Using the REFIID/void** pattern for returning COM objects for future-proofing and to avoid problems with dependencies

devblogs.microsoft.com/oldnewthing/20140710-00

July 10, 2014

Raymond Chen

Suppose you have a function that creates a reference to a COM object:

```
// pixie.h
STDAPI CreateShellItemFromPixieDust(
    const PIXIEDUST *ppd,
    IShellItem **ppsi);
```

There are a few issues with this design.

First of all, it requires that whoever uses your header file must have included `shlobj.h` first, since that's where `IShellItem` is defined. You could solve that problem by putting `#include <shlobj.h>` at the top of `pixie.h`, but that creates its own problems. For example, many header files alter their behavior based on symbols that have been `#define`d, and including that other header file as part of `pixie.h` means that it's going to use the settings that were active at the time `pixie.h` was included (which may not be what the clients of your header file are expecting). For example:

```
#include <windows.h>
#include <ole2.h>
#include <pixie.h>
#define STRICT_TYPED_ITEMIDS
#include <shlobj.h>
```

This program wants to use strict typed item IDs, so it defines the magic symbol before including `shlobj.h`. Unfortunately, that request is ignored because the `pixie.h` header file secretly included `shlobj.h` prematurely.

This can get particularly messy if somebody wants to include `shlobj.h` with particular preprocessor tricks temporarily active.

```
#include <windows.h>
#include <ole2.h>
#include <pixie.h>
// The WINDOWDATA structure added in Windows Vista conflicts
// with the one we defined back in 2000, so rename it to
// WINDOWDATA_WINDOWS.
#define WINDOWDATA WINDOWDATA_WINDOWS
#include <shlobj.h>
#undef WINDOWDATA
// Here's our version of WINDOWDATA
#include <contosotypes.h>
```

This code works around a naming conflict that was created when Windows Vista added a structure called `WINDOWDATA` to `shlobj.h`. The application already had a structure with the same name, so it has to rename the one in `shlobj.h` to some other name to avoid a redefinition error.

If you made `pixie.h` include `shlobj.h` on its own, it would do so without this fancy renaming, and the developers of that code will curse and say something like this:

```
#include <windows.h>
#include <ole2.h>
// The WINDOWDATA structure added in Windows Vista conflicts
// with the one we defined back in 2000, so rename it to
// WINDOWDATA_WINDOWS.
#define WINDOWDATA WINDOWDATA_WINDOWS
// pixie.h secretly includes shlobj.h so we have to put its #include
// under WINDOWDATA protection.
#include <pixie.h>
#include <shlobj.h>
#undef WINDOWDATA
// Here's our version of WINDOWDATA
#include <contosotypes.h>
```

Another problem with the `CreateShellItemFromPixieDust` function is that it hard-codes the output interface to `IShellItem`. When everybody moves on to `IShellItem2`, all the callers will have to follow the `CreateShellItemFromPixieDust` call with a `QueryInterface` to get the interface they really want. (Which, if your object is out-of-process, could mean another round trip to the server.)

The solution to both of these problems is to simply make the caller specify what type of object they want.

```
// pixie.h
STDAPI CreateShellItemFromPixieDust(
    const PIXIEDUST *ppd,
    REFIID riid,
    void **ppv);
```

Now that we are no longer mentioning `IShellItem` explicitly, we don't need to include `shlobj.h` any more. And if the caller wants `IShellItem2`, they can just ask for it.

Your creation function used to look like this:

```
STDAPI CreateShellItemFromPixieDust(
    const PIXIEDUST *ppd,
    IShellItem **ppsi)
{
    *ppsi = nullptr;
    IShellItem *psiResult;
    HRESULT hr = ... do whatever ...;
    if (SUCCEEDED(hr))
    {
       *ppsi = psiResult;
    }
    return hr;
}
```

You simply have to tweak the way you return the pointer:

```
STDAPI CreateShellItemFromPixieDust(
    const PIXIEDUST *ppd,
    REFIID riid,
    void **ppv)
{
    *ppv = nullptr;
    IShellItem *psiResult;
    HRESULT hr = ... do whatever ...;
    if (SUCCEEDED(hr))
    {
       hr = psiResult->QueryInterface(riid, ppv);
       psiResult->Release();
    }
    return hr;
}
```

Callers of your function would go from

```
IShellItem *psi;
hr = CreateShellItemFromPixieDust(ppd, &psi);
```

to

```
IShellItem *psi;
hr = CreateShellItemFromPixieDust(ppd, IID_PPV_ARGS(&psi));
```

If the caller decides that they really want an `IShellItem2`, they merely have to change their variable declaration; the call to the creation function is unchanged.

```
IShellItem2 *psi;
hr = CreateShellItemFromPixieDust(ppd, IID_PPV_ARGS(&psi));
```

Raymond Chen

Follow