A shell extension is a guest in someone else's house; don't go changing the code page

devblogs.microsoft.com/oldnewthing/20110803-00

August 3, 2011

Raymond Chen

A customer reported a problem with their shell extension:

> We want to format a floating point number according to the user's default locale. We do this by calling `snprintf` to convert the value from floating point to text with a period (U+002E) as the decimal separator, then using `GetNumberFormat` to apply the user's preferred grouping character, decimal separator, etc. We found, however, that if the user is running in (say) German, we find that sometimes (but not always) the `snprintf` function follows the German locale and uses a comma (U+002C) as the decimal separator with no thousands separator. This format prevents the `GetNumberFormat` function from working, since it requires the decimal separator to be U+002E. What is the recommended way of formatting a floating point number according to the user's locale?

The recommended way of formatting a floating point number according to the user's locale is indeed to use a function like `snprintf` to convert it to text with U+002E as the decimal separator (and other criteria), then use `GetNumberFormat` to apply the user's locale preferences. The `snprintf` function follows the C/C++ runtime locale to determine how the floating point number should be converted, and the default C runtime locale is the so-called `"C"` locale which indeed uses U+002E as the decimal separator. Since you're getting U+002C as the decimal separator, somebody must have called `setlocale` to change the locale from `"C"` to a German locale, most likely by passing `""` as the locale, which means "follow the locale of the environment."

> Our shell extension is running in Explorer. Under what conditions will Explorer call `setlocale(LC_NUMERIC, "")`? What should we do if the locale is not `"C"`?

As it happens, Explorer never calls `setlocale`. It leaves the locale set to the default value of `"C"`. Therefore, the call to `snprintf` should have generated a string with U+002E as the decimal separator. Determining who was calling `setlocale` was tricky since the problem was intermittent, but after a lot of work, we found the culprit: some other shell extension loaded before the customer's shell extension and decided to change the carpet by calling `setlocale(LC_ALL, "")` in its `DLL_PROCESS_ATTACH`, presumably so that its calls to `snprintf` would follow the environment locale. What made catching the miscreant more difficult was that the rogue shell extension didn't restore the locale when it was unloaded (not that that would have been the correct thing to do either), so by the time the bad locale was detected, the culprit was long gone! That other DLL used a global setting to solve a local problem. Given the problem "How do I get my calls to `snprintf` to use the German locale settings?" they decided to change *all* calls to `snprintf` to use the German locale settings, even the calls that didn't originate from the DLL itself. What if the program hosting the shell extension had done a `setlocale(LC_ALL, "French")`? Tough noogies; the rogue DLL just screwed up the host program, which wants to use French locale settings but is now being forced to use German ones. The program probably won't notice that somebody secretly replaced its coffee with Folgers Crystals. It'll be a client who notices that the results are not formatted correctly. The developers of the host program, of course, won't be able to reproduce the problem in their labs, since they don't have the rogue shell extension, and the problem will be classified as "unsolved."

What both the rogue shell extension and the original customer's shell extension should be using is the `_l` variety of string formatting functions (in this case `_snprintf_l`, although `_snprintf_s_l` is probably better). The `_l` variety lets you pass an explicit locale which will be used to format that particular string. (You create one of these `_locale_t` objects by calling `_create_locale` with the same parameters you would have passed to `set-locale`.) Using the `_l` technique solves two problems:

1. It lets you apply a local solution to a local problem. The locale you specify applies only to the specific call; the process's default locale remains unchanged.
2. It allows you to ensure that you get the locale you want even if the host process has set a different locale.

If either the customer's DLL or the rogue DLL had followed this principle of not using a global setting to solve a local problem, the conflict would not have arisen.

Raymond Chen

Follow