Subject: Re: Error Handling Advice
Posted by David Fanning on Fri, 11 Jan 2013 22:51:30 GMT
View Forum Message <> Reply to Message

Matthew Argall writes:

> I have been trying to pick up good programming practices by copying what other people do. What I have not been able to figure out, though, is error handling.
>
> Right now, I tend to check all of the input parameters and keywords to make sure they are suitable for the program that follows. But then again, there is the whole RTFM argument. When do you check a specific input parameter and when do you just throw/catch an error and return to main?

Here is what I do.

1. I assume no one can read. At least there is no evidence for it.

2. I include a CATCH error handler in every program module I write.

3. The only exception to 2 is for utility functions, for which I use an On_Error,2 condition to return to the caller of the function, which presumable DOES have a CATCH error handler.

4. I NEVER return to the main IDL level. I ALWAYS return to the caller of my program, hoping whoever wrote the damn thing that's calling my program knows enough to include error handling it it.

5. I make sure I check EVERY SINGLE ONE of my parameters and keywords to see that they are defined as *something*. If I intend to use them, I intend to have them defined.

6. The only exception to 5 is if I know that the program I'm calling checks the parameter or keyword and has decent error handling (i.e, I wrote the darn thing), then I sometimes pass the parameter or keyword along, knowing it will eventually get checked before it is used.

7. My practice is to make positional parameters required arguments, and keywords parameters optional arguments. I sometimes make a positional parameter an optional argument, if there is a compelling reason to do so, but I NEVER make a keyword parameter a required argument. Seeing the latter in other people's code makes me crazy.

8. I try to come up with values for optional parameters that "make sense". For example, one of my pet peeves for IDL 8 graphics is that when you save a graphics window as a PNG file, the PNG file is the size of my office wall. I don't think anyone wants a file that large, so I think the default value for the RESOLUTION keyword is poorly chosen. At

least I notice that on every program I've seen from ExelisVis lately
this keyword is used to get a PNG file that is a smaller size. That, to
me, makes the RESOLUTION keyword a required keyword, no matter how the
code is written. :-)

Cheers,

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

Subject: Re: Error Handling Advice
Posted by Jeremy Bailin on Fri, 11 Jan 2013 22:58:26 GMT
View Forum Message <> Reply to Message

> 4. I NEVER return to the main IDL level. I ALWAYS return to the caller
> of my program, hoping whoever wrote the damn thing that's calling my
> program knows enough to include error handling it it.

This!

Nothing makes code as hard to debug as when it throws you back to the
main level.

-Jeremy.

Subject: Re: Error Handling Advice
Posted by David Fanning on Fri, 11 Jan 2013 23:17:22 GMT
View Forum Message <> Reply to Message

David Fanning writes:

I guess I should make one more point.

9. CATCH error handlers are to catch program errors that I know
inevitably occur, but that I don't anticipate. People will use your code
in God only knows how many awful ways. I'm not omniscient. But, on the
other hand, I'm not lazy either. I check EVERY SINGLE ONE of my
parameters and I try to anticipate common errors and check for those.
Sometimes it is embarrassing how much code is error handling code in a
small program. Don't use CATCH error handlers because you are too lazy
to do any work. Even Coyote is not *that* depraved.

Cheers,

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

Subject: Re: Error Handling Advice
Posted by Matthew Argall on Sat, 12 Jan 2013 00:03:06 GMT
View Forum Message <> Reply to Message

> 3. The only exception to 2 is for utility functions

What qualifies as a utility function?

> 5. I make sure I check EVERY SINGLE ONE of my parameters and keywords to
> see that they are defined as *something*.

Just to see if they are defined, or do you make sure they are scalars, arrays, have the proper dimensions, etc? Is there a "too overboard"?

> 9. CATCH error handlers are to catch program errors that I know
> inevitably occur, but that I don't anticipate. I check EVERY SINGLE ONE of my
> parameters and I try to anticipate common errors and check for those.
> Sometimes it is embarrassing how much code is error handling code in a
> small program. Don't use CATCH error handlers because you are too lazy
> to do any work.

