Subject: Re: Warning: IDL 8.0 alters the behaviour of existing valid programs without any notice!
Posted by Karl[1] on Fri, 20 Aug 2010 16:44:44 GMT
View Forum Message <> Reply to Message

On Aug 20, 5:42 am, svhhaugan <s.v.h.hau...@gmail.com> wrote:
> On Aug 19, 9:35 pm, Chris <beaum...@hawaii.edu> wrote:
>
>> In reality, how many people actually use CATCH to handle the case of a
>> failed call to where?
>
> As I *did* point out, the CATCH was not a very likely thing
> to be used in real life, it was only for illustrational
> purposes, in real life there would just be an informative
> error message and the command line (i.e. a crash) so you could
> explore what/where/why it went wrong. It's an interactive data
> language: relying on things to crash is *not* a bug if that's
> what you want to happen.
>
>> not checking the result of where is a bug.
>
> No. Nope. Says who? Where? It isn't. Not in IDL 7.
>
> Nor is it a bug in IDL 8, as other people have pointed out, b/c
> you can do data[where(...,/null)] = 0, that's not a bug, is it?
>
> Crashing is not a bug, if that's what you want your program to do in
> certain cases. So, a program can be bug-free under IDL 7, yet
> buggy when run under IDL 8.

Gee, it is too bad that we didn't have the null array when WHERE() was
first implemented! :-)

I think that Stein has a pretty good point, among all the other good
points that were made here.

1) It isn't totally unthinkable that someone would use a CATCH to
cause an early exit when a call to WHERE returned no matches. People
accustomed to using exception handling for implementing the normal
control flow in a program might do it. (See the xerces-c C++ XML
parser for a rather perverse example.). One might have:

```
pro test,data
  catch,error
  if error ne 0 then begin
    < clean up all resources >
    catch,/cancel
    return
  end
  < allocate and init everything >
  ; Get the data I want to analyze.
  ; I am only interested in non-zero array elements.
  ; If the entire array is 0, then we jump to the catch handler
  data_to_analyze = data[where(data ne 0)]
  < analyze data_to_analyze - the algorithm will NOT work with all
elements set to 0 >
  < throw intentional error to cause cleanup code to run >
end
```

IDL 8 will do this differently than IDL 7. Debating about how much code like the above is out there and whether or not the above code is good IDL practice isn't very useful - the above idiom could be in use someplace.

2) Yeah the command line use case will be different too. If I do the same thing as above:

```
IDL> data_to_analyze = data[where(data ne 0)]
```

I'll get an error on IDL 7 if data has all zeroes in it, and perhaps [0] in IDL 8? I'm not sure that this is good. On 8 I am likely to type in my next few commands, thinking that data had some non-zero data in it.

Anyway, I think that this is enough justification for ITTVIS to consider a notice to licensees and some sort of way to turn off negative indexing. There's a patch to 8 planned anyway, right?

I don't follow IDL all that closely and it's way to late to bring this up, but I don't see a lot of value in this negative indexing anyway. Fetching the array dims is cheap and doesn't have to be done often. Yeah, it might save a line or two of code. I don't think I've ever seen array indexing like this in another language. But perhaps I'm missing the key advantages of this feature.

Also bothersome is the "breaking" of the correspondence of array indices with adjacent array elements. That is, data[-1] is (often) not adjacent to data[0]. This just doesn't feel right, even though I understand the intent.

When I first heard of negative indexing, I thought they meant:

```
a = intarr(-4:4)
```

which would allocate 9 ints with a[-4] being the "first" element. But this would break nearly everything. Arbitrary indexing would be cool, but would have had to been there from the start.