The crazy world of stripping diacritics

devblogs.microsoft.com/oldnewthing/20141124-00

November 24, 2014

Raymond Chen

Today's Little Program strips diacritics from a Unicode string. Why? Hey, I said that Little Programs require little to no motivation. It might come in handy in a spam filter, since it was popular, at least for a time, to put random accent marks on spam subject lines in order to sneak past keyword filters. (It doesn't seem to be popular any more.)

This is basically a C-ization of the C# code originally written by Michael Kaplan. Don't forget to read the follow-up discussion that notes that this can result in strange results.

First, let's create our dialog box. Note that I intentionally give it a huge font so that the diacritics are easier to see.

```
// scratch.h
#define IDD_SCRATCH 1
#define IDC_SOURCE 100
#define IDC_SOURCEPOINTS 101
#define IDC_DEST 102
#define IDC_DESTPOINTS 103
// scratch.rc
#include <windows.h>
#include "scratch.h"
IDD_SCRATCH DIALOGEX 0, 0, 320, 88
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
Caption "Stripping diacritics"
FONT 20, "MS Shell Dlg"
BEGIN
    LTEXT "Original:", -1, 4, 8, 38, 10
    EDITTEXT IDC_SOURCE, 46, 6, 270, 12, ES_AUTOHSCROLL
    LTEXT "", IDC_SOURCEPOINTS, 46, 22, 270, 12
    LTEXT "Modified:", -1, 4, 40, 38, 10
    EDITTEXT IDC_DEST, 46, 38, 270, 12, ES_AUTOHSCROLL
    LTEXT "", IDC_DESTPOINTS, 46, 54, 270, 12
    DEFPUSHBUTTON "OK", IDOK, 266, 70, 50, 14
END
```

Now the program that uses the dialog box.

```
// scratch.cpp
#define STRICT
#define UNICODE
#define _UNICODE
#include <windows.h>
#include <windowsx.h>
#include <strsafe.h>
#include "scratch.h"
#define MAXSOURCE 64
void SetDlgItemCodePoints(HWND hwnd, int idc, PCWSTR psz)
{
  wchar_t szResult[MAXSOURCE * 4 * 5];
  szResult[0] = 0;
  PWSTR pszResult = szResult;
  size_t cchResult = ARRAYSIZE(szResult);
  HRESULT hr = S_OK;
  for (; SUCCEEDED(hr) && *psz; psz++) {
    wchar_t szPoint[6];
    hr = StringCchPrintf(szPoint, ARRAYSIZE(szPoint), L"%04x ", *psz);
    if (SUCCEEDED(hr)) {
      hr = StringCchCatEx(pszResult, cchResult, szPoint, &pszResult, &cchResult, 0);
    }
  }
  SetDlgItemText(hwnd, idc, szResult);
}
```

The `SetDlgItemCodePoints` function takes a UTF-16 string and prints all the code points. This is just to help visualize the result; it's not part of the actual diacritic-removal algorithm.

```
void OnUpdate(HWND hwnd)
{
  wchar_t szSource[MAXSOURCE];
  GetDlgItemText(hwnd, IDC_SOURCE, szSource, ARRAYSIZE(szSource));
  wchar_t szDest[MAXSOURCE * 4];
  int cchActual = NormalizeString(NormalizationKD,
                                  szSource, -1,
                                  szDest, ARRAYSIZE(szDest));
  if (cchActual <= 0) szDest[0] = 0;
  WORD rgType[ARRAYSIZE(szDest)];
  GetStringTypeW(CT_CTYPE3, szDest, -1, rgType);
  PWSTR pszWrite = szDest;
  for (int i = 0; szDest[i]; i++) {
    if (!(rgType[i] & C3_NONSPACING)) {
      *pszWrite++ = szDest[i];
    }
  }
  *pszWrite = 0;
  SetDlgItemText(hwnd, IDC_DEST, szDest);
  SetDlgItemCodePoints(hwnd, IDC_SOURCEPOINTS, szSource);
  SetDlgItemCodePoints(hwnd, IDC_DESTPOINTS, szDest);
}
```

Okay, here's where the actual work happens. We put the source string into Normalization Form KD. This decomposes the diacritics so that we can identify them with `GetStringTypeW` and then strip them out.

