Think Internationalization in Everyday Design
Change Your Encoding, Change Your Company
Unicode 5.0 From 50,000 Feet
Pierre Cadieux: A Career in Internationalization
New Internationalization Features of Microsoft Vista
In five years, how much has your life changed? Since we published our first Getting Started Guide for internationalization in 2002, changes in internationalization have been coming at the speed of the internet. Unicode has proceeded from version 3.0 to 5.0. The .NET platform has grown exponentially and moved into version 2.0. Perhaps half a billion more people have internet access as China and other countries get online.

But some things haven’t changed. If you want to sell products and services in other cultures, these questions still apply. Is it a good product? How adaptable is it for other languages and cultures? Can you easily change the language and script of the display or written text? Can you change culture-specific graphics to adapt to meaningful images for other cultures? Can your product handle other number, currency, date, time, address and telephone number formats?

In this Guide, we update your knowledge of internationalization — questions to ask, points to keep in mind, pitfalls to watch for, checklists for action and what resources are available. And brace yourself for the changes coming!

The Editors
Think Internationalization in Every Day Design

Alan Horvath

Enter the global marketplace can be a daunting task for any company. It can be expensive and time consuming, so proper planning and execution are critically important. It is essential to understand the complete globalization process and this important fact — internationalize first, then localize!

This article describes some of the most common challenges within the internationalization process and how you can avoid time-consuming errors.

A matter of words

So many words are used to describe the actual process of preparing products for foreign markets that it is easy to become confused. Here are a few definitions to help clarify the process.

Globalization (g11n) includes a company’s decision to enter foreign markets, internationalize its products and localize its products.

Internationalization (i18n) is the process of designing a product such that it can handle multiple languages, cultural conventions and local infrastructures without the need for re-design.

Localization (l10n) is the process of physically, culturally and/or linguistically adapting a product for a target locale.

Translation is the process of converting words from one language to the other. An experienced translator will be able to convey the technical details accurately while instilling native nuance and style in the translated text.

Software internationalization

Let’s say you’ve written a software application for sale in the United States. Throughout your code, you’ve hard-coded terms such as dollars and cents. You’ve also hard-coded symbols such as $ and used a period (.) for the decimal point and a comma (,) to separate numbers into groups of three. Now your company wants to market your software abroad, but the terms and symbols you used are inappropriate for other currencies.

There are two ways to prepare to enter foreign markets with your software product. The first approach involves making separate copies of the software and replacing the terms and symbols with ones appropriate for each country.

But what happens if a bug is reported? Now you have multiple versions of code to correct! And if you decide to add a new feature to your product, you’ll have to make changes to all versions.

For companies that intend to enter multiple markets, this method is out of the question. It underestimates the time needed to modify files that were never meant to be localized.

And there are other issues to consider. Does your code support Unicode so that you can handle a variety of scripts as easily as your native language? Does your code correctly search and sort characters in all of the languages you expect to use? These issues and many others must be addressed before entering international markets.

The second and preferred approach consists of internationalizing your software for the global market first and then proceeding with a more streamlined localization process.

Therefore, before you begin translation of your product's interface, text and files, you need to ensure that the product is internationally ready.

Ten tips for software internationalization

Software is a component that requires a great deal of attention. Once you have made the decision to go global, every design decision — either with the code or the user interface — will be affected.

1) Use Unicode functions and methods.
2) Third-party tools. Choose your tools carefully. Some tools, APIs and add-ins might not support Unicode. If you must use them, use the right character conversions.
3) User Interface separation. Isolate your translatable resources. Hard-coded strings are very tricky to translate, and, since the code is constantly changing, they cannot be translated in parallel with the software code development. Maintain one core code base for all of the languages.
4) Concatenation. String concatenation is another source of problems for localization. Word orders and plural forms might be different in other languages. Adjectives might not agree in gender, number and case, thus creating catastrophic results.
5) Ambiguity. Keep translators in mind when writing the resource files. Add comments when a sentence could be ambiguous. It will improve the quality as well as reduce the amount of time needed for translation.
6) Expansion. Keep in mind that translated strings may expand compared to the source-language string. This will be particularly important with dialog boxes and menus. Leave at least 20% to 30% expansion room.
7) Design. Be careful when you create icons and bitmaps. You should avoid text in either because the translated words might not fit, and the cost of conversion can be expensive. Avoid any symbols with cultural connotations. They might be indecipherable or offensive in other countries. If required, make sure your product runs on different platforms — PC, Mac, Linux and so on.
8) Terminology. Check for terminology consistency. If you are not consistent in your software, the rest of the package will be even more inconsistent. Terminology management tools will help keep terminology consistent.
9) Locale testing. Check your code in the destination market to ensure that all locale issues are handled correctly.
10) Translation kit. Once your product is ready for translation, you should create a translation kit. This set of files should contain everything needed to translate and recompile the language resources as well as test the application.

You should also consider legal issues that may arise as you enter different markets. For example, there have been instances where software companies had their terms and agreements embedded in the product, but some of the terms were determined to be illegal in different jurisdictions. In many cases, the company’s
attorneys were required to define new clauses for certain parts of their contracts, therefore, you should consider the legal circumstances in all target markets before beginning the translation process. 

Also note, software containing encryption technology can be subject to export restriction, while communication software may be subject to telecommunication regulation in the target country.

Multibyte character support

Delivering translated products to Asian countries for the first time can be an exciting and challenging time for organizations, but getting it wrong can be very costly.

English software typically uses about 100 different characters to represent words and numbers. Asian languages, on the other hand, can use more than 10,000 symbols to display messages.

To facilitate this, software systems used to use what is termed multibyte or double-byte character systems (DBCS) to store text/characters. Today, Unicode has replaced these systems, but be aware that occasionally you may need to provide multibyte character support.

A common error in software development is to use third-party software — software libraries, DLL, OCX and so on — that are not Unicode enabled. Sometimes the development teams do not realize this until they begin the translation process. As a result, the product then has to be re-engineered to fix the issues at additional expense.

Documentation internationalization

Documentation internationalization does not require as much planning as does software internationalization. Some simple guidelines can help you prepare documents that can be easily localized.

Desktop publishing software. With so many choices available, it can be difficult to choose the appropriate desktop publishing application. Your choice will be affected by the type of documents you are creating, the languages that you are targeting and whether or not translation tools will be used. You will also have to decide if the documents will be printed, electronically published or both.

If you are designing marketing documents, applications such as FrameMaker, InDesign and QuarkXPress produce good results. The major drawback is that the process needed to translate these documents is more complex. Leading TM tools do an excellent job of handling these file types. In addition, if you plan on targeting Asian countries, you might have trouble finding a vendor with in-house expertise using the desktop publishing tools you selected due to the high price of the Asian software versions and the software’s complexity.

Nevertheless, if you decide to use these applications, you should keep the following guidelines in mind:

- Avoid creating unlinked text boxes throughout the document. Some files that we have processed in the past contained hundreds of unconnected stories, and processing the files with some translation tools becomes very cumbersome. Instead, flow your text from one box to the other. When the file is translated, the expanded text will move from one box to the other automatically, even from one page to the other.
- Leave plenty of white space (20% to 30%) in the pages to allow for expansion. It is a good rule for all types of documentation, but particularly important with marketing material because the number of pages for such documents is usually fixed.
- Be careful when selecting screenshots for your marketing material. The component that you choose might be the last piece translated, and the localized screenshot might not be available at the time you go to print.
- Make sure that any in-line graphics are anchored to the surrounding text so that they can move when a paragraph goes, for example, from one page to another. If you are designing long technical documents, you have more choices. You can still use applications such as InDesign or QuarkXPress, but it is not recommended because they lack the large-document handling features of applications such as FrameMaker or Word for Windows.

Graphics. Remember that the outsourcing of graphics creation and translation is expensive, particularly in Asian languages. However, careful planning and clever design of your graphics can eliminate a big part of your graphic localization costs. Removing the text from the actual graphics can reduce your costs by more than 95%.

