Subject: Passing optional parameters through a wrapper routine
Posted by bowman on Tue, 08 Feb 2000 08:00:00 GMT
View Forum Message <> Reply to Message

The _EXTRA keyword handles passing keyword parameters through a wrapper routine, but what about optional parameters? If my wrapper for TVRD has one of my own (mandatory) parameters, do I have to do the following?

```
PRO TVRD_WRAPPER, my_arg, x0, y0, nx, ny, channel, _EXTRA = tvrd_keywords

CASE N_PARAMS() OF
  1 : image = TVRD(                    _EXTRA = tvrd_keywords)
  2 : image = TVRD(x0,                 _EXTRA = tvrd_keywords)
  3 : image = TVRD(x0, y0,             _EXTRA = tvrd_keywords)
  4 : image = TVRD(x0, y0, nx,         _EXTRA = tvrd_keywords)
  5 : image = TVRD(x0, y0, nx,         _EXTRA = tvrd_keywords)
  6 : image = TVRD(x0, y0, nx, ny, channel, _EXTRA = tvrd_keywords)
  ELSE : MESSAGE, 'Incorrect number of arguments.'
ENDCASE
...
```

If I just try to pass undefined arguments through my wrapper, I get 'undefined variable' errors. (Is TVRD just counting the number of passed parameters and not checking to see if they are defined?)

Ken

--
Dr. Kenneth P. Bowman, Professor 409-862-4060
Department of Meteorology 409-862-4466 fax
Texas A&M University bowmanATcsrp.tamu.edu
College Station, TX 77843-3150 Replace AT with @

Subject: Re: Passing optional parameters through a wrapper routine
Posted by davidf on Wed, 09 Feb 2000 08:00:00 GMT
View Forum Message <> Reply to Message

Mark Hadfield (m.hadfield@niwa.cri.nz) writes:

> I have often thought that I SHOULD put more effort into recovering from
> errors gracefully but I have never been sure how to go about this. Perhaps I
> should read your book?

Well now, *there* is an idea! :-)

And since I always seem to be learning something from
your articles on this newsgroup, I just might send you one.

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting
Phone: 970-221-0438 E-Mail: davidf@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155

Subject: Re: Passing optional parameters through a wrapper routine
Posted by Craig Markwardt on Wed, 09 Feb 2000 08:00:00 GMT
View Forum Message <> Reply to Message

bowman@null.edu (Kenneth P. Bowman) writes:

```
>  The _EXTRA keyword handles passing keyword parameters through a wrapper
>  routine, but what about optional parameters?  If my wrapper for TVRD has
>  one of my own (mandatory) parameters, do I have to do the following?
>
>  PRO TVRD_WRAPPER, my_arg, x0, y0, nx, ny, channel, _EXTRA = tvrd_keywords
>
>  CASE N_PARAMS() OF
>     1 : image = TVRD(                    _EXTRA = tvrd_keywords)
>     2 : image = TVRD(x0,                 _EXTRA = tvrd_keywords)
>     3 : image = TVRD(x0, y0,             _EXTRA = tvrd_keywords)
>     4 : image = TVRD(x0, y0, nx,         _EXTRA = tvrd_keywords)
>     5 : image = TVRD(x0, y0, nx,         _EXTRA = tvrd_keywords)
>     6 : image = TVRD(x0, y0, nx, ny, channel, _EXTRA = tvrd_keywords)
>     ELSE : MESSAGE, 'Incorrect number of arguments.'
>  ENDCASE
```

I have a wrapper procedure in XFWINDOW.PRO which looks remarkably
similar to this one. To counter David, I find that it's usually the
cleanest and safest bet to mess with as *few* parameters and keywords
as possible when wrapping another function. You can get into too much
trouble trying to second-guess what the defaults should be.

