Creating a simple pidl: For the times you care enough to send the very fake

devblogs.microsoft.com/oldnewthing/20130503-00

May 3, 2013

Raymond Chen

I'll assume that we all know what pidls are and how the shell namespace uses them. That's the prerequisite for today.

A *simple pidl* is an item ID list that refers to a file or directory that may not actually exist. It's a way of playing "what if": "If there were a file or directory at this location, here is what I would have created to represent it." For the times you care enough to send the very fake.

We've seen these things in action with the `SHGFI_USEFILEATTRIBUTES` flag, which tells the `SHGetFileInfo` function, "Pretend that the file/directory exists with the attributes I specified, and tell me what the icon would be, were that item to actually exist."

Internally, the `SHGetFileInfo` function creates one of these "simple pidls", and then asks the simple pidl for its icon.

Note that a simple pidl is really a special case of a pidl created from a `WIN32_FIND_DATA`. When you parse a display name with a custom bind context, and the bind context has a `STR_FILE_SYS_FIND_DATA` bind context object, then that object is used to control the information placed into the pidl instead of getting the information from the file system.

Here's a program that creates a simple pidl and then does a few simple things with it. (Note that the Parsing with Parameters sample covers this topic too, so if you don't like the way I did it, you can look to see how somebody else did it.)

```
#define STRICT_TYPED_ITEMIDS
#include <new>
#include <windows.h>
#include <ole2.h>
#include <oleauto.h>
#include <shlobj.h>
#include <propkey.h>
#include <atlbase.h>
#include <atlalloc.h>


class CFileSysBindData : public IFileSystemBindData
{
public:
 static HRESULT CreateInstance(
  _In_ const WIN32_FIND_DATAW *pfd,
  _In_ REFIID riid, _Outptr_ void **ppv);


 // *** IUnknown ***
 IFACEMETHODIMP QueryInterface(
  _In_ REFIID riid, _Outptr_ void **ppv)
 {
  *ppv = nullptr;
  HRESULT hr = E_NOINTERFACE;
  if (riid == IID_IUnknown ||
      riid == IID_IFileSystemBindData) {
   *ppv = static_cast<IFileSystemBindData *>(this);
   AddRef();
   hr = S_OK;
  }
  return hr;
 }


 IFACEMETHODIMP_(ULONG) AddRef()
 {
  return InterlockedIncrement(&m_cRef);
 }


 IFACEMETHODIMP_(ULONG) Release()
 {
  LONG cRef = InterlockedDecrement(&m_cRef);
  if (cRef == 0) delete this;
  return cRef;
 }


 // *** IFileSystemBindData ***
 IFACEMETHODIMP SetFindData(_In_ const WIN32_FIND_DATAW *pfd)
 {
```

```
  m_fd = *pfd;
  return S_OK;
 }


 IFACEMETHODIMP GetFindData(_Out_ WIN32_FIND_DATAW *pfd)
 {
  *pfd = m_fd;
  return S_OK;
 }


private:
 CFileSysBindData(_In_ const WIN32_FIND_DATAW *pfd) :
  m_cRef(1)
 {
  m_fd = *pfd;
 }
private:
 LONG m_cRef;
 WIN32_FIND_DATAW m_fd;
};


HRESULT CFileSysBindData::CreateInstance(
 _In_ const WIN32_FIND_DATAW *pfd,
 _In_ REFIID riid, _Outptr_ void **ppv)
{
 *ppv = nullptr;
 CComPtr<IFileSystemBindData> spfsbd;
 HRESULT hr = E_OUTOFMEMORY;
 spfsbd.Attach(new (std::nothrow) CFileSysBindData(pfd));
 if (spfsbd) {
  hr = spfsbd->QueryInterface(riid, ppv);
 }
 return hr;
}
```

The `CFileSysBindData` object is extraordinarily boring. It simply implements `IFile-SystemBindData`, which is a simple interface that just babysits a `WIN32_FIND_DATA` structure. (There is also a `IFileSystemBindData2` interface which babysits a little more information, but for the purpose of this program, we're interested only in the `WIN32_FIND_DATA`.)

```
HRESULT CreateBindCtxWithOpts(
 _In_ BIND_OPTS *pbo, _Outptr_ IBindCtx **ppbc)
{
 CComPtr<IBindCtx> spbc;
 HRESULT hr = CreateBindCtx(0, &spbc);
 if (SUCCEEDED(hr)) {
  hr = spbc->SetBindOptions(pbo);
 }
 *ppbc = SUCCEEDED(hr) ? spbc.Detach() : nullptr;
 return hr;
}
```

