How did registry keys work in 16-bit Windows?

devblogs.microsoft.com/oldnewthing/20080117-00

January 17, 2008

Raymond Chen

Back in 16-bit Windows, the registry was a very different beast. Originally, the only registry hive was `HKEY_CLASSES_ROOT`, and the only things it was used for were COM object registration and file type registration. (Prior to the registry, file type registration was done in `WIN.INI`, and the only verb you could register for was "open".) The registry was stored in a single file, `REG.DAT`, which could not exceed 64KB in size. But back to registry keys. In 16-bit Windows, the registry was much simpler. The first program to call `RegOpenKey` caused the registry to be loaded, and the registry kept a running count of how many registry keys were open. When the matching number of `RegCloseKey` calls was made, the registry was written back to disk and unloaded from memory. (Yes, this means that if a program leaked a registry key, then the registry never got unloaded. Welcome to 16-bit Windows, which trusted software developers not to be stupid.) The registry key itself was just an index into the raw registry data. This backgrounder is really just a boring set-up for this little quirk of the `RegOpenKey` function in Win32:

> If this parameter [lpSubKey] is NULL or a pointer to an empty string, the function returns the same handle that was passed in.

This is an example of *over-documentation*, documenting the implementation rather than the contract. That the same handle was returned in 16-bit Windows is just a side effect of the fact that a registry key was an index. Opening the same key twice leads to the same key, which consequently has the same index (since it's the same thing). Therefore, opening the same key more than once yields the same numeric value for the handle. On the other hand, in 16-bit Windows, even if you reopened the key and got the same numeric value back, it nevertheless increased the "number of open registry keys" counter, and consequently you had to call `RegCloseKey` for each successful call to `RegOpenKey`, even if the numerical handle values were the same (which you shouldn't care about since handles are opaque). The people designing the 32-bit registry found themselves in a pickle. What should `RegOpenKey` do if asked to open the same key that was passed in? Should it return a new key which also must be closed with `RegCloseKey`? Or should it preserve the inadvertently contractual 16-bit behavior and return the same numeric key back? Since kernel handles are not reference-counted, returning the same numeric value back means that when the caller closes the key, they close both that key and the original key passed in (since they are the same thing). The

Win32 folks went for the second option: `RegOpenKey` on the same key returns the same numeric value back, which must not be closed (for doing so would close the original key as well). Personally, I would have gone for the first option (returning a new handle), but presumably the people who made this decision did so for a good reason. I suspect application compatibility played a major role.

What does this mean for you? It means that you should just plain avoid the `RegOpenKey` function, since it becomes harder to predict whether you need to close the returned key or not. Sure, if you pass a hard-coded string as the subkey name you can tell, but if the subkey name is dynamically-generated, then there's a possibility that the subkey name is a null string, in which case the returned key shouldn't be closed. Instead, use `RegOpenKeyEx`, which—since it is a new function in Win32—does not have the compatibility constraints of `RegOpenKey`. The key returned by `RegOpenKeyEx` is a brand new key and must be closed. Doesn't matter whether the subkey name is blank or not. Open with `RegOpenKeyEx` and close with `RegCloseKey`, end of story.

Raymond Chen

Follow