0600, David Fanning wrote:

> JD Smith writes:
>
>> [quoted text muted]
>
> I've been trying to follow this, but I confess, I am
> confused. Why is RETALL needed at all? I've never used
> anything but RETURN in event handler CATCH statements,
> and I have never had any problem running programs both
> from the command line and from the VM.

Sorry, my example doesn't really illustrate the original method that
well (just the problem I had with it in the VM).

Imagine you are in some deep level of the calling stack in a complex
widget program, and an error occurs. If you are running
interactively, you'd like to report the error to the user with a
dialog, and then after they dismiss it, continue running the
application. The only way to do this simply is to use RETALL, which
returns all the way to the active command line (a fact many have
discovered: when a widget app crashes, use RETALL and you're often
back in business). Widget events keep flowing, and everything is
happy. If you are running non-interactively, however, you'd like that
error to halt the processing with a call to MESSAGE. So you either
call RETALL, or MESSAGE, depending on whether you're interactive or
non-interactive. This is fine, *except* in the VM (which is of course
interactive by necessity). In the VM, RETALL returns all the way out
of the session, quitting IDL! Quite a surprise after acknowledging an
error.

> What usually screws people up is calling pop-up dialogs
> from their widget programs that rely on blocking behavior,
> rather than modal behavior. But that doesn't seem to be
> the issue here.

Nope, it's just my two-pronged "system" for handling errors doesn't
work in the VM. The larger context is my use of a "helper" class
which implements an Error method, so that at any point in my
object-widget app, I can simply say:

self->Error,'You screwed up'

and it will "do the right thing" with that error, either prompting the
user with a dialog, then returning all the way out to the active

command line and continue running (for interactive use), or issuing
the error message and halting (for scripted, non-interactive use).
The only workaround I could find was using a blocking initial call to
XManager, placed just after an explicit catch, in the VM only. Since
there's no command line, losing the active command line isn't really
important.

My code to dispatch the error then looks like:

```
    if self->IsBlocking() then begin
       ;; Send a message to be caught by the established OBJREPORT catch
       message,'OBJREPORT-ERROR',/NOPRINT
    endif else if keyword_set(ro) then return else retall
```

with IsBlocking as:

```
 ;=============================================================
==================
;  isBlocking - Is the widget part of a blocking widget, or not?
 ;=============================================================
==================
function ObjReport::IsBlocking
  return,widget_info(self.or_widget,/XMANAGER_BLOCK)
end
```

where self.or_widget is the "reporting widget" atop which alerts will
be centered. Since the widget is blocking in the VM, MESSAGE will be
called, the top level CATCH will trip, and an argumentless call to
XManager will start the event loop up again.

JD

Subject: Re: IDLVM and retall
Posted by David Fanning on Thu, 07 Sep 2006 19:14:12 GMT
View Forum Message <> Reply to Message

JD Smith writes:

> Imagine you are in some deep level of the calling stack in a complex
> widget program, and an error occurs. If you are running
> interactively, you'd like to report the error to the user with a
> dialog, and then after they dismiss it, continue running the
> application. The only way to do this simply is to use RETALL, which
> returns all the way to the active command line (a fact many have
> discovered: when a widget app crashes, use RETALL and you're often
> back in business).

But why use RETALL!? I've always just used RETURN
in these cases and widget programs continue to
run pretty much forever, VM or not. Is there someplace
you are trying to get to with RETALL that you don't
get to with RETURN?

Can this be a difference between Windows and UNIX?

