Subject: Re: REPLICATE with arrays
Posted by Craig Markwardt on Fri, 11 Feb 2000 08:00:00 GMT
View Forum Message <> Reply to Message

davidf@dfanning.com (David Fanning) writes:
> Vince Hradil (hradilv@yahoo.com) writes:
>
>> I often have the need to replicate an array, but IDL's replicate only
>> works with scalars. Does anyone have any tips on the most efficient,
>> simplest, clearest (you choose) way to do this?
>
> I am such a sucker for these kinds of articles. :-(
> ...

I'll add my implementation to the competition. Here is CMREPLICATE which takes either a scalar or array. It uses the REFORM/REBIN magic that has already been discussed, for numeric types. For other types I did have to revert to MAKE_ARRAY/copy technique, but I've sped up the copy vs a simple for loop. Obfuscatory I am sure!

It's also available at my web page (and if you are a regular check the NEWS too, since I've made a few other updates):

http://cow.physics.wisc.edu/~craigm/idl/idl.html

Craig

```
;+
; NAME:
;   CMREPLICATE
;
; AUTHOR:
;   Craig B. Markwardt, NASA/GSFC Code 662, Greenbelt, MD 20770
;   craigm@lheamail.gsfc.nasa.gov
;
; PURPOSE:
;   Replicates an array or scalar into a larger array, as REPLICATE does.
;
; CALLING SEQUENCE:
;   ARRAY = CMREPLICATE(VALUE, DIMS)
;
; DESCRIPTION:
;
;   The CMREPLICATE function constructs an array, which is filled with
;   the specified VALUE template.  CMREPLICATE is very similar to the
;   built-in IDL function REPLICATE.  However there are two
;   differences:
;
```

```
;     * the VALUE can be either scalar or an ARRAY.
;
;     * the dimensions are specified as a single vector rather than
;       individual function arguments.
;
;   For example, if VALUE is a 2x2 array, and DIMS is [3,4], then the
;   resulting array will be 2x2x3x4.
;
; INPUTS:
;
;   VALUE - a scalar or array template of any time, to be replicated.
;           NOTE: These two calls do not produce the same result:
;                 ARRAY = REPLICATE( 1,  DIMS)
;                 ARRAY = REPLICATE([1], DIMS)
;           In the first case the output dimensions will be DIMS and
;           in the second case the output dimensions will be 1xDIMS
;           (except for structures).  That is, a vector of length 1 is
;           considered to be different from a scalar.
;
;   DIMS - Dimensions of output array (which are combined with the
;          dimensions of the input VALUE template).  If DIMS is not
;          specified then VALUE is returned unchanged.
;
; RETURNS:
;   The resulting replicated array.
;
; EXAMPLE:
;   x = [0,1,2]
;   help, cmreplicate(x, [2,2])
;     <Expression>    INT       = Array[3, 2, 2]
;   Explanation: The 3-vector x is replicated 2x2 times.
;
;   x = 5L
;   help, cmreplicate(x, [2,2])
;     <Expression>    LONG      = Array[2, 2]
;   Explanation: The scalar x is replicated 2x2 times.
;
; SEE ALSO:
;
;   REPLICATE
;
; MODIFICATION HISTORY:
;   Written, CM, 11 Feb 2000
;
;-
; Copyright (C) 2000, Craig Markwardt
; This software is provided as is without any warranty whatsoever.
; Permission to use, copy, modify, and distribute modified or
```

```
; unmodified copies is granted, provided this copyright and disclaimer
; are included unchanged.
;-
function cmreplicate, array, dims

  if n_params() EQ 0 then begin
     message, 'RARRAY = CMREPLICATE(ARRAY, DIMS)', /info
     return, 0L
  endif

  if n_elements(dims) EQ 0 then return, array
  if n_elements(array) EQ 0 then $
    message, 'ERROR: ARRAY must have at least one element'

  ;; Construct new dimensions, being careful about scalars
  sz = size(array)
  type = sz(sz(0)+1)
  if sz(0) EQ 0 then return, make_array(value=array, dimension=dims)
  onedims = [sz(1:sz(0)), dims*0+1] ;; For REFORM, to extend # of dims.
  newdims = [sz(1:sz(0)), dims    ] ;; For REBIN, to enlarge # of dims.
  nnewdims = n_elements(newdims)

  if nnewdims GT 8 then $
    message, 'ERROR: resulting array would have too many dimensions.'

  if type NE 7 AND type NE 8 AND type NE 10 AND type NE 11 then begin
     ;; Handle numeric types

     ;; Argghh!  Why doesn't REBIN take an *array* of dimensions!
     ;; *Instead* we need to run EXECUTE(), with a string like this:
     ;; rebin(array1, newdims(0), newdims(1), ...)
     ;; That's what the following format string does.
     fmt = '('+strtrim(nnewdims,2)+'("newdims(",I0,")",:,","))'
     arglist = string(lindgen(nnewdims), format=fmt)
     cmd = 'return, rebin(reform([array], onedims),'+arglist+')'
     dummy = execute(cmd)

     ;; If execution reaches here then an error occurred.
     message, 'ERROR: array could not be resized'
     return, 0L

  endif else begin
     ;; Handle strings, structures, pointers, and objects separately

     ;; Handle structures, which are never scalars
     if type EQ 8 AND sz(0) EQ 1 AND n_elements(array) EQ 1 then $
       return, make_array(value=array, dimension=dims)
```

```
    nold = n_elements(array)
    nadd = 1L
    for i = 0L, n_elements(dims)-1 do nadd = nadd * long(dims(i))
    array1 = make_array(type=sz(sz(0)+1), dimension=[nold,nadd])
    array2 = reform([array], n_elements(array))

    ;; Efficient copying, done by powers of two
    array1(0,0) = array2
    stride = 1L   ;; stride increase by a factor of two in each iteration
    i = 1L & nleft = nadd - 1
    while nleft GT stride do begin
       array1(0,i) = array1(*,0:stride-1)  ;; Note sneaky IDL optimization
       i = i + stride & nleft = nleft - stride
       stride = stride * 2
    endwhile
    if nleft GT 0 then array1(0,i) = array1(*,0:nleft-1)

    return, reform(array1, newdims, /overwrite)
  endelse

end
```
