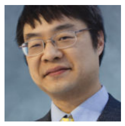
How can I detect that my window is on the current virtual desktop?

devblogs.microsoft.com/oldnewthing/20171002-00

October 2, 2017



Raymond Chen

Virtual desktops are a feature added in Windows 10 blah blah exposition.

Here is how virtual desktops work, from a programmatic standpoint:

- To switch to a virtual desktop, the system shows the windows that belong to the virtual desktop and hides the windows that do not belong to the virtual desktop. Note that the windows still all belong to the same desktop (hence the “virtual”). All we’re doing is hiding and showing windows.
- When a new window is shown, it gets placed on the current virtual desktop.
- When a window becomes foreground, the system switches to the virtual desktop that the window belongs to.

That said, there are some guidelines that programs should follow.

- Do not programatically change the current virtual desktop. The user should be the one to change virtual desktops, if that’s what they want.
- If your program decides to open a new window, then open a new window. It will be placed on the current virtual desktop.
- If your program decides to reuse an existing window (for example, if you have a tabbed user interface, and you want to open the document in a new tab), then when looking for a window to reuse, limit your search to the current virtual desktop. If you cannot find a window from the current virtual desktop, then create and show a new one, which will be placed on the current virtual desktop.
- Exception: If your program opens each document in a new window, and the user opens a document that you already have a window for, then you are allowed to switch to the virtual desktop that contains the already-open document.

Let’s start with the scratch program and make these changes.

```
#include <shlobj.h>

BOOL
OnCreate(HWND hwnd, LPCREATESTRUCT lpcs)
{
    g_hwndChild = CreateWindow("listbox", nullptr,
       WS_VISIBLE | WS_CHILD, 0, 0, 0, 0, hwnd,
       (HMENU)IntToPtr(1), g_hinst, 0);

    return TRUE;
}

void
ProcessCommandLine(LPCSTR pszMessage)
{
    ListBox_AddString(g_hwndChild, pszMessage);
}

void
OnCopyData(HWND hwnd, HWND hwndFrom, PCOPYDATASTRUCT pcds)
{
    if (pcds->dwData == 0)
    {
      // WARNING! Parameter validation is missing!
      ProcessCommandLine(reinterpret_cast<PSTR>(pcds->lpData));
    }
}

    // Add to WndProc
    HANDLE_MSG(hwnd, WM_COPYDATA, OnCopyData);

BOOL
WindowCanBeReused(HWND hwnd)
{
    // A more realistic program would have some evaluation criteria.
    return TRUE;
}

BOOL
TryHandOffToExistingInstance(LPCSTR pszMessage)
{
    HWND hwndFound = nullptr;
    while ((hwndFound = FindWindowEx(nullptr, hwndFound,
                        "Scratch", nullptr)) != nullptr) {
      if (WindowCanBeReused(hwndFound)) {
        SetForegroundWindow(hwndFound);
        COPYDATASTRUCT cds;
        cds.dwData = 0;
        cds.cbData = lstrlen(pszMessage) + 1;
        cds.lpData = const_cast<PSTR>(pszMessage);
        FORWARD_WM_COPYDATA(hwndFound, nullptr, &cds, SendMessage);
        return TRUE;
      }
    }
    return FALSE;
}


int WINAPI WinMain(HINSTANCE hinst, HINSTANCE hinstPrev,
                   LPSTR lpCmdLine, int nShowCmd)
{
    MSG msg;
    HWND hwnd;

    g_hinst = hinst;

    if (!InitApp()) return 0;

    if (SUCCEEDED(CoInitialize(NULL))) {/* In case we use COM */

      if (!lpCmdLine[0]) {
        lpCmdLine = const_cast<PSTR>("(empty command line)");
      }

      if (!TryHandOffToExistingInstance(lpCmdLine)) {

        hwnd = CreateWindow(
            TEXT("Scratch"),                /* Class Name */
            TEXT("Scratch"),                /* Title */
            WS_OVERLAPPEDWINDOW,            /* Style */
            CW_USEDEFAULT, CW_USEDEFAULT,   /* Position */
            CW_USEDEFAULT, CW_USEDEFAULT,   /* Size */
            NULL,                           /* Parent */
            NULL,                           /* No menu */
            hinst,                          /* Instance */
            0);                             /* No special parameters */

        ShowWindow(hwnd, nShowCmd);

        while (GetMessage(&msg, NULL, 0, 0)) {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
      }

      CoUninitialize();
    }

    return 0;
}
```

This is our non-virtual-desktop-aware version of the program. When it is run, it looks for an existing instance that can be reused, and if it finds one, it asks that existing instance to handle the command line.

Now let’s make this program virtual-desktop-aware.

```
IVirtualDesktopManager* g_pvdm;

BOOL
WindowCanBeReused(HWND hwnd)
{
    BOOL isCurrent;
    if (g_pvdm &&
        SUCCEEDED(g_pvdm->IsWindowOnCurrentVirtualDesktop(hwnd,
                                          &isCurrent)) &&
        !isCurrent) {
      return FALSE;
    }

    // A more realistic program would have some evaluation criteria.
    return TRUE;
}

int WINAPI WinMain(HINSTANCE hinst, HINSTANCE hinstPrev,
                   LPSTR lpCmdLine, int nShowCmd)
{
    ...

    if (SUCCEEDED(CoInitialize(NULL))) {/* In case we use COM */

      // This can fail if the system does not support virtual desktops.
      CoCreateInstance(CLSID_VirtualDesktopManager,
                       nullptr, CLSCTX_ALL, IID_PPV_ARGS(&g_pvdm));

      if (!lpCmdLine[0]) {
        lpCmdLine = const_cast<PSTR>("(empty command line)");
      }

      if (!TryHandOffToExistingInstance(lpCmdLine)) {
         ...
      }

      if (g_pvdm) g_pvdm->Release();

      CoUninitialize();
    }

    return 0;
}
```

We updated the `WindowCanBeReused` function so it takes the virtual desktop state into account. Specifically, we will not attempt to reuse windows that are not part of the current virtual desktop.

Raymond Chen

Follow