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: IDLVM and retall
Posted by JD Smith on Thu, 07 Sep 2006 22:57:58 GMT

View Forum Message <> Reply to Message

On Thu, 07 Sep 2006 13:14:12 -0600, David Fanning wrote:

> JD Smith writes:
>
>> Imagine you are in some deep level of the calling stack in a complex
>> widget program, and an error occurs. If you are running
>> interactively, you'd like to report the error to the user with a
>> dialog, and then after they dismiss it, continue running the
>> application. The only way to do this simply is to use RETALL, which
>> returns all the way to the active command line (a fact many have
>> discovered: when a widget app crashes, use RETALL and you're often
>> back in business).
>
> But why use RETALL!? I've always just used RETURN
> in these cases and widget programs continue to
> run pretty much forever, VM or not. Is there someplace
> you are trying to get to with RETALL that you don't
> get to with RETURN?

Here's a better example:

```
pro MyClass:f1, ev
  if something_is_wrong self->Error
  do_calc
```

```
end

pro MyClass:f2,ev
  self->f1,ev
  do_another_calc
end
```

> Can this be a difference between Windows and UNIX?

If you simply return (or return twice, if you were able to do that
from within self->Error), you'd be right back inside MyClass:f2 and
do_another_calc would run, which is not what you want. With RETALL,
the entire calling stack is returned from, not just one step in the
stack. A single RETURN only works if you are at the very tip of a
calling stack that will neatly fold up as soon as you go off the end.
It so happens that event callback functions work this way, so if your
errors always occur and are dealt with there, RETURN may be equivalent
to RETALL. The point is to be able to "continue with processing"
after issuing an error anywhere in the calling stack. This isn't
guaranteed to be a good idea (since the error may leave you in a state
such that further processing will continue the breakage), but if
you're careful, it's a nice way to abort whatever you were doing when
the breakage occurred.

Or am I missing something? Do you have a way of returning a widget
program to running anywere from anywhere in the calling stack? Keep
in mind that there are many many event callback procedure operating
simultaneously, so you don't want to put a CATCH statement in all of
them (plus it's slow for each motion event to call CATCH).

JD

Subject: Re: IDLVM and retall
Posted by David Fanning on Fri, 08 Sep 2006 04:00:24 GMT
View Forum Message <> Reply to Message

JD Smith writes:

```
> Here's a better example:
>
> pro MyClass:f1, ev
>   if something_is_wrong self->Error
>   do_calc
> end
>
> pro MyClass:f2,ev
>   self->f1,ev
```

>   do_another_calc
> end
>
>> Can this be a difference between Windows and UNIX?
>
> If you simply return (or return twice, if you were able to do that
> from within self->Error), you'd be right back inside MyClass:f2 and
> do_another_calc would run, which is not what you want. With RETALL,
> the entire calling stack is returned from, not just one step in the
> stack.

OK, I see what you mean. Yes, when you are trying to
merge widgets and objects things do tend to get a little
messy. I don't find myself buried this deep in my widget
programs, which is why RETURN has always worked for
me. But I do remember some messy, messy code in my
Catalyst Library to keep error messages from propagating
up the whole damn call stack! Essentially, we had to
keep track of whether the error had already been
"handled" or not, and keep issuing RETURNs if it had.
Your solution is probably better looking than this. :-)

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: IDLVM and retall
Posted by btt on Fri, 08 Sep 2006 14:57:48 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
> JD Smith writes:
>
>> Here's a better example:
>>
>> pro MyClass:f1, ev
>>   if something_is_wrong self->Error
>>   do_calc
>> end
>>
>> pro MyClass:f2,ev

>>   self->f1,ev
>>   do_another_calc
>> end
>>
>>> Can this be a difference between Windows and UNIX?
>> If you simply return (or return twice, if you were able to do that
>> from within self->Error), you'd be right back inside MyClass:f2 and
>> do_another_calc would run, which is not what you want. With RETALL,
>> the entire calling stack is returned from, not just one step in the
>> stack.
>
> OK, I see what you mean. Yes, when you are trying to
> merge widgets and objects things do tend to get a little
> messy. I don't find myself buried this deep in my widget
> programs, which is why RETURN has always worked for
> me. But I do remember some messy, messy code in my
> Catalyst Library to keep error messages from propagating
> up the whole damn call stack! Essentially, we had to
> keep track of whether the error had already been
> "handled" or not, and keep issuing RETURNs if it had.
> Your solution is probably better looking than this. :-)

