Subject: Re: Read Total lines in an ASCII file
Posted by Mark Hadfield on Mon, 16 Dec 2002 23:06:19 GMT
View Forum Message <> Reply to Message

First, I am pleased to announce that my Motley library is back on-line. Don't break out the methode traditionelle yet, because its current home is only temporary, but I do have permission from the people who own the code (and pay me my salary) to publish the code. See:

ftp://ftp.niwa.co.nz/incoming/m.hadfield/idl/README.html

Now about this "read total lines in an ASCII file" thing, I have been inspired to implement 4 methods in my MGH_TXT_RESTORE function, which reads the contents of a text file into a string array. The methods are:

0 - Read lines one at a time, accumulating them in a string array. Extend the array as necessary in increments of 70% or so (see code). Trim the result before returning.

1 - Read the file once to count the lines, create a result array of the required size, then read the file again.

2 - Create a result array of size MAX_LINES, read the file, then trim the result. This method may return fewer lines than the other methods, as all empty lines are trimmed from the end.

3 - As 0, but accumulate the results in an MGH_Vector object then move them into a string array at the end.

4 - As 0, but accumulate the results in a string array, extending the array by just one element each time.

These methods can be exercised by routine MGH_EXAMPLE_TXT_RESTORE. Here are results for a 20,000 line uncompressed file on my machine:

0 - 0.38 s
1 - 0.25 s
2 - 0.14 s
3 - 0.82 s
4 - several minutes so far @ 100% CPU utilisation.

Method 2 (the one posted by Med Bennett earlier in this thread) is the speed winner, but I don't like the fact that you need to specify the maximum file size in advance. Method 1, the two-pass method, is surprisingly quick here, but suffers if the file is on a slow network

drive. Method 0 is my favourite. Method 3 uses the MGH_Vector object, which uses the same array-growing approach as method 0; I wrote the MGH_Vector object to hide all the ugly array-growing details but in hindsight the performance penalty is too large (and the array-growing code not all that ugly, anyway). Method 4 was included to make the point that it's horrendously slow when the number of array-extending operations exceeds 1000 or so; however I can't be bothered waiting to see how long it will take.

--
Mark Hadfield "Ka puwaha te tai nei, Hoea tatou"
m.hadfield@niwa.co.nz
National Institute for Water and Atmospheric Research (NIWA)
