Chinese, Japanese, Korean & Vietnamese Computing

2nd Edition
Broader Unicode Coverage

CJKV 中日韓越 *Information Processing*

O'REILLY®

Ken Lunde

Computers/Data Processing

O'REILLY®

CJKV Information Processing

First published a decade ago, *CJKV Information Processing* quickly became the unsurpassed source of information on processing text in Chinese, Japanese, Korean, and Vietnamese. It has now been thoroughly updated to provide web and application developers with the latest techniques and tools for disseminating information directly to audiences in East Asia. This second edition reflects the considerable impact that Unicode, XML, OpenType, and operating systems such as Windows XP, Vista, Mac OS X, and Linux have had on East Asian text processing in recent years. With this book, you will:

- **Learn about CJKV writing systems and scripts, and their transliteration methods**
- **Explore trends and developments in character sets and encodings, particularly Unicode**
- **Examine the world of typography, specifically how CJKV text is laid out on a page**
- **Learn information-processing techniques, such as code conversion algorithms, and apply them using different programming languages**
- **Process CJKV text using different platforms, text editors, and word processors**
- **Become more informed about CJKV dictionaries and dictionary software**
- **Manage CJKV content and presentation when publishing in print or for the Web**

Internationalizing and localizing applications is paramount today—especially for audiences in East Asia, the fastest-growing segment of the computing world. *CJKV Information Processing*, Second Edition, will help you develop web and other applications effectively in a field that many find difficult to master.

"Our world is naturally complex, and the East Asian writing systems are perhaps doubly so. Their scripts pose challenges for all forms of communication and publishing, whether it is for the Web or for print. I highly recommend Ken Lunde's book as a companion and guide for navigating the intricacies of East Asian text processing."

—Dr. John Warnock, Co-founder, Adobe Systems Incorporated

Ken Lunde is a senior computer scientist in CJKV Type Development at Adobe Systems Incorporated. He has a Ph.D. in linguistics from the University of Wisconsin–Madison.

www.oreilly.com

US $59.99 **CAN $59.99**

ISBN: 978-0-596-51447-1

55999

9 780596 514471

SECOND EDITION

CJKV Information Processing

Ken Lunde

O'REILLY®

Běijīng · Boston · Farnham · Sebastopol · Tōkyō

CJKV Information Processing, Second Edition
by Ken Lunde

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472 USA.

Editor: Julie Steele
Production Editors: Ken Lunde and Rachel Monaghan
Copyeditor: Mary Brady
Proofreader: Genevieve d'Entremont
Indexer: Ken Lunde
Production Services: Ken Lunde
Cover Designer: Karen Montgomery
Interior Designer: David Futato
Illustrator: Robert Romano

Printing History:

January 1999:	First Edition.
December 2008:	Second Edition.

ISBN: 978-0-596-51447-1
[LSI]

This book is dedicated to the four incredible women who have touched—and continue to influence—my life in immeasurable ways:

My mother, Jeanne Mae Lunde*, for bringing me into this world and for putting me on the path I am on.*

My mother-in-law, Sadae Kudo*, for unknowingly bringing together her daughter and me.*

My wife, friend, and partner, Hitomi Kudo*, for her companionship, love, and support, and for constantly reminding me about what is truly important.*

Our daughter, Ruby Mae Lunde*, for showing me that actions and decisions made today impact and influence our future generations.*

I shall be forever in their debt….

Contents

Foreword

The noncommittal phrase *information processing* covers a lot of ground, from generating mailing lists and tabulating stock exchange transactions to editing and typesetting Lady Murasaki's *Tale of Genji*, the meditations of Lao Zi, or the poems of Han Shan. There is a lot of information in the world, and it is stored, handled, and processed in a lot of different ways.

The oldest human writing systems known—including Sumerian, early Egyptian, Chinese, and early Mayan—seem to have sprung up independently. They thrived in different places, serving unrelated languages, and they look thoroughly different from one another, but they all have something in common: they all employ large numbers of signs for meanings, supplemented with signs for sounds. The only such script that has survived in common use to the present day is Han Chinese. All the other scripts now in general use for writing natural human languages are essentially confined to the writing of sounds. They are all syllabic, consonantal, or alphabetic.

Here in the West, we often speak of Chinese as if it were a single language. Once upon a time, perhaps it was—but for thousands of years things have been more complicated than that. There are now more than a dozen Chinese languages, each with several dialects, spoken in China and by people of Chinese origin living elsewhere in the world. The most successful member of the group, often called Mandarin, is spoken as a first or second language by roughly a billion people. Add all those who speak at least one of the other Chinese languages (such as Yuè, the common language of Hong Kong, or Mĭn Nán, the most common language in Taiwan, or Wú, which is common in Shanghai), and the total gets closer to a billion and a half. Add also the speakers of Japanese, Korean, and Vietnamese, whose languages belong to different families but whose scripts and literatures have a long and rich association with Chinese, and the number is larger yet: about 25% of the human population as a whole.

You can see from this alone why a book giving clear and thorough guidance on the handling of text in Chinese, Japanese, Korean, and Vietnamese might be important. But even if these scripts weren't part of daily life for a quarter of humanity, there would be good reasons to study them. Compared to the Latin alphabet, they are wonderfully complex and polymorphous. That human speech can be recorded and transmitted in all these different ways tells us something about language, something about the mind, and something about how many different ways there are to be a human being.

There is always a certain tension between written and spoken language, because speech continues to change while writing is used to preserve it. Wherever reading and writing are normal parts of daily life, there are things that people know how to say but aren't sure how to write, and things that they know how to write, and may write fairly often, but might never find any occasion to say. (*Sincerely yours* is one example.) If we wrote only meanings and no sounds, the gulf between speech and writing would be wider, but the tension between them would often be less. This is what happens, in fact, when we use mathematical notation. We write basically nothing but symbols for meanings, along with a lot of punctuation to keep the meanings from getting confused, and what we have written can be read with equal ease, and with equal accuracy, in English, Hungarian, Arabic, or Chinese. It is not really possible to write spoken language in this way—but it is possible to write a close correlative. We can write (as logicians do when they use symbolic logic) the sequence of meanings that a string of spoken sentences would carry; then we can read what we have written by pronouncing the names of those meanings in any language we choose and filling in the holes with whatever inflections, links, and lubricants our spoken grammar requires. That in fact is how Japanese, Vietnamese, and Korean were first written: using Chinese characters to represent the meanings, then reading back the meanings of these Chinese signs in a language other than Chinese. (Most Chinese languages other than classical Mandarin are written even now in the same way.)

In time, the Japanese devised their delicate and supple syllabic scripts (*hiragana* and *katakana*) as a supplement to *kanji*, which are Han Chinese glyphs used to write Japanese; the Koreans devised their ingeniously analytical Hangul script, in which alphabetic information is nested into discrete syllabic units; and the Vietnamese, after spending a thousand years building their own large lexicon of redefined Chinese glyphs and new glyphs on the Chinese model, embraced a complex variant of the Latin alphabet instead. But in all three cultures, the old associations with Chinese script and Chinese literature run deep and have never disappeared. The connection is particularly vivid in the case of Japanese, whose script is what a linguistic geologist would call a kind of breccia or conglomerate: chunks of pure Chinese script and angular syllabics (and often chunks of Latin script as well) cemented into a matrix of cursive syllabics.

Ken Lunde is an enthusiast for all these complications and an expert on their electronic outcome—expert enough that his book is relied on by professionals and amateurs alike, in both Asia and the West.

Many North Americans and Europeans have written books about the Orient, but very few of those books have been translated into Asian languages, because so few of them can tell Asians anything about themselves. Once in a while, though, outsiders really know their stuff, and insiders see that this is so. The first edition of this book (which ran to 1,100 pages) was published in California in 1999. It was recognized at once as the definitive work in the field and was promptly translated into both Chinese and Japanese. Its shorter predecessor, *Understanding Japanese Information Processing*—Ken Lunde's first book, published in 1993, when he was only 28—had been greeted the same way: it was recognized as the best book of its kind and promptly published, unabridged, in Japanese. As a reader, I am comforted by those endorsements. The subject is, after all, complex. Some would call it daunting. I know how useful this book has been to me, and it pleases me to know that native speakers of Chinese and Japanese have also found it useful.