So, check all anticipated/potential errors, then include a CATCH for the ones you miss. When you discover one you missed, you put in another check. Is that the gist of it?

Subject: Re: Error Handling Advice
Posted by David Fanning on Sat, 12 Jan 2013 00:15:30 GMT
View Forum Message <> Reply to Message

Matthew Argall writes:

>
>> 3. The only exception to 2 is for utility functions
>
> What qualifies as a utility function?

>
>
>> 5. I make sure I check EVERY SINGLE ONE of my parameters and keywords to
>> see that they are defined as *something*.
>
> Just to see if they are defined, or do you make sure they are scalars, arrays, have the proper dimensions, etc? Is there a "too overboard"?

Well, despite all my blather, there is a "too overboard" factor at work, too. If it matters to me, I'll do more checking. But, if I write a program that works with images, I probably won't check to see if the user passed in a scalar value. Not worth my time to do so. Let CATCH handle it. But, I can tell I do a LOT more checking than most of the IDL programs I look at. :-)

>> 9. CATCH error handlers are to catch program errors that I know
>> inevitably occur, but that I don't anticipate. I check EVERY SINGLE ONE of my
>> parameters and I try to anticipate common errors and check for those.
>> Sometimes it is embarrassing how much code is error handling code in a
>> small program. Don't use CATCH error handlers because you are too lazy
>> to do any work.
>
> So, check all anticipated/potential errors, then include a CATCH for the ones you miss. When you discover one you missed, you put in another check. Is that the gist of it?

Yes, I often add more error checking if I find people are confused about how something should be used. But, I also look at the interface. If I call a program wrong several times in a row, I know I got the interface wrong, and I try to change it. One example off the top of my head is PS_START. It has a default filename of "idl.ps", so if the user wants to change it, they can use a FILENAME keyword:

```
PS_START, FILENAME='test.ps'
```

But, when I use PS_START, I always called it like this:

```
PS_START, 'test.ps'
```

I did that so often, I made an optional positional parameter to handle this correctly. FILENAME was what I can an "inefficient" keyword.

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.

Coyote's Guide to IDL Programming: http://www.idlcoyote.com/
Sepore ma de ni thue. ("Perhaps thou speakest truth.")

Subject: Re: Error Handling Advice
Posted by Matthew Argall on Sat, 12 Jan 2013 02:36:27 GMT
View Forum Message <> Reply to Message

>> What qualifies as a utility function?

I guess I will just have to look through more of your documentation to figure this one out ;-) It has been a big help so far.

> Well, despite all my blather, there is a "too overboard" factor at work,

I guess I will err on the side of too much than get stuck with point #1

> But, I also look at the interface. ...

Good point... Will keep this in mind.

Thanks for the help!

Subject: Re: Error Handling Advice
Posted by David Fanning on Sat, 12 Jan 2013 03:08:09 GMT
View Forum Message <> Reply to Message

Matthew Argall writes:

>>> What qualifies as a utility function?
>
> I guess I will just have to look through more of your documentation to figure this one out ;-) It has been a big help so far.

Ah, forgot to answer that question. A utility function has some specific task in my program. Maybe it performs an operation on a piece of data and returns the transformed data to me. I don't really expect errors in this program (since I am calling it in my main program and I wrote the darn thing, after all), but if an error occurs, I would prefer to handle it in the error handler of the main program.

Cheers,

David

--

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

Subject: Re: Error Handling Advice
Posted by John Correira on Mon, 14 Jan 2013 14:51:45 GMT
View Forum Message <> Reply to Message

On 01/11/2013 05:51 PM, David Fanning wrote:
>
> 7. My practice is to make positional parameters required arguments, and
> keywords parameters optional arguments. I sometimes make a positional
> parameter an optional argument, if there is a compelling reason to do
> so, but I NEVER make a keyword parameter a required argument. Seeing the
> latter in other people's code makes me crazy.

