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