Hi,

At risk of exposing my arrested development, I harken back to exchange I
witnessed between David and Martin Schultz eons ago. I can't remember
if it was on the newsgroup or "off-line". But I do remember clearly
that Martin and David leaned toward a FUNCTION event handling method
rather than a PROcedure - the function event handling method returned
0/1 for fail/success. If I remember rightly, the decision was driven by
the messaging aspect of events (or pseudo-events) and that functions, if
nothing else, pass messages by default.

Would that suffice to resolve the issue?

```
function MyClass::f1, ev
   if something_is_wrong then begin
     self->Error
     return,0
   endif
   return,do_calc()
end
function MyClass::f2,ev
   OK = self->f1(ev)
   if OK then OK = do_another_calc()
   return, OK
end
```

Of course, it was a long time back and I had hair and a shorter
attention span. I still have the same attention span - but the hair has
gone AWOL.

Cheers,
Ben

Subject: Re: IDLVM and retall
Posted by David Fanning on Fri, 08 Sep 2006 15:28:20 GMT
View Forum Message <> Reply to Message

JD Smith writes:

> Keep
> in mind that there are many many event callback procedure operating
> simultaneously, so you don't want to put a CATCH statement in all of
> them (plus it's slow for each motion event to call CATCH).

Ben's follow up this morning reminded me that I meant to
ask about this.

Do you have evidence that a CATCH statement is slow?
I pretty much put CATCH error handlers in ALL my
event handlers, including those used in draw widgets
for motion events. I have never noticed that these
were "slow". Or, rather, I've never thought speed was
a problem for anything I was doing with motion events.

Cheers,

David
--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: IDLVM and retall
Posted by news.qwest.net on Fri, 08 Sep 2006 15:42:21 GMT
View Forum Message <> Reply to Message

"Ben Tupper" <btupper@bigelow.org> wrote in message
news:4mdendF5h4mvU1@individual.net...
...
> At risk of exposing my arrested development, I harken back to exchange I

> witnessed between David and Martin Schultz eons ago. I can't remember if
> it was on the newsgroup or "off-line". But I do remember clearly that
> Martin and David leaned toward a FUNCTION event handling method rather
> than a PROcedure - the function event handling method returned 0/1 for
> fail/success. If I remember rightly, the decision was driven by the
> messaging aspect of events (or pseudo-events) and that functions, if
> nothing else, pass messages by default.
>
> Would that suffice to resolve the issue?
...
> Cheers,
> Ben

I was about to suggest the same thing. I have not needed to
do such a thing in IDL, but in creating very large applications
in Labview I always had this type of behaviour.
Every module had an "error input" structure, and if there was
an error, it just passed through (skipped) the module passing on the error,
and filling return values with defaults (like NAN or something appropriate).
One could append the whole routine call tree i guess too.
The errors were handled at a much higher level.

I have in mind something like the following:

```
function MyClass::f1, ev, errstruct
    if n_params() ne 2 then message:"developer error: must pass errstruct"
   if (errstruct.set eq FALSE) then begin
     ; no error passed in so perform this function
     result = do_calc(errstruct)
   endif else begin
    ; pass through errors and create default values.
      result = NULL
  endelse
  return, result
end

pro MyClass::p1, ev, errstruct
   if n_params() ne 2 then message:"developer error: must pass errstruct"
   if (errstruct.set eq FALSE) then begin
     ; no error passed in so do this routine
     do_stuff, ev,errstruct
   endif
end

function MyClass::f2,ev, errstruct
   OK = self->f1(ev, errstruct)  ; error occurs
   result = do_another_calc(ev,errstruct) ; not executed, error passed
   result2 = do_yet_another_calc(result,errstruct); not executed, error
```

```
passed
  result3 = do_yet_another_calc(result2,errstruct); not executed, error
passed
  result4 = do_yet_another_calc(result3,errstruct); not executed, error
passed
  return, result4
end
```

Coming out of f2 is the errstruct with a message such as:
"error divide by 1 occurred in myclass f1 function do_calc()"

It is a lot of work to retrofit a code base, but it would be easy
enough to implement when starting out.
Labview had the very nice feature of easily being able to demand that
errstruct always be passed (with a dataflow scheme you can do that).

