The relationship between module resources and resource-derived objects in 32-bit Windows

devblogs.microsoft.com/oldnewthing/20131003-00

October 3, 2013

Raymond Chen

Last time, we saw how 16-bit Windows converted resources attached to an EXE or DLL file (which I called *module resources* for lack of a better term) to user interface resources. As a refresher:

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

During the conversion from 16-bit Windows to 32-bit Windows, some of these rules changed. Specifically, icons, cursors, and accelerator tables are no longer references to the resource. Instead, the resource is treated as a template from which the actual user interface resource is constructed.

Resource type	Operation	Result
Icon	`LoadIcon`, etc.	Copy*

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

32-bit Resources

Uh-oh, what's up with those asterisks? Let's start with accelerator tables. In order to simulate the reference semantics of 16-bit accelerator tables, the copy is cached with a reference count, so that if you ask for the same accelerator table 1000 times, the first request creates a new accelerator table, and the other 999 requests just increment the reference count and return the same handle back. The result is that the window manager emulates reference semantics, but with an initial copy. When the reference count on an accelerator table drops to zero, then the resource is freed. Icons and cursors are the same, only weirder. If you pass the `LR_SHARED` flag, then the window manager simulates reference semantics by creating a copy of the icon or cursor the first time it is requested, and all subsequent requests with the `LR_SHARED` flag return the same handle back again.[1] The `LoadCursor` and `LoadIcon` functions are just wrappers around `LoadImage` that pass `LR_SHARED`, so applications written to the old 16-bit API still work the 16-bit way. (Even today, a lot of applications rely on the old 16-bit behavior.) If you don't pass the `LR_SHARED` flag, then you get a brand new copy of the icon or cursor. Since the only way to get this behavior is to call the new-for-Win32 function `LoadImage`, there is no compatibility issue. Based on the above discussion, we can flesh out the table a bit more:

Resource type	**Operation**	**Result**

Icon	`LoadIcon`
	`LoadImage` with `LR_SHARED`
		Cached copy

	`LoadImage` without `LR_SHARED`	Copy

Cursor	`LoadCursor`
	`LoadImage` with `LR_SHARED`
		Cached copy

	`LoadImage` without `LR_SHARED`	Copy
Accelerator	`LoadAccelerator`, etc.	Cached copy

Dialog	`CreateDialog`, etc.	Copy
Menu	`LoadMenu`, etc.	Copy
Bitmap	`LoadBitmap`, etc.	Copy
String	`LoadString`	Copy
String	`FindResource`	Reference

32-bit Resources

Another way of looking at the above table is to break it into two tables, one for operations that had a 16-bit equivalent, and one for operations that are unique to Win32:

Resource type	Operation	Result
Icon	`LoadIcon`	Simulated reference
Cursor	`LoadCursor`	Simulated reference
Accelerator	`LoadAccelerator`, etc.	Simulated reference
Dialog	`CreateDialog`, etc.	Copy
Menu	`LoadMenu`, etc.	Copy
Bitmap	`LoadBitmap`, etc.	Copy
String	`LoadString`	Copy
String	`FindResource`	Reference

32-bit Resource Creation Operations with 16-bit Equivalents

Resource type	Operation	Result
Icon	`LoadImage` with `LR_SHARED`	Simulated reference
	`LoadImage` without `LR_SHARED`	Copy
Cursor	`LoadImage` with `LR_SHARED`	Simulated reference
	`LoadImage` without `LR_SHARED`	Copy

32-bit Resource Creation Operations Without 16-bit Equivalents

Now we can answer an old question: "Do icons created from resources depend on the underlying resource?" The answer is no, at least not in 32-bit Windows. The bits are extracted from the module resource data and converted into a icon object, and if you passed the `LR_SHARED` flag, it is added to the cache of previously-created icons. [1] **Update**: If you read carefully, you'll realize that `LR_SHARED` stores the results in a cache *and pays no attention to the size*. The cache is keyed only by the resource module and ID; the size is ignored. This is why MSDN says "Do not use `LR_SHARED` for images that have nonstandard sizes." Suppose you load a resource with `LR_SHARED` and a nonstandard size. If you are the first person to load that resource, then the nonstandard size gets loaded and put into the cache. The next person to ask for that resource and who asks for a `LR_SHARED` copy will get the nonstandard-sized resource from the cache *regardless of what size they actually wanted*. Conversely, suppose a standard-size resource is already in the cache. You pass `LR_SHARED` and a nonstandard size. The cache returns you the original standard-size resource, ignoring the size you requested. To avoid this craziness, the rule is that any request for cached resources must use the standard size.

This requirement wasn't a problem in 16-bit Windows because 16-bit Windows had no way of requesting a resource at a nonstandard size. And since `LR_SHARED` is a new flag introduced in 32-bit Windows, all code which uses it can be expected to understand the Win32 rules.

Raymond Chen

Follow