.NET Internationalization: The Developer's Guide to Building Global Windows and Web Applications

By Guy Smith-Ferrier

.....

Publisher: **Addison Wesley Professional**
Pub Date: **August 07, 2006**
Print ISBN-10: **0-321-34138-4**
Print ISBN-13: **978-0-321-34138-9**
Pages: **672**

Table of Contents | Index

Overview

As business becomes more and more global, software developers increasingly need to make applications multi-lingual and culturally aware. The .NET Framework may well have the most comprehensive support for internationalization and globalization of any development platform to date, and .NET Internationalization teaches developers how to unlock and utilize that support.

Experienced international application developer Guy Smith-Ferrier covers the internationalization of both Windows Forms and ASP.NET applications, using both Versions 1.1 and 2.0 of the .NET Framework. Smith-Ferrier not only teaches you the best ways to take advantage of the globalization and internationalization features built in to the .NET Framework and Visual Studio, he also provides original code to take globalized applications to the next level of international utility and maintainability.

Key topics include

- An introduction to the internationalization process and how localization and globalization are supported in Windows and the .NET Framework
- The use of resource managers, cultures, resource DLLs, and localized strings, images, and filesincluding strongly typed resources
- Detailed coverage of form localization in Windows Forms and Web Forms
- Dealing with regional cultures and their casing, collation, and calendars
- Managing right-to-left Middle-Eastern text and pictographic East Asian languages
- How to use the book's original resource administration utilities
- How to translate resources with machine translation
- How to create custom cultures and integrate them with the .NET Framework 2.0 and Visual Studio 2005
- How resource managers work and how to write custom resource managers, including a resource manager that uses a database
- How to test your internationalization with FxCop using new and existing globalization rules

- How to effectively include the translator in the internationalization process

Whether you are a developer, architect, or manager, if you are involved in international applications with the .NET Framework, this is the one book you need to read and understand before you start development.

Guy Smith-Ferrier is an author, developer, trainer, and speaker with more than 20 years of software engineering experience. He has internationalized applications in four development platforms, including the .NET Framework. A frequent conference speaker, Guy is the author of C# and .NET courseware and has written numerous articles. You can read his blog at www.guysmithferrier.com.

.NET Internationalization: The Developer's Guide to Building Global Windows and Web Applications
By Guy Smith-Ferrier
.....
Publisher: **Addison Wesley Professional**
Pub Date: **August 07, 2006**
Print ISBN-10: **0-321-34138-4**
Print ISBN-13: **978-0-321-34138-9**
Pages: **672**

Table of Contents | Index

Copyright
Microsoft .NET Development Series
Foreword
Preface
Acknowledgments
About the Author
Chapter 1. A Roadmap for the Internationalization Process
The Operating System
The .NET Framework and Visual Studio
Languages
Resource Formats
Languages and Cultural Formatting
Windows Forms Applications
ASP.NET Applications
Globalization
Localization
Machine Translation
Resource Administration
Testing
Translation
Where Are We?

Chapter 2. Unicode, Windows, and the .NET Framework
- Unicode
- Code Pages
- Unicode Windows
- Code Page Windows
- Virtual Machines
- Windows Multilingual User Interface Pack
- Language and Locale Support
- .NET Framework Languages and .NET Framework Language Packs
- Where Are We?

Chapter 3. An Introduction to Internationalization
- Internationalization Terminology
- Cultures
- Localizable Strings
- Resource File Formats
- Resource Managers
- Localized Strings
- CurrentCulture and CurrentUICulture
- The Resource Fallback Process
- Image and File Resources
- Strongly-Typed Resources in the .NET Framework 2.0
- Strongly-Typed Resources in the .NET Framework 1.1
- Where Are We?

Chapter 4. Windows Forms Specifics
- Localizing Forms
- Setting the CurrentUICulture
- Changing the Culture During Execution
- Using Regional and Language Options to Change the Culture
- Dialogs
- Windows Resource Localization Editor (WinRes)
- ClickOnce
- Where Are We?

Chapter 5. ASP.NET Specifics
- Localizability in .NET 1.1
- Static Text
- Calendar Control
- Setting and Recognizing the Culture
- Caching Output by Culture
- Localizability in Visual Studio 2005
- Automatic Culture Recognition for Individual Pages
- Manual Culture Recognition for Individual Pages
- Application-Wide Automatic Culture Recognition
- Explicit Expressions

Global Resources
Implicit Expressions vs. Explicit Expressions
Programmatic Resource Access
Localizing ASP.NET 2 Components
Localizing the Website Administration Tool
Where Are We?
Chapter 6. Globalization
The CultureInfo Class
The RegionInfo Class
Geographical Information
String Comparisons
Casing
Sort Orders
Calendars
DateTimes, DateTimeFormatInfos, and Calendars
Numbers, Currencies, and NumberFormatInfo
International Domain Name Mapping
Environment Considerations
Extending the CultureInfo Class
Where Are We?
Chapter 7. Middle East and East Asian Cultures
Supplemental Language Support
Right-to-Left Languages and Mirroring
Input Method Editors
Where Are We?
Chapter 8. Best Practices
Font Selection
Strings and String.Format
Embedded Control Characters
Exception Messages
Hot Keys
Windows Forms Best Practices
Where Are We?
Chapter 9. Machine Translation
How Good Is It ?
Translation Engine
Pseudo Translation
Static Lookup Translator
Web Service Translators
HTML Translators
Office 2003 Research Services
Translator Evaluator
Where Are We?
Chapter 10. Resource Administration
Resource Administrator
Add Resource String Visual Studio Add-In
Reading and Writing Resources
Resource Governors