Of course, in real life, you wouldn't hard-code the array sizes like I did here, but this is just a Little Program, and Little Programs are allowed to take shortcuts.

The rest of the program is just a framework to get into that function.

```
INT_PTR CALLBACK DlgProc(HWND hwnd, UINT wm,
                         WPARAM wParam, LPARAM lParam)
{
  switch (wm)
  {
  case WM_INITDIALOG:
    return TRUE;
  case WM_COMMAND:
    switch (GET_WM_COMMAND_ID(wParam, lParam)) {
    case IDC_SOURCE:
      switch (GET_WM_COMMAND_CMD(wParam, lParam)) {
    case EN_UPDATE:
      OnUpdate(hwnd);
      break;
    }
    break;
    case IDOK:
      EndDialog(hwnd, 0);
      return TRUE;
  }
  break;
  case WM_CLOSE:
    EndDialog(hwnd, 0);
    return TRUE;
  }
  return FALSE;
}
int WINAPI wWinMain(HINSTANCE hinst, HINSTANCE hinstPrev,
                    LPWSTR lpCmdLine, int nShowCmd)
{
  DialogBox(hinst, MAKEINTRESOURCE(IDD_SCRATCH), nullptr, DlgProc);
  return 0;
}
```

Okay, let's take this program for a spin. Here are some interesting characters to try:

Original character			Resulting character		
ª	00AA	Feminine ordinal indicator	a	0061	Latin small letter a

¹	00B1	Superscript one	1	0031	Digit one
½	00BD	Vulgar fraction one half	1⁄2	0031 2044 0032	Digit one + Fraction slash + Digit two
ı	0131	Latin small letter dotless i	ı	0131	Latin small letter dotless i
Ø	00D8	Latin capital letter O with stroke			Disappears!
ł	0142	Latin small letter l with stroke	ł	0142	Latin small letter l with stroke
l·	0140	Latin small letter l with middle dot	l·	006C 00B7	Latin small letter l + middle dot
æ	00E6	Latin small letter ae	æ	00E6	Latin small letter ae
Ή	0389	Greek capital letter Eta with tonos	Η	0397	Greek capital letter Eta
А	0410	Cyrillic capital letter A	A	0410	Cyrillic capital letter A
Å	00C5	Latin capital letter A with ring above	A	0041	Latin capital letter A
Ａ	FF21	Fullwidth Latin capital letter A	A	0041	Latin capital letter A
①	2460	Circled digit one	1	0031	Digit one
➀	2780	Dingbat circled sans-serif digit one	➀	2780	Dingbat circled sans-serif digit one
®	00AE	Registered sign	®	00AE	Registered sign
Ⓡ	24c7	Circled Latin capital letter R	R	0052	Latin capital letter R
𝔭	D835 DD95	Mathematical bold Fraktur small p	p	0070	Latin small letter p
ｬ	FF6C	Halfwidth Katakana letter small Ya	ャ	30E3	Katakana letter small Ya
ャ	30E3	Katakana letter small Ya	ャ	30E3	Katakana letter small Ya
ゴ	30B4	Katakana letter Go	コ	30B3	Katakana letter Ko
“	201C	Left double quotation mark	“	201C	Left double quotation mark

”	201D	Right double quotation mark	”	201D	Right double quotation mark
„	201E	Double low-9 quotation mark	„	201E	Double low-9 quotation mark
‟	201F	Double high-reversed-9 quotation mark	‟	201F	Double high-reversed-9 quotation mark
″	2033	Double prime	″	2032 2032	Prime + Prime
`	2035	Reverse prime	`	2035	Reverse prime
‹	2039	Single left-pointing angle quotation mark	‹	2039	Single left-pointing angle quotation mark
«	00AB	Left-pointing double angle quotation mark	«	00AB	Left-pointing double angle quotation mark
—	2014	Em-dash	—	2014	Em-dash
‼	203C	Double exclamation mark	!!	0021 0021	Exclamation mark + Exclamation mark

There are some interesting quirks here. Mind you, this is what the Unicode Consortium says, so if you think they are wrong, you can take it up with them.

