Subject: Re: object newbie
Posted by Martin Schultz on Mon, 14 Aug 2000 07:00:00 GMT
View Forum Message <> Reply to Message

"J.D. Smith" wrote:
>
> Martin Schultz wrote:
>>
>>
>>
>> The GetProperty method may in fact use the special retrieval methods to
>> extract things. As long as you store only small amounts of data, it
>> won't matter if you access the campaigns structure array several times
>> and pass parts of it between methods. If you envision huge amounts of
>> data you may want to think more carefully how often these arrays must be
>> copied. I haven't dealt with these issues yet, but I would be happy to
>> hear comments.
>>
>> Cheers,
>> Martin
>
> And J.D. Smith commented:
>
>
> As far as the array copying issue, for dealing with those properties which are
> truly large, the only way to pass by reference out of your GetProperty method is
> to use pointers... which is actually good: imagine the confusion of having
> otherwise unremarkable variables as silent referents to object data members.
> Pointers are your friends.
>

I fully agree, but ... ;-)

I am currently reworking a netcdf file object (based in turn on a
geenric datafile object, but that"s a detail), and I am thinking about
how to best achieve the two conflicting goals:
* allow flexible access to subportions of the data
* minimize memory usage

What I envision is a method that would allow something like
  thefile->GetData, variables=['o3','co'], /include_dimensions,
lon=,lat=
which should return an array containing only the selected data. But this
means that I have to copy the data rather than pass a pointer back,
because otherwise the next access with a different range would overwrite
the data stored in the object and thus render the first pointer
"invalid" in the sense that it would point to something different now.
The problem arises, because I do not want to read in the whole field

first and then extract what I need but I want to take advantage of
netcdf's capabilitity to extract a subset of the data directly from the
file. So, the data will be loaded dynamically only when requested. On
the other hand it will also be possible to retrieve the complete
variable data from the file and do the extraction in memory, which may
be considerably faster if you need frequent access to various portions
of the data. For example, the dimension variables
    (for non-netcdf'ers: these are variables holding the values of the
dimensions of other variables.
    Example: a model ozone field may have dimensions LON, LAT, and LEV,
then LON will be a variable
    named LON which contains the longitude values, i.e. 2.5, 7.5, 12.5,
... 357.5 on a rather coarse grid)
will always be loaded into memory and accessed from there. Makes things
a little more convoluted if I want to support write access to the netcdf
file as well (which I do want), but it's a nice challenge, and
thanks to the beauty of objects, the user doesn't have to care a dent.

As to pointer freeing: I remember that it took me three or four attempts
(about 15 years ago) before I had a vague idea what a pointer was and
how one could use one. Well, this was Pascal (maybe Borland 3 or so),
and things have certainly changed since then (my little daughters
learned about pointers at age 2 already: "Where is pointer, where is
pointer? Here I am..."). But the key to not loosing pointers nor data is
that you should make it clear to yourself where your data actually
resides. What often helps me is the concept of a "master pointer", i.e.
the one pointer that got the data passed first. And then I usually only
allow the master pointer to free the data. All other pointers are
considered local variables (they are just unsigned longs in a sense),
and I don't need to care about memory allocation or freeing for these.
Example:

```
   master = Ptr_New(data)   ; this is the master pointer
   other = master           ; this is only a local pointer
   (*other) = (*other)*0.5  ; this operation affects the data
which is conceptually stored in master
   Ptr_Free, master         ; free memory
```

So, you don't have to worry about the fate of other at all (just that
one should of course check if it is a valid pointer before using it).

[excerpt from b2p2 = beginners' guide to pointers, unpublished material,
2000, copyright = left]

Cheers,

Martin

--
[[[