What is the proper handling of WM_RENDERFORMAT and WM_RENDERALLFORMATS?

devblogs.microsoft.com/oldnewthing/20121224-00

December 24, 2012

Raymond Chen

Jeremy points out that the documentation for `SetClipboardData` says that the clipboard owner must not call `OpenClipboard` when responding to the `WM_RENDERFORMAT` and `WM_RENDERALLFORMATS` messages. On the other hand, the documentation for `WM_RENDER-ALLFORMATS` says that the owner must call `OpenClipboard` and `EmptyClipboard`. Which is it?

It's none of them!

Let's start with `WM_RENDERFORMAT`. The reference implementation for a `WM_RENDERFORMAT` handler goes like this, with all error handling deleted for expository purposes:

```
case WM_RENDERFORMAT:
 CLIPFORMAT cf = (CLIPFORMAT)wParam;
 hData = GenerateFormat(cf);
 SetClipboardData(cf, hData);
 return 0;
```

In response to `WM_RENDERFORMAT`, you simply place the format on the clipboard. No opening is required. In fact, attempting to open will *fail* because the clipboard is already open: It has been opened by the application whose call to `GetClipboardData` triggered the delay-render!

Next comes `WM_RENDERALLFORMATS`. The original reference implementation goes like this, again with error checking deleted:

```
// code in italics is wrong -- see discussion below
case WM_RENDERALLFORMATS:
 OpenClipboard(hwnd);
 SendMessage(hwnd, WM_RENDERFORMAT, CF_FORMAT1, 0);
 SendMessage(hwnd, WM_RENDERFORMAT, CF_FORMAT2, 0);
 CloseClipboard();
 return 0;
```

In response to `WM_RENDERALLFORMATS`, you open the clipboard, then render all your formats into it, and then close the clipboard. And one to render your formats is simply to send yourself a fake `WM_RENDERFORMAT` message, which gets the code in the earlier code block to generate the format and place it on the clipboard.

So you see that everybody is wrong!

The `WM_RENDERALLFORMATS` handler *does* call `OpenClipboard`—if you tried it without the `OpenClipboard` call, you'd notice that the data never made it to the clipboard—and it *doesn't* call `EmptyClipboard`. (If you did, you'd notice that the `EmptyClipboard` would have wiped out your non-delay-rendered data!)

Where did I get these reference implementations from? I got them from the *Windows 3.1 SDK*. (And that explains the bug; read on.)

In real life, you probably would also listen for the `WM_DESTROYCLIPBOARD` message so you would know that you are no longer the clipboard owner, in which case you wouldn't bother rendering anything.

I haven't written code in a while, so let's write some code. Start with our scratch program and make these changes. We'll start by writing it incorrectly:

```
BOOL
OnCreate(HWND hwnd, LPCREATESTRUCT lpcs)
{
 if (OpenClipboard(hwnd)) {
  EmptyClipboard();
  SetClipboardData(CF_UNICODETEXT, NULL);
  CloseClipboard();
 }
 return TRUE;
}

const WCHAR c_szText[] = L"hello";
HANDLE
OnRenderFormat(HWND hwnd, UINT fmt)
{
 if (fmt == CF_UNICODETEXT)
 {
  HGLOBAL hglob;
  if (SUCCEEDED(CreateHGlobalFromBlob(
                            c_szText, sizeof(c_szText),
                            GMEM_MOVEABLE, &hglob))) {
   if (!SetClipboardData(fmt, hglob)) GlobalFree(hglob);
  }
 }
 return 0;
}
void
OnRenderAllFormats(HWND hwnd)
{
 if (OpenClipboard(hwnd)) {
  OnRenderFormat(hwnd, CF_UNICODETEXT);
  CloseClipboard();
 }
}
    HANDLE_MSG(hwnd, WM_RENDERFORMAT, OnRenderFormat);
    HANDLE_MSG(hwnd, WM_RENDERALLFORMATS, OnRenderAllFormats);
```

This program puts delay-rendered text on the clipboard when it starts up, When the request for text arrives, we just return the word `hello`. If we are asked to render all our formats, we render all our formats by calling our internal function once for each format we support. (All one of them.)

There's a tiny race condition in that implementation above, though. What if somebody takes ownership of the clipboard *while you're trying to render all your formats*? Let's force the race condition. Set a breakpoint on the `OnRenderAllFormats` function, run the program, and close the window. The breakpoint will hit.

Switch away from the debugger and open Notepad. Type `123` into Notepad, then select it and type `Ctrl`+`C` to copy it to the clipboard.

Notepad will hang for a while, since the window manager is trying to send a `WM_DESTROY-CLIPBOARD` message to tell the previous clipboard owner that it is no longer responsible for the data on the clipboard. Let the call time out, at which point Notepad will wake back up and put `123` text on the clipboard. Now resume execution of the scratch program, so that it puts the Unicode word `hello` onto the clipboard.

Okay, go back to Notepad and hit `Ctrl`+`V`. Look, it pasted `hello` instead of `123`. Oops, our delay-rendering program destroyed the clipboard as it exited. If the application had put something more complicated on the clipboard, then our scratch program would have created a mishmash of old and new data.

To protect against this race condition, make the following small change:

```
void
OnRenderAllFormats(HWND hwnd)
{
 if (OpenClipboard(hwnd)) {
  if (GetClipboardOwner() == hwnd) {
    OnRenderFormat(hwnd, CF_UNICODETEXT);
  }
  CloseClipboard();
 }
}
```

After opening the clipboard, we check if we are still the window responsible for the clipboard contents. Only if so do we render our delay-rendered formats.

Exercise: Why is the `GetClipboardOwner` test done *after* the `OpenClipboard`? Wouldn't it be better to bail out quickly if we are not the clipboard owner and avoid opening the clipboard in the first place?

Raymond Chen

Follow