The Resource Editor Control
Where Are We?
Chapter 11. Custom Cultures
Uses for Custom Cultures
Using CultureAndRegionInfoBuilder
Installing/Registering Custom Cultures
Uninstalling/Unregistering Custom Cultures
Public Custom Cultures and Naming Conventions
Supplemental Substitute Custom Cultures
Custom Culture Locale IDs
Custom Culture Parents and Children
Support for Custom Cultures
Supplemental Custom Cultures
Culture Builder Application Sample (CultureSample)
Combining Cultures
Exporting Operating System-Specific Cultures
Company-Specific Dialects
Extending the CultureAndRegionInfoBuilder Class
Custom Cultures and .NET Framework Language Packs
Custom Cultures in the .NET Framework 1.1 and Visual Studio 2003
Where Are We?
Chapter 12. Custom Resource Managers
ResourceManager.CreateFileBasedResourceManager
ResourceManager Exposed
Custom Resource Managers Examples
DbResourceManager
ResourcesResourceManager and ResXResourceManager
Writeable Resource Managers
TranslationResourceManager
StandardPropertiesResourceManager
ResourceManagerProvider
Using Custom Resource Managers in Windows Forms
Generating Strongly-Typed Resources for Sources Other Than resx Files
Using Custom Resource Managers in ASP.NET 2.0
Where Are We?
Chapter 13. Testing Internationalization Using FxCop
A Brief Introduction to FxCop
Using FxCop's Stand-Alone GUI
FxCop and ASP.NET
FxCop Globalization Rules
FxCop Spelling Rules

Overview of New FxCop Globalization Rules
Writing FxCop Globalization Rules
Where Are We?
Chapter 14. The Translator
The Translation Process
Translator or Localizer?
Translation/Localization Strategies
Resource Translation Manager
Reintegrating Resources
Where Are We?
Appendix A. New Internationalization Features in the .NET Framework 2.0 and Visual Studio 2005
Compatibility
.NET Framework Redistributable
.NET Framework Language Packs
.NET Framework
Strongly Typed Resources
Custom Cultures
Visual Studio's Resource Editor
Windows Forms
ASP.NET
Appendix B. Information Resources
Books
Resources
Magazines
Web Sites and FTP Sites
Online Machine-Translation Web Sites
Blogs
Conferences
Organizations
Commercial Machine-Translation Products
Alternatives to .NET Framework Internationalization
Index

Copyright

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U.S. Corporate and Government Sales
(800) 382-3419
corpsales@pearsontechgroup.com

For sales outside the U.S., please contact:

International Sales
international@pearsoned.com

Visit us on the Web: www.awprofessional.com

Library of Congress Cataloging-in-Publication Data:

Smith-Ferrier, Guy.

.Net internationalization : the developer's guide to building global Windows and Web applications / Guy Smith-Ferrier
p. cm.
ISBN 0-321-34138-4 (pbk. : alk. paper)
1. Microsoft .NET. 2. Microsoft .NET Framework. 3. Application software. I. Title.
QA76.76.M52S65 2006
005.2'768dc22
2006013165

Text printed in the United States on recycled paper at R.R. Donnelley in Crawfordsville, IN.

First printing, August 2006

Dedication

For Sandy, Samuel and Eloise

Microsoft .NET Development Series

John Montgomery, Series Advisor

Don Box, Series Advisor

Martin Heller, Series Editor

The Microsoft .NET Development Series is supported and developed by the leaders and experts of Microsoft development technologies including Microsoft architects. The books in this series provide a core resource of information and understanding every developer needs in order to write effective applications and managed code. Learn from the leaders how to maximize your use of the .NET Framework and its programming languages.

Titles in the Series

Brad Abrams, .NET Framework Standard Library Annotated Reference Volume 1: Base Class Library and Extended Numerics Library, 0-321-15489-4

Brad Abrams and Tamara Abrams, .NET Framework Standard Library Annotated Reference, Volume 2: Networking Library, Reflection Library, and XML Library, 0-321-19445-4

Keith Ballinger, .NET Web Services: Architecture and Implementation, 0-321-11359-4

Bob Beauchemin and Dan Sullivan, A Developer's Guide to SQL Server 2005, 0-321-38218-8

Bob Beauchemin, Niels Berglund, Dan Sullivan, A First Look at SQL Server 2005 for Developers, 0-321-18059-3

Don Box with Chris Sells, Essential .NET, Volume 1: The Common Language Runtime, 0-201-73411-7

Keith Brown, The .NET Developer's Guide to Windows Security, 0-321-22835-9

Eric Carter and Eric Lippert, Visual Studio Tools for Office: Using C# with Excel, Word, Outlook, and InfoPath, 0-321-33488-4

Eric Carter and Eric Lippert, Visual Studio Tools for Office: Using Visual Basic 2005 with Excel, Word, Outlook, and InfoPath, 0-321-41175-7

Mahesh Chand, Graphics Programming with GDI+, 0-321-16077-0

Krzysztof Cwalina and Brad Abrams, Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 0-321-24675-6

Len Fenster, Effective Use of Microsoft Enterprise Library: Building Blocks for Creating Enterprise Applications and Services, 0-321-33421-3

Sam Guckenheimer and Juan J. Perez, Software Engineering with Microsoft Visual Studio Team System, 0-321-27872-0

Anders Hejlsberg, Scott Wiltamuth, Peter Golde, The C# Programming Language, Second Edition, 0-321-33443-4