Cheers,
bob

Subject: Re: IDLVM and retall
Posted by btt on Fri, 08 Sep 2006 16:02:22 GMT
View Forum Message <> Reply to Message

R.G. Stockwell wrote:
> "Ben Tupper" <btupper@bigelow.org> wrote in message
> news:4mdendF5h4mvU1@individual.net...
> ...
>> At risk of exposing my arrested development, I harken back to exchange I
>> witnessed between David and Martin Schultz eons ago. I can't remember if
>> it was on the newsgroup or "off-line". But I do remember clearly that
>> Martin and David leaned toward a FUNCTION event handling method rather
>> than a PROcedure - the function event handling method returned 0/1 for
>> fail/success. If I remember rightly, the decision was driven by the
>> messaging aspect of events (or pseudo-events) and that functions, if
>> nothing else, pass messages by default.
>>
>> Would that suffice to resolve the issue?
> ...
>> Cheers,
>> Ben
>
> I was about to suggest the same thing. I have not needed to
> do such a thing in IDL, but in creating very large applications
> in Labview I always had this type of behaviour.
> Every module had an "error input" structure, and if there was

> an error, it just passed through (skipped) the module passing on the error,
> and filling return values with defaults (like NAN or something appropriate).
> One could append the whole routine call tree i guess too.
> The errors were handled at a much higher level.
>
> I have in mind something like the following:
>

```
> function MyClass::f1, ev, errstruct
>     if n_params() ne 2 then message:"developer error: must pass errstruct"
>    if (errstruct.set eq FALSE) then begin
>      ; no error passed in so perform this function
>      result = do_calc(errstruct)
>    endif else begin
>     ; pass through errors and create default values.
>       result = NULL
>   endelse
>   return, result
> end
>
> pro MyClass::p1, ev, errstruct
>    if n_params() ne 2 then message:"developer error: must pass errstruct"
>    if (errstruct.set eq FALSE) then begin
>      ; no error passed in so do this routine
>      do_stuff, ev,errstruct
>    endif
> end
>
> function MyClass::f2,ev, errstruct
>    OK = self->f1(ev, errstruct)  ; error occurs
>    result = do_another_calc(ev,errstruct) ; not executed, error passed
>    result2 = do_yet_another_calc(result,errstruct); not executed, error
> passed
>    result3 = do_yet_another_calc(result2,errstruct); not executed, error
> passed
>    result4 = do_yet_another_calc(result3,errstruct); not executed, error
> passed
>    return, result4
> end
```

>
> Coming out of f2 is the errstruct with a message such as:
> "error divide by 1 occurred in myclass f1 function do_calc()"
>
>
> It is a lot of work to retrofit a code base, but it would be easy
> enough to implement when starting out.
> Labview had the very nice feature of easily being able to demand that
> errstruct always be passed (with a dataflow scheme you can do that).
>

That is a really good analogy. If you really are handling events with
object methods then you could make the "errstruct" a property of the
object. You could then skip it as a second parameter. But that would
reduce the problem (or the solution really) to the one David addressed
in the Catalyst library.

That error-property-passing feature in LabView is mostly built-in, and
where it isn't I'll bet there are a lot of wrappers that add that
feature. It is wicked handy to prevent race conditions (where you don't
know or have control over which software component gets the data first).
Parallel while loops come to mind (one for acquisition and one for
processing). Most of us IDL users don't bump into race conditions
because it's a procedural language and we only handle one event at a
time. If JD is not using XMANAGER to generate events then maybe all bets
are off. Although, if I think about this hard enough... no, it's no good
trying to think, it just gets me confused.

