Registration-free COM the old-fashioned way: The car mp3 player

devblogs.microsoft.com/oldnewthing/20120406-00

April 6, 2012

Raymond Chen

Windows XP introduced Registration-Free COM, permitting you to place your COM object registrations in a manifest rather than in the registry. Very handy when you want to support xcopy deployment or running directly off a thumb drive. And very much in keeping with the principle of not using a global solution for a local problem. (If you need your COM object to be used from other programs, then global registration is not unreasonable, but if the only client is another part of your program, then you have a local problem that should employ a local solution.) Here are some articles on the subject:

Before manifest-based COM object registration was possible, you had to do things the old-school way. Mind you, old-school registration-free COM is not a very useful skill any more, falling into the same category as knowing how to churn butter or use a typewriter eraser, but since when did that ever stop me from writing about something? One old-school way is to call `DllGetClassObject` directly. This works only if you control both sides of the equation (the client and the server) because it's now your responsibility to ensure that both sides agree on the threading model. You won't have the actual COM libraries sitting in between watching out for scenarios that require marshalling, for example. For a toy project of mine, I used a different approach. My little project was an mp3 player for my laptop. Now, sure, we already have tons of mp3-playing apps out there, but mine was different: Mine was designed to be used on long driving trips. (This was back in the days when long driving trips were part of my life. I don't drive that much any more.) Here was the plan: I connected the line-out from the laptop into my car sound system, so that the music came out my car speakers. Meanwhile, all input to the program came from the mouse. Specifically, it came from the mouse buttons and the wheel. The mouse itself never moved. The idea was that I could hook up a mouse to the laptop, put the laptop on the passenger seat, and leave the mouse on the center console. I would then use the mouse buttons and wheel to navigate my music collection. I forget exactly what I ended up doing, but it was something like

- Left button click = select current item
- Right button click = go up one level
- Rotate wheel = scroll through current directory

Now remember, this program was designed to be used while driving, which means both eyes on the road. (This was long before hands-free driving laws came on the scene.) Therefore, the program provided feedback not by displaying items on the screen but by using speech synthesis to read the names of the files and directories out loud. Finding a song to play, therefore, consisted of turning the wheel and listening as the laptop read out the name of the album, then when I found the one I wanted, I would click the left mouse button, and then I would use the wheel to scroll through the songs, and when I heard the title of the one I wanted, I clicked the left mouse button again, and boom, the song started playing. I added some heuristics to the code so if consecutive tracks began with the same words (which happens often with classical music, think *Symphony #5 in c minor, first movement* followed by *Symphony #5 in c minor, second movement*) it spoke only the changes. While the song was playing, the mouse buttons served as playback controls. I think it went something like this:

- Left button click = pause / play
- Right button click = exit and choose another song
- Rotate wheel = rewind / fast-forward ten seconds
- Press middle mouse button and rotate wheel = previous/next track

(Exercise: Why didn't I need a volume control?) The easiest programming language for this was a Web page. I created a host program that simply created a Web browser control. The host program told the Web browser control to navigate to my carplay.html file, and boom, I now had an in-car playback system. I could use things like `FileSystemObject` to navigate the file system and the Windows Media Player control to do the playback. Now, this story takes place so many years ago that Internet Explorer didn't support the mouse wheel yet, so the host program also converted wheel messages into fake keyboard activity (wheel motion was converted into the up and down arrows) so that the Web page could respond to them. Once nice thing about this whole set-up is that I had the HTML DOM at my disposal. My program spewed diagnostic output to the screen like crazy, but who cares? The end user isn't looking at the screen. Therefore, the entire Web page is free real estate for debugging. The only thing missing was the speech synthesizer. There was no ActiveX control for speech synthesis, so I wrote one of my own. I let SAPI do the heavy lifting; my ActiveX control was simply some glue that let a Web page pass a string to be spoken. (Or pass a null string to shut up.) I wanted my program to be xcopy-deployable (USB thumb drives didn't exist back then) so I looked for a registration-free technique. The `DllGetClassObject` technique wouldn't work since I didn't control how Internet Explorer created COM objects; it always called `CoCreateInstance`. The technique I used was `CoRegisterClassObject`. I created a class factory for my object and explicitly registered it before creating the Web browser control. That way, when the Web page asked for my object, lo and behold, there it was: In memory, not in the registry. That was a really long-winded story with a punch line that tells you how to do something you don't need to do any more because there are easier ways of doing it nowadays. I wouldn't be surprised if you wanted a refund.

The real punch line: I spent far more time writing the program than I ever did using it.

Raymond Chen

Follow