The relationship between module resources and resource-derived objects in 16-bit Windows

devblogs.microsoft.com/oldnewthing/20131002-00

October 2, 2013

Raymond Chen

As we saw last time, in 16-bit Windows, resources attached to an EXE or DLL file (which I called *module resources* for lack of a better term) were recorded in memory as discardable global memory blocks, and the window manager accessed them directly as needed. For example, if you had an icon or a cursor, the `HICON` or `HCURSOR` was really a resource handle, and when the window manager needed to draw the icon or cursor, it would cast the icon or cursor handle to a global handle (since that's what it was under the hood), then call `LockResource` to access the raw resource data in order to copy the pixels onto the screen. Similarly, accelerator tables were simply locked and accessed directly. On the other hand, some resources were actually templates for other objects. As suggested by their names, dialog and menu templates were just the blueprints for creating a dialog or menu. When you called `CreateDialog` or `LoadMenu`, the template was read from memory, and a fresh new dialog or menu was created based on the template. Once that was done, the template was no longer used. You could modify the resulting dialog or menu all you want, and you were also on the hook for making sure it is destroyed. (Either by destroying it yourself or by transferring that obligation to somebody else.) Bitmap resources worked the same way. The resource data is a template for a new bitmap, and each time you called `LoadBitmap` (or one of its moral equivalents), a brand new bitmap was created using the resource as a template. Once that was done, the template was no longer used, and you could modify the copy to your heart's content. (And you were also responsible for destroying it when you were done.) String resources were typically copied out of the resource section, either by the `LoadString` function or explicitly by your custom string extractor. The lifetime of the copied string was therefore controlled by you, and you could modify the copied string all you like since it was just a copy. If your custom string extractor simply returned a direct pointer to the resource rather than copying, then the pointer became invalid when the module was unloaded. Okay, let's summarize in a table:

Resource type	Operation	Result
Icon	`LoadIcon`, etc.	Reference

Cursor	`LoadCursor`, etc.	Reference
Accelerator	`LoadAccelerator`, etc.	Reference
Dialog	`CreateDialog`, etc.	Copy
Menu	`LoadMenu`, etc.	Copy
Bitmap	`LoadBitmap`, etc.	Copy
String	`LoadString`	Copy
String	`FindResource`	Reference

16-bit Resources

Some of these rules changed in the conversion from 16-bit Windows to 32-bit Windows, but in a way that tried to preserve the semantics of the operations. We'll look at those changes next time.

But even before you get to that article, you have enough information to answer this customer's question:

> How do I recover the dialog ID from a dialog if I have the dialog's window handle?

This is like asking, "How do I recover the recipe book that a particular cake was made from?" The cake does not know what recipe book it was made from. You might be able to do a chemical analysis followed by a thorough survey of all cookbooks in existence to try to find a match, but even if you do, it's merely a best-guess. (And if the dialog was modified after being created, then you will never find a match. Just like you will never find a cake recipe match if somebody decided to modify the cake after it came out of the oven.)

Raymond Chen

Follow