Is there a technical reason behind not making a keyword a required
argument, or just a personal preference based on experience? I find it
harder to interpret a long series of positional parameters as opposed to
a list of somewhat descriptive keywords.

Regards,

John

Subject: Re: Error Handling Advice
Posted by David Fanning on Mon, 14 Jan 2013 15:05:05 GMT
View Forum Message <> Reply to Message

John Correira writes:

> Is there a technical reason behind not making a keyword a required
> argument, or just a personal preference based on experience? I find it
> harder to interpret a long series of positional parameters as opposed to
> a list of somewhat descriptive keywords.

It is not a technical reason so much as a visceral response. :-)

I suppose I write programs differently than most people, but I've never
had a need to define more than three required arguments in a program
EVER, as far as I can recall. I'm sure there is a reason for the 10-12 I
sometimes see in programs I haven't written, but I can't imagine what
that reason is. People who come to me with those kinds of programs are
always confused. I just assumed that was one of the reasons why. ;-)

Cheers,

David

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

Subject: Re: Error Handling Advice
Posted by Paul Van Delst[1] on Mon, 14 Jan 2013 16:45:24 GMT
View Forum Message <> Reply to Message

Hello,

On 01/11/13 19:03, Matthew Argall wrote:
>> 3. The only exception to 2 is for utility functions
>
> What qualifies as a utility function?

Dunno about DavidF's definition, but one I use is a function/procedure that is not "supposed" to be called outside the context of the file in which it is defined, or the application in which it is used. I also tend to use
  COMPILE_OPT HIDDEN
for these functions/procedures. E.g. in a file called "myfunc.pro" I might have

```
-----%<-----
; Utility stuff
pro utility1
  compile_opt hidden
  ...
end
pro utility2
  compile_opt hidden
  ...
end

..etc.. more utility-y stuff

; The main function
function myfunc, ....
  ...
  ...call utility1
```

```
   ...call utility2
end
-----%<-----
```

When you initially compile this file you won't see the "COMPILED ..." listings for the utility routines. Outta sight, outta mind.

So, on the IDL command line or in other funcs/procs I would call "myfunc", but I would not call utility1, utility2, etc. Sort of like how you don't (generally) call event handlers outside the context of your widget app.

>
>
>> 5. I make sure I check EVERY SINGLE ONE of my parameters and
>> keywords to see that they are defined as *something*.
>
> Just to see if they are defined, or do you make sure they are
> scalars, arrays, have the proper dimensions, etc?

Yes, yes, yes, and yes. Sometimes I also ensure the type.

> Is there a "too overboard"?

Probably. It depends on how you want to balance productivity over quality. Your boss may complain your productivity has dropped (because you are writing more checking code as opposed to application code), but your users might applaud that theirs has increased (because your extra checks catches their brain-dead input :o).

>
>> 9. CATCH error handlers are to catch program errors that I know
>> inevitably occur, but that I don't anticipate. I check EVERY SINGLE
>> ONE of my parameters and I try to anticipate common errors and
>> check for those. Sometimes it is embarrassing how much code is
>> error handling code in a small program. Don't use CATCH error
>> handlers because you are too lazy to do any work.
>
> So, check all anticipated/potential errors, then include a CATCH for
> the ones you miss. When you discover one you missed, you put in
> another check. Is that the gist of it?

Sure. That's as good a starting point as any. As time goes by you will likely stick to that, or change it accordingly depending on your experiences.

If in doubt, always code defensively.

cheers,

paulv

Subject: Re: Error Handling Advice
Posted by Paul Van Delst[1] on Mon, 14 Jan 2013 16:51:57 GMT
View Forum Message <> Reply to Message

hello,

On 01/14/13 10:05, David Fanning wrote:
> John Correira writes:
>
>> Is there a technical reason behind not making a keyword a required
>> argument, or just a personal preference based on experience? I find it
>> harder to interpret a long series of positional parameters as opposed to
>> a list of somewhat descriptive keywords.
>
> It is not a technical reason so much as a visceral response. :-)

Ha! Good answer!

