Subject: Re: Passing zero as a Parameter/ NOT KEYWORD_SET
Posted by Liam Gumley on Wed, 30 Jun 1999 07:00:00 GMT
View Forum Message <> Reply to Message

Martin Schultz wrote:
> And, finally: Use keyword_set when you want to make sure the value of a
> boolean flag is defined:
>   flag = keyword_set(flag)
> Then, later in the code, it's just
>   if (flag) then ...

Hi Martin,

I've done this many times myself. However it occurs to me that if FLAG
is defined on entry to the procedure, this method has the potential to
change the size and type of FLAG. For example, in the following cases,
all initial values of FLAG are valid 'true' values as far as KEYWORD_SET
is concerned:

```
flag = indgen(10)
flag = keyword_set(flag)
help, flag
FLAG            INT       =        1

flag = 'TEXT'
flag = keyword_set(flag)
help, flag
FLAG            INT       =        1

flag = [0]
flag = keyword_set(flag)
help, flag
FLAG            INT       =        1
```

In all these cases, the size, type, and value of FLAG was changed and
passed back to the caller. This could potentially lead to problems,
unless the caller is explicitly made aware that input keyword values can
be changed. I think a better method is to follow the advice DavidF gives
in his keyword usage article, which is to embed the KEYWORD_SET call in
the call to any procedures or functions where Boolean keywords are used,
e.g.

```
myprog, x, y, flag=keyword_set(flag)
```

which guards against changing the size, type, or value of the input
keyword FLAG.

The KEYWORD_SET examples above also reveal that KEYWORD_SET accepts as

'true' some pretty odd values (e.g. [0]). For this reason, I was
thinking that it might be useful to construct a function that truly
tests for Boolean true/false keyword values. In this case, I mean a
strict test for a value of zero or one, not the broad IDL definition of
true/false. That is, the keyword must pass the following tests to be
classified as 'true':
(1) It must be defined,
(2) It must not be a string,
(3) The first element of the keyword must be equal to 1L when converted
to LONG.

Here's a rough cut at the function (sans prolog):

```
;---cut here---
FUNCTION KEYWORD_BOOLEAN, KEYWORD

;- Check number of arguments

if n_params() ne 1 then message, 'Usage: RESULT =
KEYWORD_BOOLEAN(KEYWORD)'

;- Return false if keyword is undefined, a string, or first element ne 1
;- (otherwise return true)

case 1 of
  n_elements(keyword) eq 0          : return, 0
  size(keyword, /tname) eq 'STRING' : return, 0
  long(keyword[0]) ne 1L            : return, 0
  else                              : return, 1
endcase

END
;---cut here---
```

This function appears to display more 'correct' behavior, e.g.

```
print, keyword_boolean(indgen(10))
       0
print, keyword_boolean('TEXT')
       0
print, keyword_boolean([0])
       0
```

Cheers,
Liam.

--
Liam E. Gumley