Subject: Re: dynamic array workaround?
Posted by JD Smith on Mon, 15 Aug 2005 19:13:01 GMT
View Forum Message <> Reply to Message

On Sun, 14 Aug 2005 15:01:46 -0700, Jeff N. wrote:

> Hi everyone,
>
> I haven't seen anything in the help files that would lead me to believe
> that IDL has a way to do something like the following:
>
> Say I want to loop through a few lists of numbers, and create one big
> array of all numbers in the lists that meet some criteria. Since I
> don't know beforehand how many numbers will meet the criteria, I don't
> know how many elements that one big array will need.
>
> If I were writing code in a language like PERL maybe, I'd use a dynamic
> array, but I don't think that IDL supports those. Any idea as to how I
> could get around this? What I've been doing is to just initialize a
> scalar, then assign new elements to it in the loops, then just removing
> the 1st element when I'm done looping. I guess I could also loop
> twice, with the first round of loops serving to find out how many array
> elements I'd end up needing. Both approaches seem inefficient to me.
> What else could I try? I had a sneaky suspicion that the TEMPORARY
> function was what I needed, but somehow that doesn't quite look right

We've bemoaned the lack of intelligently expanding arrays in IDL; have a
look here:

http://groups.google.com/group/comp.lang.idl-pvwave/msg/42be ec7009b3d525?hl=en&

and here:

http://groups.google.com/group/comp.lang.idl-pvwave/browse_f
rm/thread/297f0ce2af6d092f/7f166469a0c1de6d

These include some snippets and links explaining how you can allocate
new memory in increasing block sizes. Note that Perl is just doing
this type of magic (and much higher magic) for you, behind the scenes.
As pointed out, for large dynamic arrays, allocating one additional
array cell at a time is horribly inefficient. It occurs to me that
something which implements a doubling strategy or otherwise could be
easily rolled into a routine (call it PUSH) which keeps track of the
allocation for you. Then you'd just have to pay the routine call
overhead for each cell (which is not *nearly* as bad as the full
"allocate N+1 bytes, copy and free N original bytes" of
concatenation).

TEMPORARY is great if your data size isn't changing, and you just want
to shuffle the data in memory itself between different IDL variables
(including pointer variables). It doesn't really help with
allocation/concatenation.

JD