Robert Bringhurst

Quadra Island, British Columbia · 21 August 2008

Preface

Close to 16 years have elapsed since *Understanding Japanese Information Processing* was published, and perhaps more importantly, 10 years have gone by since *CJKV Information Processing* was first published. A lot has changed in those 10 years. I should point out that I was first inspired to undertake the initial "CJKV" expansion sometime in 1996, during a lengthy conversation I had at a Togo's near the UC Berkeley campus with Peter Mui, my editor for *Understanding Japanese Information Processing*.

Join me in reading this thick tome of a book, which also serves as a reference book, and you shall discover that "CJKV" (Chinese, Japanese, Korean, and Vietnamese) will become a standard term in your arsenal of knowledge. But, before we dive into the details, allow me to discuss some terms with which you are no doubt familiar. Otherwise, you probably would have little need or desire to continue reading.

Known to more and more people, *internationalization*, *globalization*, and *localization* seem to have become household or "buzz" words in the field of computing and software development, and have also become very hot topics among high-tech companies and researchers due to the expansion of software markets to include virtually all parts of the planet. This book is specifically about CJKV-enabling, which is the adaptation of software for one or more CJKV locales. It is my intention that readers will find relevant and useful CJKV-enabling information within the pages of this book.

Virtually every book on internationalization, globalization, or localization includes information on character sets and encodings, but this book is intended to provide much more. In summary, it provides a brief description of writing systems and scripts, a thorough background of the history and current state of character sets, detailed information on encoding methods, code conversion techniques, input methods, keyboard arrays, font formats, glyph sets, typography, output methods, algorithms with sample source code, tools that perform useful information processing tasks, and how to handle CJKV text in the context of email and for web and print publishing. Expect to find plenty of platform-independent information and discussions about character sets, how CJKV text is encoded and handled on a number of operating systems, and basic guidelines and tips for developing software targeted for CJKV markets.

Now, let me tell you what this book is *not* about. Don't expect to find out how to design your own word-processing application, how to design your own fonts for use on a computer (although I provide sources for such tools), or how to properly handle formats for CJKV numerals, currency, dates, times, and so on. This book is not, by any stretch of the imagination, a complete reference manual for internationalization, globalization, or localization, but should serve remarkably well as a companion to such reference works, which have fortunately become more abundant.

It is my intention for this book to become the definitive source for information related to CJKV information processing issues.* Thus, this book focuses heavily on how CJKV text is handled on computer systems in a very platform-independent way, with an emphasis or bias toward Unicode and other related and matured technologies. Most importantly, everything that is presented in this book can be programmed, categorized, or easily referenced.

This book was written to fill the gap in terms of information relating to CJKV information processing, and to properly and effectively guide software developers. I first attempted to accomplish this over the course of several years by maintaining an online document that I named *JAPAN.INF* (and entitled *Electronic Handling of Japanese Text*). This document had been made publicly available through a number of FTP sites worldwide, and had gained international recognition as the definitive source for information relating to Japanese text handling on computer systems. *Understanding Japanese Information Processing* excerpted and further developed key information contained in *JAPAN.INF*. However, since the publication of *Understanding Japanese Information Processing* in 1993, *JAPAN.INF*, well, uh, sort of died. Not a horrible death, mind you, but rather to prepare for its reincarnation as a totally revised and expanded online document that I entitled *CJK.INF* (the CJK analog to *JAPAN.INF*). The work I did on *CJK.INF* helped to prepare me to write the first edition of this book, which provided updated material plus significantly more information about Chinese, Korean, and Vietnamese, to the extent that granting the book a new title was deemed appropriate and necessary. The second edition, which you have in your hands, represents a much-needed update, and I hope that it becomes as widely accepted and enjoyed as the first edition.

Although I have expended great effort to provide sufficient amounts of information for Chinese, Japanese, Korean, and Vietnamese computing, you may feel that some bias toward Japanese still lingers in many parts of this book. Well, if your focus or interest happens to be Japanese, chances are you won't even notice. In any case, you can feel at ease knowing that almost everything discussed in this book can apply equally to all of these languages. However, the details of Vietnamese computing in the context of using ideographs are still emerging, so its coverage is still somewhat limited and hasn't changed much since the first edition.

* The predecessor of this book, *Understanding Japanese Information Processing*, which had a clear focus on Japanese (hence its title) apparently became the definitive source for Japanese information processing issues, and was even translated into Japanese.

What Changed Since the First Edition?

Several important events took place during the 10 years since the first edition was published. These events could be characterized as technologies that were in the process of maturing, and have now fully matured and are now broadly used and supported.

First and foremost, *Unicode* has become the preferred way in which to represent text in digital format, meaning when used on a computer. Virtually all modern OSes and applications now support Unicode, and in a way that has helped to trivialize many of the complexities of handling CJKV text. As you read this book, you may feel a bias toward Unicode. This is for a good reason, because unless your software embraces Unicode, you are going down the wrong path. This also means that if you are using an application that doesn't handle Unicode, chances are it is outdated, and perhaps a newer version exists that supports Unicode.

Second, *OpenType* has become the preferred font format due to its cross-platform nature, and how it allows what were once competing font formats, Type 1 and TrueType, to exist in harmony. Of course, OpenType fonts support Unicode and have strong multilingual capabilities. OpenType fonts can also provide advanced typographic functionality.

Third, *PDF* (*Portable Document Format*) has become the preferred way in which to publish for print, and is also preferred for the Web when finer control over presentation is desired. In addition to supporting Unicode, PDF encapsulates documents in such a way that they can be considered a reliable digital master, and the same file can be used for displaying and printing. In short, PDF has become the key to the modern publishing workflow.

Last but not least, the *Web* itself has matured and advanced in ways that could not be predicted. The languages used to build the Web, which range from languages that describe the content and presentation of web documents, such as CSS, HTML, XHTML, and XML, to scripting languages that enable dynamic content, have matured, and they all have one thing in common: they all support Unicode. In case it is not obvious, Unicode will be a recurring theme throughout this book.

Audience

Anyone interested in how CJKV text is processed on modern computers will find this book useful, including those who wish to enter the field of CJKV information processing, and those who are already in the field but have a strong desire for additional reference material. This book will also be useful for people using any kind of computer and any type of operating system, such as FreeBSD, the various Linux distributions, Mac OS X, MS-DOS, Unix, and the various flavors of Windows.

Although this book is specifically about CJKV information processing, anyone with an interest in creating multilingual software or a general interest in I18N (*internationalization*), G11N (*globalization*), or L10N (*localization*) will learn a great deal about the issues involved in handling complex writing systems and scripts on computers. This is particularly true for people interested in working with CJKV text. Thankfully, information relating to the CJKV-enabling of software has become less scarce.

I assume that readers have little or no knowledge of a CJKV language (Chinese, Japanese, Korean, or Vietnamese) and its writing system. In Chapter 2, *Writing Systems and Scripts*, I include material that should serve as a good introduction to CJKV languages, their writing systems, and the scripts that they use. If you are familiar with only one CJKV language, Chapter 2 should prove to be quite useful for understanding the others.

Conventions Used in This Book

Kanji, hanzi, hanja, kana, hiragana, katakana, hangul, jamo, and other terms will come up, time and time again, throughout this book. You will also encounter abbreviations and acronyms, such as ANSI, ASCII, CNS, EUC, GB, GB/T, GBK, ISO, JIS, KS, and TCVN. Terms, abbreviations, and acronyms—along with many others words—are usually explained in the text, and again in Appendix D, *Glossary*, which I encourage you to study.

When hexadecimal values are used in the text for lone or single bytes, and to remove any ambiguity, they are prefixed with `0x`, such as `0x80`. When more than one byte is specified, hexadecimal values are instead enclosed in angled brackets and separated by a space, such as <`80 80`> for two instances of `0x80`. Unicode scalar values follow the convention of using the `U+` prefix followed by four to six hexadecimal digits. Unicode encoding forms are enclosed in angled brackets and shown as hexadecimal code units, except when the encoding form specifies byte order, in which case the code units are further broken down into individual bytes. `U+20000`, for example, is expressed as <`D840 DC00`> in UTF-16, but as <`D8 40 DC 00`> in UTF-16BE (UTF-16 big-endian) and as <`40 D8 00 DC`> in UTF-16LE (UTF-16 little-endian). Its UTF-8 equivalent is <`F0 A0 80 80`>. The angled brackets are generally omitted when such values appear in a table. Furthermore, Unicode sequences are expressed as Unicode scalar values separated by a comma and enclosed in angled brackets, such as <`U+304B`, `U+309A`>.

