
Subject: Re: Checking version number

Posted by [Michael Galloy](#) on Fri, 12 Jun 2009 23:14:51 GMT

[View Forum Message](#) <> [Reply to Message](#)

Kenneth P. Bowman wrote:

```
> I don't like to write version-dependent code, but with the improved PS
> driver in IDL 7.1 I would like to keep my old workarounds for 24-bit
> color available.
>
> The quantity stored in !VERSION is a string
>
> IDL> help, !version.release
> <Expression>  STRING  = '7.1'
>
> But this is not simple to convert to a number, because there are
> instances like this
>
> IDL> help, !version.release
> <Expression>  STRING  = '6.4.1'
>
> Is there an easy way to test whether the version is greater than or equal to 7.1?
>
> Ken Bowman
```

I use MG_CMP_VERSION, which is tacked onto the end of this post. It turned out to be more complicated than I thought it should be. This type of thing is making me think more about the "look before you leap" (LBYL) philosophy versus the "it's easier to ask forgiveness than permission" (EAFP) philosophy. Maybe there should be a routine which just tries to set DECOMPOSED=1 and has an error handler that does the right thing if that fails?

MG_CMP_VERSION is also part of the dist_tools package which can be grabbed via the Workbench update mechanism:

URL: http://updates.idldev.com/dist_tools

See the following article if you need help how to use the above URL:

<http://michaelgalloy.com/2009/06/03/idl-71-distributing-code.html>

There is also convenience routine MG_IDLVERSION that does what is usually required:

```
IDL> print, mg_idlversion(require='7.1')
1
```

Mike

--

www.michaelgalloy.com
Associate Research Scientist
Tech-X Corporation

; docformat = 'rst'

;+

; Compares two version numbers for the more updated number. Returns 0 for
; equal versions, 1 if version1 is later than version2, and -1 if
version1 is

; less than version2. Strings such as 'alpha' and 'beta' may be tacked
on to

; the end of a version, but are compared alphabetically.

;
;
;

;:Examples:

; For example, 1.2 is later than 1.1.2::

;

; IDL> print, mg_cmp_version('1.2', '1.1.2')

; 1

;

;:Returns:

; -1, 0, or 1

;

;:Params:

; version1 : in, required, type=string

; first version number

; version2 : in, required, type=string

; second version number

;-

function mg_cmp_version, version1, version2
compile_opt strictarr

v1parts = strsplit(version1, '.', /extract, count=v1len)

v2parts = strsplit(version2, '.', /extract, count=v2len)

nparts = v1len > v2len

v1partsValues = lonarr(nparts)

v2partsValues = lonarr(nparts)

v1partsValues[0] = long(v1parts)

v2partsValues[0] = long(v2parts)

for i = 0L, nparts - 1L do begin

if (v1partsValues[i] gt v2partsValues[i]) then return, 1

if (v1partsValues[i] lt v2partsValues[i]) then return, -1

```

if (i eq nparts - 1L) then begin
  nondigitpos1 = stregex(v1parts[i], '^[[:digit:]]')
  nondigitpos2 = stregex(v2parts[i], '^[[:digit:]]')

  if (nondigitpos1 eq -1L && nondigitpos2 eq -1L) then return, 0
  if (nondigitpos1 eq -1L) then return, 1
  if (nondigitpos2 eq -1L) then return, -1

  case 1 of
    v1parts[i] lt v2parts[i]: return, -1
    v1parts[i] gt v2parts[i]: return, 1
    else : return, 0
  endcase
endif
endfor

return, 0
end

```

; main-level example

```

v = ['1.2', '1.1.2', '1.1', '1.1alpha', '1.1beta']
colwidth = max(strlen(v)) + 2

```

```

print, "", format='(A13, $)'
print, v, format='(5A10)'

```

```

for v1 = 0L, n_elements(v) - 1L do begin
  print, v[v1] + ' > ', format='(A13, $)'
  for v2 = 0L, n_elements(v) - 1L do begin
    print, mg_cmp_version(v[v1], v[v2]), $
      format='(I10, $)'
  endfor
  print
endfor

end

```