Alex Homer, Dave Sussman, Mark Fussell, ADO.NET and System.Xml v. 2.0The Beta Version, 0-321-24712-4

Alex Homer and Dave Sussman, ASP.NET 2.0 Illustrated, 0-321-41834-4

Alex Homer, Dave Sussman, Rob Howard, ASP.NET v. 2.0The Beta Version, 0-321-25727-8

Joe Kaplan and Ryan Dunn, The .NET Developer's Guide to Directory Services Programming, 0-321-35017-0

Mark Michaelis, Essential C# 2.0, 0-321-15077-5

James S. Miller and Susann Ragsdale, The Common Language Infrastructure Annotated Standard, 0-321-15493-2

Christian Nagel, Enterprise Services with the .NET Framework: Developing Distributed Business Solutions with .NET Enterprise Services, 0-321-24673-X

Brian Noyes, Data Binding with Windows Forms 2.0: Programming Smart Client Data Applications with .NET, 0-321-26892-X

Fritz Onion, Essential ASP.NET with Examples in C#, 0-201-76040-1

Fritz Onion, Essential ASP.NET with Examples in Visual Basic .NET, 0-201-76039-8

Ted Pattison and Dr. Joe Hummel, Building Applications and Components with Visual Basic .NET, 0-201-73495-8

Dr. Neil Roodyn, eXtreme .NET: Introducing eXtreme Programming Techniques to .NET Developers, 0-321-30363-6

Chris Sells and Michael Weinhardt, Windows Forms 2.0 Programming, 0-321-26796-6

Chris Sells, Windows Forms Programming in C#, 0-321-11620-8

Chris Sells and Justin Gehtland, Windows Forms Programming in Visual Basic .NET, 0-321-12519-3

Guy Smith-Ferrier, .NET Internationalization: The Developer's Guide to Building Global Windows and Web Applications, 0-321-34138-4

Paul Vick, The Visual Basic .NET Programming Language, 0-321-16951-4

Damien Watkins, Mark Hammond, Brad Abrams, Programming in the .NET Environment, 0-201-77018-0

Shawn Wildermuth, Pragmatic ADO.NET: Data Access for the Internet World, 0-201-74568-2

Paul Yao and David Durant, .NET Compact Framework Programming with C#, 0-321-17403-8

Paul Yao and David Durant, .NET Compact Framework Programming with Visual Basic .NET, 0-321-17404-6

For more information go to www.awprofessional.com/msdotnetseries/

Foreword

I'VE BEEN WORKING IN LOCALIZATION my entire career, and I've seen countless examples of how amazingly hard it can be to create a world-ready Windows product. The early days of 32-bit Windows saw a definite lack of support for international products. Since then, so many APIs related to internationalization have been added, duplicated, replaced, and deprecated in various products and platforms that almost nobody can acquire the skills needed to know for certain if their product is properly adapted to an international market.

Of course, nobody was feeling the pain of the situation that Microsoft created more than Microsoft itself. And true to form, the company created a grand vision: a brand new development framework and runtime environment, where support for all things painful and bug-causingincluding such areas as memory management, security, and even internationalization were considered from the start. We now know the result of this effort as the .NET Framework.

And what a result it is! The difference between creating internationalized applications in .NET and Win32 is like night and day. There simply is no comparison! The amount of minute details that are abstracted away by the .NET Framework is absolutely staggering.

Still, as it turns out, Visual Studio .NET does not provide a "Make My Application World-Ready Wizard." Nor will it anytime soon. Despite the complexity hidden by the .NET Framework, the software architect and developer still requires a good understanding of the issues involved in creating a culture-independent and localizable product. There simply hasn't been any one single, overviewable source of the most essential information.

That's exactly why I'm proud of my involvement in the effort behind this book. I hope that this book can have the same impact on world-ready applications as Code Complete has had on producing quality code in general or Writing Secure Code has had on producing trustworthy applications. If this work fulfills my hope, well, time will tell. The potential is certainly there. Now it's up to you to read, learn, practice, and deliver.

Jesper Holmberg,

International Project Engineer,

Core Operation Systems Division, Microsoft Corporation

Preface

IT IS OFTEN SAID THAT THE WORLD is getting smaller every day. Cheap, fast air travel; the global economy; the global climate; the insatiable desire for standards; and, perhaps, most important of all, the Internet all play a part in the homogenization of our world. It is ironic, therefore, that this shrinking effect is not a benefit to developersin fact, it has the opposite effect. As the world community achieves greater awareness and greater tolerance, the demand for culturally aware software increases. Within the U.S. and Canada, for example, significant Hispanic, French, and Chinese populations exist.

At best, English-only Windows applications and Web sites are difficult for these cultures. At worst, these applications and Web sites exclude or even offend these populations. Such Web sites also are potentially illegal. For example, France and Quebec, Canada, both have laws prohibiting the hosting of English-only Web sites. Many countries (Wales, for example) also require that public services always be available in the native language, in addition to English. From marketing and financial viewpoints, English-only applicationsand

particularly Web sitesrepresent a massive lost market. By their very nature, Web sites are global, but an English-only Web site loses marketing opportunities to people who do not speak English. From a marketing point of view, such a lost opportunity is a criminal waste.

Good news exists, however. The .NET Framework has arguably the most comprehensive support for internationalizing .NET applications of any development platform. The .NET Framework provides a significant infrastructure for globalizing applications, and Visual Studio 2003 and 2005 provide excellent functionality for localizing Windows applications. Although Visual Studio 2003 offered little help for ASP.NET developers, rest assured that Visual Studio 2005 has thorough support for localizing Web applications.