Remove all localizable callout text from graphics and include it in the documentation’s text so that it can be added to a translation memory for re-use. Replace the callout text with numbered (not lettered) callouts, arranged clockwise on graphics, and cross-reference the numbers to the text into the main document. This allows automatic re-use of the same graphics in all localized versions.

Electronic vs. printed publishing. The obvious reason for electronic publishing is cost. It is much cheaper to create, package and ship a CD/DVD than a box of 15 manuals. The electronic book can always be printed if needed. Nevertheless, parts of your document set, such as a “Setup Guide” or an “Administrator’s Guide,” should still be provided in print format.

One-to-one page correspondence. This is an option that is rarely used but might make sense for your organization. Its main objective is to streamline the support operation. The example used is that someone in a foreign country has a problem with your application. First-tier support in that country cannot solve the problem and needs assistance from the next level. If you maintain a one-to-one page correspondence, everyone can be looking at the solution on the same page of your document. This approach also allows you to create standardized packaging materials since all your manuals will have the same dimensions. This option, however, requires more planning and may require you to sacrifice aesthetics.

An alternate approach to document internationalization. Consider using an information management system as an alternative to the traditional approach to document creation and management. In these systems, information can be entered simultaneously from any number of locations in any number of languages. Information is entered and stored in a pure and intelligent way, with no regard for final layout. The information exists only once and can be used to produce any number of publications in virtually any form — such as Help, HTML, DOC, XML — and in any language. By the very nature of this approach, information that is created and managed in this way is already “internationalized.” The result is an overall reduction in translation/localization costs with a simultaneous increase in quality and consistency.

Summary

Product internationalization is the most important step in the globalization process. Products must be designed to handle multiple languages, cultural conventions and local infrastructures without the need for re-design. Incorporating these guidelines into your internationalization strategy will save you time and money and go a long way in streamlining an otherwise complex process.
It’s a mark of greatness when a company can effectively develop products and compete worldwide. Yet software internationalization is often one of life’s painful forgotten labors that suddenly grabs intense and panicked attention as it leaks out and grinds globalization plans to a halt. You’d think that enabling technology so that it’s easily leveraged for any market opportunity would be a pretty glamorous and exciting pursuit. With rare exception, the first, second or twentieth-plus time a company does this is still a painful effort that holds back global top-line revenue opportunities.

Of course, it doesn’t have to be this way—but when you look at the nature of how software is actually developed and comes to market, unless internationalization is a very firm requirement at the project’s outset, it shouldn’t be a surprise that it gets overlooked until it’s an ugly problem. This is not one of those “if only everyone always internationalized” diatribes. I hope to describe the business issues around internationalization, including the fundamentals of what it does for a company, the competitive implications, funding the effort, and managing and maintaining global market requirements.

One point I want to get beyond quickly is the belief that you can just force your translations without internationalizing software first. I get asked about this a few times per month, often by managers who are even in the localization business. Incidentally, developers never ask this.

In the case of some limited products, it may be possible not to internationalize, but it’s a bad idea anyway. You risk having a product that doesn’t work or works poorly. In the best scenario, your software can’t be leveraged across markets or even maintained from release to release. Not internationalizing is like throwing lots of money and resources away for an inferior result which has no future. For complex applications, it’s simply not going to work. Developers nearly always accept this, but it’s an abstract concept which management can have trouble understanding. Internationalization, when done well, allows you to support any locale requirement quickly. You have one product to support over time that’s good for the whole world. Your translations are easily updated from release to release.

Business issues

Several common events push a company to expand its product development to include locale supporting requirements.

1. Somebody sold something — there has been some new marketing partnership or a new powerful customer opportunity that requires multi-locale support. A classic example is that the company gains a business contract that will necessitate supporting Japanese or another language. In some cases we’ve seen new license deals for entire countries, such as in health care or education. It’s a big hurry up to meet the customer demands.

2. Localization is realized as a competitive necessity. Perhaps the company has already invested in global sales efforts and finds growth is limited given a poor competitive position without internationalization.

3. A global company has just purchased another company or intellectual property and wants to make the new product useful for its worldwide sales efforts and product line.

4. The CEO is mandating a new global initiative. This is an important new step for the company’s evolution. You can’t go to a management conference these days without hearing about globalizing revenue opportunities and for good reason.

Top-line and bottom-line considerations

Software internationalization can have dramatic effects on top-line revenues and revenue plans as well as on bottom-line costs and profitability. It’s never just about minimizing a cost. You have to look at the whole picture to calculate return on investment and in terms of long-term changes in process.

When valuing any internationalization effort, give attention to top-line business questions such as these:
- How much does your company have riding on success in its target markets?
- What are the revenue projections over one, two and more years?
- What is the top-line cost of not having a product ready for a specific market opportunity?
- What is the impact to your company’s strategic partners or sales force if a product doesn’t work well or isn’t ready for a particular market?

- What is the result for your company’s equity value by stretching into new markets effectively?

And, of course, to get an effort funded, these bottom-line business issues need to be answered:
- What will it cost?
- How long will it take?
- Who is going to do the work?
- Do we have to give up other feature requests to prioritize internationalization?
- How can we improve the process?
- What expensive surprises do we need to watch for?
- How do we maintain the internationalized product going forward?

Learn your CFO’s language. He or she will want to understand the return on investment and may consider amortizing the effort as a capital expense. The decision isn’t about the technical issues of bits and bytes.

Globalization is never just about one customer, sale or “language.” It’s a new engineering and company process that opens opportunities.

Development issues

At Lingoport, we’ve picked up the pieces enough to see that internationalization projects have been typically frustratingly late, as in quarters to years, and rife with cost overruns.

Managing an internationalization effort, especially for the first time, can be challenging for a development team. Given the top-line sales and marketing objectives, there’s typically a shortage of time. Compounding this is that understanding the scope of requirements and detail of tasks for an internationalization effort is not an obvious thing for your development team. It’s tempting for many teams to just start out looking at embedded strings as the most obvious problem. While they are important and can be tedious, there’s much more involved. The issue of who does the work isn’t always obvious or easy. Chances are your development team members aren’t sitting on their hands looking for something to do. You’ll need to balance internationalization demands with new feature development, too.

April/May 2007 • www.multilingual.com/gsg
Building requirements starts with identifying target locale requirements. The obvious issue is language, but there’s more to it regarding culturally sensitive formatting of issues such as dates, times, numerical values, addresses and more. Changes to your database tend to have far-reaching effects into your application. Changes to programming logic or the graphical user interface further complicate things. As you might imagine, clearly built requirements will be an important pivot for all your efforts. Make sure that whoever is working on this isn’t doing it for the first or second time. There are so many pitfalls that even globalization architects who have led similar efforts many times are still learning.

For some companies, internationalization can be performed in stages. For example, it can start with supporting storage, retrieval and processing of customer data for Unicode or ISO-Latin encoding. A second possibility might be more completely internationalizing but limiting the team to addressing Western European locale requirements. Others may require a full Unicode enablement to support “double-byte” locales such as Japan, China and Korea. You can separate the ideal from the practical if need be and consider optimizing business decisions regarding budget, technologies, long-time plans and competitive market needs.

Figuring out the scope of an internationalization effort, assigning resources and planning expenses can be challenging. Architectural changes, third-party product issues such as graphics and reporting tools, installers, databases and more must be accounted for. You have to find and fix internationalization issues buried within your hundreds of thousands to millions of lines of code. Without a strong detection, extraction and refactoring tool, this alone has the potential of being an error-prone and time-consuming iterative process. Embedded strings must be quickly and easily distinguished and filtered from programmatic elements such as debug statements or SQL queries. Externalization should be automated to avoid further human error potential. Every programming language has its unique locale-limiting methods and functions as well as character encoding issues. As simple as HTML (including JSP, ASP, ASPX and so on) may be as a language, it takes some sophisticated programmatic language to comb through it. C++ has hundreds of locale-limiting issues that are highly dependent on the target encoding and supported operating systems. Even Java and C# don’t internationalize themselves, though they were built to be considerably more internationalization friendly than most other languages. You also have to effectively distribute the knowledge of internationalization complexity to your development team. Our team created a tool to help analyze source and cut time and expenses. We now offer it as a standalone product and adapt it for scalable use among large development teams.

