Macintosh Technical Notes

#211: Palette Manager Changes in System 6.0.2

See also: *Inside Macintosh*: The Palette Manager

Written by: Guillermo Ortiz October 1, 1988

This Technical Note describes the changes and enhancements to the Palette Manager in System Software 6.0.2 and future versions.

Application Palette

Applications now have the ability to define a default palette for the system to use when it needs to define the color environment (i.e., when it creates a color window without an associated palette or displays a dialog box).

The application palette feature is especially cool in cases where a color application uses old-style dialogs and alerts because without an application palette, the system will use the default palette to define the color environment. Since the system uses the default palette, the color environment may change (will change in 16-color mode) to cause some “cosmic” colors to appear in the active window. Defining a default application palette with two colors, black and white, solves this problem.

If the system needs a palette to define a color environment, it looks in the resource fork of the application for the `'pltt' ID = 0` resource and uses the palette which it contains. If the system cannot find this resource in the application’s resource fork, it will use its own default palette (resource `'pltt' ID = 0` in the System file) if present, or, if necessary, it will use the Palette Manager’s built-in palette.

Once an application has set its color environment (by calling `_InitMenus`, or `_InitPalettes` in weird instances when there are no menus) it can find the default palette by calling `GetPalette ( WindowPtr (-1) )` or change the default palette by calling `SetPalette ( WindowPtr (-1), newDefPltt, true )`. Note that the initialization of the Palette Manager with a call to `_InitMenus` is contrary to the way *Inside Macintosh*, Volume V, The Palette Manager documents it.

One Palette, Many Ports

You can now associate one palette with many `CGrafPort` and `CWindow` records, thus simplifying the use of a single palette with multiple ports and windows; System Software 6.0 and earlier require copies of the palette to use it with different windows.

Although this ability to associate one palette with multiple ports and windows will allow the use of calls like `_PmForeColor` and `_PmBackColor`, calling `_ActivatePalette` with an off-screen port does nothing, and as a result, calling it with an off-screen port will associate the palette with the port but will not cause any change in the color environment.

One important implication of this feature is that `DisposeWindow` (`_DisposWindow`) will not dispose of the associated palette automatically since it may be allocated to other ports or windows. The only exception to this behavior is when an application has used `_GetNewCWindow` to create the window, there is a `'pltt'` resource with the same ID as the window, and the application has not called `_GetPalette` for the window.

Color Updates

System version 6.0.2 also introduces a new call, `_NSetPalette`, which complements `_SetPalette`. `_NSetPalette` has the same functionality as `_SetPalette`, but the `CUpdates` parameter has been modified from a `Boolean` to an `Integer` as follows:

```
PROCEDURE NSetPalette (dstWindow: WindowPtr; srcPalette: PaletteHandle;
                      nCUpdates: INTEGER);
INLINE $AA95;
```

`_NSetPalette` changes the palette associated with `dstWindow` to `srcPalette`. It also records whether the window wants to receive updates as a result of a change to its color environment. If you want `dstWindow` to be updated whenever its color environment changes, set `nCUpdates` to `pmAllUpdates`. If you are only interested in updates when `dstWindow` is the active window, set `nCUpdates` to `pmFgUpdates`. If you are only interested in updates when `dstWindow` is not the active window, set `nCUpdates` to `pmBkUpdates`.

```
{ NSetPalette Update Constants }

pmNoUpdates =           $8000;              {no updates}
pmBkUpdates =           $A000;              {background updates only}
pmFgUpdates =           $C000;              {foreground updates only}
pmAllUpdates =          $E000;              {all updates}
```

`_SetPalette` retains its syntax and function:

```
PROCEDURE SetPalette (dstWindow: WindowPtr; srcPalette: PaletteHandle;
                      CUpdates: Boolean);
INLINE $AA95;
```

Note: The trap words for `_NSetPalette` and `_SetPalette` are identical.

CopyPalette

```
PROCEDURE CopyPalette (srcPalette, dstPalette: PaletteHandle;
                       srcEntry,dstEntry,dstLength: INTEGER);
INLINE $AAA1;
```

`_CopyPalette` is a utility procedure that copies `dstLength` entries from the source palette into the destination palette; the copy begins at `srcEntry` and `dstEntry`, respectively. `_CopyPalette` will resize the destination palette when the number of entries after the copy is greater than the original.

`_CopyPalette` does not call `_ActivatePalette`, so the application is free to do a number of palette changes without causing a series of intermediate changes to the color environment; the application should call `_ActivatePalette` after completing all palette changes.

If either of the palette handles are `NIL`, `_CopyPalette` does nothing.