Modernizing our simple program that retrieves information about the items in the Recycle Bin

devblogs.microsoft.com/oldnewthing/20110831-00

August 31, 2011

Raymond Chen

Last time, we wrote a simple program to print various properties of the items in the Recycle Bin, and we did so in the classical style, using item ID lists and `IShellFolder`s. One thing you may have noticed is that a lot of functions take the combination of an `IShellFolder` and a `PCUITEMID_CHILD`. In the shell namespace, operations on items usually happen by means of the pair (folder, child), and one of the common mistakes made by beginners is failing to keep track of the pairing and passing child pidls to the wrong parent folder.

Even if you're not a beginner and are good at keeping track of which child pidls correspond to which parent folders, it's still extra work you have to do, and it means that a lot of functions take two parameters in order to describe one thing.

Enter `IShellItem`.

The `IShellItem` encapsulates the pair (folder, child). This solves two problems:

1. You only have to pass one thing around (the `IShellItem`) instead of two (the `IShellFolder` and the `PCUITEMID_CHILD`).
2. By keeping track of the two items as a single unit, it reduces the risk that you'll accidentally use a child pidl with the wrong parent folder.

Another complexity of the classic shell interface is that there are a bunch of ways of obtaining COM objects from a shell folder:

- `IShellFolder::BindToObject`
- `IShellFolder::BindToStorage`
- `IShellFolder::CreateViewObject`
- `IShellFolder::GetUIObjectOf`
- `IUnknown::QueryInterface` (thanks to the desktop special case we saw last time).

The `IShellItem::BindToHandler` interface hides these special-cases by dealing with them under the covers so you don't have to. You just call `IShellItem::BindToHandler` and it figures out where to get the object and what weird special cases apply. (It also takes care of

the weird `S_FALSE` return value from `IShellFolder::EnumObjects`.)

And then there's the annoyance of `IShellFolder::GetDisplayNameOf` using the kooky `STRRET` structure. The `IShellItem::GetDisplayName` function encapsulates that away for you by doing the work to convert that `STRRET` into a boring string pointer.

First up in modernizing our sample program is to change `BindToCsidl` to return a shell item instead of a shell folder.

```
HRESULT BindToCsidlItem(int csidl, IShellItem ** ppsi)
{
 *ppsi = NULL;
 HRESULT hr;
 PIDLIST_ABSOLUTE pidl;
 hr = SHGetSpecialFolderLocation(NULL, csidl, &pidl);
 if (SUCCEEDED(hr)) {
  hr = SHCreateShellItem(NULL, NULL, pidl, ppsi);
  CoTaskMemFree(pidl);
 }
 return hr;
}
```

But wait, since we're modernizing, we may as well upgrade to `SHGetKnownFolderIDList`:

```
HRESULT BindToKnownFolderItem(REFKNOWNFOLDER rfid, IShellItem ** ppsi)
{
 *ppsi = NULL;
 HRESULT hr;
 PIDLIST_ABSOLUTE pidl;
 hr = SHGetKnownFolderIDList(rfid, 0, NULL, &pidl);
 if (SUCCEEDED(hr)) {
  hr = SHCreateShellItem(NULL, NULL, pidl, ppsi);
  CoTaskMemFree(pidl);
 }
 return hr;
}
```

Hey wait, there's a function for this already in Windows 7! It's called `SHGetKnownFolder-Item`. Yay, now we can delete the function entirely.

Next, we convert `PrintDisplayName` to use `IShellItem` and the item-based display name flags `SIGDN`.

```
void PrintDisplayName(IShellItem *psi, SIGDN sigdn, PCTSTR pszLabel)
{
 LPWSTR pszName;
 HRESULT hr = psi->GetDisplayName(sigdn, &pszName);
 if (SUCCEEDED(hr)) {
  _tprintf(TEXT("%s = %ws\n"), pszLabel, pszName);
  CoTaskMemFree(pszName);
 }
}
```