The ugliness above could be solved if there would be a
variable-parameter feature to the language. This might work the same
way that _EXTRA passes keywords through. For example, in

```
PRO TVRD_WRAPPER, my_arg, $EXTRA_ARGS$, _EXTRA = tvrd_keywords
  image = TVRD($EXTRA_ARGS$,          _EXTRA = tvrd_keywords)
end
```

the black box variable $EXTRA_ARGS$ would somehow contain any
additional arguments, and could be passed directly through to the
wrapped function. The use of $'s is syntactic sugar to be sure, but
it's just an example.

Craig

--
-- --------------
Craig B. Markwardt, Ph.D. EMAIL: craigmnet@cow.physics.wisc.edu
Astrophysics, IDL, Finance, Derivatives | Remove "net" for better response
-- --------------

Subject: Re: Passing optional parameters through a wrapper routine
Posted by davidf on Wed, 09 Feb 2000 08:00:00 GMT
View Forum Message <> Reply to Message

Mark Hadfield (m.hadfield@niwa.cri.nz) writes:

> I don't see anything generally wrong with passing variables on without
> knowing what they are or whether they are defined. That's what a wrapper
> routine does -- it concerns itself with some subset of the information
> passed to it and let's the "wrappee" deal with the rest. RSI in their wisdom
> invented inheritance mechanisms to do this with keywords. For a general
> wrapper routine with an unknown number of positional parameters I favour the
> "case n_params()" syntax originally proposed by Kenneth. If I get a chance
> tomorrow I may illustrate this using my (almost completely) general wrapper
> routines that report on the execution time of the wrappee.

There is nothing wrong with "wrapper" routines passing
undefined variables. I often do this too, especially
when I write a wrapper function for an object. But there
is a difference between the wrapper and the wrappee,
as it were. :-)

It just better be the case that when the undefined
variables get to the end of the line that the
program there knows what to do with them. Mine
do, because that is how I choose to write them.
Many of the built-in IDL routines do not. That's
why they issue "undefined variable" errors.

The TVRD command is a perfect example. If those
variable parameters are undefined when they come
into the program, they should be defined as 0, 0,
!D.X_Size, and !D.Y_Size, respectively. That's what

I would do if I were writing the TVRD command.
But since I didn't, and since the best I can do is
write the wrapper for it, I'd probably move the
"intelligence" up one level. :-)

Cheers,

David

P.S. I guess the CASE N_PARAMS() syntax isn't so bad,
now that I think about it. I just think it *looks* ugly. :-)

--
David Fanning, Ph.D.
Fanning Software Consulting
Phone: 970-221-0438 E-Mail: davidf@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155

Subject: Re: Passing optional parameters through a wrapper routine
Posted by thompson on Thu, 10 Feb 2000 08:00:00 GMT
View Forum Message <> Reply to Message

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 ...
```

William Thompson

Subject: Re: Passing optional parameters through a wrapper routine
Posted by edward.s.meinel on Thu, 10 Feb 2000 08:00:00 GMT
View Forum Message <> Reply to Message

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
```

Ed Meinel

Sent via Deja.com http://www.deja.com/
Before you buy.

Subject: Re: Passing optional parameters through a wrapper routine
Posted by Mark Hadfield on Thu, 10 Feb 2000 08:00:00 GMT
View Forum Message <> Reply to Message

David Fanning <davidf@dfanning.com> wrote in message
news:MPG.130b2251759d7b27989a22@news.frii.com...
> Mark Hadfield (m.hadfield@niwa.cri.nz) writes:
>
> ...
>
> There is nothing wrong with "wrapper" routines passing
> undefined variables. I often do this too, especially
> when I write a wrapper function for an object. But there
> is a difference between the wrapper and the wrappee,
> as it were. :-)

OK, that's been clarified.

> It just better be the case that when the undefined
> variables get to the end of the line that the
> program there knows what to do with them. Mine
> do, because that is how I choose to write them.
> Many of the built-in IDL routines do not. That's
> why they issue "undefined variable" errors.