What This Book Covers

This book covers the internationalization of .NET Windows Forms and ASP.NET applications. It covers both versions 1.1 and 2.0 of the .NET Framework, and both Visual Studio 2003 and Visual Studio 2005. Although the main focus of the book is on the .NET Framework 2.0 and Visual Studio 2005, it highlights differences between them and the .NET Framework 1.1 and Visual Studio 2003. Visual Studio 2003 developers can read this book by skipping the sections on Visual Studio 2005, but I advise against this. Visual Studio 2005 offers many useful new facilitiesmany of which can be retrofitted to Visual Studio 2003to provide guidance on how to design Visual Studio 2003 applications with a clear migration path to Visual Studio 2005. For a list of the new internationalization features in the .NET Framework 2.0 and Visual Studio 2005, see Appendix A, "New Internationalization Features in the .NET Framework 2.0 and Visual Studio 2005."

Chapter 1, "A Roadmap for the Internationalization Process," provides a general overview of what is involved in internationalizing an application, and includes more specific information on why some of the more advanced chapters will be of more interest to you and what solutions can be found in them. Chapter 2, "Unicode, Windows, and the .NET Framework," lays down the foundation of what Unicode is and what you can expect from the operating system and the .NET Framework. The essential mechanics of internationalization are covered in Chapter 3, "An Introduction to Internationalization," and this should be considered a prerequisite for all other chapters. From here, Windows Forms developers should read Chapter 4, "Windows Forms Specifics," and ASP.NET developers should read Chapter 5, "ASP.NET Specifics." Chapter 6, "Globalization," covers the concept of globalization in depth, along with the .NET Framework globalization classes and some solutions for globalization issues that are not covered by the .NET Framework classes. Chapter 7, "Middle East and East Asian Cultures," covers issues that are specific to right-to-left cultures (Arabic, Divehi, Farsi, Hebrew, Syriac, and Urdu) and Asian cultures (Chinese, Korean, and Japanese). Chapter 8, "Best Practices," provides internationalization guidance on a more general level, including issues such as the choice of fonts. Chapter 9, "Machine Translation," provides solutions for automatically translating your resources into other languages. Chapter 10, "Resource Administration," describes a number of utilities included in the source code for this book, to help with the administration of resources.

As applications grow beyond the simplistic examples used to illustrate concepts, the maintenance and management of applications' resources demand more dedicated solutions. Chapter 11, "Custom Cultures," describes how to create your own cultures and integrate them into the .NET Framework 2.0 and Visual Studio 2005. Custom cultures are useful for creating pseudo translations, supporting unsupported cultures, creating commercial dialects, and supporting languages outside their normal country (e.g., Spanish in the U.S., Chinese in Canada, and Urdu in the United Kingdom). Chapter 12, "Custom Resource Managers," describes how the existing resource managers work internally, and how to write new resource managers and use them in Windows Forms applications and ASP.NET applications. Custom resource managers are the solution to numerous developer issues, from changing the origin of resources (to, say, a database) to changing the functionality of resource managers (to, say, standardize specific properties throughout an application). Chapter 13, "Testing Internationalization Using FxCop," shows how to use FxCop to apply internationalization rules to your assemblies. It covers the existing FxCop globalization rules, introduces new globalization rules based on the issues raised throughout this book, and shows how to write these rules to

enable you to write your own rules. Chapter 14, "The Translator," discusses the issues and solutions involved in including the translator in the internationalization process. As noted already, Appendix A, "New Internationalization Features in .NET Framework 2.0 and Visual Studio 2005," includes a list of the new features in the .NET Framework 2.0 and Visual Studio 2005. Most of these features are covered throughout the book, so this appendix is mostly a list of pointers to chapters within the book. Appendix B, "Information Resources," is a list of books, resources, Web sites, magazines, online machine-translation Web sites, blogs, conferences, organizations, and commercial machine-translations products that will raise your awareness of the internationalization community.

Who Should Read This Book

This book is aimed at developers, team leaders, technical architectsessentially, anyone who is involved in the technical aspects of internationalizing .NET applications. The book uses C# examples, but the content is equally relevant to Visual Basic.NET developers and anyone who uses Visual Studio. The book expects that Visual Studio will be the main development environment, but many chapters focus solely on the .NET Framework. As such, the information contained within has equal value if you use an alternative development environment such as SharpDevelop or Borland Delphi 2005.

What You Need to Use This Book

To get the most from this book, you need the .NET Framework 2.0 and Visual Studio 2005. Alternatively, you can follow a large part of this book using the .NET Framework 1.1 and Visual Studio 2003. You can follow a lesser part of this book using the .NET Framework 1.1 or 2.0 and an alternative development environment.

Source Code

The complete source code for this book is available for download at http://www.dotneti18n.com. You will also find errata, updates to the code, new code examples, and additional information.

Acknowledgments

I WOULD LIKE TO THANK Jesper Holmberg, Ken Cox, Mark Blomsma, Douglas Reilly, Jason Nadal, Martin Peck, Shaun Wilde, and the Microsoft Globalization Team for their excellent help in reviewing this book; the better three quarters of 4 Chaps From Blighty (Brian Long, Steve Scott, and especially Steve Tudor, at http://www.4chapsfromblighty.com) for their excellent technical expertise and their readiness to help a friend in need; everyone who worked on this book at Addison-Wesley but notably Joan Murray, Jessica D'Amico, Curt Johnson, Antje King, Marie McKinley, Andy Beaster, and Lara Wysong for their dedication to the cause; many people at Microsoft especially "Dr. International" for their specific help and their general contribution to the internationalization world; Roy Nelson for his problem-solving skills; and Yae Nobuto for her linguistic skills. Special thanks to my brother, Paul, for too many reasons to list.