I wonder if the new IDL-IDL bridge (or any of these bridgy things) will
begin a blurring between the data-flow and procedural software ideas.

Subject: Re: IDLVM and retall
Posted by news.qwest.net on Fri, 08 Sep 2006 16:10:17 GMT
View Forum Message <> Reply to Message

"Ben Tupper" <btupper@bigelow.org> wrote in message
news:4mdigfF5p36cU1@individual.net...
...
> I wonder if the new IDL-IDL bridge (or any of these bridgy things) will
> begin a blurring between the data-flow and procedural software ideas.

Well there was VIP for a while, which was pretty much a very rough
draft of IDL as Labview.
 http://www.icaen.uiowa.edu/~dip/VIP/introduction_to_vip.html

 http://www.findarticles.com/p/articles/mi_m0EIN/is_1999_Augu st_2/ai_55322132

But is was quite painful, and not nearly as pretty as labview. :)

Cheers,
bob

Subject: Re: IDLVM and retall
Posted by JD Smith on Fri, 08 Sep 2006 16:12:33 GMT
View Forum Message <> Reply to Message

On Fri, 08 Sep 2006 09:28:20 -0600, David Fanning wrote:

> JD Smith writes:
>
>> Keep
>> in mind that there are many many event callback procedure operating
>> simultaneously, so you don't want to put a CATCH statement in all of
>> them (plus it's slow for each motion event to call CATCH).
>
> Ben's follow up this morning reminded me that I meant to
> ask about this.
>
> Do you have evidence that a CATCH statement is slow?
> I pretty much put CATCH error handlers in ALL my
> event handlers, including those used in draw widgets
> for motion events. I have never noticed that these
> were "slow". Or, rather, I've never thought speed was
> a problem for anything I was doing with motion events.

I guess I've never really measured it. It seems the actual call to
CATCH + a test on the catch error variable take about a microsecond for
me, so really probably not a problem (i.e. it will always be a tiny part
of your motion event callback budget).

What I do always wonder is how setting up a set of nested catches bogs
down IDL, since it must always keep track of where it will jump to
when an error occurs. Probably just an over-defined sense of
optimization.

JD

Subject: Re: IDLVM and retall
Posted by JD Smith on Fri, 08 Sep 2006 16:26:53 GMT

View Forum Message <> Reply to Message

On Fri, 08 Sep 2006 10:57:48 -0400, Ben Tupper wrote:

> David Fanning wrote:
>> JD Smith writes:
>>
>>> Here's a better example:
>>>
>>> pro MyClass:f1, ev
>>>   if something_is_wrong self->Error
>>>   do_calc
>>> end
>>>

```
>>>  pro MyClass:f2,ev
>>>    self->f1,ev
>>>    do_another_calc
>>>  end
>>>
>>>>  Can this be a difference between Windows and UNIX?
>>>  If you simply return (or return twice, if you were able to do that
>>>  from within self->Error), you'd be right back inside MyClass:f2 and
>>>  do_another_calc would run, which is not what you want.  With RETALL,
>>>  the entire calling stack is returned from, not just one step in the
>>>  stack.
>>
>>  OK, I see what you mean. Yes, when you are trying to
>>  merge widgets and objects things do tend to get a little
>>  messy. I don't find myself buried this deep in my widget
>>  programs, which is why RETURN has always worked for
>>  me. But I do remember some messy, messy code in my
>>  Catalyst Library to keep error messages from propagating
>>  up the whole damn call stack! Essentially, we had to
>>  keep track of whether the error had already been
>>  "handled" or not, and keep issuing RETURNs if it had.
>>  Your solution is probably better looking than this. :-)
>
>  Hi,
>
>  At risk of exposing my arrested development, I harken back to exchange I
>  witnessed between David and Martin Schultz eons ago.  I can't remember
>  if it was on the newsgroup or "off-line".  But I do remember clearly
>  that Martin and David leaned toward a FUNCTION event handling method
>  rather than a PROcedure - the function event handling method returned
>  0/1 for fail/success. If I remember rightly, the decision was driven by
>  the messaging aspect of events (or pseudo-events) and that functions, if
>  nothing else, pass messages by default.
>
>  Would that suffice to resolve the issue?
>
>  function MyClass::f1, ev
>     if something_is_wrong then begin
>       self->Error
>       return,0
>     endif
>     return,do_calc()
>  end
>  function MyClass::f2,ev
>     OK = self->f1(ev)
>     if OK then OK = do_another_calc()
>     return, OK
>  end
```