Decimal values are almost always clearly identified as such, and when used, appear as themselves without a prefix, and unenclosed. For those who prefer other notations, such as binary or octal, Appendix B, *Notation Conversion Table*, can be consulted to convert between all four notations.

Throughout this book I generically use short suffixes such as "J," "K," "S," "T," "V," and "CJKV" to denote locale-specific or CJKV-capable versions of software products. I use these suffixes for the sake of consistency, and because software manufacturers often change the way in which they denote CJKV versions of their products. In practice, you may instead encounter the suffix 日本語版 (*nihongoban*, meaning "Japanese version"), the prefix "Kanji," or the prefix 日本語 (*nihongo*, meaning "Japanese") in Japanese product names. For Chinese software, 中文 (*zhōngwén*, meaning "Chinese") is a common prefix. I also refrain from using version numbers for software described in this book (as you know, this sort of information becomes outdated very quickly). I use version numbers only when they represent a significant advancement or development stage in a product.

References to "China" in this book refer to the People's Republic of China (PRC; 中华人民共和国 *zhōnghuá rénmín gònghé guó*), also commonly known as Mainland China. References to "Taiwan" in this book refer to the Republic of China (ROC; 中華民國 *zhōnghuá mínguó*). Quite often this distinction is necessary.

Name ordering in this book, when transliterated in Latin characters, follows the convention that is used in the West—the given name appears first, followed by the surname. When the name is written using CJKV characters—in parentheses following the transliterated version—the surname appears first, followed by the given name.

"ISO 10646" and "Unicode" are used interchangeably throughout this book. Only in some specific contexts are they different.

Italic is used for pathnames, filenames, program names, new terms where they are defined, newsgroup names, and web addresses, such as domain names, URLs, and email addresses.

`Constant width` is used in examples to illustrate output from commands, the contents of files, or the text of email messages.

`Constant width bold` is used in examples to indicate commands or other text that should be typed literally by the user; occasionally it is also used to distinguish parts of an example.

The `%` (percent) character is used to represent the Unix shell prompt for Unix and similar command lines.

Footnotes are used for parenthetical remarks and for providing URLs. Sometimes what is written in the text proper has been simplified or shortened for the purpose of easing the discussion or for practical reasons (especially in Chapter 2 where I introduce the many CJKV writing systems), and the footnotes—usually, but not always—provide additional details.

How This Book Is Organized

Let's now preview the contents of each chapter in this book. Don't feel compelled to read this book linearly, but feel free to jump around from section to section. Also, the index is there for you to use.

Chapter 1, *CJKV Information Processing Overview*, provides a bird's eye overview of the issues that are addressed by this book, and is intended to give readers an idea of what they can expect to learn. This chapter establishes the context in which this book will become useful in your work or research.

Chapter 2, *Writing Systems and Scripts*, contains information directly relating to CJKV writing systems and their scripts. Here you will learn about the various types of characters that compose CJKV texts. This chapter is intended for readers who are not familiar with the Chinese, Japanese, Korean, or Vietnamese languages (or who are familiar with only one or two of those languages). Everyone is bound to learn something new in this chapter.

Chapter 3, *Character Set Standards*, describes the two classes of CJKV character set standards: coded and noncoded. Coded character set standards are further divided into two classes: national and international. Comparisons are also drawn between CJKV character set standards, and the coverage of Unicode is extensive.

Chapter 4, *Encoding Methods*, contains information on how the character set standards described in Chapter 3 are encoded on computer systems. Emphasis is naturally given to the encoding forms of Unicode, but information about legacy encoding methods is also provided. Encoding is a complex but important step in representing and manipulating human-language text in a computer. Other topics include software for converting from one CJKV encoding to another, and instructions on how to repair damaged CJKV text files.

Chapter 5, *Input Methods*, contains information on how CJKV text is input. First I discuss CJKV input in general terms, and then describe several specific methods for entering CJKV characters on computer systems. Next, we move on to the hardware necessary for CJKV input, specifically keyboard arrays. These range from common keyboard arrays, such as the QWERTY array, to ideograph tablets containing thousands of individual keys.

Chapter 6, *Font Formats, Glyph Sets, and Font Tools*, contains information about bitmapped and outline font formats as they relate to CJKV, with an emphasis toward OpenType. The information presented in this chapter represents my daily work at Adobe Systems, so some of its sections may suffer from excruciating detail, which explains the length of this chapter.

Chapter 7, *Typography*, contains information about how CJKV text is properly laid out on a line and on a printed page. Merely having CJKV fonts installed is not enough—there are rules that govern where characters can and cannot be used, and how different character classes behave, in terms of spacing, when in proximity. The chapter ends with a description of applications that provide advanced page composition functionality.

Chapter 8, *Output Methods*, contains information about how to display, print, or otherwise output CJKV text. Here you will find information relating to the latest printing and display technologies.

Chapter 9, *Information Processing Techniques*, contains information and algorithms relating to CJKV code conversion and text-handling techniques. The actual mechanics are described in detail, and, where appropriate, include algorithms written in C, Java, and other programming languages. Though somewhat dated, the chapter ends with a brief description of three Japanese code-processing tools that I have written and maintained over a period of several years. These tools demonstrate how the algorithms can be applied in the context of Japanese.

Chapter 10, *OSes, Text Editors, and Word Processors*, contains information about operating systems, text editors, and word processors that are CJKV-capable, meaning that they support one or more CJKV locale.

Chapter 11, *Dictionaries and Dictionary Software*, contains information about dictionaries, both printed and electronic, that are useful when dealing with CJKV text. Also included are tips on how to more efficiently make use of the various indexes used to locate ideographs in dictionaries.

Chapter 12, *Web and Print Publishing*, contains information on how CJKV text is best handled electronically over networks, such as when using email clients. Included are tips on how to ensure that what you send is received intact, as well as information about the Internet domains that cover the CJKV locales. Web and print publishing, through the use of HTML (*HyperText Markup Language*), XML (*Extensible Markup Language*), and PDF (*Portable Document Format*) are also discussed in detail.

Appendix A, *Code Conversion Tables*, provides a code conversion table between decimal Row-Cell, hexadecimal ISO-2022, hexadecimal EUC, and hexadecimal Shift-JIS (Japanese-specific) codes. Also included is an extension that handles the Shift-JIS user-defined range.

Appendix B, *Notation Conversion Table*, lists all 256 8-bit byte values in the four common notations: binary, octal, decimal, and hexadecimal.

Appendix C, *Perl Code Examples*, provides Perl equivalents of many algorithms found in Chapter 9—along with other goodies.

Appendix D, *Glossary*, defines many of the concepts and terms used throughout this book (and other books).

Finally, the *Bibliography* lists many useful references, many of which were consulted while writing this book.

Although not included in the printed version of this book, the following appendixes are provided as downloadable and printable PDFs. This book does include placeholder pages for them, which serve to specify their URLs.

Appendix E, *Vendor Character Set Standards*, is reference material for those interested in vendor-specific extensions to CJKV character set standards. To a great extent, Unicode has effectively deprecated these standards.

Appendix F, *Vendor Encoding Methods*, is reference material for those interested in how the vendor character sets in Appendix E are encoded.

Appendix G, *Chinese Character Sets—China*, provides character set tables, character lists, mapping tables, and indexes that relate to standards from China.

Appendix H, *Chinese Character Sets—Taiwan*, provides character set tables, character lists, mapping tables, and indexes that relate to standards from Taiwan.

Appendix I, *Chinese Character Sets—Hong Kong*, provides character set tables and character lists that relate to standards from Hong Kong.

Appendix J, *Japanese Character Sets*, provides character set tables, character lists, mapping tables, and indexes that relate to standards from Japan.

Appendix K, *Korean Character Sets*, provides character set tables, character lists, mapping tables, and indexes that relate to standards from Korea, specifically South Korea.

Appendix L, *Vietnamese Character Sets*, provides character set tables that relate to standards from Vietnam.

Appendix M, *Miscellaneous Character Sets*, provides character set tables for standards such as ASCII, ISO 8859, EBCDIC, and EBCDIK.

