

Re: Unicode and ASCII code pages confusion on user interfaces

Source:

<http://www.tech-archive.net/Archive/Win2000/microsoft.public.win2000.developer/2004-06/0056.html>

From: Richard Chambers (*john DOE_at_hotmail.com*)

Date: 06/29/04

Date: Mon, 28 Jun 2004 22:55:02 -0400

I'm working with a team that is finishing a multi-lingual application built with MS Visual C++ 6.0 for Windows XP Embedded which contains as part of the package a configuration utility which was designed to run under Windows 9x/NT.

I haven't done any work in this area lately so the writeup below is a bit rough and some details may not be right but hopefully this will point you in a good direction.

The first thing to remember about multi-lingual Windows applications is that Windows 9x does not have the UNICODE API (except for a couple of guaranteed functions like MessageBox) whereas Windows NT does. Since Windows XP Embedded is of the NT family and supports native UNICODE from the ground up, we just turned on UNICODE with a define and used TCHAR everywhere and it worked out fine.

The problem was with the configuration utility which was ported from Windows 3.x (16 bit Windows) to Win 9x/NT (and was not an MFC application so everything was direct Windows API functions) and which used ANSI rather than UNICODE since we wanted it to run on both types of Windows and the utility had to create its configuration information with UNICODE strings.

To make a long story short, we did the following with the utility:

- used WCHAR internally everywhere supporting UNICODE inside
- compiled without UNICODE turned on
- wrote our own UNICODE layer for the Windows API which does the following:

- . checks if running on Windows NT and if so, uses the UNICODE API
- . if running on Windows 9x, do conversion to multibyte then use ANSI API

I believe that one of the problems we ran into when we tried to use just the ANSI API regardless of Windows version was that when running under Windows NT, the multibyte strings were converted to UNICODE using the Windows system settings so the resulting output was wrong.

microsoft.public.win2000.developer: Re: Unicode and ASCII code pages confusion on user interfaces

We decided we had to write our own UNICODE layer because we were supporting multiple languages in a locale independent manner. In other words, someone in France may be using the utility to setup a Cyrillic or Greek system so even though they were in France using French Windows, we had to change the keyboard and display to the appropriate language.

We did pick and chose API functions so as to minimize the time spend making the UNICODE layer.

The real bear of an example would be if a South Korean was configuring a system destined for China and was using Simplified Chinese on a Korean Windows PC.

We looked at the Microsoft Unicode Layer and decided it would not do because of the system code page dependency. The Unicode Layer API will do conversions between UNICODE and multibyte to then use the ANSI API functions but only with the Windows system settings and not with an arbitrary programatically specified code page.

We could not figure out any other way to run on Windows 9x than by doing our own UNICODE to multibyte conversion, specifying the target code page as a part of the conversion call, then doing the API call with the multibyte string.

Now since the conversion utility has a multi-lingual interface, we are using resource DLLs one per supported language. When the user selects a specific language, the DLL with the resources for that language is loaded in, the appropriate internal code page is set, the appropriate font family is set, and the appropriate keyboard is selected if available. With Simplified Chinese, we use the Microsoft IME (Pinyang I think for Simplified Chinese) which is an Active-X control for Windows 9x but is built into the OS for Windows NT if the user has added that keyboard from the NT CD. If I remember correctly, the UNICODE characters were available from the IME control for Windows 9x and of course provided via standard keyboard messages from Windows NT as UNICODE.

When we popup dialogs, we send all the controls a WM_SETFONT message to set the font. You can usually tell if you missed one because you'll see the square black rectangles indicating an unsupported UNICODE character if you're displaying Simplified Chinese with a font that doesn't contain those glyphs.

Since resource strings are compiled into the resource file as UNICODE strings regardless of the Windows version, the UNICODE strings were available in the resources so all we had to do was load them in and use them internally as UNICODE since everything was kept as UNICODE until actual I/O was done.

But then we found that the LoadStringA () function does UNICODE to multibyte conversion using the system code page so we had to write our own function to do a loadstring. Found an example from a write up in MSDN about that.

Re: Unicode and ASCII code pages confusion on user interfaces

microsoft.public.win2000.developer: Re: Unicode and ASCII code pages confusion on user interfaces

So it all worked about a year ago but we haven't tested it lately on Windows 9x so maybe I should get off this news group and go check it.

Of course it may not matter much anymore since most people who would be using the utility have probably migrated to Windows XP and the rest should go ahead and get a copy anyway.

I suggest you spend some time with MSDN browsing about and get a copy of Petzold's Windows Programming book which has some info about UNICODE.