How do I customize how my application windows are grouped in the Taskbar?

devblogs.microsoft.com/oldnewthing/20120820-00

August 20, 2012

Raymond Chen

Benjamin Smedberg wants to know how to customize the icon used in the Taskbar for applications that are grouped, when the application is a runtime for multiple applications. (This is the other scenario I hinted at last time.)

Actually, customizing the icon is only part of what you want to happen when your application is a runtime. In that case, you really want each inner application to be exposed to the user as an entirely separate application. In other words, if your application is hosting Product A and Product B, you want the windows for Product A and Product B to group separately, have separate icons, maintain separate jump lists, all that stuff. Because from the user's point of view, they are separate programs. It just happens that under the covers, they're all being driven by a single EXE.

In Windows, the concept of an application is captured in something called an *Application User Model ID*, or *AppID* for short. For backward compatibility, if your application does not provide an explicit AppID, the shell will autogenerate one based on your EXE name. Therefore, the starting point for AppIDs is that an AppID maps to an EXE. But once you start customizing your AppID, you can play with this default correspondence.

All the information in this article came from the article Application User Model IDs (AppUserModelIDs) in MSDN.

Okay, so suppose your application is really a runtime for other applications. What you need to do is assign a different AppID to each of the applications you are hosting. The mechanism for this is up to you. Your applications might explicitly provide a unique ID, or you may be able to infer one. For example, if you are Internet Explorer and your "applications" are pinned Web sites, you can use the URL of the site being pinned as the unique ID.

You then get to take your unique IDs and create AppIDs for them. The format of an AppID is

```
CompanyName.ProductName.SubProduct.VersionInformation
```

where the SubProduct is optional, and the VersionInformation is present only if you want different versions of your app to be treated as distinct. (If you want an upgraded version to be a replacement for the old version, then omit the VersionInformation so that the old and new versions use the same AppID.)

Note that you have to be careful how you auto-generate your AppIDs, since the resulting AppID needs to be legal. For example, you cannot just take a URL and use it as the Sub-Product of an AppID. URLs contain embedded periods, which violates the overall format, and they can be longer than 128 characters and can contain spaces, both of which are also called out in the documentation as prohibited. Internet Explorer addresses this problem by using a hash of the URL as its SubProduct rather than the full URL.

You then assign this AppID to every window associated with the "application". You can do this for an entire process by calling `SetCurrentProcessExplicitAppUserModelID`, or you can do it on a window-by-window basis by setting the `PKEY_AppUserModel_ID` property.

Okay, let's write a program that shows how a runtime for other applications can use AppIDs to control its treatment in the taskbar. Of course, our sample won't actually be a runtime for anything; the "applications" that it hosts will simply be icons.

Start with the scratch program and make these changes:

```
#include <shellapi.h>
#include <shlobj.h>
#include <strsafe.h>
#define HOSTAPPID L"Contoso.Host"
void SetProcessAppId(LPCWSTR pszTarget)
{
  if (pszTarget[0]) {
    WCHAR szAppId[256];
    DWORD dwHash = 0;
    HashData((BYTE*)pszTarget, wcslen(pszTarget) * sizeof(WCHAR),
             (BYTE*)&dwHash, sizeof(dwHash));
    StringCchPrintfW(szAppId, ARRAYSIZE(szAppId),
                     L"%s.hosted-%08x", HOSTAPPID, dwHash);
    SetCurrentProcessExplicitAppUserModelID(szAppId);
  } else {
    StringCchPrintfW(szAppId, ARRAYSIZE(szAppId),
                     L"%s.main", HOSTAPPID);
  }
}
int WINAPI wWinMain(HINSTANCE hinst, HINSTANCE hinstPrev,
                    LPWSTR lpCmdLine, int nShowCmd)
{
  SetProcessAppId(lpCmdLine);
    ...
    ShowWindow(hwnd, SW_NORMAL);
    SetWindowText(hwnd, lpCmdLine);
    if (lpCmdLine[0]) {
      WCHAR szIcon[256];
      StringCchCopyW(szIcon, ARRAYSIZE(szIcon), ptszCmdLine);
      int iIcon = PathParseIconLocation(szIcon);
      if (iIcon == -1) iIcon = 0;
      HICON hico = ExtractIcon(hinst, szIcon, iIcon);
      SendMessage(hwnd, WM_SETICON, ICON_BIG, (LPARAM)hico);
    }
    ...
}
```

Our simple host program just hosts an icon. The path to the icon is passed on the command line in the form “path,id”, and for good measure, we put the icon path in the caption so you can see how it groups.

The real work happens in the `SetProcessAppId` function. If there is no command line, then we are running in standalone mode and set our SubProduct to `main`. If we have a command line, then we hash it and use the hash to build our SubProduct. I’m just using a four-byte hash with a simple has function; depending on how paranoid you are, you could use some other hash function, but make sure you can get the resulting AppID to fit into 128 characters. (This means that hex-encoded SHA512 is too big.)

Once we figure out what our AppID is, we set it for the entire process by calling `SetCurrentProcessExplicitAppUserModelID`.

Okay, let's take this program out for a spin. You can run it with the command lines

```
scratch %windir%\explorer.exe,0
scratch %windir%\explorer.exe,0
scratch %windir%\explorer.exe,1
scratch %windir%\explorer.exe,1
```

to see four copies of the program, two with one icon and two with another. Observe that when they group in the taskbar, the icon for the group is preserved, and that the two sets of programs group separately.

Note also that if you create shortcuts to your host program with a command line, you need to set the AppID in your shortcut, too. (Otherwise the shell won't know what the AppID of the resulting program will be, since you are setting it at runtime.)

Note also that we did not need to register the application as a host process because we explicitly set an AppID in our application and in our shortcuts. (Or at least, we said that we would. I didn't actually do it.)

Bonus reading: Developing for the Windows 7 Taskbar — Application ID.

Raymond Chen

Follow