Using Code Examples

This book is here to help you get your job done. In general, you may use the code in this book in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM or DVD of examples from this book does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*CJKV Information Processing*, Second Edition, by Ken Lunde. Copyright 2009 O'Reilly Media, Inc., 978-0-596-51447-1."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
USA
800-998-9938 (in the United States or Canada)
+1-707-829-0515 (international/local)
+1-707-829-0104 (facsimile)

There is a web page for this book, which lists errata, examples, or any additional information. You can access this page at:

http://oreilly.com/catalog/9780596514471/

To comment or ask technical questions about this book, send email to:

bookquestions@oreilly.com

Safari® Books Online

When you see a Safari Books Online icon on the cover of your favorite technology book, that means the book is available online through the O'Reilly Network Safari Bookshelf.

Safari offers a solution that's better than e-books. It's a virtual library that lets you easily search thousands of top tech books, cut and paste code samples, download chapters, and find quick answers when you need the most accurate, current information. Try it for free at *http://safaribooksonline.com/*.

Acknowledgments

To write a book or reference work this thick requires interaction with and cooperation from people across our planet. It is not possible for me to list all the people who have helped or guided me over the years—there are literally hundreds.

In some cases, people simply come to me for help on a particular subject (that's what happens, I guess, when people are aware of your email address—to ensure that I will receive a ton of email, in various parts of this book you will find that my email address is *lunde@adobe.com*). Sometimes I may not know the answer, but the question usually inspires me to seek out the truth. The truth is out there.

The year 2008 marks 17 wonderful years at Adobe Systems, a company that provides me with daily CJKV-related challenges. Its always-advancing technologies and commitment to customers is what initially attracted me, and these are the qualities and values that keep me here. Besides, they let me display on the wall of my office the antelopes that I have harvested* annually in Wyoming since 2003. I firmly believe that "diversity in the workforce" is a door that swings both ways, and the antelopes that I display in my office serve as a tribute to that belief. Speaking of tributes, all aspects of the production of this book are a tribute to Adobe Systems' publishing technologies.

To all the people who have read and criticized my previous writings, tolerated my sometimes dull but otherwise charming personality at work, pointed out errors in my work, exchanged email with me for whatever reason, or otherwise helped me to grow and become a better person: *thank you!* You should know who you are.

Special thanks go to Tim O'Reilly (the president and founder of O'Reilly Media) and Peter Mui for believing in my first book, *Understanding Japanese Information Processing*, and to Peter for serving as its editor. It was Peter who encouraged me to expand it to cover the complete CJKV framework. Thanks go to Edie Freedman for sticking with my idea

* Harvesting is another way to express hunting.

of a blowfish for the cover.[*] Ron Bilodeau graciously helped me through the layout of the book, and nurtured my desire to embrace Adobe InDesign's particular paradigm. Robert Romano also deserves a lot of credit for the work he did on the figures that are found throughout this book. Julie Steele, my editor, continually pushed and prodded me to get this book done as close to schedule as possible. Mary Brady performed the copyedit, exposing various errors and oddities that crept from my fingers to the keyboard. Genevieve d'Entremont proofread the entire book, and discovered additional gems. Rachel Monaghan reviewed the index, and also provided production assistance.

Attempting to write a book of any length, while holding down a day job and doing so as an activity above and beyond it, takes a lot of effort and perseverance. I'd like to specifically thank David Lemon, Karen Catlin, and Digby Horner for their support and encouragement while writing this book. My colleagues and coworkers at Adobe Systems, especially those in Type Development, deserve the same recognition. Miguel Sousa, a coworker in Type Development, needs to be singled out and especially thanked for providing to me custom versions of the Minion Pro and Myriad Pro fonts that include the glyphs necessary for Pinyin transliteration and for implementing tabular hexadecimal digits, both of which are used extensively throughout this book. The tabular headecimal digits are especially nice, especially for a book such as this one.

The following individuals were incredibly helpful by reviewing various parts of this book, from specific pages to entire chapters, during the various stages of its prolonged development: Tom Bishop, Jim Breen, Robert Bringhurst, Seong Ah Choi (최성아), Richard Cook, Gu Hua (顾华), Paul Hackett, Jerry Hall, Taichi Kawabata (川幡太一), John Knightley, Tatsuo Kobayashi (小林龍生), Mark Leisher, David Lemon, Lu Qin (陸勤), Nat McCully, Dirk Meyer, Charles Muller, Eric Muller, Mihai Nita, Thomas Phinney, Read Roberts, Markus Scherer, Jungshik Shin (신정식), Miguel Sousa, Frank Tang (譚永鋒), Margie Vogel, Taro Yamamoto (山本太郎), and Retarkgo Yan (甄烱輝). I am indebted to each and every one of them for providing me with useful insights and inspirational ideas, and in some deserving cases, sharp criticism. I am, of course, ultimately responsible for any errors, omissions, or oddities that you may encounter while reading this book.

Finally, I wish to thank my wonderful parents, Vernon Delano Lunde and Jeanne Mae Lunde, for all of their support throughout the years; my son, Edward Dharmputra Lunde; my step-son, Ryuho Kudo (工藤龍芳); my beautiful daughter, Ruby Mae Lunde (工藤瑠美); and my beloved and caring wife, Hitomi Kudo (工藤仁美). I treasure the time that I spend with my parents, which includes varmint and larger game hunting with my father. Having my own family has great rewards.

[*] Michael Slinn made the astute observation that the *Babel Fish* would have been more appropriate as a cover creature for this book—according to Douglas Adams' *The Hitchhiker's Guide to the Galaxy*, you simply stick a Babel Fish in your ear, it connects with your brain, and you can suddenly understand all languages.

CHAPTER 1

CJKV Information Processing Overview

Here I begin this book by stating that a lot of mystique and intrigue surrounds how CJKV—*Chinese, Japanese, Korean, and Vietnamese*—text is handled on computer systems, ranging from handheld mobile devices to mainframe computers. Although I agree with there being sufficient intrigue, there is far too much mystery in my opinion and experience. Much of this is due to a lack of information, or perhaps simply due to a lack of information written in a language other than Chinese, Japanese, Korean, or Vietnamese. Nevertheless, many fine folks, such as you, the reader of this book, would like to know how this all works. To confirm some of your worst fears and speculations, CJKV text *does* require special handling on computer systems. However, it should not be very mysterious after having read this book. In my experience, you merely need to break the so-called *one-byte-equals-one-character* barrier—most CJKV characters are represented by more than a single byte (or, to put it in another way, more than eight bits).*

English information processing was a reality soon after the introduction of early computer systems, which were first developed in England and the United States. Adapting software to handle more complex scripts, such as those used to represent CJKV text, is a more recent phenomenon. This adaptation developed in various stages and continues today.

Listed here are several key issues that make CJKV text a challenge to process on computer systems:

- CJKV writing systems use a mixture of different, but sometimes related, scripts.
- CJKV character set standards enumerate thousands or tens of thousands of characters, which is orders of magnitude more than used in the West—*Unicode now includes more than 100,000 characters, adequately covering CJKV needs.*
- There is no universally recognized or accepted CJKV character set standard such as ASCII for writing English—*I would claim that Unicode has become such a character set, hence its extensive coverage in this book.*

* For a greater awareness of, and appreciation for, some of the complexities of dealing with multiple-byte text, you might consider glancing now at the section entitled “Byte Versus Character Handling” in Chapter 9.

- There is no universally recognized or accepted CJKV encoding method such as ASCII encoding—*again, the various Unicode encoding forms have become the most widely used encodings, for OSes, applications, and for web pages.*
- There is no universally recognized or accepted input device such as the QWERTY keyboard array—this same keyboard array, through a method of transliteration, is frequently used to input most CJKV text through reading or other means.
- CJKV text can be written horizontally or vertically, and requires special typographic rules not found in Western typography, such as spanning tabs and unique line-breaking rules.

Learning that the ASCII character set standard is not as universal as most people think is an important step. You may begin to wonder why so many developers assume that everyone uses ASCII. This is okay. For some regions, ASCII is sufficient. Still, ASCII has its virtues. It is relatively stable, and it forms the foundation for many character sets and encodings. UTF-8 encoding, which is the most common encoding form used for today's web pages, uses ASCII as a subset. In fact, it is this characteristic that makes its use preferred over the other two Unicode encoding forms, specifically UTF-16 and UTF-32.

Over the course of reading this chapter, you will encounter several sections that explain and illustrate some very basic, yet important, computing concepts, such as notation and byte order, all of which directly relate to material that is covered in the remainder of this book. Even if you consider yourself a seasoned software engineer or expert programmer, you may still find value in those sections, because they carry much more importance in the context of CJKV information processing. That is, how these concepts relate to CJKV information processing may be slightly different than what you had previously learned.

