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.