And then we convert `PrintDetail` to use `IShellItem`. Oh wait, now we've hit a snag: The `IShellItem` interface doesn't have a helper method that wraps `IShellFolder2::GetDetailsEx`. Fortunately, there is a way to ask `IShellItem` to regurgitate the `IShellFolder` and `PITEMID_CHILD` that it is wrapping: You use the `IParentAndItem::GetParentAndItem` method.

```
void PrintDetail(IShellItem *psi,
    const SHCOLUMNID *pscid, PCTSTR pszLabel)
{
 IParentAndItem *ppni;
 HRESULT hr = psi->QueryInterface(IID_PPV_ARGS(&ppni));
 if (SUCCEEDED(hr)) {
  IShellFolder *psf;
  PITEMID_CHILD pidl;
  hr = ppni->GetParentAndItem(NULL, &psf, &pidl);
  if (SUCCEEDED(hr)) {
   VARIANT vt;
   hr = psf->GetDetailsEx(pidl, pscid, &vt);
   if (SUCCEEDED(hr)) {
    hr = VariantChangeType(&vt, &vt, 0, VT_BSTR);
    if (SUCCEEDED(hr)) {
     _tprintf(TEXT("%s: %ws\n"), pszLabel, V_BSTR(&vt));
    }
    VariantClear(&vt);
   }
   psf->Release();
   CoTaskMemFree(pidl);
  }
 }
}
```

Wow, it looks like we lost ground there. Ah, but Windows Vista extends `IShellItem` with the `IShellItem2` interface, and that has a bunch of new methods for retrieving properties.

```
void PrintDetail(IShellItem2 *psi,
    const SHCOLUMNID *pscid, PCTSTR pszLabel)
{
  PROPVARIANT vt;
  HRESULT hr = psi->GetProperty(*pscid, &vt);
  if (SUCCEEDED(hr)) {
   hr = VariantChangeType(&vt, &vt, 0, VT_BSTR);
   if (SUCCEEDED(hr)) {
    _tprintf(TEXT("%s: %ws\n"), pszLabel, V_BSTR(&vt));
   }
   PropVariantClear(&vt);
  }
 }
}
```

But wait, there's more. There's a special accessor just for retrieving properties as strings!

```
void PrintDetail(IShellItem2 *psi2,
    const SHCOLUMNID *pscid, PCTSTR pszLabel)
{
 LPWSTR pszValue;
 HRESULT hr = psi2->GetString(*pscid, &pszValue);
 if (SUCCEEDED(hr)) {
  _tprintf(TEXT("%s: %ws\n"), pszLabel, pszValue);
  CoTaskMemFree(pszValue);
 }
}
```

Okay, that's more like it. Now let's update the main program.

```
int __cdecl _tmain(int argc, PTSTR *argv)
{
 HRESULT hr = CoInitialize(NULL);
 if (SUCCEEDED(hr)) {
  IShellItem *psiRecycleBin;
  hr = SHGetKnownFolderItem(FOLDERID_RecycleBinFolder, KF_FLAG_DEFAULT,
                            NULL, IID_PPV_ARGS(&psiRecycleBin));
  if (SUCCEEDED(hr)) {
   IEnumShellItems *pesi;
   hr = psiRecycleBin->BindToHandler(NULL, BHID_EnumItems,
                                     IID_PPV_ARGS(&pesi));
   if (hr == S_OK) {
    IShellItem *psi;
    while (pesi->Next(1, &psi, NULL) == S_OK) {
     IShellItem2 *psi2;
     if (SUCCEEDED(psi->QueryInterface(IID_PPV_ARGS(&psi2)))) {
      _tprintf(TEXT("-----------------\n"));
      PrintDisplayName(psi2, SIGDN_PARENTRELATIVE,
                             TEXT("ParentRelative"));
      PrintDisplayName(psi2, SIGDN_NORMALDISPLAY, TEXT("Normal"));
      PrintDisplayName(psi2, SIGDN_FILESYSPATH, TEXT("FileSys"));
      PrintDetail(psi2, &SCID_OriginalLocation, TEXT("Original Location"));
      PrintDetail(psi2, &SCID_DateDeleted, TEXT("Date deleted"));
      PrintDetail(psi2, &PKEY_Size, TEXT("Size"));
      psi2->Release();
     }
     psi->Release();
    }
   }
   psiRecycleBin->Release();
  }
  CoUninitialize();
 }
 return 0;
}
```

Okay, so now we know how to enumerate the contents of the Recycle Bin and obtain properties of the items in it. How do we purge or restore items? We'll look at that next time.

Raymond Chen

Follow