Of what use is the RDW_INTERNALPAINT flag?

devblogs.microsoft.com/oldnewthing/20130621-00

June 21, 2013

Raymond Chen

For motivational purposes, let's start with a program that displays a DWM thumbnail.

Start with the scratch program and add the following:

```
#include <dwmapi.h>


HWND g_hwndThumbnail;
HTHUMBNAIL g_hthumb;


void UpdateThumbnail(HWND hwndFrame, HWND hwndTarget)
{
 if (g_hwndThumbnail != hwndTarget) {
  g_hwndThumbnail = hwndTarget;
  if (g_hthumb != nullptr) {
   DwmUnregisterThumbnail(g_hthumb);
   g_hthumb = nullptr;
  }


  if (hwndTarget != nullptr) {
   RECT rcClient;
   GetClientRect(hwndFrame, &rcClient);
   if (SUCCEEDED(DwmRegisterThumbnail(hwndFrame,
                         g_hwndThumbnail, &g_hthumb))) {
    DWM_THUMBNAIL_PROPERTIES props = {};
    props.dwFlags = DWM_TNP_RECTDESTINATION | DWM_TNP_VISIBLE;
    props.rcDestination = rcClient;
    props.rcDestination.top += 50;
    props.fVisible = TRUE;
    DwmUpdateThumbnailProperties(g_hthumb, &props);
   }
  }
 }
}
```

The `UpdateThumbnail` function positions a thumbnail of the target window inside the frame window. There’s a small optimization in the case that the target window is the same one that the thumbnail is already viewing. Overall, no big deal.

```
void
OnDestroy(HWND hwnd)
{
 UpdateThumbnail(hwnd, nullptr);
 PostQuitMessage(0);
}
```

When our window is destroyed, we need to clean up the thumbnail, which we do by updating it to a null pointer.

For the purpose of illustration, let’s say that pressing the `1` key changes the thumbnail to a randomly-selected window.

```
struct RANDOMWINDOWINFO
{
 HWND hwnd;
 int cWindows;
};


BOOL CALLBACK RandomEnumProc(HWND hwnd, LPARAM lParam)
{
 if (hwnd != g_hwndThumbnail &&
     IsWindowVisible(hwnd) &&
     (GetWindowStyle(hwnd) & WS_CAPTION) == WS_CAPTION) {
  auto prwi = reinterpret_cast<RANDOMWINDOWINFO *>(lParam);
  prwi->cWindows++;
  if (rand() % prwi->cWindows == 0) {
   prwi->hwnd = hwnd;
  }
 }
 return TRUE;
}


void ChooseRandomWindow(HWND hwndFrame)
{
 RANDOMWINDOWINFO rwi = {};
 EnumWindows(RandomEnumProc, reinterpret_cast<LPARAM>(&rwi));
 UpdateThumbnail(hwndFrame, rwi.hwnd);
}


void OnChar(HWND hwnd, TCHAR ch, int cRepeat)
{
 switch (ch) {
 case TEXT(‘1’):
  ChooseRandomWindow(hwnd);
  break;
 }
}


 HANDLE_MESSAGE(hwnd, WM_CHAR, OnChar);
```

The random window selector selects among windows with a caption which are visible and which are not already being shown in the thumbnail. (That last bit is so that when you press `1`, it will always pick a *different* window.)

Run this program, and yippee, whenever you press the `1` key, you get a new thumbnail.

Okay, but usually your program shows more than just a thumbnail. It probably incorporates the thumbnail into its other content, so let’s draw some other content, too. Say, a single-character random message.

```
TCHAR g_chMessage = ‘?’;


void
PaintContent(HWND hwnd, PAINTSTRUCT *pps)
{
 if (!IsRectEmpty(&pps->rcPaint)) {
  RECT rcClient;
  GetClientRect(hwnd, &rcClient);
  DrawText(pps->hdc, &g_chMessage, 1, &rcClient,
           DT_TOP | DT_CENTER);
 }
}


void ChooseRandomMessage(HWND hwndFrame)
{
 g_chMessage = rand() % 26 + TEXT(‘A’);
 InvalidateRect(hwndFrame, nullptr, TRUE);
}


void OnChar(HWND hwnd, TCHAR ch, int cRepeat)
{
 switch (ch) {
 case TEXT(‘1’):
  ChooseRandomWindow(hwnd);
  break;
 case TEXT(‘2’):
  ChooseRandomMessage(hwnd);
  break;
 }
}
```

Now, if you press 2, we change the random message. There is a small optimiztion in `PaintContent` that skips the rendering if the paint rectangle is empty. Again, no big deal.

