Manipulating the positions of desktop icons

devblogs.microsoft.com/oldnewthing/20130318-00

March 18, 2013

Raymond Chen

Today's little program demonstrates how you can manipulate the positions of desktop icons.

The entire program is just scaffolding to get us far enough that we can call `IFolderView::GetItemPosition` and `IFolderView::SelectAndPositionItems`.

First, we adapt the code we saw some time ago that extracts the `IFolderView` from a window.

Reminder: These "Little Programs" do no error checking because they are intended as demonstrations, not production-ready applications.

```
void FindDesktopFolderView(REFIID riid, void **ppv)
{
 CComPtr<IShellWindows> spShellWindows;
 spShellWindows.CoCreateInstance(CLSID_ShellWindows);

 CComVariant vtLoc(CSIDL_DESKTOP);
 CComVariant vtEmpty;
 long lhwnd;
 CComPtr<IDispatch> spdisp;
 spShellWindows->FindWindowSW(
     &vtLoc, &vtEmpty,
     SWC_DESKTOP, &lhwnd, SWFO_NEEDDISPATCH, &spdisp);

 CComPtr<IShellBrowser> spBrowser;
 CComQIPtr<IServiceProvider>(spdisp)->
     QueryService(SID_STopLevelBrowser,
                  IID_PPV_ARGS(&spBrowser));

 CComPtr<IShellView> spView;
 spBrowser->QueryActiveShellView(&spView);

 spView->QueryInterface(riid, ppv);
}
```

The `FindDesktopFolderView` function takes the code from that earlier article and uses it to extract the shell view for the desktop. Everything here should look familiar (just in a different costume), aside from the call to `FindWindowSW`, because we are looking for a specific window by location rather than just enumerating through all of them.

The first parameter to `FindWindowSW`. is the folder we are looking for. In our case, we are looking for the desktop.

The second parameter is reserved and must be `VT_EMPTY`.

The third parameter describes the types of windows we are looking for. We use the special `SWC_DESKTOP` flag (available starting in Windows Vista) to say, “Hey, I know the desktop isn’t the sort of thing people think of when they go looking for Explorer windows, but I know what I’m talking about, so let me have it.”

The fourth parameter receives the window handle, which is of no interest to us, but the parameter is mandatory, so we have to give it something.

The fifth parameter specifies the search options. We use `SWFO_NEEDDISPATCH` to say, “Please return the `IDispatch` in the sixth parameter.” And the sixth parameter is where we want the `IDispatch` to be returned.

Okay, we already have enough to be able to enumerate all the desktop icons and print their names and locations.

```
#define UNICODE
#define _UNICODE
#include <windows.h>
#include <shlobj.h>
#include <exdisp.h>
#include <shlwapi.h>
#include <atlbase.h>
#include <atlalloc.h>
#include <stdio.h>

// CCoInitialize incorporated by reference

int __cdecl wmain(int argc, wchar_t **argv)
{
 CCoInitialize init;
 CComPtr<IFolderView> spView;
 FindDesktopFolderView(IID_PPV_ARGS(&spView));
 CComPtr<IShellFolder> spFolder;
 spView->GetFolder(IID_PPV_ARGS(&spFolder));

 CComPtr<IEnumIDList> spEnum;
 spView->Items(SVGIO_ALLVIEW, IID_PPV_ARGS(&spEnum));
 for (CComHeapPtr<ITEMID_CHILD> spidl;
      spEnum->Next(1, &spidl, nullptr) == S_OK;
      spidl.Free()) {
  STRRET str;
  spFolder->GetDisplayNameOf(spidl, SHGDN_NORMAL, &str);
  CComHeapPtr<wchar_t> spszName;
  StrRetToStr(&str, spidl, &spszName);

  POINT pt;
  spView->GetItemPosition(spidl, &pt);

  wprintf(L"At %4d,%4d is %ls\n", pt.x, pt.y, spszName);
 }
 return 0;
}
```

After getting the `IFolderView`, we also ask for the corresponding `IShellFolder`. This isn't actually necessary for enumerating the icons, but it lets us print their names.

We ask the view for its `Items` enumeration, then proceed to enumerate each of the items. For each item, we ask the `IShellFolder` for its name, and we ask the `IFolderView` for its position. Then we print the results.

