Subject: Re: Structure field concatenation
Posted by John-David T. Smith on Mon, 11 Sep 2000 18:59:59 GMT
View Forum Message <> Reply to Message

Martin Schultz wrote:
>
> Now, if an american is getting pedantic on this issue, I would really
> like to be thorough here ;-)
>
> Let's state the "problem" again:
> We already have a pointer variable, and we want to reassign a new
> value to it. But, as typical for "real life" situations, we don't know
> for sure if the pointer that we have is valid or not.
>
> This leads to two possible solutions:
>
> dwf:
```
>    IF NOT Ptr_Valid(myptr) THEN Ptr_New(myptr)
>    *myptr = thingy
```
>
> mgs:
```
>    IF Ptr_Valid(myptr) THEN Ptr_Free, myptr
>    myptr = Ptr_New(thingy, /No_Copy)
```
>
> Both solutions should lead to the same result and with approximately
> the same speed (at least if I add the No_Copy keyword to my solution).
> So, the difference is that thingy is still accessible in David's
> solution while it becomes undefined in mine (and David's solution
> saves a couple of keystrokes which you can then use for
> documentation).
>
> Well, if there were no other hidden differences, I would declare
> victory to David. BUT, there is in fact a difference!! If you want to
> point to an undefined variable (well, who wants to do this anyway),
> David's solution breaks, whereas my way happily creates a new pointer
> pointing to an undefined variable. Thus, the fail-safe dwf solution
> would be:
>
```
>    IF NOT Ptr_Valid(myptr) THEN Ptr_New(myptr)
>    IF N_Elements(thingy) GT 0 THEN *myptr = thingy $
>    ELSE ...  ; either stop, print a warning, create a new pointer, or
> ...
```
>
> And this gives me leeway for more documentation ;-)
>

Time to double-down the pedantry. Both methods you mention are perfectly valid
and useful, in different ways, but your argument is unrelated to the topic I was

addressing. The notion I attempted to dispel was the need for a "safety net" of any kind when reassigning an already allocated pointer's value, as if IDL's memory management in this case could not be trusted. By arguing that this pointer heap variable management was tantamount to traditional variable memory management, I showed that if you trust the latter (a condition I cannot prove), then you implicity trust the former.

And now to address your slightly rescoped points along with David and Ben's rejoinders...

You could slightly improve your method by changing the first line to simply:

```
Ptr_Free, myptr
```

which is faster than an "if" test in the case of myptr being null (and saves you even more keystrokes). This allows very simple pointer cleanup when used appropriately (as I discussed a few weeks back).

As for David and Ben's argument for undefined ptr values, I argue that a non-existent pointer is just as valid a placeholder of a null field or empty list as a pointer pointing to an undefined value. The test simply changes from:

```
IF Size(theValue, /TName) EQ 'UNDEFINED' THEN
```

to

```
IF NOT ptr_valid(theValue) THEN
```

Also, how is an undefined pointer created manually after the fact -- i.e., how do you "empty" am already filled list? Something awkward like:

```
ptr_free, theValue
theValue=ptr_new(/alloc)
```

where I would only need the first line. To be fair to the advantages of the Undefined method... if changing the pointed-to value altogether, they need only:

```
*theValue=newvalue
```

whereas I require:

```
if ptr_valid(theValue) then *theValue=newvalue else theValue=ptr_new(newvalue)
```

A remaining issue which affects me by far the most, is one of appending data to a pointer heap variable which is an array (possibly not yet existing).   I end up with much code looking like:

```
if ptr_valid(self.arr) then *self.arr=[*self.arr,newval] else $
 self.arr=ptr_new([newval])
```

Which seems wordy. Unfortunately, and undefined value method does not help:

```
IDL> arr=ptr_new(/alloc)
IDL> *arr=[*arr,1] ; oops, won't work!
% Variable is undefined: <PtrHeapVar4>.
```

you end up needing:

```
if Size(self.arr, /TName) EQ 'UNDEFINED' THEN *self.arr=[newval] else $
 *self.arr=[*self.arr,newval]
```

not any better.

And what about Martin's method of pointer rebirth? It simply doesn't work for extending pointed-to arrays without an awkward temp variable.

```
tmp=*self.arr
ptr_free,self.arr
if Size(tmp, /TName) EQ 'UNDEFINED' then self.arr=ptr_new(newval) else $
  self.arr=ptr_new([tmp,newval])
```

So, as is usual with these arguments, there is no right or wrong answer. Here are the choices with the pros and cons:

1. I prefer keeping my heap variables in place, and indicate lists are empty with null pointers (by freeing the pointers as appropriate), which becomes quite easy. My "empty" lists take up no memory on the heap. I pay for this emptying ease when reassigning a pointed-to value, where I require a test to ensure the pointer heap variable exists (creating one if necessary). Appending to pointed-to arrays is reasonably easy.

2. David and Ben prefer to keep their variables around even longer, indicating empty lists with pointers to undefined variables. They can easily reassign pointed-to values (since the heap variable will always be there), but manually emptying a list is more awkward, requiring a variable to be freed and reassigned to a newly created undefined heap variable (ptr_new(/alloc)). Appending to pointed-to arrays is about as hard as method #1.

3. Martin prefers continuously recreating his pointer heap variables, which allows him to assign undefined variables, but not really more easily than in #2, since he's always freeing them anyway just as when emptying a list in that method. It does allow him to assign *unexpectedly* undefined variables (like typos? ... hmm, not sure if this is a pro or a con, or unpassed arguments, as David comments), since the semantics are exactly the same either way, in contrast to #2. The empty list could reasonably be either that of #1 or #2.

Appending to (possibly nonexistent) arrays is very awkward compared to the former two.

So there you have it. None exist in exclusion of the others, and each can borrow from the other as appropriate. But note that it is very convenient to adopt a single "list is empty" paradigm so your code easily interoperates.

JD

```
--
 J.D. Smith                             /*\    WORK: (607) 255-6263
 Cornell University Dept. of Astronomy  \*/          (607) 255-5842
 304 Space Sciences Bldg.               /*\     FAX: (607) 255-5875
 Ithaca, NY 14853                       \*/
```