Writing Systems and Scripts

CJKV text is typically composed of a mixture of different scripts. The Japanese writing system, as an example, is unique in that it uses four different scripts. Others, such as Chinese and Korean, use fewer. Japanese is one of the few, if not the only, languages whose writing system exhibits this characteristic of so many scripts being used together, even in the same sentence (as you will see very soon). This makes Japanese quite complex, orthographically speaking, and poses several problems.*

Unicode's definitions of *script* and *writing system* are useful to consider. *Script* is defined as a collection of letters and other written signs used to represent textual information in one or more writing systems. For example, Russian is written with a subset of the Cyrillic script; Ukranian is written with a different subset. The Japanese writing system uses several scripts. *Writing system* is defined as a set of rules for using one or more scripts to write a particular language. Examples include the American English writing system, the British English writing system, the French writing system, and the Japanese writing system.

* *Orthography* is a linguistic term that refers to the writing system of a language.

The four Japanese scripts are *Latin characters*, *hiragana*, *katakana*, and *kanji* (collectively referred to as *ideographs* regardless of the language). You are already familiar with Latin characters, because the English language is written with these. This script consists of the upper- and lowercase Latin alphabet, which consists of the characters often found on typewriter keys. Hiragana and katakana are native Japanese syllabaries (see Appendix D for a definition of "syllabary"). Both hiragana and katakana represent the same set of syllables and are collectively known as *kana*. Kanji are ideographs that the Japanese borrowed from China over 1,600 years ago. Ideographs number in the tens of thousands and encompass meaning, reading, and shape.

Now let's look at an example sentence composed of these four scripts, which will serve to illustrate how the different Japanese scripts can be effectively mixed:

EUC等のエンコーディング方法は日本語と英語が混交しているテキストをサポートします。

In case you are curious, this sentence means "Encoding methods such as EUC can support texts that mix Japanese and English." Let's look at this sentence again, but with the Latin characters underlined:

<u>EUC</u>等のエンコーディング方法は日本語と英語が混交しているテキストをサポートします。

In this case, there is a single abbreviation, EUC (short for *Extended Unix Code*, which refers to a locale-independent encoding method, a topic covered in Chapter 4). It is quite common to find Latin characters used for abbreviations in CJKV texts. Latin characters used to transliterate Japanese text are called ローマ字 (*rōmaji*) or ラテン文字 (*raten moji*) in Japanese.

Now let's underline the katakana characters:

EUC等の<u>エンコーディング</u>方法は日本語と英語が混交している<u>テキスト</u>を<u>サポート</u>します。

Each katakana character represents one syllable, typically a lone vowel or a consonant-plus-vowel combination. Katakana characters are commonly used for writing words borrowed from other languages, such as English, French, or German. Table 1-1 lists these three underlined katakana words, along with their meanings and readings.

Table 1-1. Sample katakana

Katakana	**Meaning**	**Reading**[a]
エンコーディング	encoding	*enkōdingu*
テキスト	text	*tekisuto*
サポート	support	*sapōto*

a. The macron is used to denote long vowel sounds.

Note how their readings closely match that of their English counterparts, from which they were derived. This is no coincidence: it is common for the Japanese readings of borrowed words to be spelled out with katakana characters to closely match the original.

Next we underline the hiragana characters:

EUC 等のエンコーディング方法は日本語と英語が混交しているテキストをサポートします。

Hiragana characters, like katakana as just described, represent syllables. Hiragana characters are mostly used for writing grammatical words and inflectional endings. Table 1-2 illustrates the usage or meaning of the hiragana in the preceding example.

Table 1-2. Sample hiragana

Hiragana	Meaning or usage	Reading
の	Possessive marker	*no*
は	Topic marker	*wa*[a]
と	and (conjunction)	*to*
が	Subject marker	*ga*
している	doing… (verb)	*shite-iru*
を	Object marker	*o*
します	do… (verb)	*shimasu*

a. This hiragana character is normally pronounced *ha*, but when used as a topic marker, it becomes *wa*.

That's a lot of grammatical stuff! Japanese is a postpositional language, meaning that grammatical markers, such as the equivalent of prepositions as used in English, come after the nouns that they modify. These grammatical markers are called *particles* (助詞 *joshi*) in Japanese.

Finally, we underline the ideographs (called *hànzì* in Chinese, *kanji* in Japanese, *hanja* in Korean, and *chữ Hán* and *chữ Nôm* in Vietnamese):

EUC 等のエンコーディング方法は日本語と英語が混交しているテキストをサポートします。

At first glance, ideographs appear to be more complex than the other characters in the sentence. This happens to be true most of the time. Ideographs represent meanings and are often called *Chinese characters*, *Han characters*, *pictographs*, or *logographs*.* Ideographs are also assigned one or more readings (pronunciations), each of which is determined by context. While their readings differ depending on the language (meaning Chinese, Japanese, Korean, or Vietnamese), ideographs often have or convey the same or similar meaning. This makes it possible for Japanese to understand—but not necessarily pronounce—

* Being a widespread convention, this is beyond critique. However, linguists use these terms for different classes of ideographs, depending on their etymology.

some very basic Chinese, Korean, and Vietnamese texts. Table 1-3 provides a listing of the underlined ideographs and ideograph compounds (words composed of two or more ideographs) from our example sentence, and supplies their meanings and readings.

Table 1-3. Sample ideographs and ideograph compounds

Ideographs	Meaning	Reading
等	such as…	*nado*
方法	method	*hōhō*
日本語	Japanese (language)	*nihongo*
英語	English (language)	*eigo*
混交	(to) mix	*konkō*

Of course, this example includes only those types of characters that are used in Japanese—other locales use different types of characters. Table 1-4 lists the four CJKV locales, along with what scripts their writing systems use.

Table 1-4. CJKV locales and their scripts

Locale	Scripts
China	Latin and hanzi (simplified)
Taiwan	Latin, zhuyin, and hanzi (traditional)
Japan	Latin, hiragana, katakana, and kanji
Korea[a]	Latin, jamo, hangul, and hanja
Vietnam	Latin (Quốc ngữ), chữ Nôm, and chữ Hán

a. Jamo are the alphabet-like components that make up hangul.

Table 1-5 lists some sample characters from each of the scripts used in CJKV locales. We discuss the scripts of these writing systems in much greater detail in Chapter 2.

Table 1-5. Sample CJKV characters

Script	Sample characters		
Latin characters	A B C D E F G H I J	…	q r s t u v w x y z
Zhuyin	ㄅㄆㄇㄈㄉㄊㄋㄌㄍㄎ	…	ㄠㄡㄢㄣㄤㄥㄦㄧㄨㄩ
Hiragana	ぁあぃいぅうぇえぉお	…	りるれろゎわゐゑをん
Katakana	ァアィイゥウェエォオ	…	ロヮワヰヱヲンヴヵヶ
Jamo	ㄱㄲㄳㄴㄵㄶㄷㄸㄹㄺ	…	ㆅㆆㆇㆈㆉㆊㆋㆌㆍㆎ
Hangul syllables	가각간갇갈갉갊감갑값	…	흽힁히힉힌힐힘힙힛힝

Table 1-5. Sample CJKV characters

Script	Sample characters		
Hanzi (simplified)	啊阿埃挨哎唉哀皑癌蔼	…	黪黯鼢鼬鼯鼹鼷鼽鼾齄
Hanzi (traditional)	一乙丁七乃九了二人儿	…	驫鱺鸝灩灪爩麤齾齉龘
Kanji	亜唖娃阿哀愛挨姶逢葵	…	齶龕龜龠堯槇遙瑤凜熙
Hanja	伽佳假價加可呵哥嘉嫁	…	晞曦熙熹熺犧禧稀羲詰

But, how frequently are each of these scripts used? Given an average sampling of Japanese writing, one normally finds 30% kanji, 60% hiragana, and 10% katakana. Actual percentages depend on the nature of the text. For example, you may find a higher percentage of kanji in technical literature, and a higher percentage of katakana in fields such as fashion and cosmetics, which make extensive use of loan words written in katakana. Most Korean texts consist of nothing but hangul syllables, and most Chinese texts are composed of only hanzi.[*] Latin characters are used the least, except in Vietnam, where they represent the primary script.