FWIW I apply the IDL parameter/keyword type for my Fortran95/2003 code.
E.g. if I have

```
  subroutine mysub(a,b,c,d)
    real          :: a, b
    real, optional :: c, d
    ...
  end subroutine mysub
```

the documentation for this routine's calling sequence is always listed
sort-of like an IDL routine:

```
  !  call mysub(a, b, c=c, d=d)
```

to indicate that "a" and "b" are mandatory arguments, and "c" and "d"
are optional.

> I suppose I write programs differently than most people, but I've never
> had a need to define more than three required arguments in a program
> EVER, as far as I can recall. I'm sure there is a reason for the 10-12 I
> sometimes see in programs I haven't written, but I can't imagine what
> that reason is. People who come to me with those kinds of programs are
> always confused. I just assumed that was one of the reasons why. ;-)

My rule of thumb: More that 3-5 required arguments -> collect them in a

structure! :o)

cheers,

paulv

Subject: Re: Error Handling Advice
Posted by David Fanning on Mon, 14 Jan 2013 17:05:00 GMT
View Forum Message <> Reply to Message

Paul van Delst writes:

>> It is not a technical reason so much as a visceral response. :-)
>
> Ha! Good answer!

I should probably elaborate. Some things are just wrong. Required keywords fall into that category for me. (Not to say I haven't occasionally used one, but certainly not in any code that has escaped my office.) I found one in some ENVI code not too long ago. It just drives me crazy, that's all. It's a moral argument. ;-)

Cheers,

David

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

Subject: Re: Error Handling Advice
Posted by Matthew Argall on Mon, 14 Jan 2013 18:30:03 GMT
View Forum Message <> Reply to Message

Ok, I think I have reasoned with myself enough to know when to use ON_ERROR, 2 and when to use CATCH, but most CATCH blocks that I see just print an error message and return to the calling program, anyway, without doing anything special.

For example, take the following

----------%<----------
CATCH, Error_status

```
IF Error_status NE 0 THEN BEGIN
    CATCH, /CANCEL
    PRINT, 'Error index: ', Error_status
    PRINT, 'Error message: ', !ERROR_STATE.MSG
    RETURN
ENDIF
-----------%<----------------
```

Is there any reason for not using ON_ERROR, 2 in this case?

From what I gather, the CATCH blocks are normally used to make a program "fail gracefully", in that it resets the device and color table, closes files, frees pointers, etc. But without anything to reset, is ON_ERROR, 2 sufficient?

I guess I will start with some with trial and ::ehem:: error ;-)

Subject: Re: Error Handling Advice
Posted by David Fanning on Mon, 14 Jan 2013 18:44:08 GMT
View Forum Message <> Reply to Message

Matthew Argall writes:

> Ok, I think I have reasoned with myself enough to know when to use ON_ERROR, 2 and when to use CATCH, but most CATCH blocks that I see just print an error message and return to the calling program, anyway, without doing anything special.
>
> For example, take the following
>
```
> -----------%<----------------
> CATCH, Error_status
>
> IF Error_status NE 0 THEN BEGIN
>     CATCH, /CANCEL
>     PRINT, 'Error index: ', Error_status
>     PRINT, 'Error message: ', !ERROR_STATE.MSG
>     RETURN
> ENDIF
> -----------%<----------------
```
>
> Is there any reason for not using ON_ERROR, 2 in this case?

In this case, probably not. That CATCH error handler sucks. :-)

What you normally want to do is print out some kind of useful information about where the error occurred in the program (what line number) and what the error was. Otherwise, you will have a devil of a

time fixing it. My standard error handler looks like this:

```
Catch, theError
IF theError NE 0 THEN BEGIN
   Catch, /CANCEL
   void = Error_Message()
   RETURN
ENDIF
```

Error_Message is a Coyote Library routine that will display a pop-up dialog (if appropriate) and write the error traceback out to the command line where you can read the message and do something about the error. The Catch error handler also clears the error status, so you return to the caller with a clear slate. This is especially important in widget programs, where you often like to keep the programs running, even if errors occur.

