

---

Subject: Re: Procedures and Functions - Checking Input Data for Validity

Posted by [Craig Markwardt](#) on Sat, 17 Apr 2010 04:15:05 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

On Apr 16, 11:27 pm, Aram Panasenco <panasenco...@gmail.com> wrote:

> I tried searching IDL Help for a way to simplify checking for validity  
> of passed in arguments, but the only thing I was able to find was  
> documentation of some C macros (please correct me if there actually is  
> something that can easily allow me to validate function/procedure  
> input). So I thought of a way to be able to quickly check if the data  
> falls into a certain type and range.  
> I propose creating a function that checks input for validity in  
> accordance with a string of rules. You can think of it as an analogue  
> to a FORMAT string. The data and the string both get passed into a  
> function called something along the lines of VALIDATE\_INPUT  
> (preferably something shorter). The function returns 1 if the data  
> matches the rules defined by the string, and 0 if it doesn't. Here's  
> what I think the syntax of the string should be:

>  
> [-]T[-][R]

>  
> Where:  
> - is an optional flag, dictating whether the types specified after it  
> are inclusive (if the [-] flag is not set) or exclusive (if the [-]  
> flag is set)  
> T is a list of type codes/names separated by commas. The [-] flag  
> determines whether T specifies what the data should ("inclusive") or  
> shouldn't ("exclusive") be like.  
> - is a second optional flag, dictating whether the range(s) specified  
> after it should be exclusive or inclusive  
> R is an optional range specification. It can be written in standard  
> mathematical notation (round or square braces)

>  
> Type codes/names:  
> Elements of T can be either type codes from 0 to 15 or type names such  
> as UNDEFINED, DOUBLE, or POINTER. Since it is very easy for a rule  
> processor to differentiate between the two, I think it can be ok to  
> use either one.  
> All the type codes and names can be found in the documentation for the  
> SIZE function in the IDL reference guide.

>  
> Examples of T:

> " - NaN"  
> All data that contains NaN will return 0  
> " Double, Float "  
> Only Double and Float data will return 1  
> " - 12,13,14,15"  
> All unsigned number types will return 0

```

>
> Range Notation:
> Range defines the numerical boundaries of the data. Range
> specification will be ignored for non-numerical data. Range will be
> written in the form ( MIN,MAX ) or (N), where the braces can be either
> round or square. A round brace indicates an open boundary (the number
> is not part of the range), and a square brace indicates a closed
> boundary (the number is part of the range). The range will accept
> numbers OR the IDL constant system variables !pi, !dpi, !dtor, and !
> radeg OR the keyword !infinity (not an actual IDL keyword).
>
> Examples of R:
> " (-!pi , !pi) "
>   All data within the range, excluding -pi and pi, will return 1
> " (0,100] "
>   All data within the range, excluding 0, will return 1
> " - (0) "
>   For single numbers, it won't matter whether you use round or curly
> brackets.
>   In this case, all data that's not equal to 0 will return 1
> " [-100,0), (0,100] "
>   All data from -100 to 100, excluding 0, will return 1
> " - (0,!infinity)"
>   All data from 0 to infinity, excluding 0, will return 0 (all
> numbers greater than 0 will return 0)
>   Note that it's ok, while not recommended (for good math's sake),
> to brace !infinity with a square bracket
>
> Example of using the function (named VALIDATE_INPUT here):
>
> ; I have a spherical polygon that I want to
> ; move around while keeping maximum
> ; precision. The polygon's spherical
> ; coordinates are kept in a variable called
> ; SphVertices, and I don't want the longitude
> ; values to go out of the range (0,2*!pi) and
> ; the latitude values to go out of range
> ; (-!pi,!pi). I can use VALIDATE_INPUT:
>
> pro RenderPolygon, sphVertices
>   e = VALIDATE_INPUT(sphVertices[0,*] , "DOUBLE (0,2*!pi)") and $
>       VALIDATE_INPUT(sphVertices[1,*] , "DOUBLE (-!pi,!pi)")
>   if (~e) then HandleError("INVALID INPUT: Polygon Vertices")
>   ...
>   ...
> end
>
> ; I have a routine that controls a list of files.

