U4. Color Utility Commands

ViewIt supports several commands to help programmers get and set colors, plus a single command to switch the color palette associated with a program. The need for help with getting and setting colors stems from the fact that the color environments available on the Mac range from B & W w/o Color QuickDraw to the latest version of 32-bit QuickDraw running on 32-bit deep screens. By using utility commands to get and set colors, drawing can be done without needing to be aware of the current color environment.

Name Number Parameters & Variables used

GetFgC 178 a,b,c,d,uRGB,uResult,uMenuID, uMenuItem,ul2

GetBkC 179 a,b,c,d,uRGB,uResult,uMenuID, uMenuItem,ul2

Returns the foreground, GetFgC, or background, GetBkC, color from the designated port or color table. When getting a color in a color table, the only difference between GetFgC and GetBkC is that the former returns black and the latter white when a color cannot be found. All forms of GetFgC and GetBkC work in all color & non-color environments.

◦ if b = 0 then...
a = source window or port
0 = front modal or active modeless window
1 = current port
other = WindowPtr or GrafPtr
c & d are not used (pass 0)

◦ if b = -1 then...
c = clut ID or handle
d = clut index (1-based)
a is not used (pass 0)

◦ if b = -2 then...
c = clut ID or handle
d = clut value (partID)
a is not used (pass 0)

where the unusual use of parameter b to determine the type of action follows that seen in the SetFgC/BkC commands.

For both color and non-color ports, the color is returned in both uRGB and in the three 4-byte integer variables uResult, uMenuID, and uMenuItem, where the latter either contain an old-style color constant in uResult and -1 in uMenuID and uMenuItem, or the three RGB color components in the low word of each variable.

When getting a color by index from a clut, ul2 returns the entry's partID. When getting a color by partID, and the partID cannot be found in the clut, then ViewIt will get the color from clut 1211 which contains a copy of the standard control colors.

ViewIt includes clut 1210 which contains the 8 old-style colors as RGB colors with part IDs equal to the old-style color constants. This facilitates finding the RGB equivalent of an old-style color by calling GetFgC with b = -2, c = 1210, and d = old-style color constant (= the partID).

SetFgC 180 a,b,c,d,uRGB,uResult

SetBkC 181 a,b,c,d,uRGB,uResult

Resets the foreground, SetFgC, or background, SetBkC, color-related fields of the designated port's (C)GrafPort record, where parameters b, c, and d specify the color. All forms of SetFgC and SetBkC will work in all color and non-color environments. Black (foreground) or white (background) is used as the default color if the specified color is not supported in the current environment. If the port's color is changed, ViewIt returns uResult ≠ 0.

a = target window or port
0 = front modal or active modeless window
1 = current port
other = WindowPtr or GrafPtr

◦ if b = c = d = -1 then...
  uRGB is used as source of color
◦ if b = -1 then...
  c = clut ID or handle
  d = clut index (1-based)
◦ if b = -2 then...
  c = clut ID or handle
  d = clut value (partID)
◦ if b > 0 and c = d = -1 then...
  b = address of RBG color OR old-style color constant:
      33 = black, 30 = white, 205 = red, 341 = green,
      409 = blue, 273 = cyan, 137 = magenta, 69 = yellow
◦ if b ≥ 0, c ≥ 0, d ≥ 0 then...
  b = red component (in low word, 0 to 65535)
  c = green component (in low word, 0 to 65535)
  d = blue component (in low word, 0 to 65535)
where the unusual use of parameter b arose from efforts to maintain backward compatibility with older versions.
  When searching for a "partID" (typically used by control drivers to set control part colors), and the partID cannot be found in the designated clut, then ViewIt will try getting the color from clut 1211 which contains a copy of the standard control colors. This makes it easy for control drivers to set control part colors without having to worry about the color environment in use.
  To help understand the use of b, c, and d, consider the following examples which all specify the color white:
  b = c = d = -1, uRGB.red = green = blue = $FFFF
  b = -1, c = 1211, d = 2 (the 2nd color in clut 1211)
  b = 30, c = d = $FFFFFFFF = -1
  b = c = d = $0000FFFF = 65535
NOTE: *When creating complex pictures or pixmaps with thousands of different colors, it is usually not advisable to use SetFgC or SetBkC to set colors since they are more time-consuming than the corresponding toolbox calls.*

