Subject: Re: Q: Array of Structures
Posted by J.D. Smith on Fri, 12 Jun 1998 07:00:00 GMT
View Forum Message <> Reply to Message

David Fanning wrote:

```
>
> Saeid.Zoonematkermani@sunysb.edu writes:
>
>> Thanks for the response. I think there is more to this problem than is
>> apparent at first. I believe that my structures are created identically.
>> It is the output of a complicated procedure. The structure is defined
>> inside the procedure and they should be the same each time. And just to
>> check this, I ran your example from above and here is the result:
>>
>> IDL> a = { data:FltArr(10), name:'First Structure' }
>> IDL> b = { data:FltArr(10), name:'Second Structure' }
>> IDL> c = [a, b]
>> % Conflicting data structures: B,concatenation.
>> % Execution halted at:  $MAIN$
>>
>> Rather curious. I wonder if this is a MacOS version problem!
>
> Oh, oh. Another one of those days. I *tested* this. I have the
> file to prove it! But you are right, I can't reproduce it. :-(
>
> I guess string lengths in structures have to be exactly the
> same, too. Which surprises me and goes contrary to what happens,
> for example, when you replicate named structures with the
> REPLICATE command.
>
> But, look. I just tried this:
>
>    a = { data:FltArr(10), name:'' }
>    b = { data:FltArr(10), name:'' }
>    c = [a, b]
>    % Conflicting data structures: B,concatenation.
>    % Execution halted at:  $MAIN$
>
> But now, I try this:
>
>    a = { data:FltArr(10), name:'' }
>    b = a
>    c = [a, b]
>
> No problem!
>
> Very, very weird. I'll see if RSI can explain this. :-)
>
```

> Cheers,
>
> David

As far as I know, you cannot, *in general*, concatenate anonymous structures (I mean into a list, not into a larger structure which is their superset, which can be accomplished with create_struct). I have a possible guess as to what may be happening.

For concatenation of structures, IDL probably tests the *types* of structures to see whether the concatenation is allowed, rather than the fields and field types contained in those structures (and much less the actual data members).

E.g.

```
IDL> a={S1, name:'name'}
IDL> b={S1, name:'alongname'}
IDL> c=[a,b]
```

...no problem, no matter what data lengths are involved, as long as those types are the same. On the other hand, when you create an anonymous structure, IDL likely gives it some random internal structure type -- not for public consumption, but merely as a convenience. For replicate, as for your second example above, the exact same structure (of some mysterious "internal anonymous type") is being *copied*, and no new "internal anonymous type" is created. This type of anonymous structure concatenation succeeds, since the structure types (albeit unknown to us) are the same, being mere copies of a original anonymous structure. On the other hand, two successive anonymous structure creations yield two *different* internal types, which, according to the rules, cannot be concatenated.

Why do anonymous structures need this internal type? Likely because if they didn't have it, replicate and other such functions couldn't work with the same code for both named and anonymous structures. Anonymous structures clearly cannot all have the *same* internal type, since then something like:

```
IDL> a=[{test:5},{test:[1,2,3,4,5]}]
```

would be allowed. The only options are either to treat anonymous structures specially, or to use fields and data member types (not structure type) to determine when structure concatenation is allowed. The former might be a pain in the neck. The latter would significantly slow down structure operations, increasing the burden of structure type-checking from one time only (at the time of creation) to possibly very many times.

There may not be any other compelling reasons, and a new way of dealing with anonymous structures might even get implemented, if we make enough noise. As a reminder, this may be a totally uninformed guess of the way IDL works (seems plausible to me, though :)).

For a solution to your problem, just use a named structure when you need dynamically
modified lists of them. They are more inconvenient, but also more reliable in these
situations. If you do object oriented programming, an easy place to define them is
in the same class__define procedure used to prototype your object class.

JD

--
 J.D. Smith |*| WORK: (607) 255-5842
 Cornell University Dept. of Astronomy |*| (607) 255-4083
 206 Space Sciences Bldg. |*| FAX: (607) 255-5875
 Ithaca, NY 14853 |*|