> From what I gather, the CATCH blocks are normally used to make a program "fail gracefully", in that it resets the device and color table, closes files, frees pointers, etc. But without anything to reset, is ON_ERROR, 2 sufficient?

I'm afraid the CATCH block doesn't do ANY of these things, unless you explicitly do it yourself in the CATCH block. Freeing up your pointers is a good idea!

> I guess I will start with some with trial and ::ehem:: error ;-)

Yep. :-)

Cheers,

David

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

Subject: Re: Error Handling Advice
Posted by Matthew Argall on Mon, 14 Jan 2013 20:02:41 GMT
View Forum Message <> Reply to Message

> I guess I will start with some with trial and ::ehem:: error ;-)

So, after a little work, I get a cascade of error messages

```
---------------------------
function add, a, b
   Catch, theError
   IF theError NE 0 THEN BEGIN
      Catch, /CANCEL
      void = Error_Message()
      RETURN, !Null
   ENDIF

   RETURN, a + b
end

pro call_add
   Catch, theError
   IF theError NE 0 THEN BEGIN
      Catch, /CANCEL
      void = Error_Message()
      RETURN
   ENDIF

   sum = add(a, b)

   IF sum EQ !Null THEN message, 'Sum not valid'
END
--------------------------------
```

Should there be a STOP in there somewhere?

Is this were "add" would require an ON_ERROR, 2? I can imagine some cases where you would want to keep both CATCH statements...

Subject: Re: Error Handling Advice
Posted by David Fanning on Mon, 14 Jan 2013 20:58:33 GMT
View Forum Message <> Reply to Message

Matthew Argall writes:

```
>
>>  I guess I will start with some with trial and ::ehem:: error ;-)
>
>  So, after a little work, I get a cascade of error messages
>
>  ---------------------------
>  function add, a, b
>      Catch, theError
```

```
>     IF theError NE 0 THEN BEGIN
>        Catch, /CANCEL
>        void = Error_Message()
>        RETURN, !Null
>     ENDIF
>
>     RETURN, a + b
>  end
>
>  pro call_add
>     Catch, theError
>     IF theError NE 0 THEN BEGIN
>        Catch, /CANCEL
>        void = Error_Message()
>        RETURN
>     ENDIF
>
>     sum = add(a, b)
>
>     IF sum EQ !Null THEN message, 'Sum not valid'
>  END
>  --------------------------------
>
>  Should there be a STOP in there somewhere?
```

No, there are a TON of errors in this program. You might just as well
find all of them before you move on. :-)

> Is this were "add" would require an ON_ERROR, 2? I can imagine some cases where you would want to keep both CATCH statements...

I would probably write this particular set of routines like this.

```
function add, a, b, SUCCESS=success
   Catch, theError
   IF theError NE 0 THEN BEGIN
      Catch, /CANCEL
      void = Error_Message()
      success = 0
      RETURN, !Null
   ENDIF

   success = 1
   result = a + b
   RETURN, result
end
```

```
pro call_add
   Catch, theError
   IF theError NE 0 THEN BEGIN
      Catch, /CANCEL
      void = Error_Message()
      RETURN
   ENDIF

   sum = add(a, b, SUCCESS=success)
   IF ~success THEN RETURN

   IF sum EQ !Null THEN message, 'Sum not valid'
END
```

Cheers,

David

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

Subject: Re: Error Handling Advice
Posted by David Fanning on Mon, 14 Jan 2013 21:02:28 GMT
View Forum Message <> Reply to Message

David Fanning writes:

```
>  I would probably write this particular set of routines like this.
>
>  function add, a, b, SUCCESS=success
>     Catch, theError
>     IF theError NE 0 THEN BEGIN
>        Catch, /CANCEL
>        void = Error_Message()
>        success = 0
>        RETURN, !Null
>     ENDIF
>
>     success = 1
>     result = a + b
>     RETURN, result
>  end
>
```

```
>  pro call_add
>     Catch, theError
>     IF theError NE 0 THEN BEGIN
>        Catch, /CANCEL
>        void = Error_Message()
>        RETURN
>     ENDIF
>
>     sum = add(a, b, SUCCESS=success)
>     IF ~success THEN RETURN
>
>     IF sum EQ !Null THEN message, 'Sum not valid'
>  END
```

