

The Microsoft Layer for Unicode on Windows 95/98/Me Systems

Michael Kaplan
President and Lead Developer
Trigeminal Software, Inc.

Cathy Wissink
Program Manager, Windows Globalization
Microsoft Corporation

Introduction

With the July 2001 Platform SDK (coinciding with the release of Windows XP's RC1 drop), Microsoft released a component known as *The Microsoft Layer for Unicode on Windows 95/98/Me Systems* (MSLU for short), whose stated purpose (according to the Platform SDK documentation) is:

[to provide] a complete set of Unicode functions on Microsoft Windows® 95, Windows 98, and Windows Millennium Edition (Windows Me). With this, Unicode applications can run on Microsoft Windows NT®, Windows 2000, Windows XP, and Windows 95/98/Me.¹

However, this layer provides more than just this very useful functionality to people involved with Unicode and multilingual applications: it also underscores Microsoft's continued future support of Unicode.

This paper will briefly explain what this layer offers the development community, how MSLU fits in with the other Microsoft technological solutions, and finally, describes what the importance of Unicode for Microsoft is and how MSLU helps support the standard.²

Motivation for MSLU

Many of the questions people ask about MSLU relate to the purpose of developing this layer so long after the delivery of platforms for which the layer was created. A full discussion on the importance of Unicode in Microsoft

¹About the Microsoft Layer for Unicode, July 2001 Platform SDK

² For a more comprehensive look at MSLU, please see our article in MSDN Magazine (<http://msdn.microsoft.com/msdnmag/issues/01/10/MSLU/MSLU.asp>).

platforms can be found in Cathy Wissink's paper Unicode and Windows XP (in these proceedings), but to briefly summarize the situation:

- NT platforms (including Windows 2000 and the new Windows XP) are underlyingly Unicode;
- Windows 95/98 and Me are all based on code pages.

As a result of this dichotomy between the Unicode based NT platform and the code page based Win9x platform, developers have been hamstrung by their need to stay compatible with the Win9x platform and its non-Unicode APIs. It became crucial for Microsoft to try to help people write applications for both platforms that would provide appropriate multilingual support through support of Unicode.

Other Microsoft products

This issue applied to MS-internal customers as well. Many Microsoft products released over the last few years have migrated to Unicode, namely:

- Microsoft SQL Server
- Microsoft Word
- Microsoft Excel
- Microsoft Access
- Microsoft FrontPage
- Microsoft Visual Studio.Net

This list continues to grow. Since these products were all supported on Win9x, there had to be a way to support a Unicode application on these platforms, and each of the product groups had to individually determine how they would add that support. Although the development process of such a Win9x translation layer had been outlined in an MSJ article³, the method described in the article did require the developer to do all of the work to provide a wrapper layer over the Win32 APIs needed by the application; in other words, there was no tool which internal developers could leverage. As a result, at least 32 different partial layers of this type were written over the years among the many application groups, in an effort to make products available in a platform-agnostic manner to all users. Note that in many cases, functionality was limited on Win9x, but the translation work was done so that the product groups could create a single binary for all platforms. Unfortunately, each layer was only partially implemented and developers tend to dislike writing the same code over and over again; this made

³ Bishop, F. Avery, "Design a Single Unicode App that Runs on Both Windows 98 and Windows 2000." Microsoft Systems Journal, April 1999.

the myriad of wrapper layers an imperfect solution at best. When you consider that external developers were not assisted at all, it became clear that *something* had to be done; a layer that both external and internal customers could use to leverage Unicode APIs needed to be written.

Migration

New platforms, no matter how compelling the features may be, are only relevant to those users who do, in fact, migrate to those platforms. The issue of what developers must do when some or even most of their users have not yet embraced Windows 2000 or Windows XP is one that is acutely felt, both inside and outside of Microsoft. Developers have many choices:

- *Force everyone to upgrade:* By only supporting a product on specific platforms, a developer can design a Unicode application by only allowing it to run on Unicode platforms. The downside of this approach is that you will probably have fewer customers.
- *Create two versions of the application:* By creating both "A" ("ANSI", or code page-based) and "W" ("wide", or Unicode) versions of an application, Unicode support can be leveraged whenever the appropriate version of the application is run on the correct operating system. The downside of this approach is that it requires much more work to write code that will compile differently, depending on UNICODE and _UNICODE defines in the code.
- *Provide a Unicode version for all platforms:* A Unicode version of the application can be created to run on all platforms. The downside of this approach is how much work it takes to provide Unicode support on all platforms when so many of the Win9x operating system's features do not offer a way to provide this support.

