

Re: Owner Draw STATIC Window with XP Themes

Source: <http://www.tech-archive.net/Archive/VC/microsoft.public.vc.mfc/2004-11/0832.html>

From: Jeff Partch [MVP] (jeffp_at_mvps.org)

Date: 11/11/04

Date: Thu, 11 Nov 2004 06:46:30 -0600

"Doug Harrison [MVP]" <dsh@mvps.org> wrote in message
news:lop5p05fiqtce38g174dfg1g642gtqbp9h@4ax.com...

> *Jonathan Wood wrote:*

>

> > *Well, IsAppThemed will return TRUE even though a manifest resource is not*

> > *present so I questioned its reliability. But I guess you know if you are*

> > *including a manifest resource or not. Hopefully, it is reliable given*

> *that*

> > *the manifest is included.*

>

> *IME, it is. You can test by turning themes off on a system-wide basis and*

> *also just for your specific application through the .exe's properties.*

Don't get too annoyed by my jumping in here, but I think you are both right.
My recollected understanding of IsThemeActive and IsAppThemed is:

- 1) they will both return TRUE when themes are active system wide
- 2) except that IsAppThemed will return FALSE only in the case where themes are disabled in the compatibility properties for the application.

Neither concerns itself with whether this particular app has a manifest and is supposed to use themes and neither is particularly determinate as to whether this particular control is supposed to use themes --- only that it **could** --- not not that it **should**. That leaves the similarly flawed GetThemeAppProperties, and GetWindowTheme which I discuss below.

Another hack I use for the standard windows controls and which I've posted before is to test whether the GetClassLongPtr(hWnd, GCLP_HMODULE) is equal to GetModuleHandle(_T("Comctl32.dll")), and which I think does consider the "has a manifest" issue, but probably not the SetThemeAppProperties(~STAP_ALLOW_CONTROLS) and SetWindowTheme(L"", L"") cases. And IIRC, there are folks doing things with the activation context to disable theming after the fact even in an otherwise themed app with a manifest.

> *Here's the function which helps maintain the theme state, called during*
> *PreSubclassWindow and in response to WM_THEMECHANGED:*

```
>  
> void  
> OdButton::UpdateTheme()  
> {  
> if (OsVersion::AtLeastXP())  
> {  
> CloseThemeData(m_hTheme);  
> m_hTheme = (UserEx::IsAppThemed())  
> ? OpenThemeData(m_hWnd, L"button")  
> : 0;  
> }  
> }
```

FYI, According to Dave Anderson, Microsoft Developer Support, you can't really do this, Doug...

```
"  
> You should open a theme once per window... open it when the window is  
> created and close the handle when the window is destroyed. There is a  
> limit  
> of 64K handles per process. The bug is when this limit is reached,  
> OpenThemeData does not return NULL.  
"
```

Even allowing an exception for WM_THEMECHANGED, you're now doing it twice per window. :)

Presumably a subclass should use `GetWindowTheme` to obtain, "the most recent theme handle from `OpenThemeData`". But...

Back in July, I reported what I thought was a bug regarding `GetWindowTheme`...

"So here's the situation: I'm working on this `NM_CUSTOMDRAW` handler for a themed pushbutton on XPSP1. I'm using VC7.1 and a MFC7 dialog-based application (although I don't believe MFC is the culprit). Now, in my `NM_CUSTOMDRAW` handler, I have occasion to call `GetWindowTheme`..., but every time the button gains and loses the focus the call returns `NULL`".

So if the question the control is asking is, "as an independent component of any application running on any OS under any externally imposed or inherited variable conditions, am I supposed to be using themes right now"? my subjective conclusion is that there is no *one* API that answers this question, and even in consort all of them together may not answer the question with absolute certainty. At best, you end up with an increasingly more educated guess. FWIW.

:)

--

microsoft.public.vc.mfc: Re: Owner Draw STATIC Window with XP Themes

Jeff Partch [VC++ MVP]