Or, even simpler:

```
function add, a, b
   Catch, theError
   IF theError NE 0 THEN BEGIN
      Catch, /CANCEL
      void = Error_Message()
      RETURN, !Null
   ENDIF

   RETURN, a + b
end

pro call_add
   Catch, theError
   IF theError NE 0 THEN BEGIN
      Catch, /CANCEL
      void = Error_Message()
      RETURN
   ENDIF

   sum = add(a, b)

   IF sum EQ !Null THEN RETURN
END
```

Cheers,

David

--
David Fanning, Ph.D.

Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.idlcoyote.com/
Sepore ma de ni thue. ("Perhaps thou speakest truth.")

Subject: Re: Error Handling Advice
Posted by Matthew Argall on Mon, 14 Jan 2013 21:33:07 GMT
View Forum Message <> Reply to Message

> No, there are a TON of errors in this program. You might just as well
> find all of them before you move on. :-)

I guess my point was to make a case WITH a ton of errors to create the cascade and see how CATCH works. I could check to see if "a" and "b" are defined (one of your earlier points) but I would still have to use RETURN in CALL_ADD instead of MESSAGE.

If I am 6 programs deep into the stack, I do not want to generate 6 error messages as my Catch blocks work their way back up. Also, putting RETURN instead of MESSAGE does not seem to be a general solution to errors. (... or is it?)

Maybe my definition of "utility function" is too strict and I should think about using ON_ERROR, 2 more.

Will keep pondering...

Subject: Re: Error Handling Advice
Posted by David Fanning on Mon, 14 Jan 2013 21:47:52 GMT
View Forum Message <> Reply to Message

Matthew Argall writes:

> I guess my point was to make a case WITH a ton of errors to create the cascade and see how CATCH works. I could check to see if "a" and "b" are defined (one of your earlier points) but I would still have to use RETURN in CALL_ADD instead of MESSAGE.
>
> If I am 6 programs deep into the stack, I do not want to generate 6 error messages as my Catch blocks work their way back up.

I would say your concept of "error handling" just needs to grow a little. If you are calling a routine that can fail and return a NULL value, then part of your error handing, outside of Catch and ON_Error and all the rest of it, is to check the return value for the NULL condition and do something reasonable. If you NEED that value in the rest of the program, then you HAVE to check that you have it. Otherwise, errors *will* cascade.

Anyway, the only programs I know of that are 6 programs deep on the
stack are function graphics routines, and I think they routinely use
silent error handlers to deal with that. ;-)

Cheers,

David

P.S. Maybe it only *seems* like they use silent error handlers.
In any case, those routines are pretty much impossible for a layman to
debug.

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

Subject: Re: Error Handling Advice
Posted by Paul Van Delst[1] on Mon, 14 Jan 2013 22:21:52 GMT
View Forum Message <> Reply to Message

Hello,

On 01/14/13 16:33, Matthew Argall wrote:
>> No, there are a TON of errors in this program. You might just as
>> well find all of them before you move on. :-)
>
> I guess my point was to make a case WITH a ton of errors to create
> the cascade and see how CATCH works. I could check to see if "a" and
> "b" are defined (one of your earlier points) but I would still have
> to use RETURN in CALL_ADD instead of MESSAGE.
>
> If I am 6 programs deep into the stack, I do not want to generate 6
> error messages as my Catch blocks work their way back up.

Um... isn't that exactly what you want? It's the programmed equivalent
of a traceback.

I find the cascade/traceback errors I get in my code extremely helpful
in debugging where things went wrong.

> Also,
> putting RETURN instead of MESSAGE does not seem to be a general
> solution to errors. (... or is it?)

>
> Maybe my definition of "utility function" is too strict and I should
> think about using ON_ERROR, 2 more.
>
> Will keep pondering...