>
> Of course, it was a long time back and I had hair and a shorter
> attention span. I still have the same attention span - but the hair has
> gone AWOL.

The features I want in my error handling system are:

1. The Error method prompts the user with an error (either printing it of popping up a dialog), and either stops processing (non-interactive), or clears the entire calling stack to continue the application (interactive).

2. Can be called from *anywhere* on the calling stack, not just an event handler, but any method function or procedure, at any arbitrary depth.

3. The routine which calls Error doesn't have to check its return value, return anything special or do anything else to signal an error.

4. Doesn't require use of the traditional event callback system to work, since I use an object messaging framework which expands the concept of widget events to any generic "message" with which two objects can communicate.

For this reason, the standard tricks using the event handler didn't work for me, and that's why I came to RETALL. I've recently found that running your primary event loop in XManager while in the VM (vs. the active command line, for the non-blocking version) with a top level catch is a perfectly acceptable equivalent solution, if and only if you have exactly one entry point into your application (i.e. one predictable "first" call to XManager). Luckily, this is true for a compiled binary app, which is where RETALL doesn't work.

It's particularly nice just to say:

```
pro MyClass:Foo

  if x lt y then self->Error,'X must be GE Y'
  x-=y
  ...
end
```

and not worry about the details, no other handling required.

I should re-emphasize that this only works if you can cleanly unwind the entire calling stack at the point the error is signaled and still have a runnable program. Objects help a lot with this, given their

persistence and slow "churn" in instance data (vs. a state structure
which might be moved in and out of local scope thousands of times a
second). If you are careful and only change important instance data when
the operation has succeeded, it's easy to make methods which can be
interrupted in the middle, and then later called again with no ill effects.

JD

Subject: Re: IDLVM and retall
Posted by JD Smith on Fri, 08 Sep 2006 16:30:06 GMT
View Forum Message <> Reply to Message

