Why don't I use any class libraries in my sample code?

devblogs.microsoft.com/oldnewthing/20070216-03

February 16, 2007

Raymond Chen

As a general rule, I avoid using any class libraries in my sample code. This isn't because I'm opposed to class libraries, but rather because I don't want to narrow my audience to "people who use MFC" (to choose one popular class library). If I were to start using MFC for all of my samples, I'd probably lose all the people who don't use MFC. "Oh, but those people can just translate the MFC code into whatever class library they use." Well, sure, they could do that, but first they would have to learn MFC. I wouldn't be talking about `HWND`s and `HDC`s any more but rather `CWnd`s and `CDC`s. I would write "Add this to your `OnDropEx` handler", and all the non-MFC people would say, "What are you talking about? I'm not using MFC. What is the Win32 equivalent to `OnDropEx`?" (Suppose my article on using accessibility to read the text under the mouse cursor were titled "How to use MFC to retrieve text under the mouse cursor." Would you have read it?) "Well, fine, don't use MFC, but still it wouldn't kill you to use a smart pointer library." But which one? There's MFC's `CIP`, ATL's `CComPtr`, STL^H^H^Hthe C++ standard library's `auto_ptr`, the Microsoft compiler's built-in `_com_ptr_t` (which you get automatically if you use the nonstandard `#import` directive), and boost's grab bag of smart pointer classes `scoped_ptr`, `shared_ptr`, `weak_ptr`, `intrusive_ptr` ... And they all behave differently. Sometimes subtly incompatibly. For example, MFC's `CIP::CreateObject` method uses `CLSCTX_INPROC_SERVER`, whereas ATL's `CComPtr::CreateInstance` method uses `CLSCTX_ALL`. When you're chasing down a nasty COM marshalling problem, these tiny details matter, and if you're an ATL programmer looking at MFC code, these tiny details are also something you're going to miss simply due to lack of familiarity. (And woe unto you if your preferred language is VB or C# or some other popular non-C++ language. Now you have double the translation work ahead of you.) Instead of hiding the subtleties behind a class library, I put them right out on the table. Those of you who have a favorite class library can convert the boring error-prone plain C++ code into your beautiful class library. In fact, I almost expect you to do it.

(On a related note, some people are horrified at the rather dense code presentation I use here. I don't write code like that in real life; I'd be just as horrified as you if I saw that code in a real program. I just use that style here because of the nature of the medium. A great way to lose people's interest is to make them plow through 100 lines of boring code before they reach the good stuff.)

Raymond Chen

Follow