Finally, for the avoidance of doubt, the fictional character Frodo Potter does not appear anywhere in this book.

About the Author

Guy Smith-Ferrier is an author, developer, trainer, and speaker with more than 20 years of software engineering experience. He has internationalized applications in four development platforms, including the .NET Framework. He has spoken at numerous conferences on three continents and been voted Best Speaker twice. He is the author of C#/.NET course-ware and the official Borland courseware for COM and ADO. He has written articles for numerous magazines, has co-authored an application-development book, and is the author of the ADO chapter of Mastering Delphi 6 (Sybex, 2001). He lives in the U.K. with his wife and two children. His blog is at http://www.guysmithferrier.com.

Chapter 1. A Roadmap for the Internationalization Process

LEARNING IS LIKE CLIMBING a mountain. When you stand at the bottom of a mountain, you cannot actually see the top; the side of the mountain gets in the way. What you see is the first peak on the horizon. As you climb over that peak, you see the next peak on the horizon. You keep doing this until the last peak is the summit and your climb is over. I have always felt that technical subjects seem simple when I don't know anything about them. As I climb each peak, I see the other subjects behind my initially imperfect view and realize that there is more to climb. Each peak reveals new peaks that I was not aware of.

Internationalization is like a mountain. From the outside, the subject might seem simpleyou just translate all the text, don't you? But as each problem is solved, it allows us to see other problems on the horizon. From the bottom of the mountain, it is difficult to see all the peaks before we reach the top. The purpose of this chapter is to provide an aerial view of the mountain. Its goal is to help project managers and developers appreciate some of the higher-level decisions that must be made in the internationalization of an application. It poses many questions and leaves most unanswered. The answers lie in the subsequent chapters of this book. If you are a developer, lend your copy of this book to your project manager (or buy him or her a new copy) and ask him or her to read this chapter; with this overview, managers should get a broad grasp of your world.

The Operating System

Your choice of the version of Windows to use for your application has a significant impact upon the functionality of your application. As always, the most recent operating systems provide the best support; this is especially true in internationalization. A clear distinction exists between Unicode versions of Windows (Windows NT 3.51, Windows 2000, Windows XP, and above) and non-Unicode versions of Windows (Windows 95, 98, and Me); I advise using a Unicode version of Windows, at the very least. The version of the operating system also affects support for mirroring, which is used in right-to-left cultures (such as Arabic, Hebrew, and Persian [Farsi]), font availability and font functionality, the number of supported cultures (languages and regions), and support for complex scripts (some writing systems such as Thai require considerably more complex support for rendering than comparatively simple Latin languages).

The .NET Framework and Visual Studio

Your choice of which version of the .NET Framework to use has perhaps the greatest impact of any decision under your control. Obviously, the .NET Framework 2.0 has significantly better internationalization support

than the .NET Framework 1.1. Take a look at Appendix A, "New Internationalization Features in the .NET Framework 2.0 and Visual Studio 2005," for a complete list of the differences. Some of these new features can be retrofitted into the .NET Framework 1.1 (e.g., strongly typed resources and IDN mapping); others can be simulated with varying degrees of success (e.g., WinRes Visual Studio File Mode and custom cultures). However, others are solely the domain of the newer version of the framework (for example, `TableLayoutPanel`).

The version of the .NET Framework (i.e., 1.1 or 2.0) dictates the version of Visual Studio (2003 or 2005, respectively). The functionality of Visual Studio 2005 is obviously superior to that of Visual Studio 2003. The Resource Editor has greater functionality; support for strongly typed resources is built into the IDE; and Windows Forms support a new property reflection model. Of greatest significance to ASP.NET applications is that Visual Studio 2005 supports localizing ASP.NET applications. In Visual Studio 2003, localizing an ASP.NET application is all your own work.

Languages

The languages that your application supports dictate many of the considerations that arise when internationalizing your application. Latin languages, such as French, German, and Spanish, make the fewest demands on your application and require a relatively low level of technology to support. The further you go from Latin languages, the greater awareness you need of issues that Latin languages do not require.

Right-to-left languages (such as Arabic, Hebrew, and Persian [Farsi]) require support for reading text right to left (instead of left to right) and mirroring. In a mirrored form or Web page, controls are repositioned from one side of a form to the other. In addition, controls are rendered right to left (e.g., scrollbars are on the left of a control, and menus start from the right and expand to the left).

Your choice of operating system affects the level of font technology available to support your languages. Chinese, Japanese, and Korean (often abbreviated to CJK) languages use glyphs that are not present in most fonts. Windows 2000 and above support font linking. This allows a font, such as MS Mincho, to be linked to another, such as Microsoft Sans Serif, so that when a glyph is required that is not found in the base font, the other linked fonts are searched for the required glyph. This technology takes away much of the burden of choosing a suitable font because you can stay with a single font, such as Microsoft Sans Serif, and rely on font linking to find the missing glyphs. Similarly, other languages (e.g., Devanagari) use scripts that require complex rendering, which is not found in Latin scripts. This complex rendering is achieved automatically if your operating system supports font fallback (which Windows 2000 and above do) and this support is installed.