Any tool won’t help you find what’s not in your code. You still have to be savvy with your architecture, with a long-term eye towards your product life cycle.

Localization, testing and beyond

Unless your company is only internationalizing to support managing multi-locale customer data but not localizing the database, you’ll likely be interested in when the role of localization comes in. It’s quite reasonable to dovetail string extraction efforts with your localization vendor so that your localized releases aren’t dependent on first completing the entire internationalization effort. For initial testing you can use pseudo-localization. To do this, add new characters from your target locales to either side of a string, expanding the string as needed. This lets you make sure that your product supports extended characters, resizing and the like, without having to wait for localization testing or needing to have your tester speak the target language. You’ll want to use pseudo-localization for the interface, as well as passing data and locale-formatted variables through your application’s database and functions. Once you’ve received the translations from your localization company, you’ll need to perform linguistic testing as well. Expect that some translations may need to be adjusted — they may be technically correct but not the best choice given the specifics of your interface, product domain or word usage.

Finally, you need to create a sustained plan for systematically auditing new code development, making sure that it doesn’t break new internationalization requirements over the years. Make sure you have strong documentation on your internationalization architecture and procedures. That way you have a legacy that can be clearly followed over the years.

Through it all, I can’t overestimate the need to communicate. Most development efforts fail due to lack of clear requirements and ongoing communication. You’re going to have to blend internationalization objectives with new features. That means a clear development path, source control practices, testing processes, education, tools and cooperation among developers. You’ll have a whole world of new clients and worldwide stakeholders to support. And that fundamentally changes a company with the opportunity to further make it great.
By now, there’s probably no one reading this magazine who hasn’t at least heard of Unicode. In its 15-year history, Unicode has become the character encoding standard of choice in new applications. It’s the default encoding of HTML and XML; it’s the fundamental character type in programming languages such as Java, C# and Javascript; and it’s the internal character encoding in the Windows and Macintosh operating systems. Virtually all Unix flavors include support for it, too. Unicode is to computing in the twenty-first century what the American Standard Code for Information Interchange (ASCII) was to computing in the twentieth century.

If you’re just getting into software internationalization, Unicode is something you want to know about. It can make your life much easier, but it’s important to keep in mind just what Unicode is and isn’t. Just what it means to say you’re Unicode-based or Unicode-compatible can be rather squishy and is highly dependent on just what your application does. More importantly, it’s important to keep in mind that supporting Unicode is neither necessary nor sufficient to writing an internationalized program. Unicode and internationalization are related, but very different concepts. Unicode makes it easier to write internationalized programs, but you can write them without using Unicode. And you can very easily write Unicode-based programs that still aren’t internationalized.

Many articles in this guide will help you get up to speed on just what it means to write an internationalized program. The purpose of this article is to help you get up to speed on just what it means to support Unicode and which problems it does and doesn’t solve.

At first glance, Unicode can be quite an intimidating beast. The latest version — Version 5.0 — sprawls across a 1,400-page book and a CD full of appendices, character property databases, and other supplemental material and comprises nearly 100,000 character assignments.

This article can’t possibly cover all that, so what we’ll try to do is take the proverbial “50,000-foot view” of what Unicode is and what problems it solves. To go further, there are several good “introduction to Unicode” resources and a useful “cheat sheet,” and the standard itself is actually quite accessibly written.

Character encoding standards

Unicode is a character encoding standard. Computers don’t have any innate knowledge of text or characters or images or sounds; all computers really understand are numbers. To represent text in software, you adopt a convention where each character you need to represent is given a number. You decide that in your application, any time you see, say, the number 1 in a memory location you know is supposed to hold text, you interpret it as the letter A. When you see the number 2, it’s B and so on. Sequences of these numbers represent sequences of characters.

Further, text is so common that rather than having each developer adopt his or her own convention for representing text with numbers, the industry issues standards, official documents that define conventions for assigning numbers to characters. If two applications follow the same standard for representing text, they can pass text back and forth between each other, and they’ll both be able to do things with it properly.

The problem, of course, is that there are so many different standards. Most modern computing systems use the ASCII or something based on it. ASCII was published in the 1960s by what is now the American National Standards Institute (ANSI) and uses the values from 32 to 126 to represent the 26 uppercase and lowercase letters of the English alphabet, the 10 digits, and various punctuation marks and symbols. The values from 0 to 31 and the value 127 were reserved for various control signals, and byte values from 128 to 255 weren’t used.

However, ASCII only includes codes for the letters in the English alphabet. Speakers of other languages don’t have codes for the letters of their alphabets. Even other languages that use the Latin alphabet, such as French, are missing codes for the accented versions of the letters that they use. Since the byte values from 128 to 255 weren’t standardized by ASCII, computer vendors, national governments and other bodies came up with other standards that used these code values for the letters of other alphabets.

Now there’s a plethora of character encoding standards out there, each of which defines code values for a single language or a small group of related languages. There are several problems with this: 1) The standards are mutually incompatible. While you can usually count on the value 65 representing the capital letter A, the value 215 can represent lots of different characters, depending on the encoding standard. 2) Because most legacy encoding standards only encode a small number of characters for a small number of languages, mixing languages in a single document frequently requires changing from one encoding standard to another in the middle of the document, and there are oftentimes no mechanisms in the software for doing that or for reliably interchanging such documents with other applications.
3) Often, encoded text travels across media without any external indication of the encoding standard it follows. Software receiving a message of unknown encoding has to guess or simply assume. Many times, the sending software intends for a particular numeric value to represent some character, and the receiving software interprets it as something totally different, thus leading to garbage. If you’ve ever received an e-mail message with strange characters where you expect dashes or quotation marks to be, you’ve seen this problem in action.

Unicode was designed to solve these problems. The idea was to use a larger data type than a byte for each character and then give every character in every language its own unique numeric representation. This means you can mix languages freely in a document without the software having to worry about mixed encodings, and you can send text from one system to another without worrying about it getting mangled on the other end — as long as the sending and receiving systems both support Unicode.

It should be clear that Unicode doesn’t solve all your internationalization problems. You still have to translate the text. You still have to remember to call number-formatting and date-formatting routines that can produce different output for users of different languages. All Unicode does is make it possible to represent text in many different languages without having to keep track of the encoding or deal with data loss in interchange.

WHAT UNICODE DOES

Unicode is unique among character encoding standards in the sheer number of characters to which it assigns numbers — nearly 100,000 in the most recent version. Those 100,000 character assignments cover all of the characters in all of the writing systems for all the languages in common business use today, as well as the characters needed for many minority languages and obsolete writing systems, and a whole host of mathematical, scientific and technical symbols. Whatever the character you need, the chances are overwhelming that Unicode has it. And, if it doesn’t, no other encoding standard in reasonably wide use is going to have it either. This comprehensiveness makes it possible to represent text in any Unicode-encoded language or combination of languages without having to worry about specifying which character encoding standard your application or document is following and without having to be concerned about changing that encoding standard in the middle of your document or going without characters because you can’t change encodings.

More importantly, Unicode is unique in approaching the business of assigning numbers to characters with far more rigor than any other encoding standard has attempted. The further away from the Latin alphabet you get, the less clear-cut using numbers to represent text becomes. In many writing systems, the letters don’t march in a nice orderly fashion from the left-hand side of the page to the right. In some, they go from right to left. In some, they knot together in complex ways. In some, they’re adorned with various accent, tone or vowel marks that attach to the letters in many different places. Straightening this out into a one-dimensional sequence of numbers is complex, and the right answer is often ambiguous.