So, how many characters do you need to learn in order to read and write CJKV languages effectively? Here are some *very* basic guidelines:

- You must learn hiragana and katakana if you plan to deal with Japanese—this constitutes approximately 200 characters.
- Learning hangul is absolutely necessary for Korean, but you can get away with not learning hanja.
- You need to have general knowledge of about 1,000 kanji to read over 90% of the kanji in typical Japanese texts—more are required for reading Chinese texts because only hanzi are used.

If you have not already learned Chinese, Japanese, Korean, or Vietnamese, I encourage you to learn one of them so that you can better appreciate the complexity of their writing systems. Although I discuss character dictionaries, and learning aids to a lesser extent, in Chapter 11, they are no substitute for a human teacher.

Character Set Standards

A character set simply provides a common *bucket*, *repertoire*, or *collection* of characters. You may have never thought of it this way, but the English alphabet is an example of a character set standard. It specifies 52 upper- and lowercase letters. Character set standards are used to ensure that we learn a minimum number of characters in order to communicate with others in society. In effect, they limit the number of characters we need to

* Well, you will also find symbol-like characters, such as punctuation marks.

learn. There are only a handful of characters in the English alphabet, so nothing is really being limited, and as such, there really is no character set standard per se. In the case of languages that use ideographs, however, character set standards play an especially vital role. They specify which ideographs—out of the tens of thousands in existence—are the most important to learn for each locale. The current Japanese set, called *Jōyō Kanji* (常用漢字 *jōyō kanji*), although advisory, limits the number of ideographs to 1,945.* There are similar character sets in China, Taiwan, and Korea. These character set standards were designed with education in mind, and are referred to as *noncoded* character sets.

Character set standards designed for use on computer systems are almost always larger than those used for the purpose of education, and are referred to as *coded* character sets. Establishing coded character set standards for use with computer systems is a way to ensure that everyone is able to view documents created by someone else. ASCII is a Western character set standard, and ensures that their computer systems can communicate with each other. But, as you will soon learn, ASCII is not sufficient for the purpose of professional publishing (neither is its most common extension, ISO 8859-1:1998).

Coded character set standards typically contain characters above and beyond those found in noncoded ones. For example, the ASCII character set standard contains 94 printable characters—42 more than the upper- and lowercase alphabet. In the case of Japanese, there are thousands of characters in the coded character sets in addition to the 1,945 in the basic noncoded character set. The basic coded Japanese character set standard, in its most current form, enumerates 6,879 characters and is designated JIS X 0208:1997. There are four versions of this character set, each designated by the year in which it was established: 1978, 1983, 1990, and 1997. There are two typical compatibility problems that you may encounter when dealing with different versions of the same character set standard:

- Some of these versions contain different numbers of characters—later versions generally add characters.
- Some of these versions are not 100% compatible with each other due to changes.

In addition, there may be an extended character set standard, such as Japan's JIS X 0212-1990, that defines 6,067 additional characters, most of which are kanji, or China's GB 18030-2005, which adds tens of thousands of characters to its original GB 2312-80 standard.

Additional incompatibility has occurred because operating system (OS) developers took these coded character set standards one step further by defining their own extensions. These vendor character set standards are largely, but not completely, compatible, and almost always use one of the national standards as their base. When you factor in vendor character set standards, things appear to be a big mess. Fortunately, the broad adoption of Unicode has nearly brought the development of vendor-specific character sets to a halt. This book documents these character sets, primarily for historical purposes.

* The predecessor of this character set, *Tōyō Kanji* (当用漢字 *tōyō kanji*), was prescriptive.

Encoding Methods

Encoding is the process of mapping a character to a numeric value, or more precisely, assigning a numeric value to a character. By doing this, you create the ability to uniquely identify a character through its associated numeric value. The more unique a value is among different encoding methods, the more likely that character identification will be unambiguous. Ultimately, the computer needs to manipulate the character as a numeric value. Independent of any CJKV language or computerized implementations thereof, indexing encoded values allows a numerically enforced ordering to be mapped onto what might otherwise be a randomly ordered natural language.

While there is no universally recognized encoding method, many have been commonly used—for example, ISO-2022-KR, EUC-KR, Johab, and Unified Hangul Code (UHC) for Korean. Although Unicode does not employ a single encoding form, it is safe to state that the encoding forms for Unicode—UTF-8, UTF-16, and UTF-32—have become universally recognized. In addition, each one has become the preferred encoding form for specific uses. For the Web, UTF-8 is the most common encoding form. Applications prefer to use the UTF-16 encoding form internally for its overall space-efficiency for the majority of multilingual text. OpenType fonts, when they include glyphs that map from characters outside the *Basic Multilingual Plane* (BMP), make use of and prefer the UTF-32 encoding form.

Data Storage Basics

First, before describing these encoding methods, here's a short explanation of how memory is allocated on computer systems. Computer systems process data called *bits*. These are the most basic units of information, and they can hold or store one of two possible values: *on* or *off*. These are usually mapped to the values 1 or 0, respectively. Bits are strung together into units called *bytes*. Bytes are usually composed of 7 or 8 bits. Seven bits allow for up to 128 unique combinations, or values; 8 bits allow for up to 256. While these numbers are sufficient for representing most characters in Western writing systems, it does not even come close to accommodating large character sets whose characters number in the thousands, such as those required by the CJKV locales. It is also possible to use more than 8 bits, and some encoding methods use 16 or 32 bits as their code units, which are equivalent to 2 and 4 bytes, respectively.

The first attempt to encode an Asian language script on computer systems involved the use of Japanese half-width katakana characters. This is a limited set of 63 characters that constitutes a minimal set for representing Japanese text. But there was no support for kanji. The solution to this problem, at least for Japanese, was formalized in 1978, and employed the notion of using 2 bytes to represent a single character. This did not eliminate the need for one-byte characters, though. The Japanese solution was to extend the notion of one-byte character encoding to include two-byte characters. This allows for text with mixed one- and two-byte characters. How one- and two-byte characters are distinguished depends on the encoding method. Two bytes equal 16 bits, and thus can provide up to

65,536 unique values. This is best visualized as a 256×256 matrix. See Figure 1-1 for an illustration of such a matrix.