East Asian languages (Chinese, Japanese, and Korean) make further demands on the application. These languages have considerably more characters than are available on a QWERTY keyboard (there are 5,000-odd Japanese Kanji characters), so a software component called an Input Method Editor (IME) is used to enter characters. Typically, this issue is more relevant to Windows Forms applications than ASP.NET applications: IMEs are used with both Windows Forms applications and ASP.NET applications, but Windows Forms applications can optionally offer help in controlling the IME. The question in terms of development is, do you offer help in controlling the IME, and, if so, how much help do you offer?

The languages that you need to support also dictate whether you need to use custom cultures. A custom culture is a language or a language and region (or a replacement of an existing language or language and region) that is not already known to the .NET Framework. For example, if you wanted to support Bengali in Bangladesh, you would have to create a custom culture. Similarly, if you wanted to support a known language outside of its "known" region (e.g., Spanish in the United States), then you would have to create a custom culture. Although custom cultures can be created in the .NET Framework 1.1, support for this is very low and

you are strongly advised to use the .NET Framework 2.0.

The languages that you need to support also affect your testing process. For accurate testing, you must test your application on the version of the operating system that your users will be running. The "version" means not only the release (Windows 2000, XP, 2003, and so on), but also the language. If your users will be running German Windows XP Professional, then your testing is not complete if you test it in English Windows XP Professional. You should consider two testing scenarios: Windows Multiple User Interface and Virtual PC. Both are covered in Chapter 2, "Unicode, Windows, and the .NET Framework."

Resource Formats

The strings, bitmaps, icons, audio, and other files that can be translated or localized in your application are collectively called resources. During development, these resources are moved out of the source code and into a separate location. One of the fundamental decisions that you must make is what format these resources will be stored in. This decision has far-reaching implications. The most commonly used format, and the one offered by default in Visual Studio 2003 and 2005, is resx files. A resx file is an XML file. The .NET Framework 2.0 offers resx, resources, txt, and restext as choices, but you can choose a completely new format as well.

A common alternative to these is to use a database to store all resources, but you could equally use XLIFF (XML Localization Interchange File Format, a format used to exchange localization data between tools). As with all decisions, you must weigh the pros and cons. The .NET Framework and Visual Studio lend themselves to using resx files. Their support for this format is complete. resx files can be manipulated using all .NET Framework SDK tools and all parts of Visual Studio, including, notably, Visual Studio's Resource Editor. All other formats have less support to some degree and involve some additional effort to support. This does not mean that a database, for example, cannot be used. In Chapter 12, "Custom Resource Managers," we create a resource manager for a database. In the same chapter, we create a utility for creating strongly typed resources, for resources in any format. In Chapter 10, "Resource Administration," we create a Resource Editor replacement that, among other uses, enables you to read and write resources in any format.

One of the benefits of storing resources in a database is that resources can be changed easily at runtime. (By contrast, unless you are writing an ASP.NET 2.0 application, changes to resx files require a recompilation for the changes to be seen.) Thus, a database offers a great solution for translators because they can immediately see the results of their translation. In addition, a database is suitable if the strings of an application are rapidly changing, making a recompilation of the application unsuitable.

Languages and Cultural Formatting

One of the first decisions that you must make is how users will specify what language or language and region they want the application to use. This question involves several parts.

First, .NET Framework applications make a distinction between the language and region used for the user interface and the language and region used for cultural formatting. The user interface refers to the text and images that are shown to the user. Cultural formatting refers to issues such as the formatting of dates, numbers, and currencies. The first part of this decision involves whether you should allow your users to make a distinction between these two settings. If they can make a distinction between these two settings, it would be possible to have the user interface in, say, Spanish but use cultural formatting for the U.S. This would be useful for the millions of Spanish-speaking people who live and work in the U.S. (although you might argue that creating a custom culture is a better solution).

Second, you should be aware that languages vary from region to region. English as spoken in the United States is not exactly the same as English as spoken in the U.K. The same is true for French in Canada compared with French in France, and, perhaps more important, Portuguese in Brazil compared with Portuguese in Portugal. One of the decisions you must make is how specific your translations will be. If you translate from English to French, will you use a single French translation for all French-speaking countries, regardless of their region? Imagine that you had to use an application in which the phrase, "The bank has not honored this check," is displayed as "The bank has not honoured this cheque" (which uses the British English spelling). In the United States, you would probably get support calls complaining of spelling errors in the application.

Windows Forms Applications

Windows Forms applications have their own considerations. You must decide whether users are allowed to specify which language and region the application should use. Certainly, giving users a choice is always good from a functionality point of view. If you use the .NET Framework dialog controls (including `MessageBox`), however, you might not be able to deliver on this promise. These controls draw resources from the operating system and the .NET Framework Language Packs. Consequently, if your application is running on a Spanish version of Windows with a Spanish .NET Framework Language Pack, the dialogs will be in Spanish. This is a great time saver if your application's user interface should be in Spanish. However, if you have given users a choice of what language the application should use, they could choose a different language from the operating system (say, French), making the user interface schizophrenic (part would be in French, and part would be in Spanish).

If you decide to give your users a choice, you will also need to decide how users specify that choice. You might choose to let users specify this in the Regional and Language Options Control Panel applet. This is convenient, but you should also consider what would happen if they change this setting while the application is running. An alternative is to offer some menu setting inside the application. If you do this, you should consider whether a change to the language here means that the setting applies immediately or when the application next starts. If it applies immediately, you will need a way of refreshing the entire application's user interface (see Chapter 4, "Windows Forms Specifics," for a number of solutions).

