Subject: Re: Structure field concatenation
Posted by John-David T. Smith on Mon, 11 Sep 2000 19:22:39 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
>
> Ben Tupper (btupper@bigelow.org) writes:
>
>> Martin Schultz wrote:
>>
>>>
>>> If you want to
>>> point to an undefined variable (well, who wants to do this anyway),
>>>
>>
>> Hello,
>>
>> I hate to pipe up because I'm the gear-slipping dope that may have started this mess
>> (although I'm learning much just listening.)   I find utility in a pointer to an
>> undefined variable useful when working with lists of things that a user can
>> completely empty. (Like a base map with or without any number of overlays, or the
>> datasets have not been loaded yet.) Isn't it analogous to a container object that is
>> waiting for additions?
>
> Maybe Ben *used* to be a "gear-slipping dope", but
> that hardly describes him lately. Let's just say I've
> been pushed over my knowledge horizon more times than
> I like to admit in the past couple of months by his
> thought-provoking questions. :-)
>
> In any case, I'd like to provide more support for the
> utility of pointers to undefined variables. Let me give
> you a specific example: CW_FIELD.
>
> If you set CW_FIELD up so that it will accept, say,
> integer values, and you happen to leave the field
> blank, then when you go and get the value in the field
> it will return a 0 to you. What's wrong with that?, you
> say.
>
> What is wrong is that a 0 is a valid integer value.
> So you go willy-nilly on with your code thinking that
> you have got something decent. But suppose the number
> were the X Size of an image. And now suppose you want
> to Congrid your image into this size:
>
>   displayImage = Congrid(image, xsize, ysize)
>

```
> This causes an error. But now the error message is
> *very* strange (probably impossible to understand if
> you don't have a lot of IDL experience) and is one
> step (and many lines of code) removed from where the
> error really occurred.
>
> If CW_FIELD had returned an undefined variable (well,
> what else would it be if the field was blank?), then
> the error would have been something that is easily
> understood. What is more, it is something that could
> be easily checked for:
>
>    Widget_Control, fieldID, Get_Value=theValue
>    IF Size(theValue, /TName) EQ 'UNDEFINED' THEN $
>       Message, 'Whoops. Field is blank. Try again.'
>
> You could argue that you could as easily check to see
> if the size is 0, and you would probably be right,
> except in those cases where 0 is a valid value. Then
> you are really out of luck. (Unless you decide, as
> the authors of CW_FIELD did, that 0 is the default
> value if the value is undefined. Dubious, at best.)
>
> In any case, I find it *much* more useful to get
> an undefined variable when the field is undefined,
> so that is why FSC_INPUTFIELD, which is my CW_FIELD
> replacement that looks editable on Windows machines,
> uses pointers to store the value. If the field is
> undefined, then the pointer points to an undefined
> variable.
```

Using undefined variables as placeholders for empty lists or null fields is a fine idea on the surface (usually we must resort to things like -1, c.f. where()). The problem is you can't return them from functions or assign them to variables. If so I could have:

```
wh=where(indgen(10) lt 0)

if defined(wh) then blah
```

but alas, I must use -1 as the test. If we collected all of the semantics for functions which need to indicate that they are returning nothing, the list would be frightfully long...

-1 where >0 is expected
0 where >1 is expected
any scalar where an array is expected
any number where a string is expected

any string (like '') where a number is expected
a number (like 0) where a pointer is expected
...

and so on. A natural value for *all* of these is "undefined", which can be returned through arguments (keyword or positional), but not by functions. We wouldn't have to scratch our heads every time we wanted to rest a return value. I myself use a collection of the above methods in my own code.... wasted time thinking about which test to use.

If we wanted to live with the present situation, we could adopt the policy that any routine which is to return a value which may be null or empty should be implemented as a procedure. But wait, even that won't work. Why? Consider:

```
pro getit, var, SOMETHING=something
 if something then var=1
end

IDL> getit,var
IDL> print,size(var,/TNAME)
UNDEFINED
IDL> getit,var,/SOMETHING
IDL> print,size(var,/TNAME)
INT
IDL> getit,var
IDL> print,size(var,/TNAME)
INT
```

Uh oh, we can't tell our variable wasn't set on the last one... it retains its previous value. The pass-by-reference of arguments has defeated us. This is where a language like Perl excels. There you could simply say "var=undef". If we had a little more control over when and where our variables get undefined, we'd be much better off. Heck I might even switch to the pointer-to-undefined-heap-var for my empty lists on the heap.

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                    \*/
```