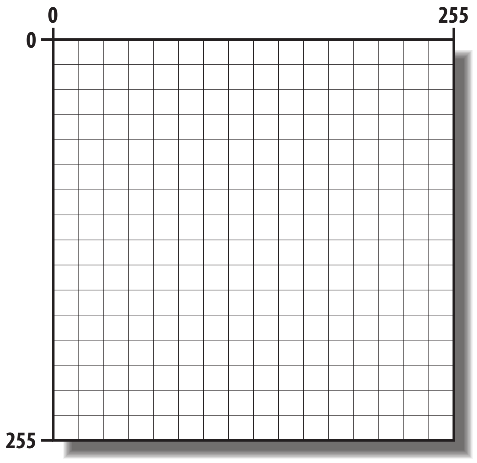


Figure 1-1. 256×256 encoding matrix

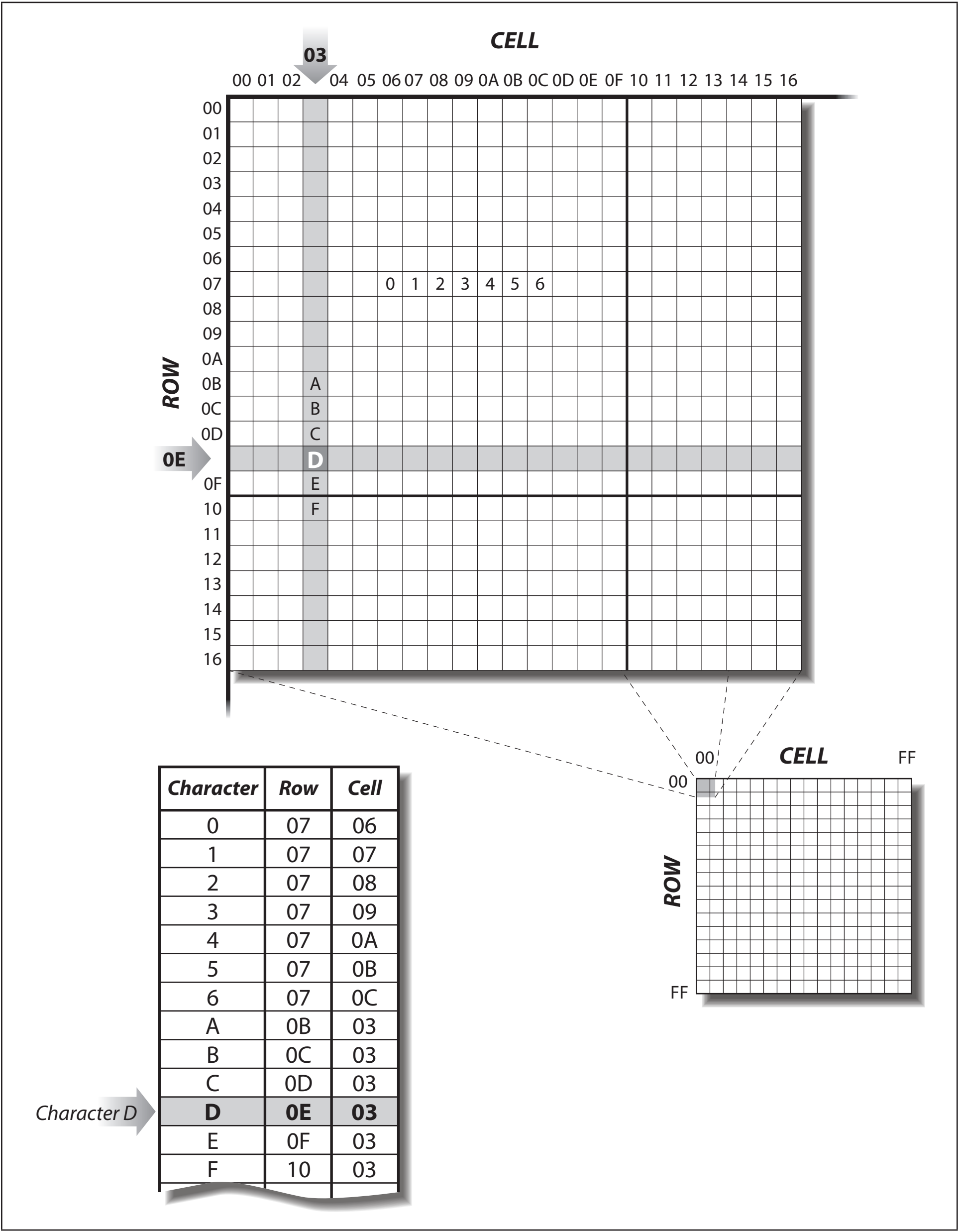
However, not all of these 65,536 cells can be used for representing displayable characters. To enable the mixture of one- and two-byte characters within a single text stream, some characters needed to be reserved as control characters, some of which then serve as the characters that signify when a text stream shifts between one- and two-byte modes. In the case of ISO-2022-JP encoding, the upper limit of displayable characters was set at 8,836, which is the size of the code space made from a 94×94 matrix.*

But why do you need to mix one- and two-byte characters anyway? It is to support existing one-byte encoding standards, such as ASCII, within a two-byte (or sometimes larger) encoding system. One-byte encoding methods are here to stay, and it is still a rather efficient means to encode the characters necessary to write English and many other languages. The most common encoding method for web pages, a mixed one- through four-byte encoding form called UTF-8, includes ASCII as its one-byte portion. However, languages with large character sets—those spoken in the CJKV locales—require two or more bytes to encode characters. Some encoding methods treat all characters, including ASCII, the same, meaning that they consume or require the same amount of encoding space. UTF-16 and UTF-32 use 16- and 32-bit code units, meaning that ASCII "A" (`0x41`) is represented by 16 or 32 bits: <`0041`> or <`00000041`>.

Along with discussions about character sets and encodings, you will encounter the terms "row" and "cell" again and again throughout this book. These refer to the axes of a matrix

* Code space refers to the area within the (usual) 256×256 encoding matrix that is used for encoding characters. Most of the figures in Chapter 4 and Appendix F illustrate code spaces that fall within this 256×256 matrix.

used to hold and encode characters. A matrix is made up of rows, and rows are made up of cells. The first byte specifies the row, and the second specifies the cell of the row. Figure 1-2 illustrates a matrix and how characters' positions correspond to row and cell values.


CELL
03
00 01 02 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16
ROW
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0F 10 11 12 13 14 15 16
0E
0 1 2 3 4 5 6
A B C D E F
00 CELL FF
00 ROW FF
Character D

Character	Row	Cell
0	07	06
1	07	07
2	07	08
3	07	09
4	07	0A
5	07	0B
6	07	0C
A	0B	03
B	0C	03
C	0D	03
D	**0E**	**03**
E	0F	03
F	10	03

Figure 1-2. Indexing an encoding matrix by row and cell

In an attempt to allow for a mixture of one- and two-byte characters, several CJKV encoding methods have been developed. As you will learn in Chapter 4, these encoding methods are largely, but not completely, compatible. You will also see that there are encoding methods that use three or even four bytes to represent a single character!

The most common Japanese encoding methods are ISO-2022-JP, Shift-JIS, and EUC-JP. ISO-2022-JP, the most basic, uses seven-bit bytes (or, seven bits of a byte) to represent characters, and requires special characters or sequences of characters (called *shifting characters* or *escape sequences*) to shift between one- and two-byte modes. Shift-JIS and EUC-JP encodings make generous use of eight-bit characters, and use the value of the first byte as the way to distinguish one- and multiple-byte characters. Now, Unicode has become the most common encoding for Japanese.

Input Methods

Those who type English text have the luxury of using keyboards that can hold all the keys to represent a sufficient number of characters. CJKV characters number in the thousands, though, so how does one type CJKV text? Large keyboards that hold thousands of individual keys exist, but they require special training and are difficult to use. This has led to software solutions: *input methods* and the *conversion dictionaries* that they employ.

Most Chinese and Japanese text is typically input in two stages:

1. The user types raw keyboard input, which the computer interprets by using the input method and the conversion dictionary to display a list of *candidate* characters (the word "candidate" here refers to the character or characters that are mapped to the input string in the conversion dictionary).
2. The user selects one choice from the list of candidate characters, or requests more choices.

How well each stage of input is handled on your computer depends greatly on the quality (and vintage) of the input software you are using. Typical Korean and Vietnamese input is much more direct, and it varies by the keyboard layout that is used.

Software called an *input method* handles both of these input stages. It is so named because it grabs the user's keyboard input before any other software can use it (specifically, it is the first software to process keyboard input).

The first stage of input requires keyboard input, and can take one of two usual forms:

- Transliteration using Latin characters (type "k" plus "a" to get か, and so on), which is common for Chinese and Japanese, but relatively rare for Korean.
- Native-script input—zhuyin for Chinese as used in Taiwan, hiragana for Japanese, hangul for Korean, and so on.

The form used depends on user preference and the type of keyboard in use. For Japanese, the input method converts transliterated Japanese into hiragana on-the-fly, so it doesn't really matter which keyboard you are using. In fact, studies show that over 70% of Japanese computer users prefer transliterated Japanese input.

Once the input string is complete, it is then parsed in one of two ways: either by the user during input or by a parser built into the input method. Finally, each segment is run through a conversion process that consists of a lookup into a conversion dictionary. This is very much like a *key-value* lookup. Typical conversion dictionaries have tens of thousands of entries. It seems that the more entries, the better the conversion quality. However, if the conversion dictionary is too large, users are shown a far too lengthy list of candidates. This reduces input efficiency.

Can ideographs be input one at a time? While single-ideograph input is possible, there are three basic units that can be used. These units allow you to limit the number of candidates from which you must choose. Typically, the larger the input unit, the fewer candidates. The units are as follows:

- Single ideograph
- Ideograph compound
- Ideograph phrase

Early input programs required that each ideograph be input individually, as single ideographs. Nowadays it is much more efficient to input ideographs as they appear in compounds or even phrases. This means that you may input two or more ideographs at once by virtue of inputting their combined reading. For example, the ideograph compound 漢字 (the two ideographs for writing the word meaning "ideograph") can be input as two separate characters, 漢 (pronounced *kan* in Japanese) and 字 (pronounced *ji* in Japanese). Table 1-6 shows the two target ideographs, along with other ideographs that share the same reading.

Table 1-6. Single ideograph input—Japanese

Character	Reading	Ideographs with identical readings
漢	KAN	乾侃冠寒刊勘勧巻喚堪姦完官寛干幹患感慣憾換敢柑桓棺款歓汗漢澗潅環甘監看竿管簡緩缶翰肝艦莞観諌貫鑑鑑間閑関陥韓館舘
字	JI	事似侍児字寺慈持時次滋治爾璽痔磁示而耳自蒔辞

You can see that there are many other ideographs with the same readings, so you may have to wade through a long list of candidate ideographs before you find the correct one. A more efficient way is to input them as one unit, called an ideograph compound. This produces a much shorter list of candidates from which to choose. Table 1-7 illustrates the two ideographs input as a compound, along with candidate compounds with the same reading.