If your application is a Windows Forms 2.0 application and you intend to use `ClickOnce` to deploy and update your application, you should also consider whether the `ClickOnce` user interface should be localized. The question here is this:

Is the user interface of the deployment and update mechanism separate from that of the application? For example, if you write a Spanish application, should the `ClickOnce` interface also be in Spanish? If the answer is yes you will also need to localize `ClickOnce` (see Chapter 4).

ASP.NET Applications

The single biggest factor affecting the internationalization of ASP.NET applications is whether you use Visual Studio 2003 or Visual Studio 2005. Visual Studio 2005 has considerable support, whereas Visual Studio 2003 offers no support beyond what the .NET Framework already offers. That is not to say that you cannot internationalize ASP.NET 1.1 applications, just that such internationalization is a manual process with no additional help from the tools.

You also must decide how users will specify their language and region. A common approach to this problem is to adopt users' language preferences from their browser (this information is passed in the HTTP header of all requests). A refined variation on this theme is to adopt users' initial settings from the browser and then let the user override the settings in a user profile. You should also decide whether users should be allowed to specify a different language and region for the user interface to the language and region for their cultural formatting. In an ASP.NET application, such different settings will lead to a schizophrenic user interface if you use the Calendar control. For example, if the user specifies that the user interface is in Spanish, but the cultural formatting is United States, a page would display in Spanish but a Calendar control on the same page would display in English.

You also need to consider exactly what gets handed over to your translator to translate. Do you give the translators just the resources or the aspx pages as well? Handing over the aspx pages introduces new problems, not the least of which is that these critical files have to be locked while with the translator (which could be days or weeks). A better solution is to ensure that all static text has been removed from your aspx pages and placed in resources. Of course, this means that the issue of formatting is now outside the hands of the translator and in the hands of the developers.

If your application caches pages (for better performance), your caching process must be language/regionaware. If it is not, the first user to access a page will have the page cached, and subsequent users will get the same cached page even if they use a different language.

You also must decide whether to use absolute or relative positioning of controls. By default, Visual Studio 2003 uses absolute positioning and Visual Studio 2005 uses relative positioning. Pros and cons exist for both approaches, but in internationalization terms, consider that absolute positioning causes significant problems for localization; this is not only because controls do not automatically reposition and resize, but also because mirroring does not work when controls are absolutely positioned. As a general rule, you should use relative positioning in internationalized applications; if you intend to use mirroring (for right-to-left languages such as Arabic, Hebrew, and Persian [Farsi]), you should consider this a requirement.

Globalization

Globalization is the process of adapting an application so that it does not have cultural preconceptions. The most common example is the different date formats used throughout the world. An application that displays dates as "MM/dd/yy" assumes the U.S. date format. Globalization is not limited to dates, however, and covers many lesser-known subjects. How many of the following, for example, do you know?

Not all languages have a concept of upper case.

A conversion to upper case does not result in the same string in all cultures.

Not all regions use the Gregorian calendar.

Some regions use a 13-month calendar.

"AM" and "PM" suffixes are not "AM" and "PM" in all languages.

In some languages, month names take a different form when the month can be said to "own" a day.

Several regions have more than one way of sorting data.

Thai does not use spaces to separate words (words are not separated at all).

> The Windows Program Files folder isn't always called "Program Files" on non-English versions of Windows.

The good news is that the .NET Framework understands all these issues (and many more), and the advice is quite clear: Always use the .NET Framework classes. This will solve a very large part of the globalization issues of your application. You need a very good reason to use your own classes instead of those provided by the .NET Framework. For more details, see Chapter 6, "Globalization."

One decision that you might need to make early in the development of your application is how to refer to cultures. A culture is a language or a language-and-region combination. If you need to store a culture or a list of cultures (for, say, a user's preferred settings), you must decide whether to store them as names or as numbers. If you are using the .NET Framework 2.0, the answer is simple: Always store culture identifiers as strings (because strings can identify all cultures, regardless of whether they are custom cultures, cultures for alternate sort orders, or any other culture). Unfortunately, if you are using the .NET Framework 1.1, there is no easy answer to this; your decision will depend upon the functionality your application requires. If you use alternate sort orders, you must store these cultures as numbers. If you don't use alternate sort orders, you should store cultures as strings. Of course, you could store alternate sort orders as numbers and all other cultures as strings; in that case, you would need to be able to read and write cultures as both names and numbers.

Localization

Localization is the process of creating resources for a given language or a language and region. When you create a French version of your application, you have localized it. Localization brings with it some more decisions.

You must decide what language you want your exceptions to be thrown in. You might at first think that exceptions should be in the same language as the user interface. Alternatively, you might take the approach that all exceptions should be handled by an application-wide handler, with the exception logged and sent to the support team, and a different (localized) message displayed to the user. In this scenario, you might decide that all exceptions should be in the original developer's language. After all, if your developers are English, how much use is an error message in German? In an ASP.NET application, you might opt for this second approach. In a Windows Forms application, you might adopt both approaches. Consider that an "Unable to find the specified file" exception might be of immediate use to the user, so it could be argued that it should be localized. However, a "Value does not fall within the expected range" exception is of no value to the user (unless the application is a developer tool) and is intended for the developer, so it should probably not be localized.

You also must consider how much or how little your translator will do. Primarily, should the translator redesign forms or Web pages? When text is translated, it is often larger than the original text. You need to decide how to cope with this expansion. The traditional approach has been to let translators redesign forms (moving controls out of the way and resizing controls, for example) so that collisions do not occur. An alternative approach is to design forms with room for expansion. The .NET Framework 2.0 offers an additional solution that allows Windows Forms applications to behave in a similar way to HTML, and bump controls out of the way or wrap text as necessary.

