Subject: Re: string definition question
Posted by Paul Van Delst[1] on Tue, 14 Jan 2003 18:04:27 GMT
View Forum Message <> Reply to Message

mwvogel wrote:
>
> As my news server refuses my post, I'll paste it here :-)
>
> ////////////////////
> I would try KEYWORD_PRESENT; with A defined as 'IDL', B as '' and C
> undefined I get the following :
> IDL> A = 'IDL' & B = '' & PRINT, KEYWORD_SET(A), KEYWORD_SET(B),
> KEYWORD_SET(C)
>
> 1 0 0
>
> I guess that works in routines too.

D'oh! Why didn't I think of that!?

Thanks!

paulv

>
> Mika
> ////////////////////
>
> Mika
>
> "Paul van Delst" <paul.vandelst@noaa.gov> schreef in bericht
> news:<3E2432EC.18E46318@noaa.gov>...
>> Hello there,
>> Although I should probably know the answer to this, since all my IDL
>> reference books have
>> been borrowed, hope you don't mind me asking here.
>> I'm a bit anal about argument checking in IDL. After establishing that the
>> correct number
>> of arguments has been passed using:
>> n_arguments = 1
>> IF (N_PARAMS() LT n_arguments) THEN $
>> MESSAGE, 'Invalid number of arguments.', $
>> /NONAME, /NOPRINT
>>
>> My standard method for checking string arguments (like filenames to read)

>> is something
>> like:
>> IF (N_ELEMENTS(FileNAME) EQ 0) THEN BEGIN
>> MESSAGE, 'Input FileNAME argument not defined!', $
>> /NONAME, /NOPRINT
>>
>> IF (STRLEN(FileNAME) EQ 0) THEN $
>> MESSAGE, 'Input FileNAME argument not defined!', $
>> /NONAME, /NOPRINT
>> If I pass a zero-length string, e.g. FileNAME='', the N_ELEMENTS() test
>> passes so I always
>> test for a non-zero string length (it's happened). If I combine the two
>> tests using AND,
>> then if the variable is undefined, the STRLEN() function generates an
>> errors (since its
>> argument must be defined).
>> The above works great, but I have always felt that it should be
>> unnecessary. Is there a
>> one-step method to test that the argument is actually defined AND that the
>> string length
>> is not zero? Would the ARG_PRESENT function be useful here somehow? I read
>> the on-line
>> manual, but I really don't grok the text.
>> Thanks for any info,
>> paulv
>> -- Paul van Delst
>> CIMSS @ NOAA/NCEP/EMC
>> Ph: (301)763-8000 x7274
>> Fax:(301)763-8545

--
Paul van Delst
CIMSS @ NOAA/NCEP/EMC
Ph: (301)763-8000 x7274
Fax:(301)763-8545

Subject: Re: string definition question
Posted by btupper on Tue, 14 Jan 2003 18:13:41 GMT
View Forum Message <> Reply to Message

On Tue, 14 Jan 2003 10:55:24 -0500, Paul van Delst
<paul.vandelst@noaa.gov> wrote:

> Hello there,
>
> Although I should probably know the answer to this, since all my IDL reference books have
> been borrowed, hope you don't mind me asking here.

>
> I'm a bit anal about argument checking in IDL. After establishing that the correct number
> of arguments has been passed using:
>
> n_arguments = 1
> IF (N_PARAMS() LT n_arguments) THEN $
>   MESSAGE, 'Invalid number of arguments.', $
>   /NONAME, /NOPRINT
>
>
> My standard method for checking string arguments (like filenames to read) is something
> like:
>
> IF (N_ELEMENTS(FileNAME) EQ 0) THEN BEGIN
>   MESSAGE, 'Input FileNAME argument not defined!', $
>   /NONAME, /NOPRINT
>
> IF (STRLEN(FileNAME) EQ 0) THEN $
>   MESSAGE, 'Input FileNAME argument not defined!', $
>   /NONAME, /NOPRINT
>
> If I pass a zero-length string, e.g. FileNAME='', the N_ELEMENTS() test passes so I always
> test for a non-zero string length (it's happened). If I combine the two tests using AND,
> then if the variable is undefined, the STRLEN() function generates an errors (since its
> argument must be defined).
>
Hi Paul,

I can't think how to get away from the two step test. Usually the
ARG_PRESENT() test is reserved for optional output keywords. I would
stick to N_ELEMENTS() and N_PARAMS()