```

```

> ; The routine should never be passed a
> ; negative number or a non-integer. I can
> ; use VALIDATE_INPUT:
>
> function SelectFileFromList, index
>   e = VALIDATE_INPUT(index, "INT,UINT [0,!infinity)")
>   if (~e) then HandleError("INVALID INPUT: Invalid Index for
> Filelist")
>   ...
>   ...
> end
>
> Please share your thoughts on a function like that. Would it be
> worthwhile to code something like that. Can you think of any
> improvements?

```

Parameter checking is certainly an area that IDL deserves better support.

There is an IDL Astronomy Library procedure called ZPARCHECK which does parameter type and dimension checking but not range checking, which might be a starting point.

I think your proposal is quite thorough. However, my opinion is that writing a parser of your "format string", which handles all possibilities, will be pretty difficult. In essence, you would spend a lot of time validating your validator. Beyond that, string parsing is not IDL's forte', and such a routine may cause performance issues.

My opinion is that it would be better to take advantage of IDL's existing keyword parsing.

For example something like this,

```

e = validate_input(X, /float, /double)
e = validate_input(X, min=5, max=20)
e = validate_input(X, /invert, value=0)

```

A weakness of ZPARCHECK and your approach is that there is no way to specify a default value. The parameter validation stage is also the obvious stage to optionally attach a default value if no value is specified explicitly.

Also, what to do with vector values? Most routines in IDL are designed to accept vector arguments, so one would probably want to consider the ramifications of that more carefully.