The last option is the best from a feature standpoint, since it offers consistent functionality across multiple platforms, but it is also the most challenging. To support a Unicode version of an application across multiple platforms, there need to be translation layers that can be used over any non-Unicode element, and this is where MSLU fits in. Rather than requiring developers to do all of the work themselves, a layer (MSLU) is provided that allows Unicode applications to be written on all platforms.

MSLU, however, is not the only Microsoft technology that offers Unicode support across both Windows NT and Windows 9x platforms. The following are technologies that provide some form of Unicode support (either full or in part) on Win9x systems:

The Microsoft Layer for Unicode on Windows 95/98/Me Systems

- *The Windows Common Controls*: Starting with version 5.80, this helpful library has full support for Unicode, including many commonly used user interface elements such as the TreeView and the ListView. This version can either be automatically installed with Internet Explorer 5.0 (or later) or can be provided by a separate redistributable download associated with the Platform SDK.
- *Uniscribe*: Starting with Windows 2000, Uniscribe provides a powerful means for supporting complex scripts⁴. It can be installed on NT 4.0 and Win9x machines by installing Internet Explorer (5.0 or later).
- *RichEdit*: The RichEdit control provides one of the most powerful interfaces available to developers for rich edit display. All versions after 2.0 support Unicode text (version 3.0 provides significant bidirectional and other complex script support).
- *MLang*: The MultiLanguage (MLang) object provides a rich set of APIs that assist with Internet software internationalization. It is available to all platforms by installing Internet Explorer (5.0 or later).
- *TSF*: The Windows Text Services Framework (TSF) enables advanced, source-independent text input. Expanding on earlier components like the Global IME, it provides a rich means of multilingual support, even on Windows 98 and Windows Me platforms⁵.
- *WinForms*: The forms package for Visual Studio.Net provides full Unicode support on all platforms, including Windows 98 and Windows Me⁶.
- *The .NET Common Language Runtime (CLR)*: Both the CLR's System.Globalization and System.Text namespaces provide valuable support for both NLS features and text encoding/conversion in a platform-independent manner; like WinForms, it is available on both Windows 98 and Windows Me⁷.

As a result of these technologies, there already was a precedent for Unicode support on downlevel (i.e., Win9x) platforms. However, one of the biggest stumbling blocks to development of Unicode on Win9x was the lack of support for Unicode APIs. Although it was not intended, this lack of support could often be seen as a way of undermining the standard, and Windows XP is the first

⁴ A complex script is a writing system that needs additional processing in order to be displayed correctly. Some scripts considered complex are Thai (which has complex word-breaking algorithms and a filter for illegal character combinations), Devanagari (which needs reordering of glyphs), and Arabic (which has contextual shaping of glyphs).

⁵ For more information, see Kevin Gjerstad's paper in the IUC 19 proceedings: [Windows Text Services Framework](#). (September 2001)

⁶ For more information, see Achim Ruopp's paper [Building International Applications with Visual Studio.NET](#) and François Liger's papers [Enabling Globalization with Microsoft .NET Framework](#) and [Building Localized Applications with Microsoft .NET Framework](#), all in these proceedings.

⁷ Again, see Achim and François' papers.

consumer-oriented version of Windows that has full Unicode support. In order to ensure that developers were leveraging Unicode fully and to also support the standard, it was crucial to provide an easy and extensible way to provide Unicode support on all platforms.

What the layer is

MSLU had many specific design criteria to meet the issues described above. Some of them were based on who was doing and planning the work (the Windows XP team), others were based on what company was doing the work (Microsoft), and still others were based on who the layer was designed for (both internal and external development teams targeting the WinNT and Win9x platforms). As a result of all these factors, the basic design criteria were to:

- Provide a superset of all APIs needed by the application-specific layers previously written by Microsoft-internal groups;
- Keep this set of APIs agnostic enough to be used by a diverse group of application developers;
- Be easy to use, by keeping the size small and not monopolizing system resources;
- Be free of major dependencies (e.g., components or registration routines) while still working properly with other components;
- Not adversely impact performance;
- Have the ability to be overridden where necessary or desired;
- Ideally not provide any new functionality; that is, be a translation layer pure and simple.

