Subject: Re: Passing optional parameters through a wrapper routine
Posted by John-David T. Smith on Fri, 11 Feb 2000 08:00:00 GMT

William Thompson wrote:
```
>
>  edward.s.meinel@aero.org writes:
>
>> In article <950129121.690143@clam-55>,
>>   "Mark Hadfield" <m.hadfield@niwa.cri.nz> wrote:
>
>>>  That's an interesting point David. The first few lines
>>>  of my routines tend to look something like this:
>>>
>>>   if n_elements(arg1) then message, 'You haven't defined arg1'
>
>> ...
>
>>>  2. The principle that in scientific programming
>>>  (as opposed, say, to Web page programming)
>>>  it is much better for programs to crash than to continue
>>>  and return bad data.
>
>> Ugh, I *hate* MESSAGE. Why cause a crash when it is easy to exit nicely?
>> How about:
>
>>   IF N_ELEMENTS(arg1) EQ 0 THEN BEGIN
>>       print, 'You haven't defined arg1'
>>       RETURN
>>   ENDIF
>
>> or even:
>
>>   IF N_ELEMENTS(arg1) EQ 0 THEN BEGIN
>>       dummy = DIALOG_MESSAGE('You haven't defined arg1')
>>       RETURN
>>   ENDIF
>
>> This way the user gets the message, but the program doesn't crash. This
>> is especially helpful when the procedure is used in a widget -- I don't
>> have to manually clean up everything before trying again.
>
>>>   if size(arg2, /TNAME) ne 'STRING' then message, 'Arg2 must be string
>>>  specifying the file name'
>
>> ...
>
>>>  if in doubt, stop and call for help.
```

```
>
>> Right, but you can stop and ask for help without forcing a crash. How
>> about:
>
>> IF SIZE(arg2, /TNAME) NE 'STRING' THEN BEGIN
>
>> ; Oooops! forgot the file name.
>
>>   arg2 = DIALOG_PICKFILE(set_the_appropriate_keywords)
>>   IF arg2 EQ '' THEN BEGIN
>>     dummy = DIALOG_MESSAGE( $
>>             'You must provide a file name as the second argument')
>>     RETURN
>>   ENDIF
>> ENDIF
>
>  I tend to agree with Mark Hadfield that it's better to crash than to not catch
>  the error and let the program continue on.  If one is operating in a
>  user-driven environment, then bringing it to the user's attention, such as
>  popping up an error widget as described above, is a good way to handle it.
>  However, one must also think about the case where data analysis software is
>  allowed to run in batch mode.
>
>  One trick I've adopted in many of my programs is to use an error message
>  keyword, called ERRMSG.  Then, instead of using something like
>
>         MESSAGE, 'You haven't defined arg1'
>
>  I substitute
>
>         MESSAGE = 'You haven't defined arg1'
>         GOTO, HANDLE_ERROR
>
>  At the end of the program, I have lines like
>
>                 GOTO, FINISH
>         ;
>         ;  Error handling point.
>         ;
>         HANDLE_ERROR:
>                 IF N_ELEMENTS(ERRMSG) NE 0 THEN                 $
>                         ERRMSG = 'My_Routine: ' + MESSAGE ELSE  $
>                         MESSAGE, MESSAGE
>         ;
>         ;  Exit point.
>         ;
>         FINISH:
>                 RETURN
>               END
>
>  That way, if the calling routine calls "My_Routine" without passing the ERRMSG
>  keyword, then messages are handled with the MESSAGE facility.  However, if
>  ERRMSG is passed, then the error message is passed back to the calling routine
>  and it's then responsible for deciding what to do about it.  The only drawback
>  to this scheme is that one has to define ERRMSG first, so that "My_Routine"
>  knows that it was passed, e.g
>
>         ERRMSG = ''
>         My_Routine, ERRMSG=ERRMSG, ...
>         IF ERRMSG NE '' THEN ...
```

That's why arg_present() was invented! It can detect undefined but nevertheless passed-in parameters, available by reference from the calling level. No need to define them beforehand. It also saves you from the silly user who does:

```
IDL> my_routine,ERROR_MESSAGE='This is a fine message'
```

You would change your code to:

```
IF arg_present(ERRMSG) THEN $
ERRMSG = 'My_Routine: ' + MESSAGE ELSE  $
MESSAGE, MESSAGE
```

Another side-benefit of arg_present() is that you can use it to annoy David Fanning by forcing him to contradict bold statements such as "it is NOT possible to reliably determine if a keyword was used in a call to your program", when in fact the test:

```
n_elements(k) ne 0 OR arg_present(k)
```

will tell you precisely this ;). This might be useful if k is a flag, which you'd like to set if anything, even an undefined variable, is passed it.

JD

--
J.D. Smith |*| WORK: (607) 255-5842
Cornell University Dept. of Astronomy |*| (607) 255-6263
304 Space Sciences Bldg. |*| FAX: (607) 255-5875
Ithaca, NY 14853 |*|