Subject: Re: IDL 8.0 bug -- line number of errors not given
Posted by penteado on Wed, 13 Oct 2010 04:38:22 GMT
View Forum Message <> Reply to Message

On Oct 12, 11:46 pm, David Fanning <n...@dfanning.com> wrote:
> Paulo Penteado writes:
>> I was wondering if this was the case, since I agree this was
>> desirable.
>
> Humm. In what sense of the word do you find this "desirable"?

The way I see it, debugging when developing the code is usually most helped by the default on_error (0), so that one is left at the scope and line of the error, and can navigate between scopes to inspect things.

Using 2 for on_error could be appropriate only when one expects that the only source of problems occurring in the routine is in the parameters it gets passed, and every error inside the routine is being handled. So it would just get back to the caller, using message to inform the caller what was wrong with the parameters. In that situation, it would be less confusing for the user to know that the problem was with the parameters used in the routine call (through the message passed), instead of seeing a long sequence of errors from some routine called several levels below, which may not make it obvious why that routine call created a problem. In that sense, this is locating the source of the error more precisely, as it was the parameters used in the routine call, not some obscure code deep in some other routine, where the error was thrown.

The trouble with 2 for on_error is when the error is unhandled. Then just getting some like "% Specified offset to array is out of range: A" is really unhelpful. In such a situation, I would argue that 'on_error,0' should have been used.

>
>> Not showing the full trace is consistent with returning to the caller.
>
> Returning to the caller for what purpose?

For the purpose of saying (through message) that the problem was the way the routine was called, not leaving some unhandled error to happen inside the routine.

>
>> That is, the code below behaves as I expected, and agree that should
>> be the case:
>

> Why is this what you expected? Because of your experience
> with other software? What other software acts like this?
> You thought IDL's behavior in the past was inconsistent with
> good programming practices?

When all errors get handled in the routine, it can use 'on_error,2' to expose only an informative error message telling why the parameters used on its call caused trouble. What is publicly exposed is only the routine's interface and its error messages, not unhandled errors that are only informative if one knows the implementation.

I just thought that showing the full trace when a simple informative error message is given and scope is returned to the caller makes things a little more confusing. If one wants to see all the details, it is better to remain in the scope (and line) of the error, not return to the caller. In a dynamic language like IDL, just statically (out of the scope) looking at the line where the error was thrown may not say much anyway: it would be necessary to still be in scope, to inspect the variables used in that line, to figure out what happened. In such a situation, the traceback would help most to know where to add a breakpoint, to run the program again and stop in the line of trouble. That is, to get the same as if on_error was 0 and execution stopped at the error.

Returning to the caller is more consistent with just saying that the caller is at fault (and why) instead of showing what was going on inside.

Java has a much more elaborate error handling structure, and its checked exceptions, declared with throws, seem to me a similar idea, of a routine telling that it may give some error back to the caller (supposedly with an informative message), to let one know how what the problem was with the routine call.

Or, to put it another way, add an 'on_error,2' to a routine is somewhat like turning off traceback on a compiled language, to rely solely on the program's messages to inform of problems.

>
>> function test1
>> On_error,2
>> a = bindgen(32)
>> c = long(a,30,1)
>> return,1
>> end
>
>> pro test
>> print,test1()

>> return
>> end
>
>> IDL> test
>> % Compiled module: TEST.
>> % Specified offset to array is out of range: A.
>> % Execution halted at: TEST 9 /home/penteado/idl/
>> test.pro
>> % $MAIN$
>
>> As the trace shows where the place where the routine with 'on_error,2'
>> was called.
>
> Well, I agree this is the place it was called. But what possible
> good is this information? How would you use this information?
>
> Why would you use ON_ERROR, 2? Should it be eliminated from
> the language? How would you use it now that it is working
> as you expect it to?

In this case it is no good because the error was not handled, there is
no message to inform of the problem. In this case I think it would be
better to leave 0 for on_error.
