Further adventures in trying to guess what encoding a file is in

devblogs.microsoft.com/oldnewthing/20150223-00

February 23, 2015

Raymond Chen

The `IsTextUnicode` function tries to guess the encoding of a block of memory purporting to contain text, but it can only say "Looks like Unicode" or "Doesn't look like Unicode", and there some notorious examples of where it guesses wrong.

A more flexible alternative is `IMultiLanguage2::DetectCodepageInIStream` and its buffer-based equivalent `IMultiLanguage2::DetectInputCodepage`. Not only can these methods detect a much larger range of code pages, they also can report multiple code pages, each with a corresponding confidence level.

Here's a Little Program that takes the function out for a spin. (Remember, Little Programs do little to no error checking.)

```
#define UNICODE
#define _UNICODE
#include <windows.h>
#include <shlwapi.h>
#include <ole2.h>
#include <mlang.h>
#include <shlwapi.h>
#include <atlbase.h>
#include <stdio.h>

bool IsHtmlFile(PCWSTR pszFile)
{
 PCWSTR pszExtension = PathFindExtensionW(pszFile);
 return
  CompareStringOrdinal(pszExtension, -1,
                       L".htm", -1, TRUE) == CSTR_EQUAL ||
  CompareStringOrdinal(pszExtension, -1,
                        L".html", -1, TRUE) == CSTR_EQUAL;
}

int __cdecl wmain(int argc, wchar_t **argv)
{
 if (argc < 2) return 0;
 CCoInitialize init;
 CComPtr<IStream> spstm;
 SHCreateStreamOnFileEx(argv[1], STGM_READ, 0, FALSE, nullptr, &spstm);

 CComPtr<IMultiLanguage2> spml;
 CoCreateInstance(CLSID_CMultiLanguage, NULL,
     CLSCTX_ALL, IID_PPV_ARGS(&spml));

 DetectEncodingInfo info[10];
 INT cInfo = ARRAYSIZE(info);

 DWORD dwFlag = IsHtmlFile(argv[1]) ? MLDETECTCP_HTML
                                    : MLDETECTCP_NONE;
 HRESULT hr = spml->DetectCodepageInIStream(
     dwFlag, 0, spstm, info, &cInfo);
 if (hr == S_OK) {
  for (int i = 0; i < cInfo; i++) {
   wprintf(L"info[%d].nLangID = %d\n", i, info[i].nLangID);
   wprintf(L"info[%d].nCodePage = %d\n", i, info[i].nCodePage);
   wprintf(L"info[%d].nDocPercent = %d\n", i, info[i].nDocPercent);
   wprintf(L"info[%d].nConfidence = %d\n", i, info[i].nConfidence);
  }
 } else {
  wprintf(L"Cannot determine the encoding (error: 0x%08x)\n", hr);
 }
 return 0;
}
```

Run the program with a file name as the command line argument, and the program will report all the detected code pages.

One thing that may not be obvious is that the program passes the `MLDETECTCP_HTML` flag if the file extension is `.htm` or `.html`. That is a hint to the detector that it shouldn't get faked out by text like `<body>` and think it found an English word.

Here's the output of the program when run on its own source code:

```
info[0].nLangID = 9
info[0].nCodePage = 20127
info[0].nDocPercent = 100
info[0].nConfidence = 83
info[1].nLangID = -1
info[1].nCodePage = 65001
info[1].nDocPercent = -1
info[1].nConfidence = -1
```

This says that its first guess is that the text is in language 9, which is `LANG_ENGLISH`, code page 20127, which is US-ASCII, That text occupies 100% of the file, and the confidence level is 83.

The second guess is that the text is in code page 65001, which is UTF-8, but the confidence level for that is low.

The language-guessing part of the function is not very sophisticated. For a higher-quality algorithm for guessing what language some text is in, use Extended Linguistic Services. I won't bother writing a sample application because MSDN already contains one.

Raymond Chen

Follow