Exploring possible solutions to the inconsistency in how Windows searches case-insensitively for named resources

devblogs.microsoft.com/oldnewthing/20250723-00/?p=111403

July 23, 2025

Some time ago, I explained why Windows has trouble finding Win32 resources with accented characters. It boils down to an ambiguity in the Portable Executable specification: It says that the names are treated as case-insensitive, but it does not specify which case-insensitive comparison algorithm to use. The Resource Compiler uses the C locale (which considers the lowercase Latin letters a-z to be counterparts to the uppercase Latin letters A-Z), whereas the resource subsystem uses the user's current locale's case mapping table (which can vary from user to user, but which usually considers lowercase accented letters to be equivalent to the same accent applied to the uppercase version of the lowercase base character).

Commenter Jan Ringoš wondered if this could be fixed and offered a few possibilities.

The first option is to change the behavior of `FindResource` to use the C locale first and then use the user locale if there is no match according to the C locale. "I don't think it would break compatibility."

This change will definitely break compatibility. As well as hurting performance.

The performance penalty is obvious: Every resource search now takes place twice, once with C locale capitalization and once with user locale capitalization. Now, you could avoid the second search if you notice that the result of the user locale capitalization is the same as the C locale capitalization, and the second search is never performed if the first search succeeds, so the penalty is probably mitigated significantly. You'd have to do some performance testing to see how often the second search occurs in practice.

The compatibility break is more serious. It means that if a resource name matches the C locale capitalization but not the user locale capitalization, the revised version finds it, but the original version either failed or found the user locale version instead.

Returning a different resource entirely is clearly a problem. The program used to find the user locale capitalized resource, and after a Windows update, it now finds the C locale capitalized resource.

“But who would be so crazy as to have a C locale capitalized resource that is never found?”

In my experience, these types of “who would be so crazy as to…” questions are often answered with “because they were relying it by accident, not on purpose.”

Suppose you have designed your system so that when it loads a DLL, it looks for a resource with a name provided by a configuration file or some data. For example, the Internet Explorer `res:` protocol behaves this way. If somebody specifies a resource of the form `res://mydll.dll/#10/Städte`, the `res:` protocol handler will call `FindResource` with the string `Städte`, which means “cities” in German.

Somebody who wants to use this feature might add the cities resource to their Resource Compiler script:

```
Städte RCDATA L"Aachen\0Berlin\0\0Chemnitz"

⟦ ... hundreds of other resource statements ... ⟧
```

They then build the binary and find that it doesn’t work. (They don’t realize that the reason is that the resource is capitalized by the Resource Compiler as `STäDTE`, but the resource loader looks for `STÄDTE`. Not that we’d expect them to know this, seeing as how Maurice Kayser didn’t know it, and I see no reason to believe that Maurice’s experience isn’t typical of all other developers.)

As part of their “just keep trying everything you can think of in the hopes that one of them works”, they found that if they add

```
STÄDTE RCDATA L"Aachen\0Berlin\0\0Chemnitz"
```

at the very end of their resource script, then the cities list starts working.

As the program is developed, they kept updating the `STÄDTE` cities list, adding seven more cities, but the first one atrophied because it was far away and nobody noticed it.

If the behavior of `FindResource` were to change to prefer the C locale capitalization, then this program’s city list will regress to the three-city version, rather than the ten-city version they intended.

“Okay, so reverse the search. Look for the user local version first and fall back to the C locale.”

That still would break compatibility.

Suppose this program decided that it didn't want any cities at all, so they deleted the `STÄDTE` resource. The `FindResource` fails, and the cities list is blank. If you were to add the C locale fallback, then this change would find the old, long-forgotten `Städte` version, and three cities would show up in the list when they intended none.

Another proposal from Jan was to add a switch to the Resource Compiler to tell it to store the resource names as-is (no capitalization) and then change `FindResource` to search for an exact match first and then use the user locale if there is no exact match.

Again, this is a breaking change. Consider that program that was compiled ten years ago that had both `Städte` and `StÄdte` resources. Today, a search for `Städte` finds `StÄdte`, but with Jan's second proposal, it would find `Städte`, which is a change in behavior.

For compatibility reasons, the behavior of `FindResource` cannot change.

What you can do is have a switch for the Resource Compiler that says, "Capitalize the strings according to the user locale instead of the C locale." Only programs that have been recompiled with the new switch are affected, so they explicitly opted into the new behavior. Old programs (which were necessarily compiled without the new switch) continue to behave as before.

But if you think about it, this switch to control capitalization isn't really necessary. You can just capitalize the strings yourself in the resource script! (Or instead of doing the capitalization manually, maybe you write a preprocessing step that converts all the resource names to user-locale uppercase.)

The fact that you can already address the problem yourself without any tooling changes reduces the likelihood that the tooling will add a switch to change the capitalization rules: That flag would just be a convenience rather than providing essential functionality that isn't obtainable any other way.

Furthermore, since the resource loader uses the user locale, you can't really be sure what those strings are going to capitalize to, because you don't control the user's locale. So even in the absence of the locale mismatch, you still would be better off explicitly capitalizing your named resources, to remove the unpredictable user locale from the picture.

Or just stick to the uppercase letters A through Z for your resource names. I believe all locales treat them as already uppercase, so the uppercasing through the user locale is a no-operation.

Or just don't use named resources at all. Stick to numbers. Everybody agrees on numbers.