The superscript-like characters are converted to their plain versions. Enclosed alphabetics are also converted, but not the ® symbol. Fullwidth forms of Latin letters are converted to their halfwidth equivalents. On the other hand, halfwidth Katakana characters are expanded to their fullwidth equivalents. But small Katakana does not convert to their large equivalents.

The Ø disappears completely! What's up with that? The character code for Ø is reported as `C3_ALPHA | C3_NONSPACING | C3_DIACRITIC`, and since we are removing nonspacing characters, this causes it to be removed. (Why is Ø nonspacing? It occupies space!) For whatever reason, it does not decompose into O + Combining Solidus Overlay. On the other hand, the Polish ł remains intact because it is reported as `C3_ALPHA | C3_DIACRITIC`. Poland wins and Norway loses?

The diacritic removal ignores linguistic rules. The Swedish Å decomposes into a capital A and a combining ring above, even though in Swedish, the character is considered nondecomposable. (Just like the capital letter Q in English does not decompose into an O and a tail.) Katakana Go suffers a similar ignoble fate, converting to Katakana Ko, which is linguistically nonsensical. But then again, removing diacritics *is already linguistically nonsensical*. Nonsensical operation is nonsensical.

There is no attempt to unify look-alike characters from different scripts. Look-alike characters in the Greek and Cyrillic alphabets are not mapped to their Latin doppelgängers.

The infamous Turkish dotless i does not turn into a dotted i. (And the lowercase Latin i does not decompose into a combining dot and a dotless i.)

Finally, I tried a selection of punctuation marks. Most of them pass through unchanged, with the exception of the double prime and double exclamation mark which each decompose into a pair of singles. (But double quotation marks do not decompose into a pair of singles.)

Okay, but the goal of this exercise was spam detection, so we are actually interested in mapping as far as possible all the way down to plain ASCII. We'd like to convert, for example, the look-alike characters in the Cyrillic and Greek alphabets to the Latin characters they resemble.

So let's try something else. If we want to convert to ASCII, then just convert to ASCII!

```
#define CP_ASCII 20127
void OnUpdate(HWND hwnd)
{
  wchar_t szSource[MAXSOURCE];
  GetDlgItemText(hwnd, IDC_SOURCE, szSource, ARRAYSIZE(szSource));
  char szDest[MAXSOURCE * 2];
  int cchActual = WideCharToMultiByte(CP_ASCII, 0, szSource, -1,
                              szDest, ARRAYSIZE(szDest), 0, 0);
  if (cchActual <= 0) szDest[0] = 0;
  SetDlgItemTextA(hwnd, IDC_DEST, szDest);
  SetDlgItemCodePoints(hwnd, IDC_SOURCEPOINTS, szSource);
}
```

We can extend the table above with a new column.

Original character			KD character			ASCII character		
ª	00AA	Feminine ordinal indicator	a	0061	Latin small letter a	a	0061	Latin small letter a
¹	00B1	Superscript one	1	0031	Digit one	1	0031	Digit one
½	00BD	Vulgar fraction one half	1⁄2	0031 2044 0032	Digit one + Fraction slash + Digit two	?		No conversion
ı	0131	Latin small letter dotless i	ı	0131	Latin small letter dotless i	i	0069	Latin small letter i

