On the proper care and feeding of the enigmatic Get-DistanceOfClosestLanguageInList function

devblogs.microsoft.com/oldnewthing/20230320-00

March 20, 2023

Raymond Chen

The `GetDistanceOfClosestLanguageInList` function takes a BCP47 language tag and tries to match it against a list of BCP47 language tags, reporting the result as a floating point number in the range 0.0 (no match at all) to 1.0 (exact match found). It doesn't tell you which language in the list was the closest match; it just tells you the quality of the match. What is the intended use of this function?

The idea here is that you have translated your program into several languages, and you want to choose the one that best matches the user's preferences. You take each of your available translations and see how well it matches the user's language preference list, and then you pick the best match.

We start with a helper function.

```
auto GetThreadPreferredUILanguageList()
{
    ULONG count;
    ULONG bufferSize = 0;
    THROW_IF_WIN32_BOOL_FALSE(
        GetThreadPreferredUILanguages(MUI_LANGUAGE_NAME,
            &count, nullptr, &bufferSize));
    std::unique_ptr<wchar_t[]> buffer(new wchar_t[bufferSize]);
    THROW_IF_WIN32_BOOL_FALSE(
        GetThreadPreferredUILanguages(MUI_LANGUAGE_NAME,
            &count, buffer.get(), &bufferSize));
    return buffer;
}
```

This helper function retrieves the current thread's preferred UI language list, in the form of a double null-terminated string. By an amazing coincidence, this happens to be the format that works best for the `GetDistanceOfClosestLanguageInList` function.

```
auto ChooseBestLanguage(std::vector<std::wstring> const& languages)
{
    auto userLanguages = GetThreadPreferredUILanguageList();

    double bestScore = 0.0;
    std::wstring bestLanguage;
    for (auto&& language : languages) {

        double score;
        auto hr = GetDistanceOfClosestLanguageInList(
            language.c_str(), userLanguages.get(), 0, &score);
        if (hr == HRESULT_FROM_WIN32(ERROR_NO_MATCH)) {
            // Ignore if language doesn't match anything.
            continue;
        }
        THROW_IF_FAILED(hr);

        if (score > bestScore) {
            bestScore = score;
            bestLanguage = language;

            // Can't improve on perfection.
            if (score == 1.0) break;
        }
    }
    return bestLanguage;
}
```

To choose the best language from a list of candidate languages, we call `GetDistanceOf-ClosestLanguageInList`, asking it to evaluate each candidate against the user's preferences, and we keep the one with the best score.

As a small optimization, we stop if we get a score of `1.0`, because that indicates a perfect match, and you can't improve on perfection.

Note that it's possible that we make it through the list without any candidate matching, in which case we return an empty string. When this happens, programs usually pick some default fallback language (which tends to be English).

Bonus chatter: My code sample checks explicitly for `HRESULT_FROM_WIN32(ERROR_NO_MATCH)` to detect and reject languages that do not match anything on the user's preference list. Since a failed match provides a score of zero, a failed match would not pass the `score > bestScore` test anyway. However, we do have to detect that specific error case so we don't treat it as a fatal error.

```
auto ChooseBestLanguage()
{
    auto userLanguages = GetThreadPreferredUILanguageList();

    double bestScore = 0.0;
    std::wstring bestLanguage;
    for (auto&& language : GetCandidateLanguages()) {

        double score;
        auto hr = GetDistanceOfClosestLanguageInList(
            language.c_str(), userLanguages.get(), 0, &score);
        THROW_HR_IF(FAILED(hr) &&
            hr != HRESULT_FROM_WIN32(ERROR_NO_MATCH));

        if (score > bestScore) {
            bestScore = score;
            bestLanguage = language;

            // Can't improve on perfection.
            if (score == 1.0) break;
        }
    }
    return bestLanguage;
}
```

In practice, though this ends up being a pointless optimization because you still have to test against the special value and skip over the error path, so you didn't really improve much. There's still a comparison and a jump. It's just that instead of jumping to the loop control if the match failed, you jump to the score test (which will fail).

Bonus bonus chatter: This function is misnamed. The value it calculates is not a "distance". If it were a distance, then zero would mean "closest", but this function returns 1.0 for the closest possible match and uses 0.0 to mean that there is no match at all. A better name would be something like `GetClosenessOfClosestLanguageInList`, where zero means "not close at all" and one means "the best closeness possible."