SetPal 185 a,b,c,d,uResult

  Resets the current program-wide color palette or palette characteristics according to the parameters a, b, c, and d. SetPal is ignored if Color QuickDraw is not supported, and uResult returns a negative value if an error occurs.
  a = scope of palette change, or source clut resource
    < -1 = - ID of clut resource or System color table
    -1 = apply to all palette entries
    0 to 255 = palette entry number
    > 255 = clut handle in memory
  b = palette entry usage
    0 = pmCourteous
    2 = pmTolerant
    4 = pmAnimated
    8 = pmExplicit
    10 = pmTolerant + pmExplicit (System ≥ 7 or 32-bit QD)
    12 = pmAnimated + pmExplicit (System ≥ 7 or 32-bit QD)
  c = palette entry tolerance
  d = target palette
    0 = the program-wide palette
    other = a palette handle
where a "clut" is a resource that contains a table of colors that can be created/edited with ResEdit or other resource editor.
  If the scope of changes applies to the entire palette (a < 0 or a > 255), then all visible windows in all screens are erased and invalidated. In cases where your program keeps control after calling SetPal, you

may want to call DoUpdt to force all of the invalidated windows to be redrawn.

The most common use of SetPal is to use it to force screen pixel values to correspond to specific RGB colors so that pixels in pixmaps can be poked directly (a complete discussion of this is presented in the GrafCt documentation). If, for example, clut 1001 contained 256 grays, then the following call would force an 8-bit monitor to display the 256 grays with the index value of each gray from the clut equal to the pixel value (0-255):

```
FaceIt(nil,SetPal,-1001,10,0,0);
```

where b = 10 ensures the correspondence between RGB color and pixel value. To "undo" this palette change without quitting the program, simply reset the palette to the System color table corresponding to the current screen depth:

```
FaceIt(nil,SetPal,-8,2,0,0); force color change
FaceIt(nil,SetPal,-1,0,0,0); minimize redrawing
```

where "-8" refers to the 8-bit deep, 256-color System color table, and SetPal is called a second time to reset the usage of these new colors to "courteous" to minimize window redrawing when switching between programs.

Several other points about SetPal and the program-wide color palette are worth noting:

◦ If FaceIt is in use and SetPal is called while the program is in the background under MultiFinder or System 7, then ViewIt keeps control and delays executing SetPal until the program is brought forward.

◦ If the screen depth supports fewer colors than are in the palette, then only the first colors from the palette will be available. The 16-color System palette (ID = 4), for example, contains the colors white, black, light gray, dark gray, plus 12 other common colors (in that order). This produces reasonable colors at all screen depths:

1 bit/pixel --> white & black displayed
2 bits/pixel --> white, black, light gray, dark gray
4 bits/pixel --> all 16 colors from clut 1000
> 4 bits/pixel --> all 16 colors plus others

• Experienced programmers can obtain a handle to the current program-wide palette by calling "GetPalette" with an argument of -1 (i.e., "GetPalette(WindowPtr(-1))"). This handle can then be used with other Color QuickDraw calls to directly manipulate the contents of the palette.

• Although each window in a program can have its own color palette, we recommend using the single program-wide palette for the following reasons: (1) it minimizes the amount of window updating that occurs when different windows in the same program are brought to the front, (2) it solves the problem of floating windows which would otherwise have to be assigned the palette of the active window, and (3) it greatly simplifies the manipulation and switching of color palettes (you only have one to deal with).