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'

    if size(arg2, /TNAME) ne 'STRING' then message, 'Arg2 must be string specifying the file name'
```

OK, I do supply values for undefined arguments when there are sensible defaults, but my fallback approach to handling bad arguments and other errors is, if in doubt, stop and call for help. The reasons for this approach are:

1. Laziness

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.

3. The expectation that the person running the routine (usually me) will be able to work out what to do.

4. It is always possible for the calling routine to CATCH errors and handle them itself if it thinks it knows better.

I have often thought that I SHOULD put more effort into recovering from errors gracefully but I have never been sure how to go about this. Perhaps I should read your book?

Mark Hadfield
m.hadfield@niwa.cri.nz http://katipo.niwa.cri.nz/~hadfield/
National Institute for Water and Atmospheric Research
PO Box 14-901, Wellington, New Zealand

Subject: Re: Passing optional parameters through a wrapper routine
Posted by John-David T. Smith on Fri, 11 Feb 2000 08:00:00 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
>
> J.D. Smith (jdsmith@astro.cornell.edu) writes:
>
>> 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 ;).
>
> Actually, I've toned the rhetoric down in my IDL
> courses, even if I haven't gotten around to updating
> my web pages.
>
> I now say "It's NOT possible to reliably determine if
> a keyword was used in a call, without resorting to a
> bunch of nonsense I can never remember and you
> shouldn't have to worry about anyway if you do it
> the way I'm telling you." :-)
>

See, I told you it annoys him :)

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 |*|

Subject: Re: Passing optional parameters through a wrapper routine
Posted by davidf on Fri, 11 Feb 2000 08:00:00 GMT
View Forum Message <> Reply to Message

J.D. Smith (jdsmith@astro.cornell.edu) writes:

> 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 ;).

Actually, I've toned the rhetoric down in my IDL courses, even if I haven't gotten around to updating my web pages.

I now say "It's NOT possible to reliably determine if a keyword was used in a call, without resorting to a bunch of nonsense I can never remember and you shouldn't have to worry about anyway if you do it the way I'm telling you." :-)

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting
Phone: 970-221-0438 E-Mail: davidf@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155

Subject: Re: Passing optional parameters through a wrapper routine
Posted by John-David T. Smith on Fri, 11 Feb 2000 08:00:00 GMT
View Forum Message <> Reply to Message

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 |*|

Subject: Re: Passing optional parameters through a wrapper routine
Posted by Martin Schultz on Fri, 11 Feb 2000 08:00:00 GMT

edward.s.meinel@aero.org wrote:
>
>
> 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

You can still use MESSAGE to print out your warning. Just use the
/CONTINUE
keyword. What I like about message is that it tells you the routine
you're in.

>> 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

Oh, I think one could write whole books on error treatment (just that no
one would
read them, I fear ;-). I try to group errors into categories like:

FATAL AND FUNDAMENTAL: stop right here and there and tell the user to
ring me up in the middle of the night or at least send me mail

FATAL USER ERROR: tell the user what an idiot he/she is and quit
execution gracefully
 (message,...,/Continue & return)

SERIOUS ERROR: When in interactive mode, stop gracefully and complain;
when in batch mode, log the error into a file and continue.

MILD ERROR: Complain and try to take corrective actions - in interactive mode let the user decide how to continue (e.g. by showing the pickfile dialog)

WARNING: Complain and continue

And maybe one should add
DEBUG INFO: produce verbose output when debugging

Well, that's my philosophy - but I must admit, it's not always easy to discipline myself to actually implement it...

Cheers,
Martin

--
[[[[[[[[[[
[[Dr. Martin Schultz   Max-Planck-Institut fuer Meteorologie   [[
[[Bundesstr. 55, 20146 Hamburg   [[
[[phone: +49 40 41173-308   [[
[[fax: +49 40 41173-298   [[
[[martin.schultz@dkrz.de   [[
[[[[[[[[[[

Subject: Re: Passing optional parameters through a wrapper routine
Posted by thompson on Mon, 14 Feb 2000 08:00:00 GMT
View Forum Message <> Reply to Message

You're right, arg_present() is a much better way of doing it. Unfortunately, most of our software was written before IDL/v5. Even more unfortunately, we're still required (at present) to write to version 4.0.

Thanks, though.

Bill Thompson