Yet another consideration is how to translate hotkeys. Many controls have hotkey assignments to allow users to jump to controls using a set of keystrokes. For example, Alt+C might jump to a "Country" text box. If the form is localized and the prompt is now, for example, "Pays" instead of "Country," clearly, a hotkey of Alt+C is meaningless. The decision that you need to make is whether to ask your translator to make these hotkey assignments or whether to automate assignments of hotkeys in code. The problem with placing this in the

hands of the translator is that the translator might not always be able to see hotkey assignment clashes if he or she cannot see the form on which the text lies. The problem with automating this process is that the most logical hotkey is not always the hotkey that gets assigned.

Machine Translation

Machine translators are programs that translate text. They can be Windows programs, Web services, or Web sites. They provide a way of automating the translation of your application. You must decide whether to use them, and, if so, how much to use them. Start with the premise that machine translation will not be perfect. It will make mistakes. If you use a machine translation, you must also follow up the machine translation with a review by a human translator who can catch all the machine's translation mistakes. That said, many of the translations will be correct, and this approach can significantly reduce the amount of time that it takes for the overall translation effort (and, therefore, the cost of that effort). In addition, this approach is very useful for getting a demo version of the application up and running with little or no translation cost. Consider that if you demo the Greek version of your application using a machine translation, it will be indistinguishable from a human translation to someone who cannot read Greek.

Another very important use for machine translation is to create a pseudo translation of your application. A pseudo translation is a translation to a pseudo language that looks very similar to the original language but that can also be identified as not the original language. The benefit of a pseudo translation is that it allows developers to test that the application is localizable while still being able to read the prompts in the application so that it can be used without having to learn another language. This approach is a very useful strategy in testing an internationalized application. However, it requires you to choose a culture that is used for the pseudo translation. If you are using the .NET Framework 2.0, the best solution to this is to create a custom culture specifically for this purpose (this is one of the examples in Chapter 11, "Custom Cultures"). If you are using .NET Framework 1.1, then the best solution is to hijack an existing culture.

Resource Administration

The basic process of maintaining resourcesthat is, adding, editing, and deleting text and imagesis simple. Visual Studio provides a Resource Editor (with lesser functionality in Visual Studio 2003) to maintain resources. However, in a large application, the maintenance of these resources is a significant part of the development process. The problem is that if you delete an entry from one language, you will probably want to delete the same entry from all the others. The same can also be true for adding and editing resources. It is easy for resources to get out-of-synch with each other. A tool is needed to help manage the numerous changes made to resources and to keep those resources in synch. Chapter 12 provides the Resource Administrator for this purpose. In addition to this, unlike the Visual Studio Resource Editor, it is not restricted to using resx files. Furthermore, it can automatically translate resources as they are added. So if you add a new string for a `MessageBox` for "Collect additional company information?", the French resource is also updated with a string for "Rassemblez l'information additionnelle de compagnie?"

Testing

I have already pointed out that the most accurate testing is performed only by running the application on the language version of the intended operating system, and also that using a pseudo translation is a great way to find localizability issues before the application is released beyond the developers. Another tool in the testing

armory is FxCop. Throughout this book, I make various recommendations or point out choices that you might want to adopt in the internationalization of your application. FxCop is a static analysis tool included with Visual Studio 2005 Team System and available for download for the .NET Framework 1.1 and 2.0. It enables you to create rules that enforce the decisions and choices you have made. Chapter 13, "Testing Internationalization Using FxCop," contains a collection of rules that enforce the recommendations made in this book. You can enable or disable individual rules according to your decisions.

Translation

One of the decisions you must make with regard to translating your application is how to provide immediate (or very fast) visual feedback of the translator's translations. For translators to achieve a speed of translation that is financially viable, they will sometimes translate long lists of strings one after another. This gives good productivity for the translator, but the translations occur out of context. That is, the translator does not see the context in which the strings are used. Often the context in which the string is used (not to mention its length) can change the vocabulary or grammar used in the translation. If the translator simply translates a list of strings and does not get to see the translations used within the application until the strings are returned to the developers, reintegrated into the application, and sent back to the translator, there can be an unnecessarily high number of translation iterations. One solution is to provide feedback to the translator as soon as possible, or even to allow the translator to translate text in context.

To provide feedback as soon as possible, you have a number of strategies, which are investigated at length in Chapter 14, "The Translator." Clearly, using a database for resources is a good solution in this respect because the resources are held in only a single format and do not need to be converted from one format to another (as resx files do). However, Chapter 14 investigates other solutions.

In a Windows Forms application, you can use the Windows Resource Localization Editor (WinRes). This is a forms designer that has similar functionality to the Visual Studio Forms Designer, which your translator can use to translate strings and redesign forms. This tool has the benefit that it allows translators to see their translations in context. WinRes has pros and cons (see Chapters 4 and 14)you must decide whether the pros outweigh the cons.

Where Are We?

We are looking down on the mountain with a high-level map in our hands. I have presented many of the issues on which decisions must be made. The details behind those decisions are found in the remaining chapters of this book. The technical issues that get us from the bottom of the mountain to the summit can also be found within these chapters. Good luck on your ascent.

Chapter 2. Unicode, Windows, and the .NET Framework

OUR INDUSTRY HAS A HUMBLE BEGINNING. The American Standard Code for Information Interchange (ASCII) character set that underlies the American National Standards Institute (ANSI) upon which DOS was built included upper and lower characters for the English language alone. We have come a long way since then. In this chapter, you will learn about the operating environment in which your .NET