A bind context is basically a string-indexed associative array of COM objects. There is also a `BIND_OPTS` (or `BIND_OPTS2`) structure in there, but the things most people care about are the object parameters. They provide an extensible method of passing arbitrary parameters to a function. (Think of it as the COM version of the JavaScript convention of jamming random junk into an Options parameter.) You start with a `IBindCtx` parameter, and any time you need to add a new flag or parameter, you just stuff it into the `IBindCtx`. If you just want to add a new boolean flag, you can even ignore the contents of the object parameter and merely base your behavior on whether the parameter exists at all.

```
HRESULT AddFileSysBindCtx(
 _In_ IBindCtx *pbc, _In_ const WIN32_FIND_DATAW *pfd)
{
 CComPtr<IFileSystemBindData> spfsbc;
 HRESULT hr = CFileSysBindData::CreateInstance(
  pfd, IID_PPV_ARGS(&spfsbc));
 if (SUCCEEDED(hr)) {
  hr = pbc->RegisterObjectParam(STR_FILE_SYS_BIND_DATA,
                                spfsbc);
 }
 return hr;
}
```

To add a file system bind parameter, you just create an object which implements `IFile-SystemBindData` and register it with the bind context with the string `STR_FILE_SYS_FIND_DATA`.

```
HRESULT CreateFileSysBindCtx(
 _In_ const WIN32_FIND_DATAW *pfd, _Outptr_ IBindCtx **ppbc)
{
 CComPtr<IBindCtx> spbc;
 BIND_OPTS bo = { sizeof(bo), 0, STGM_CREATE, 0 };
 HRESULT hr = CreateBindCtxWithOpts(&bo, &spbc);
 if (SUCCEEDED(hr)) {
  hr = AddFileSysBindCtx(spbc, pfd);
 }
 *ppbc = SUCCEEDED(hr) ? spbc.Detach() : nullptr;
 return hr;
}
```

The `CreateFileSysBindCtx` function simply combines the two steps of creating a bind context and then adding a file system bind parameter to it. In casual conversation, a bind context is often named after the parameter inside it. In this case, we have a bind context with a file system bind parameter, so we call it a "file system bind context".

```
HRESULT CreateSimplePidl(
 _In_ const WIN32_FIND_DATAW *pfd,
 _In_ PCWSTR pszPath, _Outptr_ PIDLIST_ABSOLUTE *ppidl)
{
 *ppidl = nullptr;
 CComPtr<IBindCtx> spbc;
 HRESULT hr = CreateFileSysBindCtx(pfd, &spbc);
 if (SUCCEEDED(hr)) {
  hr = SHParseDisplayName(pszPath, spbc, ppidl, 0, nullptr);
 }
 return hr;
}
```

This is where everything comes together. To create a simple pidl, we take the `WIN32_FIND_DATAW` containing the metadata we want to use, put it inside a file system bind context, then use that bind context to parse the file name. The presence of a file system bind context tells the parser, "Trust me on this, just go with what's in the bind context." It suppresses all disk access, and the final pidl will describe an item that exactly matches the metadata you provided, whether that accurately reflects reality or not. (You can also pass the bind context to `SHCreateItemFromParsingName` if you prefer to get an `IShellItem`.)

Okay, let's take this out for a spin.

```
void DoStuffWith(_In_ PCIDLIST_ABSOLUTE pidl)
{
 // Print the file name
 wchar_t szBuf[MAX_PATH];
 if (SHGetPathFromIDListW(pidl, szBuf)) {
  wprintf(L"Path is \"%ls\"\n", szBuf);
 }


 // Print the file size
 CComPtr<IShellFolder2> spsf;
 PCUITEMID_CHILD pidlChild;
 if (SUCCEEDED(SHBindToParent(pidl,
                        IID_PPV_ARGS(&spsf), &pidlChild))) {
  CComVariant vt;
  if (SUCCEEDED(spsf->GetDetailsEx(pidlChild,
                        &PKEY_Size, &vt))) {
   if (SUCCEEDED(vt.ChangeType(VT_UI8))) {
    wprintf(L"Size is %I64u\n", vt.ullVal);
   }
  }
 }
}


int __cdecl wmain(int argc, PWSTR argv[])
{
 CCoInitialize init;
 if (SUCCEEDED(init)) {
  WIN32_FIND_DATAW fd = {};
  fd.dwFileAttributes = FILE_ATTRIBUTE_NORMAL;
  fd.nFileSizeLow = 42;
  CComHeapPtr<ITEMIDLIST_ABSOLUTE> spidlSimple;
  if (SUCCEEDED(CreateSimplePidl(&fd,
                L"Q:\\Whatever.txt", &spidlSimple))) {
   DoStuffWith(spidlSimple);
  }
 }
 return 0;
}
```

Our test program asks for a simple pidl to `Q:\Whatever.txt`, and then prints information from it. Observe that the creation of the simple pidl succeeds even though you probably don't have a Q: drive, and even if you did, the code never tried to access it. And when we ask the pidl, "Hey, what's the file size?" it retrieves the fake value 42 we passed in the `WIN32_FIND_DATAW` structure.

Sure, that was kind of artificial, but so-called simple pidls are handy if you want to talk about an object on slow media (such as a network share) without actually accessing the target device.

Exercise: What changes are necessary in order to create a simple pidl that refers to a file with illegal characters in its name? Hint: `STR_NO_VALIDATE_FILENAME_CHARS`.

Raymond Chen

Follow