Best wishes,  
Craig

---



---

Subject: Re: Procedures and Functions - Checking Input Data for Validity  
Posted by [Aram Panasenco](#) on Sat, 17 Apr 2010 08:09:05 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Apr 16, 9:15 pm, Craig Markwardt <craig.markwa...@gmail.com> wrote:  
> On Apr 16, 11:27 pm, Aram Panasenco <panasenco...@gmail.com> wrote:  
>  
>  
>  
>  
>  
>  
>> I tried searching IDL Help for a way to simplify checking for validity  
>> of passed in arguments, but the only thing I was able to find was  
>> documentation of some C macros (please correct me if there actually is  
>> something that can easily allow me to validate function/procedure  
>> input). So I thought of a way to be able to quickly check if the data  
>> falls into a certain type and range.  
>> I propose creating a function that checks input for validity in  
>> accordance with a string of rules. You can think of it as an analogue  
>> to a FORMAT string. The data and the string both get passed into a  
>> function called something along the lines of VALIDATE\_INPUT  
>> (preferably something shorter). The function returns 1 if the data  
>> matches the rules defined by the string, and 0 if it doesn't. Here's  
>> what I think the syntax of the string should be:  
>  
>> [-]T[-][R]  
>  
>> Where:  
>> - is an optional flag, dictating whether the types specified after it  
>> are inclusive (if the [-] flag is not set) or exclusive (if the [-]  
>> flag is set)  
>> T is a list of type codes/names separated by commas. The [-] flag  
>> determines whether T specifies what the data should ("inclusive") or  
>> shouldn't ("exclusive") be like.  
>> - is a second optional flag, dictating whether the range(s) specified  
>> after it should be exclusive or inclusive  
>> R is an optional range specification. It can be written in standard  
>> mathematical notation (round or square braces)  
>  
>> Type codes/names:  
>> Elements of T can be either type codes from 0 to 15 or type names such  
>> as UNDEFINED, DOUBLE, or POINTER. Since it is very easy for a rule  
>> processor to differentiate between the two, I think it can be ok to  
>> use either one.  
>> All the type codes and names can be found in the documentation for the  
>> SIZE function in the IDL reference guide.  
>  
>> Examples of T:  
>> " - NaN"

```

>> All data that contains NaN will return 0
>> " Double, Float "
>> Only Double and Float data will return 1
>> " - 12,13,14,15"
>> All unsigned number types will return 0
>
>> Range Notation:
>> Range defines the numerical boundaries of the data. Range
>> specification will be ignored for non-numerical data. Range will be
>> written in the form ( MIN,MAX ) or (N), where the braces can be either
>> round or square. A round brace indicates an open boundary (the number
>> is not part of the range), and a square brace indicates a closed
>> boundary (the number is part of the range). The range will accept
>> numbers OR the IDL constant system variables !pi, !dpi, !dtr, and !
>> raddeg OR the keyword !infinity (not an actual IDL keyword).
>
>> Examples of R:
>> " (-!pi , !pi) "
>> All data within the range, excluding -pi and pi, will return 1
>> " (0,100] "
>> All data within the range, excluding 0, will return 1
>> " - (0) "
>> For single numbers, it won't matter whether you use round or curly
>> brackets.
>> In this case, all data that's not equal to 0 will return 1
>> " [-100,0), (0,100] "
>> All data from -100 to 100, excluding 0, will return 1
>> " - (0,!infinity)"
>> All data from 0 to infinity, excluding 0, will return 0 (all
>> numbers greater than 0 will return 0)
>> Note that it's ok, while not recommended (for good math's sake),
>> to brace !infinity with a square bracket
>
>> Example of using the function (named VALIDATE_INPUT here):
>
>> ; I have a spherical polygon that I want to
>> ; move around while keeping maximum
>> ; precision. The polygon's spherical
>> ; coordinates are kept in a variable called
>> ; SphVertices, and I don't want the longitude
>> ; values to go out of the range (0,2*!pi) and
>> ; the latitude values to go out of range
>> ; (-!pi,!pi). I can use VALIDATE_INPUT:
>
>> pro RenderPolygon, sphVertices
>>   e = VALIDATE_INPUT(sphVertices[0,*] , "DOUBLE (0,2*!pi)") and $
>>       VALIDATE_INPUT(sphVertices[1,*] , "DOUBLE (-!pi,!pi)")
>>   if (~e) then HandleError("INVALID INPUT: Polygon Vertices")

```

```

>> ...
>> ...
>> end
>
>> ; I have a routine that controls a list of files.
>> ; The routine should never be passed a
>> ; negative number or a non-integer. I can
>> ; use VALIDATE_INPUT:
>
>> function SelectFileFromList, index
>>     e = VALIDATE_INPUT(index , "INT,UINT [0,!infinity)")
>>     if (~e) then HandleError("INVALID INPUT: Invalid Index for
>> Filelist")
>>     ...
>>     ...
>> end
>
>> Please share your thoughts on a function like that. Would it be
>> worthwhile to code something like that. Can you think of any
>> improvements?
>
> Parameter checking is certainly an area that IDL deserves better
> support.
>
> There is an IDL Astronomy Library procedure called ZPARCHECK which
> does parameter type and dimension checking but not range checking,
> which might be a starting point.
>
> I think your proposal is quite thorough. However, my opinion is that
> writing a parser of your "format string", which handles all
> possibilities, will be pretty difficult. In essence, you would spend
> a lot of time validating your validator. Beyond that, string parsing
> is not IDL's forte', and such a routine may cause performance
> issues.
>
> My opinion is that it would be better to take advantage of IDL's
> existing keyword parsing.
> For example something like this,
> e = validate_input(X, /float, /double)
> e = validate_input(X, min=5, max=20)
> e = validate_input(X, /invert, value=0)
>
> A weakness of ZPARCHECK and your approach is that there is no way to
> specify a default value. The parameter validation stage is also the
> obvious stage to optionally attach a default value if no value is
> specified explicitly.
>
> Also, what to do with vector values? Most routines in IDL are

```

> designed to accept vector arguments, so one would probably want to  
> consider the ramifications of that more carefully.  
>  
> Best wishes,  
> Craig

I think that as far as the function should be concerned, even one "garbage" value in a matrix should make it return 0. Therefore, all the range/value testing will be done on each value of the matrix (probably using the WHERE() function).