You could squish the two statements into one - but there are still two
tests involved.

```
 If (n_elements(FileName) NE 0) Then $
    If (StrLen(FileName[0]) EQ 0) Then $
    Print, "Ain't nuttin here, Paul"  Else $
    Print, "Here's the string" + fileNAME[0] Else $
    Print, "fileNAME argument is required, ya' know"
```

Ben

Subject: Re: string definition question
Posted by thompson on Tue, 14 Jan 2003 18:45:03 GMT
View Forum Message <> Reply to Message

Paul van Delst <paul.vandelst@noaa.gov> writes:

> mwvogel wrote:
>>
>> As my news server refuses my post, I'll paste it here :-)
>>
>> ////////////////////
>> I would try KEYWORD_PRESENT; with A defined as 'IDL', B as '' and C
>> undefined I get the following :
>> IDL> A = 'IDL' & B = '' & PRINT, KEYWORD_SET(A), KEYWORD_SET(B),
>> KEYWORD_SET(C)
>>
>> 1 0 0
>>
>> I guess that works in routines too.

I've always been disappointed that the KEYWORD_SET() routine does not follow the same logic as the rest of IDL for deciding whether something is true or false. According to the definition of true and false in the documentation

> Definition of True and False
>
> The condition of the IF statement can be any scalar expression. The definition of true and false for the different data types is as follows:
>
> * Byte, integer, and long: odd integers are true, even integers are false.
>
> * Floating-Point, double-precision floating-point, and complex: non-zero values are true, zero values are false. The imaginary part of complex numbers is ignored.
>
> * String: any string with a nonzero length is true, null strings are false.

However, the KEYWORD_SET() documentation simply says

> The KEYWORD_SET function returns a nonzero value if Expression is defined and nonzero or an array, otherwise zero is returned. This function is especially useful in user-written procedures and functions that process keywords that are interpreted as being either true (keyword is present and nonzero) or false (keyword was not used, or was set to zero).

In other words, KEYWORD_SET() treats integer and floating point equally, while they're treated differently in conditional statements. I've always found that

troublesome. On the other hand, the treatment of strings is consistent between
the two, although it's undocumented for KEYWORD_SET().

William Thompson

Subject: Re: string definition question
Posted by Paul Van Delst[1] on Tue, 14 Jan 2003 19:28:03 GMT
View Forum Message <> Reply to Message

Ben Tupper wrote:
>
> On Tue, 14 Jan 2003 10:55:24 -0500, Paul van Delst
> <paul.vandelst@noaa.gov> wrote:
>
>> Hello there,
>>
>> Although I should probably know the answer to this, since all my IDL reference books have
>> been borrowed, hope you don't mind me asking here.
>>
>> I'm a bit anal about argument checking in IDL. After establishing that the correct number
>> of arguments has been passed using:
>>
>>   n_arguments = 1
>>   IF (N_PARAMS() LT n_arguments) THEN $
>>   MESSAGE, 'Invalid number of arguments.', $
>>   /NONAME, /NOPRINT
>>
>>
>> My standard method for checking string arguments (like filenames to read) is something
>> like:
>>
>>   IF (N_ELEMENTS(FileNAME) EQ 0) THEN BEGIN
>>   MESSAGE, 'Input FileNAME argument not defined!', $
>>   /NONAME, /NOPRINT
>>
>>   IF (STRLEN(FileNAME) EQ 0) THEN $
>>   MESSAGE, 'Input FileNAME argument not defined!', $
>>   /NONAME, /NOPRINT
>>
>> If I pass a zero-length string, e.g. FileNAME='', the N_ELEMENTS() test passes so I always
>> test for a non-zero string length (it's happened). If I combine the two tests using AND,
>> then if the variable is undefined, the STRLEN() function generates an errors (since its
>> argument must be defined).
>>
> Hi Paul,
>
> I can't think how to get away from the two step test.   Usually the

> ARG_PRESENT() test is reserved for optional output keywords. I would
> stick to N_ELEMENTS() and N_PARAMS()
>
> You could squish the two statements into one - but there are still two
> tests involved.
>
> If (n_elements(FileName) NE 0) Then $
> If (StrLen(FileName[0]) EQ 0) Then $
> Print, "Ain't nuttin here, Paul" Else $
> Print, "Here's the string" + fileNAME[0] Else $
> Print, "fileNAME argument is required, ya' know"