On Fri, 08 Sep 2006 09:42:21 -0600, R.G. Stockwell wrote:

```
>
> "Ben Tupper" <btupper@bigelow.org> wrote in message
> news:4mdendF5h4mvU1@individual.net...
> ...
>> At risk of exposing my arrested development, I harken back to exchange I
>> witnessed between David and Martin Schultz eons ago.  I can't remember if
>> it was on the newsgroup or "off-line".  But I do remember clearly that
>> Martin and David leaned toward a FUNCTION event handling method rather
>> than a PROcedure - the function event handling method returned 0/1 for
>> fail/success. If I remember rightly, the decision was driven by the
>> messaging aspect of events (or pseudo-events) and that functions, if
>> nothing else, pass messages by default.
>>
>> Would that suffice to resolve the issue?
> ...
>> Cheers,
>> Ben
>
> I was about to suggest the same thing. I have not needed to
> do such a thing in IDL, but in creating very large applications
> in Labview I always had this type of behaviour.
> Every module had an "error input" structure, and if there was
> an error, it just passed through (skipped) the module passing on the error,
> and filling return values with defaults (like NAN or something appropriate).
> One could append the whole routine call tree i guess too.
> The errors were handled at a much higher level.
>
> I have in mind something like the following:
>
> function MyClass::f1, ev, errstruct
>     if n_params() ne 2 then message:"developer error: must pass errstruct"
>    if (errstruct.set eq FALSE) then begin
>      ; no error passed in so perform this function
>      result = do_calc(errstruct)
>    endif else begin
>     ; pass through errors and create default values.
>       result = NULL
>   endelse
>   return, result
> end
>
> pro MyClass::p1, ev, errstruct
>    if n_params() ne 2 then message:"developer error: must pass errstruct"
>    if (errstruct.set eq FALSE) then begin
>      ; no error passed in so do this routine
>      do_stuff, ev,errstruct
>    endif
> end
>
> function MyClass::f2,ev, errstruct
>    OK = self->f1(ev, errstruct)  ; error occurs
>    result = do_another_calc(ev,errstruct) ; not executed, error passed
>    result2 = do_yet_another_calc(result,errstruct); not executed, error
> passed
>    result3 = do_yet_another_calc(result2,errstruct); not executed, error
> passed
>    result4 = do_yet_another_calc(result3,errstruct); not executed, error
> passed
>    return, result4
> end
>
> Coming out of f2 is the errstruct with a message such as:
> "error divide by 1 occurred in myclass f1 function do_calc()"
>
>
> It is a lot of work to retrofit a code base, but it would be easy
> enough to implement  when starting out.
> Labview had the very nice feature of easily being able to demand that
> errstruct always be passed (with a dataflow scheme you can do that).
```

This is so much work that I would probably abandon a consistent error framework just to avoid it ;). I have many hundreds of individual methods, and might need to signal an error from any one of them. With my technique and some care to make changes to instance data "atomic", i.e. (quasi-)guaranteed either to fail entirely, or succeed completely, handling errors becomes so much simpler.

JD

Subject: Re: IDLVM and retall
Posted by JD Smith on Fri, 08 Sep 2006 16:35:54 GMT
View Forum Message <> Reply to Message

On Fri, 08 Sep 2006 12:02:22 -0400, Ben Tupper wrote:
> If JD is not using XMANAGER to generate events then maybe all bets
> are off. Although, if I think about this hard enough... no, it's no good
> trying to think, it just gets me confused.

I use XManager, I just use many individual instances of it, all of which are non-blocking and therefore just set up yet more widgets to be polled in the event loop and get out of the way. However, behind the scenes, all objects constantly communicate with each other regarding various things using non-event "messages", like "I've just changed my array value here, in case you're interested". So the pattern of method calls is very different from a traditional "up the widget tree" program structure. Messages are based on subscription lists which can and do change dynamically during runtime (Obj1 to Obj2: I'm interested in messages about array changes now, but not later, etc.).

JD

Subject: Re: IDLVM and retall
Posted by David Fanning on Fri, 08 Sep 2006 16:48:43 GMT
View Forum Message <> Reply to Message

JD Smith writes:

> 4. Doesn't require use of the traditional event callback system to
>   work, since I use an object messaging framework which expands the
>   concept of widget events to any generic "message" with which two
>   objects can communicate.

Now here is a project for ITTVIS. If they would incorporate widgets into an object system with "messages" I would *definitely* spring for the new IDL version. That would be something worth having. And, no doubt, their iTools would even be a little more comprehensible. :-)

```
> It's particularly nice just to say:
>
> pro MyClass:Foo
>
>   if x lt y then self->Error,'X must be GE Y'
>   x-=y
>   ...
```

> end
>
> and not worry about the details, no other handling required.

Does your ERROR method provide a way of getting a traceback?
That was my biggest problem with error methods. Sure, I could
record and report the error, I just couldn't figure out where
it was to fix the darn thing!!

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Sepore ma de ni thui. ("Perhaps thou speakest truth.")