I see your point and I have to agree that it probably would've been a little too hard to process parameters through a string, especially as far as processing elements like "2\*!pi". However, I personally find using procedures/functions that include a lot of keywords extremely hard to read (since there's no way of grouping them). I think extremely clear readability is very important for a function like that, because it is practically the interface of the routine. I am going to adopt your idea of using keyword parsing, but with a little tweak. I think that a lot of /double,/string,/float,etc. keywords in your suggestion (plus a keyword like /exclude) can still be replaced by a string containing the names/codes of the keywords. Compare:

```
e = VALIDATE(data, /exclude, /double,/float,/int)
to
e = VALIDATE(data, ex_type="double,float,int")
```

and

```
e = VALIDATE(data, /struct,/object)
to
e = VALIDATE(data, in_type="struct,object")
```

It's not very hard to process an in\_type/ex\_type string in 4 simple steps:

- 1) Remove whitespace (STRCOMPRESS)
- 2) Convert to uppercase (STRUPCASE)
- 3) Split by commas (STRSPLIT)
- 4) Run a loop for each substring: Identify if it's a type code or type name (using FIX and ON\_IOERROR) and compare each against the data's type code and name (EQ or STRCMP).

The argument against using strings to process range is strong, though. There's no way I am going to be able to imitate the way IDL processes constants (2\*!pi, 3D, etc.). Besides, what if the user wants to use a variable name? No, I am going to have to use keywords: in\_range, ex\_range, in\_value, ex\_value. in\_range and ex\_range are going to be 2xN arrays of numbers - representing inclusive MIN-MAX pairs, and

in\_value and ex\_value are going to be 1xN arrays of numbers - representing inclusion and exclusion values.

```
e = VALIDATE(data, in_range=[-100,100]) ; <-- Range from -100 to 100
e = VALIDATE(data, in_range=[-!pi,!pi], ex_value = [-!pi,0,!pi]) ; <--
Range from -pi to pi, excluding 0 and the endpoints
```

Specifying the default value for data can be important in non-critical applications. DEFAULT should definitely be a keyword for the validator. Note that with the DEFAULT keyword set, the function always returns 1.

```
e = VALIDATE(data, ex_range="NaN", default=0)
```

All in all it comes down to the following:

The VALIDATE function (I decided to remove the \_input part, because the user might want to use it not just for processing input values) takes in the mandatory data parameter and several optional keywords:

- in\_type and ex\_type : strings that contain (inclusive or exclusive) type codes or names for the data (separated by commas). If both keywords are passed, in\_type is given priority.
- in\_range and ex\_range : 2xN arrays that contain N MIN-MAX pairs specifying the inclusive or exclusive numeric ranges. Apply to numerical data only.
- in\_value and ex\_value : 1xN arrays that contain N inclusive or exclusive numeric values. Apply to numerical data only.
- default : value used to replace all invalid data. If the default keyword is set, VALIDATE always returns 1.

Examples:

```
;Original example 1
;(RenderPolygon)
```

```
pro RenderPolygon, sphVertices
  e = VALIDATE(sphVertices[0,*], in_type="double", in_range=[0,2*!
pi], ex_value=[0,2*!pi]) and $
    VALIDATE(sphVertices[1,*], in_type="double", in_range=[-!pi,!
pi], ex_value=[-!pi,!pi])
  if (~e) then HandleError("INVALID INPUT: Polygon Vertices")
  ...
  ...
end
```

```
;Original example 2
;(SelectFileFromList)
```



```

function SelectFileFromList, index
    e = VALIDATE(index, in_type="int", in_range =
[0,N_LISTELEMENTS-1])
    if (~e) then HandleError("INVALID INPUT: Invalid Index for
Filelist")
    ...
    ...
end