Okay, but sometimes there are times where your program wants to update the thumbnail *and* the message at the same time. Like this:

```
void OnChar(HWND hwnd, TCHAR ch, int cRepeat)
{
 switch (ch) {
 case TEXT('1'):
  ChooseRandomWindow(hwnd);
  break;
 case TEXT('2'):
  ChooseRandomMessage(hwnd);
  break;
 case TEXT('3'):
  ChooseRandomWindow(hwnd);
  ChooseRandomMessage(hwnd);
  break;
 }
}
```

Run this program and press `3` and watch the thumbnail and message change simultaneously.

And now we have a problem.

You see, the `ChooseRandomWindow` function updates the thumbnail immediately, since the thumbnail is presented by DWM, whereas the `ChooseRandomMessage` function updates the message, but the new message doesn't appear on the screen until the next paint cycle. This means that there is a window of time where the new thumbnail is on the screen, but you still have the old message. Since painting is a low-priority activity, the window manager is going to deliver other messages to your window before it finally gets around to painting, and the visual mismatch between the thumbnail and the message can last for quite some time. (You can exaggerate this in the sample program by inserting a call to `Sleep`.) What can we do to get rid of this visual glitch?

One solution would be to delay updating the thumbnail until the next paint cycle. At the paint cycle, we update the thumbnail *and* render the new message. Now both updates occur at the same time, and you get rid of the glitch. To trigger a paint cycle, we can invalidate a dummy 1×1 pixel in the window.

```
HWND g_hwndThumbnailWanted;


void
PaintContent(HWND hwnd, PAINTSTRUCT *pps)
{
 UpdateThumbnail(hwnd, g_hwndThumbnailWanted);


 if (!IsRectEmpty(&pps->rcPaint)) {
  RECT rcClient;
  GetClientRect(hwnd, &rcClient);
  DrawText(pps->hdc, &g_chMessage, 1, &rcClient,
           DT_TOP | DT_CENTER);
 }
}


void ChooseRandomWindow(HWND hwndFrame)
{
 RANDOMWINDOWINFO rwi = {};
 EnumWindows(RandomEnumProc, reinterpret_cast(&rwi));
 g_hwndThumbnailWanted = rwi.hwnd;
 RECT rcDummy = { 0, 0, 1, 1 };
 InvalidateRect(hwndFrame, &rcDummy, FALSE);
}
```

Now, when we want to change the thumbnail, we just remember what thumbnail we want (the "logical" current thumbnail) and invalidate a dummy pixel in our window. The invalid dummy pixel triggers a paint cycle, and in our paint cycle, we call `UpdateThumbnail` to synchronize the logical current thumbnail with the physical current thumbnail. And then we continue with our regular painting (in case there is also painting to be done, too).

But it sure feels wasteful invalidating a pixel and forcing the `DrawText` to occur even though we really didn't update anything. Wouldn't it be great if we could just say, "Could you fire up a paint cycle for me, even though there's technically nothing to paint? Because I actually do have stuff to paint, it's just something outside your knowledge since it is not rendered by GDI."

Enter the `RDW_INTERNALPAINT` flag.

If you pass the `RDW_INTERNALPAINT` flag to `RedrawWindow`, that means, "Set the 'Yo, there's painting to be done!' flag. I know you think there's no actual painting to be done, but trust me on this." (It's not actually a flag, but you can think of it that way.)

When the window manager then get around to deciding whether there is any painting to be done, before it concludes, "Nope, this window is all valid," it checks if you made a special `RDW_INTERNALPAINT` request, and if so, then it will generate a dummy `WM_PAINT` message for you.

Using this new flag is simple:

```
g_hwndThumbnailWanted = rwi.hwnd;
// ~~RECT rcDummy = { 0, 0, 1, 1 };~~
// ~~InvalidateRect(hwndFrame, &rcDummy, FALSE);~~
RedrawWindow(hwndFrame, nullptr, nullptr,
             RDW_INTERNALPAINT);
```

Now, when the program wants to update its thumbnail, it just schedules a fake-paint message with the window manager. These fake-paint messages coalesce with real-paint messages, so if you do an internal paint and an invalidation, only one actual paint message will be generated. If the paint message is a fake-paint message, the `rcPaint` will be empty, and you can test for that in your paint handler and skip your GDI painting.

Raymond Chen

Follow