It’s also not always clear just when two different squiggles are the same character and when they’re different. For example, in many writing systems, the shape of a letter can change dramatically depending on the letters around it, or two letters can merge into a totally different shape when they appear together. Different encoding decisions may need to be made for different scripts, yet you still have to be able to mix them in a document and have things work sensibly. There are many characters with similar appearances, leading to potential security issues that have to be addressed. You also can’t infer much about a character from its position in the code space or its appearance in the code charts. There are too many characters for that, with more being added all the time.
Because of these and many other issues, the Unicode standard goes far beyond any other character encoding standard in describing just how those 100,000 character assignments get used together to represent real text and how software should carry out various processes on the characters. For example, since you can’t infer things from a character’s position in the encoding space, the standard includes a very large database of character properties that lays out in tremendous detail the exact meaning of a character code: Is it a letter, a digit or a punctuation mark? If it’s a letter, is it uppercase or lowercase? Which character is its partner in the opposite case? If the character is a number, which numeric value does it represent? If it’s a diacritical mark, how does it attach to its base character? Is the character part of a right-to-left writing system? Does it join cursorily to other characters? What other characters, both in Unicode and in other standards, is it equivalent to? How does it sort when compared to other characters? And so on and so on.

Unicode also includes many rules on how to do different things with encoded text. For example, because there are more assigned character codes than can fit in a single 16-bit word, Unicode includes methods of representing text using sequences of 8-, 16- and 32-bit values. There are also rules and guidelines not just for how to display a sequence of character codes, but for determining when two strings are the same, locating line and word boundaries, mapping strings to equivalent representations in other encoding standards, performing regular-expression searches or language-sensitive sorts, using Unicode in programming-language identifiers and much, much more.

What Unicode means for text handling

All of these things make it possible to handle more languages and to handle more languages well than any other character encoding standard. The various rules and guidelines in the standard help with many of the processes needed in writing internationalized text, thereby making them easier or more powerful. There are many comprehensive software-internationalization packages that use Unicode as their base. Because of this, it’s frequently possible to write internationalized text without having to know all the nitty-gritty details of the Unicode standard. You can often stand on the shoulders of experts who have done most of the heavy lifting for you.

Tremendous blood, sweat and tears have gone into those 100,000 character assignments and their accompanying rules, guidelines and property databases. Unicode is not just the largest collection of characters ever encoded in a single standard. It’s the most comprehensive collection of rules, guidelines and best practices ever compiled for handling text in computer software.

You could write an internationalized application without using Unicode, but why would you?
INTERNATIONALIZATION

Pierre Cadieux: A Career in Internationalization

Nancy A. Locke

In the language industry, precise terminology — as both a means and an end — ranks high on the list of priorities. Ironically, the persistent ambiguity of the terminology that describes the processes used by the language industry also ranks high as a challenge. Buzzilicious euphemisms aside, the fuzziness of the terms globalization, localization and internationalization — perfectly serviceable words in other contexts that were then co-opted and re-purposed by the language industry — can be downright annoying. In a recent interview, Montréal-based internationalization consultant Pierre Cadieux describes how he wound up in internationalization, a field that by any measure is relatively new and, despite increasing awareness, not well understood. Not surprisingly, terminology on many levels emerges as a core issue.

Development of an internationalization engineer

When people dream of being their own bosses, a dream that often includes ditching the commute, the cube farm and “casual Fridays” to work from the casual comfort of home, they may well dream of what for Cadieux is a reality. When he is not traveling, he works from his home on a quiet, tree-lined street within walking distance of the antique shops, boutiques, restaurants and cafés in the city’s chic Plateau district. Red brick walls, hardwood floors, cathedral ceilings, exposed beams, a fireplace and, in the courtyard, a hot tub create a warm living environment. The office, a large room equipped with a battery of computers, multiple screens and cold steel bookcases, is all business.

Cadieux holds bachelor’s and master’s degrees in computer science, both from the Université de Montréal, in a concentration called Systems, which he describes as “compilers, operating systems, basically hard-core programming.” He spent eight years focusing on assembly languages beginning with PASCAL and FORTRAN. When asked if his university training included anything resembling internationalization, Cadieux laughs. “That would be a no,” he confirms. He adds that twenty years later he suspects that internationalization still does not figure in his alma mater’s computer science curriculum. He also suspects that the reason the subject is not taught is that it’s misunderstood. “They probably think it’s too easy, too simple, not worth a whole course.”

Luckily, since its launch in 2002, the Localization Certificate Program offered by the Faculty of Continuing Education at the Université de Montréal has recognized the importance of teaching internationalization. The program requires students to take an introductory course in the subject taught by Cadieux. Although the course is designed as a multidisciplinary program that accepts students with translation, computer science and business orientations, Cadieux says that an overwhelming majority of his students are translators.

The road to internationalization

During his seven years as a student, Cadieux also worked a minimum of 20 hours a week as a consulting programmer, so making the transition from a student life to a professional career posed no problems. He says, for example, that as a student, “I was a subcontractor to a teacher who got a contract on the space shuttle project trying to figure out the motion equations for the Canadian arm. And he figured it out.”

Cadieux also gained on-the-job training in building teams to tackle larger projects. At one point, he employed four people in order to fulfill a contract for Farrés Mattar, an “amazing” physicist whose focus was optical lasers.

After completing his master’s degree in 1982 and because he already had a solid background and track record in scientific programming, Cadieux was hired by Météo, French shorthand for the Meteorological Service of Canada. Although at the time Météo boasted one of the first, fully-functioning machine translation (MT) applications, Cadieux’s work did not bring him into contact with that aspect of the important government agency. He did, however, get to work on a supercomputer.

“I was one of the first, if not the first Canadian to actually develop software on a Cray,” he says proudly. According to a Wikipedia entry, in the 1970s and 1980s “IBM Corp. and Cray Research competed to be the maker of the fastest computer on earth. Cray won every time. . . .” The problem was that Météo’s Cray hadn’t been delivered yet, so Cadieux worked remotely on a Cray housed at the manufacturer’s site in Minnesota — an arrangement that slowed things down some. “It was slow and painful,” he admits. “But I managed to port my software to Cray, test it and it worked.”

After six months and completing what he had set out to do at Météo, Cadieux went on to join former university colleagues who had found jobs with Alis Technologies. Compared to the huge and complex governmental environment, Alis was a small, young and dynamic company that embodied more closely Cadieux’s notions of an optimal work environment.

“My dream has always been to travel and work and be with a company where I could make a difference,” he says. As “employee number 8” at Alis, Cadieux made a difference. His responsibilities included tasks well beyond programming. “ Needless to say, I designed, I developed, I documented, I supported, I sold, I traveled.” At Alis, in addition to
realizing at least part of his career objectives, Cadieux also encountered internationalization for the first time.

Founded by Bachir Halimi, in 1983 Alis had already created a niche for itself in the globalization, internationalization, localization and translation domain — the “GILT space” in Cadieux’s parlance — by creating bidirectional, specifically Arab-language, products. One of the company’s first notable accomplishments was the “arabization” of Multiplan, one of the first spreadsheet programs introduced by Microsoft and a precursor of the now ubiquitous Excel. The success of the Multiplan project and the superior macro-driven flexibility of the Planet technology developed by Cadieux led to Alis winning the contract for the arabization of MS-DOS in 1987. Cadieux describes his brainchild as “a first attempt at a general-purpose text-rendering” tool that went on to include multilingual keyboards and menus and became a “general-purpose library to create bidirectional and multilingual devices.”

In search of new challenges, Cadieux left Alis in 1995 and worked three years

as the chief architect of the medical knowledge base for Purkinje, a developer of electronic medical record systems, where he first recognized the centrality of terminology to GILT processes. When asked the provenance of the name Purkinje, Cadieux waxes poetic. Jan Evangelista Purkinje (1787–1869) was a Czech doctor who discovered certain fibers that occur in both the heart and the brain. “It’s such a beautiful name for a computer science company for that reason,” Cadieux says.