```

;More Examples:

```

e = VALIDATE(denominator, ex_value=0)
e = VALIDATE(somedata, ex_type = "0,1,6,8,9,10,11", ex_range = [-1,1],
in_value = 0)
e = VALIDATE(magfield, ex_type="NaN",default=min(magfield, /NaN))

```

Thank you Craig for helping me improve (I think) on my original idea  
Please continue to post feedback! What other changes/fixes can I make  
to the design before I start coding away?

~Aram Panasenco

Subject: Re: Procedures and Functions - Checking Input Data for Validity  
Posted by [Aram Panasenco](#) on Sat, 17 Apr 2010 17:39:54 GMT  
[View Forum Message](#) <> [Reply to Message](#)

Update: I've decided to ditch string processing altogether, and pass  
in arrays of strings or integers to in\_type and ex\_type:

```

e = VALIDATE(data, ex_type=["NaN"])
e = VALIDATE(data, in_type=["Double","Float"])
e = VALIDATE(data, ex_type=[10,11,12,13,14,15])

```

I have started working on the actual function, expect it soon!

~Aram Panasenco

Subject: Re: Procedures and Functions - Checking Input Data for Validity  
Posted by [David Fanning](#) on Sat, 17 Apr 2010 17:48:35 GMT  
[View Forum Message](#) <> [Reply to Message](#)

Aram Panasenco writes:

```

> Update: I've decided to ditch string processing altogether, and pass
> in arrays of strings or integers to in_type and ex_type:
>

```

```
> e = VALIDATE(data, ex_type=["NaN"])
> e = VALIDATE(data, in_type=["Double","Float"])
> e = VALIDATE(data, ex_type=[10,11,12,13,14,15])
>
> I have started working on the actual function, expect it soon!
```

Aram, just a little practical advice. Don't get too excited. Ten seconds after you publish your masterpiece it will get trashed by someone you didn't even know \*read\* the IDL newsgroup. If you can fight through the disappointment, you MAY be able to improve it substantially. :-)

Cheers,

David

--

David Fanning, Ph.D.  
Fanning Software Consulting, Inc.  
Coyote's Guide to IDL Programming: <http://www.dfanning.com/>  
Sepore ma de ni thui. ("Perhaps thou speakest truth.")

---

---

Subject: Re: Procedures and Functions - Checking Input Data for Validity  
Posted by [Aram Panasenco](#) on Sat, 17 Apr 2010 18:42:40 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Apr 17, 10:48 am, David Fanning <n...@dfanning.com> wrote:

```
> Aram Panasenco writes:
>> Update: I've decided to ditch string processing altogether, and pass
>> in arrays of strings or integers to in_type and ex_type:
>
>> e = VALIDATE(data, ex_type=["NaN"])
>> e = VALIDATE(data, in_type=["Double","Float"])
>> e = VALIDATE(data, ex_type=[10,11,12,13,14,15])
>
>> I have started working on the actual function, expect it soon!
>
> Aram, just a little practical advice. Don't get too
> excited. Ten seconds after you publish your masterpiece
> it will get trashed by someone you didn't even know
> *read* the IDL newsgroup. If you can fight through
> the disappointment, you MAY be able to improve it
> substantially. :-)
>
```

> Cheers,  
>  
> David  
>  
> --  
> David Fanning, Ph.D.  
> Fanning Software Consulting, Inc.  
> Coyote's Guide to IDL Programming:<http://www.dfanning.com/>  
> Sepore ma de ni thui. ("Perhaps thou speakest truth.")

All the better than no response at all! If someone takes their time to thoroughly thrash a function, then they probably care about its quality. ;)

~Aram Panasenco

---