Subject: Re: undefined keyword variables
Posted by J.D. Smith on Mon, 01 Nov 1999 08:00:00 GMT
View Forum Message <> Reply to Message

Craig Markwardt wrote:
>
> davidf@dfanning.com (David Fanning) writes:
>>
>> Mark Fardal (fardal@weka.astro.umass.edu) writes:
>>
>>> A question: should you always be able to pass undefined variables as
>>> keywords to IDL routines?
>>
>> Oh, this is absolutely normal behavior. (At least under the
>> usual standards by which such things are judged in IDL.)
>> Is it *correct* behavior? Don't know. But I would doubt it.
>> Seems to me *any* optional input keyword should be capable of
>> accepting an undefined variable as an argument. I would run
>> it by RSI for confirmation.
>>
>>> In general I don't know why you should be able to safely feed
>>> undefined variables to routines and expect them to work.
>>
>> Well, because you expect decent programmers to test any
>> variable they expect to receive and define default values
>> if one is not passed in. (As well as testing for data type
>> and structure, but who among us does this except under
>> exceptional conditions?)
>
> I am never sure any more how undefined keywords are passed. It seems
> to make a difference whether it's a built-in routine, or an IDL
> routine. It seems to make a difference whether it's IDL 4 or 5. It
> seems to make a difference if you refer to the variable by name before
> calling the procedure (not necessarily setting its value). All these
> factors make it hard to handle pass-through keywords consistently. It
> would be nice (no, crucial!) to have this more carefully documented by
> RSI.

It's really not a difference between built-in and compiled routines,
just well-written and poorly written routines. Back when I first
noticed this phenomenon of built-in routines recognizing undefined
variables, I immediately knew that RSI programmers had access to some
argument functionality we in compiled-land did not. Thus was
arg_present() born. I can now write a compiled routine which can:

1) Discern if a keyword is passed at all.
2) Discern if a keyword is passed with a value.
3) Discern if a keyword is passed which has scope in the passing level

(by reference).

Both 2 & 3 can be simultaneously true. So, since the introduction of arg_present, we can make programs which handle undefined submitted keywords gracefully, in whatever way necessary. This doesn't mean we *will*. Here is an example which demonstrates the various possibilities. Note that keyword_set is a really a subset of n_elements, and so isn't explicity included, though it can be useful.

```
pro testkey,KEY1=k1
  case arg_present(k1)+2L*(n_elements(k1) ne 0) of
    0: print,'Nothing was passed through the keyword.'
    1: print,'An undefined variable was passed.'
    2: print,'A value without scope in the passing level was passed.'
    3: print,'A defined and valued variable was passed.'
  endcase
end

IDL> testkey
Nothing was passed through the keyword.
IDL> testkey,KEY1=1
A value without scope in the passing level was passed.
IDL> testkey,KEY1=undef_var
An undefined variable was passed.
IDL> undef_var=[1,2,3]
IDL> testkey,KEY1=undef_var
A defined and valued variable was passed.
```

RSI programmers have similar (and perhaps more) functionality for writing built-in programs. This doesn't mean they'll use it consistently or correctly.

I can easily produce a routine which fails on some keywords and not on others when passed undefined variables. So can RSI. The problem is there isn't always a correct thing to do... maybe an error is actually appropriate in some cases, but consistency should be policy.

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 |*|