Subject: Possible to create 'keyword_used' type function?
Posted by Brian Devour on Mon, 09 Jun 2014 21:11:35 GMT
View Forum Message <> Reply to Message

Hi all,

So, as noted here:

http://www.idlcoyote.com/tips/keyword_check.html

checking if keywords were used is kinda a pain in the butt, thanks to the difference between keywords passed by value or reference and the misleadingly-named functions used for these purposes.

At the bottom of the page, there's a nice example of a bit of code that will reliably check if a keyword was used, regardless of whether it was passed by value or reference:

```
pro testme, KEY=k
      if n_elements(k) ne 0 OR arg_present(k) then  $
       print,'You used KEY!' else $
       print,'You neglected KEY!'
   end
```

The operative part, of course, is the test:

```
if n_elements(k) ne 0 OR arg_present(k) then...
```

I can and have used this in my programs to check keywords. However, just on an aesthetic level it somewhat annoys me to have a long compound test, as compared to how simple 'if keyword_set(k) then...' is. I would like to create a function named something like keyword_used that would duplicate the functionality of the compound test above, but in a more compact and quicker to type format, so that I could simply do:

```
if keyword_used(key) then...
```

in my program instead of doing:

```
if n_elements(key) ne 0 OR arg_present(key) then...
```

So, I tried to do this. The first and simplest way to try this that came to mind was this:

```
function keyword_used, key1
if n_elements(key1) ne 0 or arg_present(key1) then return, 1 else return, 0
end
```

This doesn't work; 'keyword_used(key)' in a program always returns 1 regardless of whether 'key' was actually used or not in the program call. I *think* this is because when 'key' doesn't exist in the program, IDL obligingly passes it into keyword_used by reference, hence causing 'arg_present(key1)' inside keyword_used to return true. (I'm not sure if I'm describing that correctly; these things still confuse me somewhat.)

I did some more experimentation with trying to use scope_varfetch and routine_info to try and figure out what variable names were used in various calls and to go up a level and examine the variables existing inside the routine that calls keyword_used, but I relatively quickly got lost in the complexities thereof. (I'm an astronomer, not a programmer, and it shows...) I did some searches on this general topic, but I didn't see anywhere anyone trying this particular task. Is it even possible to write a simple function to do this?