At Purkinje, Cadieux says, the company’s “so-called medical knowledge base” was “really a terminology base, a massive amount of medical terminology, about 300,000 terms that were structured in such a way that a doctor could use a pen computer and quickly enter clinical information.” While a real departure from Alis, the job at Purkinje had some interesting similarities including creating basic terminology. Programmers, Cadieux says, work in one programming language while software architects, as the job title suggests, must lay the groundwork of a durable structure. An essential aspect of developing software architecture is “establishing some basic concepts and some basic terms and defining them.”

While working at Purkinje, Cadieux also grew to understand and appreciate the essentially “organic” nature of human language. Working closely with translators has only increased his respect for the complexities and difficulties involved in translation and language-based processes.

Entering internationalization for real

After leaving Purkinje, Cadieux reworked his curriculum vitae and decided to highlight his experience in internationalization. The rapid and enthusiastic response to the service offering somewhat surprised him. In short order, he won a contract to work with Bowne Global Solutions (BGS). Unfortunately, the dot-com debacle, which had a devastating impact across the board on the emerging language industry, cut short the company’s plans to develop a production unit in Los Angeles headed up by Cadieux.

In November 2000, however, an internationalization workshop in Montréal —
organized and animated by Cadieux and promoted by BGS — created new possibilities. The workshop attracted the attention of the Localisation Industry Standards Association (LISA), which invited Cadieux to present workshops at its events. Eventually, Cadieux was also invited to be the technical editor of The Globalization Insider, an online newsletter published by LISA. In collaboration with Bert Esselink, he developed and published a formula and a definition in an effort to clarify two oft-used and frequently misused terms:

Formula:  
\[ \text{Globalization} = \text{Internationalization} + N \times \text{Localization} \]  
\( (N \text{ is the number of targeted locales}) \)

Definition  
Internationalization of a thing consists in any and all preparatory tasks that will facilitate subsequent localization of said thing.

Since his first workshop in 2000, workshops have become an important source of revenue for Cadieux’s company, i18n inc., in addition to internationalization audits and the development of software tools designed to support translation and localization professionals with an emphasis on redundancy analysis.

Cadieux still takes a lively interest in the issues related to bidirectional languages, and he has developed a fascination with and an expertise in Asian languages. Living in Québec means that the official French <-> English language pair gets a fair amount of his attention. While Cadieux’s client list is heavy on software developers, his visibility at LISA, UNICODE and other industry conferences has also made him the “go-to guy” when language service providers are asked to provide internationalization services.

He explains that any effort to create a “universal” requires that a certain level of specificity be compromised.

The future of internationalization  
As for future trends in internationalization, Cadieux focuses on Unicode, which forms “the foundation” of internationalization. He describes Unicode as “a major success which I respect enormously.” That said, he notes that work remains to be done.

To start with, he explains that any effort to create a “universal” requires that a certain level of specificity be compromised. And, logically, the burden of compromise falls most heavily on the new languages now being added. In addition, he says that the emergence of multiple encodings such as UTF8, UTF12, UTF32 with BE and LE variants — and the inconsistent application of existing rules — create incompatibilities that will need to be addressed and may require a significant revamping, if not tomorrow, at some point.

Aside from Unicode, Cadieux stresses the importance of m-computing (mobile computing). Smaller devices mean smaller display screens, which represent a future challenge. Also, smaller mobile devices tend to be geared towards more personal needs, which will require a more significant internationalization effort.

Finally, Cadieux has noticed a significant if slow boom developing in the demand for internationalization services. Slowly but surely, he says, companies are waking up to the need for internationalization and the importance of addressing it earlier in the development cycle. Despite lingering confusion, the upsurge in demand may mean that the terminology and processes employed by the language industry are finally gaining currency.

Logrus Technology:

- Multilingual localization, engineering and testing
- Rare languages and complex scripts
- Large production site in Russia
- Complex projects, large and small
- Authoring, test case writing, tools development
- Multilingual DTP in 36+ languages, including ME and FE
- Most field area specializations: ERP, mechanical, engineering, telecom, marcom, finance, medical, consumer and more

We provide multilingual services into a large number of languages, including complex scripts and rare tongues. We specialize in large and highly technical projects requiring unique technical experience, a high level of self-sufficiency, and outstanding problem-solving capabilities. We go beyond simple, standard, separate work pieces and complete solutions of several of them combined, saving your time with very nice extras which include out-of-the-box workarounds, writing tools to cut hundreds of hours of effort, managing in-country approval, doing design, layout, DTP, sound recording and so much more.

Standing above a background of expectations

www.logrus.ru
management@logrus.ru
+1 (215) 947-4773

page 12
Vista is Microsoft’s latest operating system for the desktop. Along with Vista’s new appearance and features are additions to its supported locales, a newer version (3.0) of .NET, and a fix to a relatively unknown but useful pair of internationalization APIs for C++ programmers. Left over from Windows XP is a promising but not yet complete set of routines providing interesting information about world regions.

New locales in Vista

The locales supported on an operating system are important to companies that create software products since a wider audience becomes potentially available. Non-profit and charitable organizations are also interested for a variety of reasons, including providing an opportunity to those otherwise lacking the resources to participate in the world of computing. On Windows, the place to search is the Regional and Language Options Applet, where you can find a list of currently supported locales. The Vista operating system now provides 205, an increase of almost 30% over Windows XP. In addition, the locale naming system has been regularized to the format found in .NET — a combination of language and region or language, region, and script. As a result, approximately 70 of the previous locales have new names.

For example, a typical older form of a locale name in Windows XP is Afrikaans, whereas the new one is Afrikaans (South Africa); in place of Serbian (Cyrillic) are both Serbian (Cyrillic, Serbia) and Serbian (Latin, Serbia). In fact, the only remaining Windows locale with a single, language-only name is Persian, formerly Farsi. For the most part, a display name follows the Language (Region) or Language (Script, Region) patterns provided by the DisplayName property of the corresponding CultureInfo object in Microsoft .NET. Although the new locale names are certain much more informative and consistent, the name changes may affect your previous globalization efforts if they served as an identifier for retrieval of information or used for some other purpose in your code. If so, you may want to review your current strategy when you begin a Vista project. Table 1 shows a few examples of old and new formats.

Of course, the other interesting items are the new locales. The list is fairly extensive (40+ new locales), so we have broken down the list into related groups. The information is contained in a series of tables with some comments about the language and occasionally the region itself. You might want to keep in mind that in the future, your organization may be required to support some of these locales.

Minority languages in Europe

Alsation (France) — Low Alemannic German language spoken in the Alsace region of France.

Breton (France) — Celtic language of Brittany with about 250,000 speakers. Closely related to Cornish (revived in the twentieth century) and Welsh.

Corsican (France) — Corsican is closely related to Italian. The language is used at all levels of education in Corsica. Regional radio broadcast service available. Books and occasional newspaper articles published in the language.

Frisian (Netherlands) — Second official language in The Netherlands. Genetically most closely related to English: “Good butter en green tzieze (cheese) is good English

Irish (Ireland) — Although a minority language today, Irish is recognized by the constitution as the national and first official language of Ireland. It is also an official language of the European Union and is recognized in Northern Ireland.

Lower Sorbian (Germany) — Spoken by a Slavic minority in Brandenburg by about 10,000 people. Sometimes called Wendish or Lusatian. There is also a Lusatian community in Texas.

Luxembourgish (Luxembourg) — Luxembourgish is a West Germanic language spoken in Luxembourg. It is one of three official languages; the other two are French and German. About 300,000 people speak Luxembourgish worldwide.

