Subject: Re: Error Handling Advice
Posted by Paul Van Delst[1] on Mon, 14 Jan 2013 16:45:24 GMT
View Forum Message <> Reply to Message

Hello,

On 01/11/13 19:03, Matthew Argall wrote:
>> 3. The only exception to 2 is for utility functions
>
> What qualifies as a utility function?

Dunno about DavidF's definition, but one I use is a function/procedure that is not "supposed" to be called outside the context of the file in which it is defined, or the application in which it is used. I also tend to use
  COMPILE_OPT HIDDEN
for these functions/procedures. E.g. in a file called "myfunc.pro" I might have

```
-----%<-----
; Utility stuff
pro utility1
  compile_opt hidden
  ...
end
pro utility2
  compile_opt hidden
  ...
end

..etc.. more utility-y stuff

; The main function
function myfunc, ....
  ...
  ...call utility1
  ...call utility2
end
-----%<-----
```

When you initially compile this file you won't see the "COMPILED ..." listings for the utility routines. Outta sight, outta mind.

So, on the IDL command line or in other funcs/procs I would call "myfunc", but I would not call utility1, utility2, etc. Sort of like how you don't (generally) call event handlers outside the context of your widget app.

>
>
>> 5. I make sure I check EVERY SINGLE ONE of my parameters and
>> keywords to see that they are defined as *something*.
>
> Just to see if they are defined, or do you make sure they are
> scalars, arrays, have the proper dimensions, etc?

Yes, yes, yes, and yes. Sometimes I also ensure the type.

> Is there a "too overboard"?

Probably. It depends on how you want to balance productivity over quality. Your boss may complain your productivity has dropped (because you are writing more checking code as opposed to application code), but your users might applaud that theirs has increased (because your extra checks catches their brain-dead input :o).

>
>> 9. CATCH error handlers are to catch program errors that I know
>> inevitably occur, but that I don't anticipate. I check EVERY SINGLE
>> ONE of my parameters and I try to anticipate common errors and
>> check for those. Sometimes it is embarrassing how much code is
>> error handling code in a small program. Don't use CATCH error
>> handlers because you are too lazy to do any work.
>
> So, check all anticipated/potential errors, then include a CATCH for
> the ones you miss. When you discover one you missed, you put in
> another check. Is that the gist of it?

Sure. That's as good a starting point as any. As time goes by you will likely stick to that, or change it accordingly depending on your experiences.

If in doubt, always code defensively.

cheers,

paulv