Here was my solution:

```
FUNCTION Valid_String, Input_String
  IF ( N_ELEMENTS( Input_String ) EQ 0 ) THEN RETURN, 0
  IF ( STRLEN( Input_String ) EQ 0 ) THEN RETURN, 0
  RETURN, 1
END
```

Use of the KEYWORD_SET() function (as suggested by Mika) would reduce this to a single line. I would rather imbed the use of KEYWORD_SET() for this purpose in the same routine above so later readers of the software (including, or especially, me) don't get confused since the string to test may be a regular, non-keyword argument.

Anyway....in the routine where I want to test the filename string:

```
PRO testybits, FileNAME

  CATCH, Error_Status
  IF ( Error_Status NE 0 ) THEN BEGIN
    CATCH, /CANCEL
    MESSAGE, !ERROR_STATE.MSG, /CONTINUE
    IF ( N_ELEMENTS( FileID ) NE 0 ) THEN FREE_LUN, FileID
    RETURN
  ENDIF

  IF ( Valid_String( FileNAME ) EQ 0 ) THEN $
    MESSAGE, 'Input FileNAME argument not defined!', $
           /NONAME, /NOPRINT

  PRINT, FileNAME

  CATCH, /CANCEL

END
```

paulv

p.s. I guess I should really call it PVD_Valid_String. ha bloody ha. :o)

--
Paul van Delst
CIMSS @ NOAA/NCEP/EMC
Ph: (301)763-8000 x7274
Fax:(301)763-8545

Subject: Re: string definition question
Posted by JD Smith on Tue, 14 Jan 2003 19:36:42 GMT
View Forum Message <> Reply to Message

On Tue, 14 Jan 2003 11:45:03 -0700, William Thompson wrote:

> Paul van Delst <paul.vandelst@noaa.gov> writes:
>
>> mwvogel wrote:
>>>
>>> As my news server refuses my post, I'll paste it here :-)
>>>
>>> ////////////////////
>>> I would try KEYWORD_PRESENT; with A defined as 'IDL', B as '' and C
>>> undefined I get the following :
>>> IDL> A = 'IDL' & B = '' & PRINT, KEYWORD_SET(A), KEYWORD_SET(B),
>>> KEYWORD_SET(C)
>>>
>>> 1 0 0
>>>
>>> I guess that works in routines too.
>
>
> I've always been disappointed that the KEYWORD_SET() routine does not
> follow the same logic as the rest of IDL for deciding whether something
> is true or false. According to the definition of true and false in the
> documentation
>
> Definition of True and False
>
> The condition of the IF statement can be any scalar expression. The
> definition of true and false for the different data types is as
> follows:
>
> * Byte, integer, and long: odd integers are true, even integers are
> false.
>

> * Floating-Point, double-precision floating-point, and complex:
> non-zero values are true, zero values are false. The imaginary part of
> complex numbers is ignored.
>
> * String: any string with a nonzero length is true, null strings are
> false.
>
> However, the KEYWORD_SET() documentation simply says
>
> The KEYWORD_SET function returns a nonzero value if Expression is
> defined and nonzero or an array, otherwise zero is returned. This
> function is especially useful in user-written procedures and functions
> that process keywords that are interpreted as being either true
> (keyword is present and nonzero) or false (keyword was not used, or was
> set to zero).
>
> In other words, KEYWORD_SET() treats integer and floating point equally,
> while they're treated differently in conditional statements. I've
> always found that troublesome. On the other hand, the treatment of
> strings is consistent between the two, although it's undocumented for
> KEYWORD_SET().

Strangely enough, this is precisely the reason I *do* like KEYWORD_SET.
Had IDL inherited a more useful definition of TRUE and FALSE than the
FORTRAN versions, a separate logic for KEYWORD_SET wouldn't be necessary,
but do you really want to test for non-zero status in your keywords with:

```
 if keyword_set(key) then if key gt 0 then do_something
```

This would not really be a savings over:

```
 if n_elements(key) gt 0 then if key gt 0 then do_something
```

And the only time you'd profit from the altered definition would be
discriminating even/odd integers... hardly that common an operation
(for me at least):

```
 if keyword_set(key) then print,"It's odd"
```

which in real IDL would need to be:

```
 if n_elements(key) gt 0 then if key then print,"It's odd"
```

I agree that the variety of TRUE/FALSE meanings scattered throughout
IDL is somewhat disconcerting, but in this case, I think it's well
worth it!