Occitan (France) — Occitan is spoken in Occitania (Southern France and Monaco) and in a few valleys of Italy as well as in the Aran Valley in Spain. Fewer than 500,000 proficient speakers live in France.

Romansh (Switzerland) — One of four national languages of Switzerland. The number of speakers is about 50,000 to 70,000 in the canton of Grisons. It is the smallest of the official languages of Switzerland.

Table 1: A few of the differences between the old (Windows XP) and new (Vista) formats.

<table>
<thead>
<tr>
<th></th>
<th>Windows XP</th>
<th>Vista</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Afrikaans</td>
<td>Afrikaans (South Africa)</td>
<td>Language only replaced by language and region.</td>
<td></td>
</tr>
<tr>
<td>Serbian (Cyrillic)</td>
<td>Serbian (Cyrillic, Serbia)</td>
<td>Region information is available.</td>
<td></td>
</tr>
<tr>
<td>Zulu</td>
<td>isiZulu (South Africa)</td>
<td>Change in language name. “isi” is a prefix for language.</td>
<td></td>
</tr>
<tr>
<td>Farsi</td>
<td>Persian</td>
<td>The only locale with a language only name in Vista.</td>
<td></td>
</tr>
<tr>
<td>Faeroese</td>
<td>Faroese (Faroe Islands)</td>
<td>Spelling change of language. Both are acceptable forms.</td>
<td></td>
</tr>
<tr>
<td>Norwegian (Bokmål)</td>
<td>Norwegian, Bokmål (Norway)</td>
<td>Language and dialect format.</td>
<td></td>
</tr>
<tr>
<td>Northern Sotho</td>
<td>Sesotho sa Leboa (South Africa)</td>
<td>Change in language name and a reference to region.</td>
<td></td>
</tr>
</tbody>
</table>

en good Frieze.” The modern versions of both languages have diverged over the several hundred years of separation.

Irish (Ireland) — Although a minority language today, Irish is recognized by the constitution as the national and first official language of Ireland. It is also an official language of the European Union and is recognized in Northern Ireland.

Swedish (Finland) — Wikipedia gives a count of about 300,000 speakers. The current Finnish alphabet recognizes the special characters required for Swedish. It may seem strange to classify Swedish as a minority language but the concept is relative to the region where it is spoken.

Upper Sorbian (Germany) — Spoken by a Slavic minority in Saxony of about
40,000 people. Sometimes called Wendish or Lusatian. There is also a Lusatian community in Texas.

New in the Balkans

Bosnian (Cyrillic, Bosnia and Herzegovina) — Windows XP supported a Bosnian locale based on Latin script. Windows Vista has extended the script support to Cyrillic.

New Russian locales

Bashkir (Russia) — Spoken in the Republic of Bashkortostan and other parts of the Russian federation. A Turkic language, it is currently written with Cyrillic characters.

Yakut (Russia) — Sometimes known as Sakha. A North Turkic language family including Shor, Tuvan, and Dolgan. Uses the Cyrillic script.

Additional locales in the Indian subcontinent

Generally, if you can handle Hindi, you should be able to manage these languages as well.

Assamese (India) — Indian language spoken in the state of Assam (Northeast India) as well as Bangladesh and Bhutan. The language is Indo-Aryan written with a version of Bengali. About 20 million speakers.

Bengali (Bangladesh) — An Indo-Aryan language with two literary styles (elegant and current), about 200 million speakers, and spoken in Bangladesh, West Bengal, and some western countries such as the United States and United Kingdom.

Nepali (Nepal) — Indo-Aryan language spoken in Nepal, Bhutan and some parts of India and Burma.

Oriya (India) — An Eastern Indo-Aryan language spoken mainly in the state of Orissa as well as other regions such as West Bengal and Jharkhand. About 31 million speakers.

Sinhala (Sri Lanka) — Sinhala is an Indo-Aryan language of about 12 million non-Tamil people of Sri Lanka.

Minority languages in the People's Republic of China (PRC)

If you have been involved with GB 18030 certification, you will certainly recognize these four languages.

Mongolian (Traditional Mongolia, PRC) — Mongolian is written in both Cyrillic and traditional scripts. Few speakers know the traditional script, but it is beginning to be taught once again in schools. In the inner Mongolia Autonomous Region of China, the Classical Mongol script is still used.

Tibetan (PRC) — The language is derived from the Brahmi script but the classification is as a member of the Sino-Tibetan languages. The population is estimated to be about 7.3 million.

Uighur (PRC) — Uighur is a language in the eastern branch of the Turkic group. The people in the PRC live in the western part of China. Estimated population is supposed to be over 8 million.

Yi (PRC) — Yi is a family of related Tibeto-Burman languages spoken by the Yi people. It has about 6 million speakers.

New locales for Central Asia

Dari (Afghanistan) — Dari is a member of the Indo-Iranian group of languages, which includes Pashto as well.

Pashto (Afghanistan) — Together, Dari and Pashto make up the two official languages of Afghanistan.

Tajik (Cyrillic, Tajikistan) — Indo-European language of the Iranian group and the official language of the country. The language is normally written in Cyrillic, but other scripts are possible.

Turkmen (Turkmenistan) — Until 1991 a republic of the Soviet Union. Turkmen is a member of the Turkic family of languages.

New Southeast Asian locales

Filipino (Philippines) — Before Vista, the only locale available for the Philippines was English (Philippines).

Khmer (Cambodia) — Language of the Khmer people and the official language of Cambodia. The script is based on the Pallava script of India. The language is not tonal.

Lao (Lao People's Democratic Republic) — A tonal language of the Tai family. Lao is based on the same script as Thai. A second script, Tham, is also used and was derived from the one in Lan Na before the Thai script was standardized.

New locales for Africa

One has a name change; the others are new. Note also that the Setswana locale in Vista was the Tswana locale in Windows XP. So, Setswana (South Africa) is not a new locale.

Amharic (Ethiopia) — The official language of Ethiopia.

Hausa (Latin, Nigeria) — Hausa speakers are located in Niger and the north of Nigeria. Hausa also acts as a lingua franca in West Africa.

Igbo (Nigeria) — Spoken in Nigeria by about 18 million people. Dialects are many but generally mutually intelligible. Interest is building in standardizing the written language.

Kinyarwanda (Rwanda) — Main spoken language of Rwanda. Also spoken in the eastern Congo and southern Uganda.

Kiswahili (Kenya) — Bantu language spoken by the people of eastern and central Africa. It is a national language in Kenya, Tanzania, and Uganda. Also called Swahili.

Tamazight (Latin, Algeria) — Berber languages (Tamazight) are spoken in Morocco and Algeria. Latin characters are the norm. Much discussion is taking place on standardizing the numerous language variations.

Wolof (Senegal) — Language spoken in Senegal, Gambia and Mauritania. Native language of the Wolof people.

Yoruba (Nigeria) — Native tongue of the Yoruba. Spoken in Nigeria, Benin, and Togo and in communities of Brazil, Sierra Leone (called Oku), and Cuba (called Nago).

New English locales

An Indian film director once said that “English is just another Indian language.” The concept has apparently been extended in the other two regions as well.

English (India) — Don't expect to find the same formats for number, currency, time and date (long and short)!

English (Malaysia) — There is a strong demand for English teachers in Malaysia.

English (Singapore) — A government campaign encourages the local population to speak good English rather than “Singlish.”

A useful addition to the Spanish locales

Spanish (United States) — A locale that has long been needed.

New indigenous languages of Central and South America

K’iche (Guatemala) — Part of the Mayan language family spoken by nearly a million people in the highlands of Guatemala. Has many dialects but most are mutually intelligible.

Mapudungun (Chile) — Spoken in central Chile and west central Argentina by the Mapuche people. About 400,000 speakers.
Indigenous languages from Greenland and North America

Greenlandic (Greenland) — Eskimo-Aleut language spoken in Greenland. Sometimes called the Eastern Eskimo language. It is written using Latin letters, with a number of doubled vowels and consonants.