Ø	00D8	Latin capital letter O with stroke			Disappears!	O	004F	Latin capital letter O
ł	0142	Latin small letter l with stroke	ł	0142	Latin small letter l with stroke	l	006C	Latin small letter l
l·	0140	Latin small letter l with middle dot	l·	006C 00B7	Latin small letter l + middle dot	?		No conversion
æ	00E6	Latin small letter ae	æ	00E6	Latin small letter ae	a	0061	Latin small letter a
Ή	0389	Greek capital letter Eta with tonos	H	0397	Greek capital letter Eta	?		No conversion
А	0410	Cyrillic capital letter A	A	0410	Cyrillic capital letter A	?		No conversion
Å	00C5	Latin capital letter A with ring above	A	0041	Latin capital letter A	A	0041	Latin capital letter A
A	FF21	Fullwidth Latin capital letter A	A	0041	Latin capital letter A	A	0041	Latin capital letter A
①	2460	Circled digit one	1	0031	Digit one	?		No conversion
①	2780	Dingbat circled sans-serif digit one	①	2780	Dingbat circled sans-serif digit one	?		No conversion
®	00AE	Registered sign	®	00AE	Registered sign	R	0052	Latin capital letter R
Ⓡ	24c7	Circled Latin capital letter R	R	0052	Latin capital letter R	?		No conversion
𝔭	D835 DD95	Mathematical bold Fraktur small p	p	0070	Latin small letter p	??		No conversion
ｬ	FF6C	Halfwidth Katakana letter small Ya	ャ	30E3	Katakana letter small Ya	?		No conversion
ャ	30E3	Katakana letter small Ya	ャ	30E3	Katakana letter small Ya	?		No conversion

ゴ	30B4	Katakana letter Go	コ	30B3	Katakana letter Ko	?		No conversion
“	201C	Left double quotation mark	“	201C	Left double quotation mark	“	0022	Quotation mark
”	201D	Right double quotation mark	”	201D	Right double quotation mark	“	0022	Quotation mark
„	201E	Double low-9 quotation mark	„	201E	Double low-9 quotation mark	“	0022	Quotation mark
‟	201F	Double high-reversed-9 quotation mark	‟	201F	Double high-reversed-9 quotation mark	?		No conversion
″	2033	Double prime	″	2032 2032	Prime + Prime	?		No conversion
′	2032	Prime	′	2032	Prime	‘	0027	Apostrophe
`	2035	Reverse prime	`	2035	Reverse prime	`	0060	Grave accent
‹	2039	Single left-pointing angle quotation mark	‹	2039	Single left-pointing angle quotation mark	<	003C	Less-than sign
«	00AB	Left-pointing double angle quotation mark	«	00AB	Left-pointing double angle quotation mark	<	003C	Less-than sign
—	2014	Em-dash	—	2014	Em-dash	–	002D	Hyphen-minus
	203C	Double exclamation mark	!!	0021 0021	Exclamation mark + Exclamation mark	?		No conversion

There are some interesting differences here.

Some characters fail to convert to ASCII outright. This is not unexpected for the Japanese characters, is mildly unexpected for the look-alikes in the Cyrillic and Greek alphabets, and is surprising for some characters like double prime, double exclamation point, enclosed alphanumerics, and vulgar fractions because they had ASCII decompositions in Normalization Form KD, but converting directly into ASCII refused to use them.

But the dotless i gets its dot back.

Another weird thing you might notice is that the æ converts to just the a. This goes contrary to the expectations of American English, because words which historically use the æ and œ are largely respelled in American English to use just the e. (Encyclopædia → encyclopedia, fœtus → fetus.) Mysteries abound.

If your real goal is to map every character to its nearest ASCII look-alike, then all these code page games are just beating around the bush. The way to go is to use the Unicode Confusables database. There is a huge data file and instructions on how to use it. There's also a nice Web site that lets you explore the confusables database interactively.

Or you could just take the sledgehammer approach: If there are a significant number of characters outside the Latin alphabet and punctuation and you are expecting English text, then just reject it as likely spam.

ಠ_ಠ

Raymond Chen

Follow