JD

Subject: Re: string definition question
Posted by Mark Hadfield on Tue, 14 Jan 2003 20:32:57 GMT
View Forum Message <> Reply to Message

"Paul van Delst" <paul.vandelst@noaa.gov> wrote in message
news:3E2432EC.18E46318@noaa.gov...
> ..
> I'm a bit anal about argument checking in IDL. After establishing that the
correct
> number of arguments has been passed using:
>
> n_arguments = 1
> IF (N_PARAMS() LT n_arguments) THEN $
> MESSAGE, 'Invalid number of arguments.', $
> /NONAME, /NOPRINT

I see your actual question has been answered by others, so permit me to take another tack. Why do you set the NONAME & NOPRINT keywords? And why check the number of parameters? Isn't it better to check each argument to see that it's been defined (with N_ELEMENTS) or that it's available for output (with ARG_PRESENT) as necessary. The additional N_PARAMS check lets you distinguish arguments that have been given an undefined value from those that are completely missing; I don't think this is a very interesting distinction.

I ask because I feel that I have never really sorted out error checking in IDL. I guess that I lean towards a minimal approach: if a piece of code requires that a value be defined then I'll learn soon enough if it's not. (It's the code that silently gives you the wrong answer that you've got to look out for.)

Mark Hadfield "Ka puwaha te tai nei, Hoea tatou"
m.hadfield@niwa.co.nz
National Institute for Water and Atmospheric Research (NIWA)

Subject: Re: string definition question
Posted by Mark Hadfield on Tue, 14 Jan 2003 20:39:03 GMT
View Forum Message <> Reply to Message

"William Thompson" <thompson@orpheus.nascom.nasa.gov> wrote in message
news:b01lrf$1fq$1@skates.gsfc.nasa.gov...
> I've always been disappointed that the KEYWORD_SET() routine does
> not follow the same logic as the rest of IDL for deciding whether
something
> is true or false. According to the definition of true and false in the
> documentation:

> [snip]

There's a new tip on David's site, written by me, on this very subject (and touching on the relationship between logical and bitwise operations):

http://www.dfanning.com/code_tips/bitwiselogical.html

Feedback & suggestions for modification are welcome.

--
Mark Hadfield "Ka puwaha te tai nei, Hoea tatou"
m.hadfield@niwa.co.nz
National Institute for Water and Atmospheric Research (NIWA)

Subject: Re: string definition question
Posted by Paul Van Delst[1] on Tue, 14 Jan 2003 21:27:01 GMT
View Forum Message <> Reply to Message

Mark Hadfield wrote:
>
> "Paul van Delst" <paul.vandelst@noaa.gov> wrote in message
> news:3E2432EC.18E46318@noaa.gov...
>> ..
>> I'm a bit anal about argument checking in IDL. After establishing that the
> correct
>> number of arguments has been passed using:
>>
>> n_arguments = 1
>> IF (N_PARAMS() LT n_arguments) THEN $
>> MESSAGE, 'Invalid number of arguments.', $
>> /NONAME, /NOPRINT
>
> I see your actual question has been answered by others, so permit me to take
> another tack. Why do you set the NONAME & NOPRINT keywords?

The very first thing I do in *all* my "serious" IDL procedures is this:

```
CATCH, Error_Status
IF ( Error_Status NE 0 ) THEN BEGIN
  CATCH, /CANCEL
  MESSAGE, !ERROR_STATE.MSG, /CONTINUE
  RETURN
ENDIF
```

and in my functions, this:

```
@error_codes
```

```
  CATCH, Error_Status
  IF ( Error_Status NE 0 ) THEN BEGIN
    CATCH, /CANCEL
    MESSAGE, !ERROR_STATE.MSG, /CONTINUE
    RETURN, FAILURE
  ENDIF
```

where in the second example, the values for SUCCESS, INFORMATION, WARNING, and FAILURE are
defined in the include file "error_codes.pro".

