Subject: Re: IDL 8.0 compile_opt changes
Posted by H. Evans on Mon, 21 Dec 2009 13:15:37 GMT
View Forum Message <> Reply to Message

On Dec 21, 6:33 am, ddegr...@stny.rr.com wrote:
> On Dec 19, 2:11 pm, Chris Torrence <gorth...@gmail.com> wrote:
>
>
>
>> Another point of information: We are adding a lot of new language
>> features to IDL 8.0. Things like a new "foreach" operator, negative
>> array subscripting a[0:-1],
>
>> More thoughts? More eggnog?
>
> No! negative index is a Bad Idea!
>
> z= where(bad ne 0)
> if z[0] ne -1 then begin
>   lots of stuff
> endif
>
> I have a lot of code like this, which would no longer be valid if -1
> were a legal array index. I could go thought and include a count in
> the where function but that's a lot more work than fixing all the ().
> Actually I was surprised that you could still use () for arrays. I
> tell my students to always use [].
>
> david

I also have large amounts of code that use exactly this type of range
limit checking. The completely correct solution is to get the number
of elements found in the where command and check on that. This,
though, doesn't solve the basic problem of having to envelop all
subsequent code in an IF/THEN/ELSE construct, which I find becomes
unwieldy, almost as unwieldy as x[N_ELEMENTS(x)-1], which looks much
nicer as x[-1].

I've often thought the WHERE function should be provided so it can be
used as follows:

```
  IF ( WHERE( z eq bad_data, i) ) THEN z[i] = !values.f_nan
```

I find it more succinct, less clumsy and more obvious than:

```
  I = WHERE( z EQ bad_data, count)
  IF ( count GT 0) z[i] = !values.f_nan
```

Obviously it's simple enough to implement a basic rewrite:

```
  FUNCTION IS_FOUND, condition, indices, COUNT=COUNT, _EXTRA=_EXTRA
    indices = WHERE( condition, count, _EXTRA=_EXTRA)
    RETURN, count GT 0
  END
```

But the underlying problem is how to maintain backwards compatibility
and provide a new syntax for indexing from the end of the array. I'm
not sure that the new file extension solution would work here, as it's
possible that the newer code would output correct -ve indices that
would be used as input to older code that would cease to function.

In the example you give above, it should be a matter of just ensuring
that the WHERE function continues to work as it currently does, but
the user must be aware that the "-1" return value is not to be used in
indexing. This is likely to break older code that doesn't check that
the WHERE found any valid indices, i.e.:

```
  i = where( x eq bad_data)
  x[i] = !values.f_nan
```

But then, this sort of code is quite likely to be a source of problems
in the first place and would be improved by the new coding
restrictions.

Alternatively, could a new data type be created for indexing, e.g.
ARRAY_INDEX? An index is more than just an ordinal. Typecasting could
be used transparently to convert much of the existing code to the
newer index type. The type could have an equivalent undefined/NaN
value for no indices found, and while -ve numbers could refer to
reverse indexing a more robust solution could be implemented. This
could also then be shoehorned into the old way of working with a
suitable pragma option, set either through the file name extension or
the compile_opt statements. For .pro files, the undefined value would
be returned as -1L, and for the .prx files it could be genuinely
undefined. This should stop old code from breaking and still allow for
the extension of the reverse indexing.

I believe the opportunity for addressing many of the existing
shortcomings in the language should be grasped if the option for a new
file extension is chosen.

H.