Inuktitut (Latin, Canada) — The language is used in schools and local government to some degree and on radio and television. It can be written using English letters.

Inuktitut (Syllabics, Canada) — A parallel locale but using the Inuktitut syllabary.

Mohawk (Mohawk) — Mohawk belongs to the Iroquoian group of Native American languages. The script uses a basic collection of Latin letters. It has been taught in schools since the early 1970s, and there has been a standard form of the written language since 1993.

Geographical information

GetGeoInfo is a promising Windows API that seems to be a work in progress. Available in Windows XP, it appears not to have been upgraded — except that the documentation tells you explicitly that the official languages and time zone information is not yet implemented. But even the information available is useful. Especially interesting was the hope of being able to read time zone and official language information around the globe. But just in case you are interested in what works, here are a couple of examples in the current output.

<table>
<thead>
<tr>
<th>GEO_NATION: 5</th>
<th>GEO_NATION: 20</th>
</tr>
</thead>
<tbody>
<tr>
<td>GEO_LATITUDE: 40.356</td>
<td>GEO_LATITUDE: 32.303</td>
</tr>
<tr>
<td>GEO_LONGITUDE: 47.869</td>
<td>GEO_LONGITUDE: -64.752</td>
</tr>
<tr>
<td>GEO_ISO2: AZ</td>
<td>GEO_ISO2: BM</td>
</tr>
<tr>
<td>GEO_ISO3: AZE</td>
<td>GEO_ISO3: BMU</td>
</tr>
<tr>
<td>GEO_RFC1766: fr-az</td>
<td>GEO_RFC1766: fr-bm</td>
</tr>
<tr>
<td>GEO_LCID: 0000040C</td>
<td>GEO_LCID: 0000040C</td>
</tr>
<tr>
<td>GEO_FRIENDLYNAME: Azerbaijan</td>
<td>GEO_FRIENDLYNAME: Bermuda</td>
</tr>
<tr>
<td>GEO_OFFICIALNAME: Republic of Azerbaijan</td>
<td>GEO_OFFICIALNAME: Bermuda</td>
</tr>
<tr>
<td>GEO_TIMEZONES: 0</td>
<td>GEO_TIMEZONES: 0</td>
</tr>
<tr>
<td>GEO_OFFICIALLANGUAGES: 0</td>
<td>GEO_OFFICIALLANGUAGES: 0</td>
</tr>
</tbody>
</table>

You have probably noticed the peculiar RFC-1766 identifiers as well. The documentation says you can fix this, but it is tiresome to determine exactly what they mean, especially when you have a tight publishing deadline.

Finding grapheme boundaries without using .NET

If you program in .NET, you know that there are a pair of classes, TextElementEnumerator and StringInfo, that work together to help you find grapheme boundaries in Unicode strings. Earlier versions of Windows also had a pair of C++ API's, CharNext and CharPrev, that provided some of this information, and if you are not using .NET, this may be your only means (unless you fall back on ICU).

With Windows XP, they were present but did not work although they were functional in earlier versions of Windows. I don’t know when they became broken, but apparently I was one of the first to notice. I passed on the information to Mihai Nita at Adobe, who verified the problem and sent it on to Microsoft. While the APIs are still not functional in XP, they are working reasonably in Vista.

It is easy to understand the usefulness of CharNext and CharPrev if this is your only tool. Let’s create a string of letters with the first having two non-spacing diacritic elements — in this case a combining candrabindu followed by a combining minus sign below. In the demonstration program Figure 1, you can see the values as well as the composed form of the string. Also note that only three characters actually appear in the output string even though the string length is five. The first character is clearly a grapheme cluster having three elements. The next two are simply single characters (B and C).

Now let's test CharNext using this string. In the program, you enter the Unicode code points as a series of 16-bit hex values as parameters. Thus, charw 0041 0310 0320 0042 0043 is the execution line. The important output is shown in Figure 2.

<table>
<thead>
<tr>
<th>Result in XP</th>
<th>Result in Vista</th>
</tr>
</thead>
<tbody>
<tr>
<td>0041 0310 1</td>
<td>0041 0042 3</td>
</tr>
<tr>
<td>0310 0320 1</td>
<td>0042 0043 1</td>
</tr>
<tr>
<td>0320 0042 1</td>
<td>0043 0000 1</td>
</tr>
</tbody>
</table>

References

www.microsoft.com/globaldev/vista/whats_new_vista.mspx
www.microsoft.com/globaldev/vista/vistahome.mspx

An appendix to this article, which is a table showing old and new forms of existing locales in Windows XP and Vista, is part of the downloadable Getting Started Guide available online at www.multilingual.com/gsg

April/May 2007 • www.multilingual.com/gsg
Old and new forms of existing locales in Windows XP and Vista