Many of the specific implementation details related to these criteria can be found in an article written by Michael Kaplan and Cathy Wissink for MSDN Magazine⁸.

What the layer is not

It is important to also define what MSLU is not, such that developer expectations are properly set regarding what it can and cannot do.

The Microsoft Layer for Unicode was designed to be a translation layer for NT-based Unicode APIs on Win9x, as noted earlier. The layer, however, is not a complete rewrite of Win9x intended to fix every bug, nor is it some type of "NT emulator" for the Win9x platform. It does not provide support for "Unicode-only" scripts like Devanagari or Georgian, or for any of the new supplementary

⁸ For a more comprehensive look at MSLU, please see our article in MSDN Magazine (<http://msdn.microsoft.com/msdnmag/issues/01/10/MSLU/MSLU.asp>).

characters that have been added to Unicode (specifically, UTF-16) as surrogate pairs.⁹ It also cannot provide extended international support beyond the platform (or default system code page) that it runs on, since it is relying on the operating system for any particular international functionality that developers may want to leverage.¹⁰

In short, it is clearly intended as a migratory aid: its purpose is and will remain to provide an easy to use and extensible method for writing Unicode applications today, without waiting until there are no more Win9x machines at all.

Beyond the initial release

Since MSLU first shipped in July 2001, the DLL has been updated several times through the Platform SDK release mechanism. These updates were mainly to fix minor bugs reported by developers who were using the DLL, but there was one major feature that was added after the original release. Originally, it was assumed that the main customers would be leveraging C/C++, and therefore using the MSLU loader; they would never need to call the DLL on the NT platform. On other languages such as Visual Basic or C#, the onus was on the developer to make sure they never called the DLL on the wrong platform. This assumption was incorrect. Many developers decided to always call the DLL, causing many problems on NT due to assumptions MSLU made about the memory, windows, resources, and other issues. To help correct this issue for customers, some work was done to make sure that the DLL would always forward the calls to the original API if you were running on the NT platform.

The fact that this change was made due to customer need underscores one very important issue: Microsoft is paying attention to customers of this DLL (like those who post messages to the MSLU newsgroup and those who send questions to Dr. International!) and making sure that their questions are answered and that their needs are met. This component will continue to be an important one as long as Win9x is an important customer platform; it is crucial that developers have a means of successfully migrating their applications to Unicode on the Win9x platform.

Implications

Clearly, Microsoft has moved to Unicode for its future support of languages and scripts: they are generally not creating or implementing new code pages, and

⁹ For a specific definition of Unicode supplementary characters and “surrogates”, please see the Unicode glossary at <http://www.unicode.org/glossary/index.html>

¹⁰ For example, it does not provide updated versions of the cp_*.nls files that contain support for code pages, nor does it add support on Windows 95 for GetLocaleInfo LCTypes for which the OS has no specific information.

they are not producing new versions of the Win9x code base (in fact, many enhancements to Windows Me, such as USB support, were accomplished by taking features from the NT code base). The advantages to this are numerous for developers of multilingual applications, because the NT code base provides a much better platform for applications. However, this is only true if developers are actually writing Unicode applications -- those relying on the backwards-compatible support for "A" APIs will not see many of those benefits.

MSLU is not intended to be a permanent means of supporting Unicode for anyone; if it were, MSLU would consist more of actual code ported from Windows XP, rather than wrappers around Win9x functions. However, this would have been much more challenging to implement. Most of the wrapper functions simply convert text from Unicode and call the non-Unicode version of the specific API, which limits the actual support for these APIs to a specific code page, and as a result, still limits the international functionality to that available on the system code page on a Win9x platform. What it *does* provide, however, is the ability to create a single Unicode application on all platforms that will provide full multilingual support on Unicode platforms without requiring extra code to be written by the developer.

Conclusion

The eventual move to Unicode by all of Microsoft's major applications (Internet Explorer, Office, SQL Server, Visual Studio, etc.) sent a clear message about the company's support on Unicode applications and their importance for the future. However, in migrating to Unicode, most of these products developed proprietary layers that external customers could not use; as such, a migration solution was needed for the development community. The Microsoft Layer for Unicode on Windows 95/98/Me Systems (MSLU) provides the first easy way for a developer armed with nothing more than the Win32 Platform SDK to produce Unicode applications that will run on all platforms. Its release by the Windows division helps to underscore the strategic importance of Unicode to Microsoft for its own products, its external developers, and its customers.