Okay, that was neat, but you can do more than just query the positions. You can also modify them.

```
int __cdecl wmain(int argc, wchar_t **argv)
{
 CCoInitialize init;
 CComPtr<IFolderView> spView;
 FindDesktopFolderView(IID_PPV_ARGS(&spView));

 CComPtr<IEnumIDList> spEnum;
 spView->Items(SVGIO_ALLVIEW, IID_PPV_ARGS(&spEnum));
 for (CComHeapPtr<ITEMID_CHILD> spidl;
      spEnum->Next(1, &spidl, nullptr) == S_OK;
      spidl.Free()) {
  POINT pt;
  spView->GetItemPosition(spidl, &pt);
  pt.x += (rand() % 5) - 2;
  pt.y += (rand() % 5) - 2;

 PCITEMID_CHILD apidl[1] = { spidl };
 spView->SelectAndPositionItems(
     1, apidl, &pt, SVSI_POSITIONITEM);
 }
 return 0;
}
```

This time, instead of printing the item's name and position, we jiggle the icon position by a few pixels randomly, then set the jiggled coordinates as the new position.

Turn off *Auto arrange icons* and *Align icons to grid* on the desktop, and then run this program. Hey, look, your icons shifted randomly by a few pixels.

For extra hijinx, drop a call to `spView->SetCurrentFolderFlags(FWF_AUTOARRANGE | FWF_SNAPTOGRID, 0)` before you enter the loop (to programmatically turn off auto-arrange and snap-to-grid), then put this program in a loop, and slip it onto a friend's (or enemy's) computer.

More seriously, we can we put the two pieces together to make a program that saves and restores desktop icon positions.

Second reminder: These "Little Programs" do no error checking because they are intended as demonstrations, not production-ready applications.

```
void SavePositions(IFolderView *pView, PCWSTR pszFile)
{
 CComPtr<IStream> spStream;
 SHCreateStreamOnFileEx(pszFile, STGM_CREATE | STGM_WRITE,
     FILE_ATTRIBUTE_NORMAL, TRUE, nullptr, &spStream);
 CComPtr<IEnumIDList> spEnum;
 pView->Items(SVGIO_ALLVIEW, IID_PPV_ARGS(&spEnum));
 for (CComHeapPtr<ITEMID_CHILD> spidl;
      spEnum->Next(1, &spidl, nullptr) == S_OK;
      spidl.Free()) {
  IStream_WritePidl(spStream, spidl);
  POINT pt;
  pView->GetItemPosition(spidl, &pt);
  IStream_Write(spStream, &pt, sizeof(pt));
 }
}
```

The `SavePositions` function enumerates all the icons in a view and writes their identities and positions to a file.

```
void RestorePositions(IFolderView *pView, PCWSTR pszFile)
{
 CComPtr<IStream> spStream;
 SHCreateStreamOnFileEx(pszFile, STGM_READ,
     FILE_ATTRIBUTE_NORMAL, FALSE, nullptr, &spStream);
 POINT pt;
 for (CComHeapPtr<ITEMID_CHILD> spidl;
      SUCCEEDED(IStream_ReadPidl(spStream, &spidl)) &&
      SUCCEEDED(IStream_Read(spStream, &pt, sizeof(pt)));
      spidl.Free()) {
  PCITEMID_CHILD apidl[1] = { spidl };
  pView->SelectAndPositionItems(1, apidl, &pt, SVSI_POSITIONITEM);
 }
}
```

The `RestorePositions` function does the reverse. It reads the identities and positions from the file and calls `IFolderView::SelectAndPositionItems` to move the item to its previously-saved position.

```
int __cdecl wmain(int argc, wchar_t **argv)
{
 if (argc != 3) {
  wprintf(L"Usage: %ls save filename\n"
          L"       %ls restore filename\n", argv[0], argv[0]);
  return 0;
 }
 CCoInitialize init;

 CComPtr<IFolderView> spView;
 FindDesktopFolderView(IID_PPV_ARGS(&spView));

 if (wcscmp(argv[1], L"save") == 0) {
  SavePositions(spView, argv[2]);
 } else if (wcscmp(argv[1], L"restore") == 0) {
  RestorePositions(spView, argv[2]);
 }
 return 0;
}
```

And all that's left is to write the main program that calls either the `SavePositions` or `RestorePositions` function based on the command line parameters.

Exercise: Discuss what happens if you rename an item on the desktop, and then try to restore its position. What could be done to address this?

Raymond Chen

Follow