<table>
<thead>
<tr>
<th>Windows XP</th>
<th>Vista</th>
</tr>
</thead>
<tbody>
<tr>
<td>Afrikaans</td>
<td>Afrikaans (South Africa)</td>
</tr>
<tr>
<td>Albanian</td>
<td>Albanian (Albania)</td>
</tr>
<tr>
<td>Armenian</td>
<td>Armenian (Armenia)</td>
</tr>
<tr>
<td>Azerbaijani (Cyrillic)</td>
<td>Azeri (Cyrillic, Azerbaijan)</td>
</tr>
<tr>
<td>Azerbaijani (Latin)</td>
<td>Azeri (Latin, Azerbaijan)</td>
</tr>
<tr>
<td>Basque</td>
<td>Basque (Basque)</td>
</tr>
<tr>
<td>Belarusian</td>
<td>Belarusian (Belarus)</td>
</tr>
<tr>
<td>Bulgarian</td>
<td>Bulgarian (Bulgaria)</td>
</tr>
<tr>
<td>Catalan</td>
<td>Catalan (Catalan)</td>
</tr>
<tr>
<td>Croatian</td>
<td>Croatian (Croatia)</td>
</tr>
<tr>
<td>Croatian (Bosnia and Herzegovina)</td>
<td>Croatian (Latin, Bosnia and Herzegovina)</td>
</tr>
<tr>
<td>Czech</td>
<td>Czech (Czech Republic)</td>
</tr>
<tr>
<td>Danish</td>
<td>Danish (Denmark)</td>
</tr>
<tr>
<td>Divehi</td>
<td>Divehi (Maldives)</td>
</tr>
<tr>
<td>English (Philippines)</td>
<td>English (Republic of the Philippines)</td>
</tr>
<tr>
<td>English (Trinidad)</td>
<td>English (Trinidad and Tobago)</td>
</tr>
<tr>
<td>Estonian</td>
<td>Estonian (Estonia)</td>
</tr>
<tr>
<td>Farsi</td>
<td>Persian</td>
</tr>
<tr>
<td>Faeroese</td>
<td>Faeroese (Faroe Islands)</td>
</tr>
<tr>
<td>Finnish</td>
<td>Finnish (Finland)</td>
</tr>
<tr>
<td>FYRO Macedonian</td>
<td>Macedonian (Former Yugoslav Republic of Macedonia)</td>
</tr>
<tr>
<td>Galician</td>
<td>Galician (Galician)</td>
</tr>
<tr>
<td>Georgian</td>
<td>Georgian (Georgia)</td>
</tr>
<tr>
<td>Greek</td>
<td>Greek (Greece)</td>
</tr>
<tr>
<td>Gujarati</td>
<td>Gujarati (India)</td>
</tr>
<tr>
<td>Hebrew</td>
<td>Hebrew (Israel)</td>
</tr>
<tr>
<td>Hindi</td>
<td>Hindi (India)</td>
</tr>
<tr>
<td>Hungarian</td>
<td>Hungarian (Hungary)</td>
</tr>
<tr>
<td>Icelandic</td>
<td>Icelandic (Iceland)</td>
</tr>
<tr>
<td>Indonesian</td>
<td>Indonesian (Indonesia)</td>
</tr>
<tr>
<td>Xhosa</td>
<td>isiXhosa (South Africa)</td>
</tr>
<tr>
<td>Zulu</td>
<td>isiZulu (South Africa)</td>
</tr>
<tr>
<td>Japanese</td>
<td>Japanese (Japan)</td>
</tr>
<tr>
<td>Kannada</td>
<td>Kannada (India)</td>
</tr>
<tr>
<td>Kazakh</td>
<td>Kazakh (Kazakhstan)</td>
</tr>
<tr>
<td>Konkani</td>
<td>Konkani (India)</td>
</tr>
<tr>
<td>Korean</td>
<td>Korean (Korea)</td>
</tr>
<tr>
<td>Kyrgyz</td>
<td>Kyrgyz (Kyrgyzstan)</td>
</tr>
<tr>
<td>Latvian</td>
<td>Latvian (Latvia)</td>
</tr>
<tr>
<td>Lithuanian</td>
<td>Lithuanian (Lithuania)</td>
</tr>
<tr>
<td>Maltese</td>
<td>Maltese (Malta)</td>
</tr>
<tr>
<td>Maori</td>
<td>Maori (New Zealand)</td>
</tr>
<tr>
<td>Marathi</td>
<td>Marathi (India)</td>
</tr>
<tr>
<td>Mongolian (Cyrillic)</td>
<td>Mongolian (Cyrillic, Mongolia)</td>
</tr>
<tr>
<td>Norwegian (Bokmål)</td>
<td>Norwegian, Bokmål (Norway)</td>
</tr>
<tr>
<td>Norwegian (Nynorsk)</td>
<td>Norwegian, Nynorsk (Norway)</td>
</tr>
<tr>
<td>Polish</td>
<td>Polish (Poland)</td>
</tr>
<tr>
<td>Punjabi</td>
<td>Punjabi (India)</td>
</tr>
<tr>
<td>Romanian</td>
<td>Romanian (Romania)</td>
</tr>
<tr>
<td>Russian</td>
<td>Russian (Russia)</td>
</tr>
<tr>
<td>Sanskrit</td>
<td>Sanskrit (India)</td>
</tr>
<tr>
<td>Serbian (Cyrillic)</td>
<td>Serbian (Cyrillic, Serbia)</td>
</tr>
<tr>
<td>Serbian (Latin)</td>
<td>Serbian (Latin, Serbia)</td>
</tr>
<tr>
<td>Slovak</td>
<td>Slovak (Slovakia)</td>
</tr>
<tr>
<td>Slovenian</td>
<td>Slovenian (Slovenia)</td>
</tr>
<tr>
<td>Swedish</td>
<td>Swedish (Sweden)</td>
</tr>
<tr>
<td>Syriac</td>
<td>Syriac (Syria)</td>
</tr>
<tr>
<td>Tamil</td>
<td>Tamil (India)</td>
</tr>
<tr>
<td>Tatar</td>
<td>Tatar (Russia)</td>
</tr>
<tr>
<td>Telegu</td>
<td>Telegu (India)</td>
</tr>
<tr>
<td>Thai</td>
<td>Thai (Thailand)</td>
</tr>
<tr>
<td>Turkish</td>
<td>Turkish (Turkey)</td>
</tr>
<tr>
<td>Ukrainian</td>
<td>Ukrainian (Ukraine)</td>
</tr>
<tr>
<td>Urdu</td>
<td>Urdu (Islamic Republic of Pakistan)</td>
</tr>
<tr>
<td>Uzbek (Cyrillic)</td>
<td>Uzbek (Cyrillic, Uzbekistan)</td>
</tr>
<tr>
<td>Uzbek (Latin)</td>
<td>Uzbek (Latin, Uzbekistan)</td>
</tr>
<tr>
<td>Vietnamese</td>
<td>Vietnamese (Vietnam)</td>
</tr>
<tr>
<td>Welsh</td>
<td>Welsh (United Kingdom)</td>
</tr>
<tr>
<td>Northern Sotho</td>
<td>Sesotho sa Leboa (South Africa)</td>
</tr>
</tbody>
</table>
Code Got You Stranded?

Enterprise tools for internationalizing large applications over multiple technologies.
- Find and fix internationalization issues fast
- Save over 50% on engineering time and budget
- Free Trial at www.lingoport.com/passport

Typically found in 1 million Lines of Code: over 40K embedded strings, over 10K locale-limiting issues and more!

Lingoport is your passport for Internationalization

Get the benefits of our Globalyzer Enterprise Software combined with WorldReady development services

Complete Globalization solutions from code to market. Find out how we can help you achieve your release objectives. Visit lingoport.com/passport or email us at passport@lingoport.com.
This guide is a component of the magazine MultiLingual, formerly MultiLingual Computing & Technology. With a new look and a new sense of purpose, MultiLingual continues to lead the world in keeping track and informing its readers of the latest in the electronic universe.

In addition to the coverage that it provided before, the new magazine provides more insights from industry leaders, an improved news section and expanded calendar, as well as basic industry terminology and references.

MultiLingual’s eight issues a year are filled with news, technical developments and language information for people who are interested in the role of language, technology and translation in our twenty-first-century world. A ninth issue, the Resource Directory and Index, provides listings of companies in the language industry and a key to the previous year’s content.

Four issues each year include Getting Started Guides like this one, which are primers for moving into new territories both geographically and professionally.

The magazine itself covers a multitude of topics.

Translation
How are translation tools changing the art and science of communicating ideas and information between speakers of different languages? Translators are vital to the development of international and localized software. Those who specialize in technical documents, such as manuals for computer hardware and software, industrial equipment and medical products, use sophisticated tools along with professional expertise to translate complex text clearly and precisely. Translators and people who use translation services track new developments through articles and news items in MultiLingual.

Language technology
From multiple keyboard layouts and input methods to Unicode-enabled operating systems, language-specific encodings, systems that recognize your handwriting or your speech in any language — language technology is changing day by day. And this technology is also changing the way in which people communicate on a personal level — changing the requirements for international software and changing how business is done all over the world.

MultiLingual is your source for the best information and insight into these developments and how they will affect you and your business.

Global web
Every website is a global website, and even a site designed for one country may require several languages to be effective. Experienced web professionals explain how to create a site that works for users everywhere, how to attract those users to your site and how to keep the site current. Whether you use the internet and worldwide web for e-mail, for purchasing services, for promoting your business or for conducting fully international e-commerce, you’ll benefit from the information and ideas in each issue of MultiLingual.

Managing content
How do you track all the words and the changes that occur in a multilingual website? How do you know who’s doing what and where? How do you respond to customers and vendors in a prompt manner and in their own languages? The growing and changing field of content management and global management systems (CMS and GMS), customer relations management (CRM) and other management disciplines is increasingly important as systems become more complex. Leaders in the development of these systems explain how they work and how they work together.

Internationalization
Making software ready for the international market requires more than just a good idea. How does an international developer prepare a product for multiple locales? Will the pictures and colors you select for a user interface in France be suitable for users in Brazil? Elements such as date and currency formats sound like simple components, but developers who ignore the many international variants find that their products may be unusable. You’ll find sound ideas and practical help in every issue.

Localization
How can you make your product look and feel as if it were built in another country for users of that language and culture? How do you choose a localization service vendor? Developers and localizers offer their ideas and relate their experiences with practical advice that will save you time and money in your localization projects.

And there’s much more
Authors with in-depth knowledge summarize changes in the language industry and explain its financial side, describe the challenges of computing in various languages, explain and update encoding schemes and evaluate software and systems. Other articles focus on particular countries or regions; translation and localization training programs; the uses of language technology in specific industries — a wide array of current topics from the world of multilingual computing.

MultiLingual is a critical business asset in our electronic world. Readers of MultiLingual explore language technology and its applications, project management, basic elements and advanced ideas with the people who are building the future.