The format of icon resources

devblogs.microsoft.com/oldnewthing/20120720-00

July 20, 2012

Raymond Chen

It's been a long time since my last entry in the continuing sporadic series on resources formats. Today we'll look at icons.

Recall that an icon file consists of two parts, an *icon directory* (consisting of an icon directory header followed by a number of icon directory entries), and then the icon images themselves.

When an icon is stored in resources, each of those parts gets its own resource entry.

The icon directory (the header plus the directory entries) is stored as a resource of type `RT_GROUP_ICON`. The format of the icon directory in resources is slightly different from the format on disk:

```
typedef struct GRPICONDIR
{
    WORD idReserved;
    WORD idType;
    WORD idCount;
    GRPICONDIRENTRY idEntries[];
} GRPICONDIR;
typedef struct GRPICONDIRENTRY
{
    BYTE  bWidth;
    BYTE  bHeight;
    BYTE  bColorCount;
    BYTE  bReserved;
    WORD  wPlanes;
    WORD  wBitCount;
    DWORD dwBytesInRes;
    WORD  nId;
} GRPICONDIRENTRY;
```

All the members mean the same thing as in the corresponding `ICONDIR` and `IconDirectoryEntry` structures, except for that mysterious `nId` (which replaces the `dwImageOffset` from the `IconDirectoryEntry`). To unravel that mystery, we need to look at where the rest of the icon file went.

In the icon file format, the `dwImageOffset` represented the location of the icon bitmap within the file. When the icon file is converted to a resource, each icon bitmap is split off into its own resource of type `RT_ICON`. The resource compiler auto-assigns the resource IDs, and it is those resource IDs that are stored in the `nId` member.

For example, suppose you have an icon file with four images. In your resource file you say

```
42 ICON myicon.ico
```

The resource compiler breaks the file into five resources:

Resource type	Resource Id	Contents

`RT_GROUP_ICON`	42	`GRPICONDIR.idCount = 4`
		`GRPICONDIRENTRY[0].nId = 124`
		`GRPICONDIRENTRY[1].nId = 125`
		`GRPICONDIRENTRY[2].nId = 126`
		`GRPICONDIRENTRY[3].nId = 127`

`RT_ICON`	124	Pixels for image 0
`RT_ICON`	125	Pixels for image 1
`RT_ICON`	126	Pixels for image 2
`RT_ICON`	127	Pixels for image 3

Why does Windows break the resources into five pieces instead of just dumping them all inside one giant resource?

Recall how 16-bit Windows managed resources. Back in 16-bit Windows, a resource was a handle into a table, and obtaining the bits of the resource involved allocating memory and loading it from the disk. Recall also that 16-bit Windows operated under tight memory constraints, so you didn't want to load anything into memory unless you really needed it.

Therefore, looking up an icon in 16-bit Windows went like this:

- Find the icon group resource, load it, and lock it.
- Study it to decide which icon image is best.
- Unlock and free the icon group resource since we don't need it any more.
- Find and load the icon image resource for the one you chose.
- Return that handle as the icon handle.

Observe that once we decide which icon image we want, the only memory consumed is the memory for that specific image. We never load the images we don't need.

Drawing an icon went like this:

- Lock the icon handle to get access to the pixels.
- Draw the icon.
- Unlock the icon handle.

Since icons were usually marked discardable, they could get evicted from memory if necessary, and they would get reloaded the next time you tried to draw them.

Although Win32 does not follow the same memory management model for resources as 16-bit Windows, it preserved the programming model (find, load, lock) to make it easier to port programs from 16-bit Windows to 32-bit Windows. And in order not to break code which loaded icons from resources directly (say, because they wanted to replace the icon selection algorithm), the breakdown of an icon file into a directory + images was also preserved.

You now know enough to solve this customer's problem:

> I have an icon in a resource DLL, and I need to pass its raw data to another component. However, the number of bytes reported by `SizeOfResource` is only 48 instead of 5KB which is the amount actually stored in the resource DLL. I triple-checked the resource DLL and I'm sure I'm looking at the right icon resource.
>
> Here is my code:
>
> ```
> HRSRC hrsrcIcon = FindResource(hResources,
>                      MAKEINTRESOURCE(IDI_MY_ICON), RT_GROUP_ICON);
> DWORD cbIcon = SizeofResource(hResources, hrsrcIcon);
> HGLOBAL hIcon = LoadResource(hResources, hrsrcIcon);
> void *lpIcon = LockResource(hIcon);
> ```

Raymond Chen

Follow