The *last* thing I do in procedures is:

```
  CATCH, /CANCEL
END
```

and in functions

```
  CATCH, /CANCEL
  RETURN, SUCCESS
END
```

Now - any error checking I do I use something like:

```
  MESSAGE, 'An error occurred! Oh no!', $
          /NONAME, /NOPRINT
```

All this does is set the !ERROR_STATE.MSG which I then actually print out in my CATCH error handler - all errors tripped using the MESSAGE, 'xxxx', /NONAME, /NOPRINT get sent to the CATCH. I do this so I *always* have only one SUCCESSful exit point and only one FAILed exit point.

> And why check
> the number of parameters? Isn't it better to check each argument to see that
> it's been defined (with N_ELEMENTS) or that it's available for output (with
> ARG_PRESENT) as necessary.

I do both. My simple reasoning is if all the required arguments aren't defined then issue an error stating that.

> The additional N_PARAMS check lets you
> distinguish arguments that have been given an undefined value from those
> that are completely missing; I don't think this is a very interesting
> distinction.

Hmm - maybe, but I prefer to err on the side of verbosity. I would rather the error message state "invalid number of arguments" rather than "argument X is not defined" when what really happened was that argument X wasn't even passed into the routine. When the

error occurs I want to know *exactly* what occurred - did I forget to pass the argument or did I forget to define it.

> I ask because I feel that I have never really sorted out error checking in
> IDL. I guess that I lean towards a minimal approach: if a piece of code
> requires that a value be defined then I'll learn soon enough if it's not.

Ahh - therein lies the difference in our attitudes. I'm ridiculously anal about checking stuff and issuing error messages every chance I get. I have code consisting of 10's of lines of code and only 1-3 lines are actually the working, non-error checking parts. Minimalism in coding isn't my strong point. :o\

> (It's the code that silently gives you the wrong answer that you've got to
> look out for.)

oh yeah - you betchya. Every programmers nightmare. But your statement that you'll "learn soon enough" if something is wrong is not always the case. A number of times I've found answers to be enticingly correct - only to find out later (sometimes by someone else...gasp! horror!) they were quite bogus.

paulv

--
Paul van Delst
CIMSS @ NOAA/NCEP/EMC
Ph: (301)763-8000 x7274
Fax:(301)763-8545

Subject: Re: string definition question
Posted by sheryn.gillin on Tue, 14 Jan 2003 22:12:25 GMT
View Forum Message <> Reply to Message

Hi Paul,

> Is there a one-step method to test that the argument is actually
defined AND >that the string length is not zero?

I tend to use the SIZE function - that way you can get both 'length' [or number of dimensions in an array] and type (0 if undefined) back, and do my checking on the result.

Cheers
Sheryn

Subject: Re: string definition question

Posted by thompson on Wed, 15 Jan 2003 15:31:53 GMT
View Forum Message <> Reply to Message

JD Smith <jdsmith@as.arizona.edu> writes:

> On Tue, 14 Jan 2003 11:45:03 -0700, William Thompson wrote:

(stuff deleted)

>> In other words, KEYWORD_SET() treats integer and floating point equally,
>> while they're treated differently in conditional statements. I've
>> always found that troublesome. On the other hand, the treatment of
>> strings is consistent between the two, although it's undocumented for
>> KEYWORD_SET().

> Strangely enough, this is precisely the reason I *do* like KEYWORD_SET.
> Had IDL inherited a more useful definition of TRUE and FALSE than the
> FORTRAN versions, a separate logic for KEYWORD_SET wouldn't be necessary,
> but do you really want to test for non-zero status in your keywords with:

(stuff deleted)

But I don't want to test for non-zero status! I want to test for *Boolean*
status--that's what KEYWORD_SET() is supposed to be for! The current
KEYWORD_SET() fails to correctly treat boolean parameters formed out of
operations such as AND, OR, and NOT. Try this in an IDL program

```
A = 3
B = 3
TEST_EQUAL = A EQ B
MYPROC, MYKEYWORD=(NOT TEST_EQUAL)
```

and see what you get for KEYWORD_SET(MYKEYWORD). I know I've been bitten by
that one.

> I agree that the variety of TRUE/FALSE meanings scattered throughout
> IDL is somewhat disconcerting, but in this case, I think it's well
> worth it!

And I agree that the definition of TRUE/FALSE used in IDL's Boolean logic is
somewhat byzantine, but the problem is created by using different definitions
in different places.

Perhaps KEYWORD_SET() should have a /BOOLEAN keyword to force compliance with
how True and False are used elsewhere in IDL.

William Thompson