Table 1-7. Ideograph compound input—Japanese

Compound	Reading	Compounds with identical readings
漢字	KANJI	漢字 感じ 幹事 監事 完司

Note how the list of ideograph compounds is much shorter in this case. There is an even higher-level input unit called an *ideograph phrase*. This is similar to inputting two or more ideographs as a single compound, but it adds another element, similar to a preposition in English, that makes the whole string into a phrase. An example of an ideograph phrase is 漢字は, which means "the ideograph" in Japanese. Because Chinese-language text is composed solely of hanzi, ideograph phrases apply only to Japanese, and possibly Korean.

Some of you may know of input software that claims to let you convert whole sentences at once. This is not really true. Such software allows you to input whole sentences, but the sentence is then parsed into smaller units, usually ideograph phrases, and then converted. Inputting whole sentences before any conversion is merely a convenience for the user.

Korean input has some special characteristics that are related to how their most widely used script, hangul syllables, is composed. Whether input is by a QWERTY or a Korean keyboard array, Korean input involves entering hangul elements called *jamo*. As the jamo are input, the operating system or input software attempts to compose hangul syllables using an automaton. Because of how hangul syllables are composed of jamo, the user may have up to three alternatives for deleting characters:

- Delete entire hangul syllable
- Delete by jamo
- Delete by word

This particular option is specific to Korean, and depends on whether the input method supports it.

Typography

Typography is a broad topic, and covers the way in which characters are set together, or composed, to form words, lines, and pages. What makes CJKV text different from Western text is the fact that it can usually be written or set in one of two orientations, described as follows:

- Left to right, top to bottom—horizontal setting, as in this book
- Top to bottom, right to left—vertical setting

Chapter 7 provides plenty of examples of horizontal versus vertical writing. More often than not, vertical writing orientation causes problems with Western-language applications. This is because vertical writing does not come for free, and some nontrivial amount of effort is required to implement this support in applications. Luckily, it is generally acceptable to set CJKV text in the same horizontal orientation as most Western languages. Traditional novels and short stories are often set vertically, but technical materials, such as science textbooks and the like, are set horizontally.

Vertically set CJKV text is not a simple matter of changing writing direction. Some characters require special handling, such as a different orientation (90˚ clockwise rotation) or a different position within the *em-square*.* Chapter 7 provides some sample text set both horizontally and vertically, and it illustrates the characters that require special treatment.

In addition to the two writing directions for CJKV text, there are other special layout or composition considerations, such as special rules for line breaking, special types of justification, metrics adjustments, methods for annotating characters, and so on.

Basic Concepts and Terminology FAQ

Now I'll define some basic concepts which will help carry you through this entire book. These concepts are posed as questions because they represent questions you might raise as you read this book. If at any time you encounter a new term, please glance at the glossary toward the back of the book: new terms are included and explained there.

What Are All These Abbreviations and Acronyms?

Most technical fields are flooded with abbreviations and acronyms, and CJKV information processing is no exception. Some of the more important, and perhaps confusing, ones are explained in the following section, but when in doubt, consult Appendix D.

What is the difference between GB and GB/T? What about GBK?

Most references to "GB" refer to the GB 2312-80 character set standard, which represents the most widely implemented character set for Chinese. Now, of course, a much larger character set, designated GB 18030-2005, is considered the standard character set.

GB stands for *Guo Biao* (国标 *guóbiāo*), which is short for *Guojia Biaozhun* (国家标准 *guójiā biāozhǔn*), and simply means "National Standard."

Because GB/T character set standards are traditional analogs of existing GB character set standards, some naturally think that the "T" stands for *Traditional*. This represents yet another myth to expose as untrue. The "T" in "GB/T" actually stands for *Tui* (推 *tuī*), which is short for *Tuijian* (推荐 *tuījiàn*) and means "recommended" in the sense that it is the opposite of "forced" or "mandatory."

The "K" in GBK (an extension to GB 2312-80) comes from the Chinese word 扩展 (*kuòzhǎn*), which means "extension." As you will learn in Chapter 3, while GBK is an

* The term *em-square* refers to a square-shaped space whose height and width roughly correspond to the width of the letter "M." The term *design space* is actually a more accurate way to represent this typographic concept.

extension to GB 2312-80, and thus appropriately named, GBK itself was extended to become GB 18030-2005.

What are JIS, JISC, and JSA? How are they related?

In much of the literature in the field of Japanese information processing, you will quite often see references to JISC, JIS, and JSA. The most common of these is JIS; the least is JISC. What these refer to can sometimes be confusing and is often contradicted in reference works.

JIS stands for *Japanese Industrial Standard* (日本工業規格 *nihon kōgyō kikaku*), the name given to the standards used in Japanese industry.[*] The character 〄 is the original symbol for JIS, which was used through September 30, 2008. The character ⒿⒾⓈ is the new symbol for JIS, which was issued on October 1, 2005, and its use became mandatory on October 1, 2008. JIS can refer to several things: the character set standards established by JISC, the encoding method specified in these character set standards, and even the keyboard arrays described in specific JIS standards. Context should usually make its meaning clear. Of course, JIS appears frequently in this book.

JISC stands for *Japanese Industrial Standards Committee* (日本工業標準調査会 *nihon kōgyō hyōjun chōsakai*).[†] This is the name of the governing body that establishes JIS standards and publishes manuals through JSA. The committee that develops and writes each JIS manual is composed of people from Japanese industry who have a deep technical background in the topic to be covered by the manual. Committee members are listed at the end of each JIS manual.

JSA stands for *Japanese Standards Association* (日本規格協会 *nihon kikaku kyōkai*).[‡] This organization publishes the manuals for the JIS standards established by JISC, and generally oversees the whole process.

JIS is often used as a blanket term covering JIS, JISC, and JSA, but now you know what they *genuinely* mean.

Several JIS "C" series standards changed designation to "X" series standards on March 1, 1987. Table 1-8 lists the JIS standards—mentioned in this book—that changed designation from "C" to "X" series.

* There is even a JIS standard for manufacturing toilet paper! It is designated JIS P 4501:2006 and is entitled トイレットペーパー (*toiretto pēpā*). Its English title is *Toilet tissue papers*. The "P" series JIS standards are for the pulp and paper industries.

† *http://www.jisc.go.jp/*

‡ *http://www.jsa.or.jp/*

Table 1-8. JIS standard designation changes

JIS "C" series	JIS "X" series
JIS C 6220	JIS X 0201
JIS C 6228	JIS X 0202
JIS C 6225	JIS X 0207
JIS C 6226	JIS X 0208
JIS C 6233	JIS X 6002
JIS C 6235	JIS X 6003
JIS C 6236	JIS X 6004
JIS C 6232	JIS X 9051
JIS C 6234	JIS X 9052

Because these changes took place well over two decades ago, they have long been reflected in software and other documentation.

What is KS?

KS simply stands for *Korean Standard* (한국 공업 규격/韓國工業規格 *hanguk gongeop gyugyeok*). All Korean character set standard designations begin with the two uppercase letters "KS." The character ㉿ is the symbol for KS.

All KS standards also include another letter in their designation. Those that are discussed in this book all include the letter "X," which now indicates electric or electronic standards.*

Several KS "C" series standards changed designation to "X" series standards on August 20, 1997. Table 1-9 lists the KS standards—mentioned in this book—that changed designation from the "C" to "X" series.

Table 1-9. KS standard designation changes

KS "C" series	KS "X" series
KS C 5601	KS X 1001
KS C 5657	KS X 1002
KS C 5636	KS X 1003
KS C 5620	KS X 1004
KS C 5700	KS X 1005

* Other letter designations for KS standards include "B" (mechanical), "D" (metallurgy), and "A" (general guidelines).

Table 1-9. KS standard designation changes

KS "C" series	KS "X" series
KS C 5861	KS X 2901
KS C 5715	KS X 5002

It took several years until these new KS standard designations were consistently reflected in software and documentation. Especially for KS X 1001, its original designation, KS C 5601, is still seen often.

Are VISCII and VSCII identical? What about TCVN?

Although both VISCII and VSCII are short for *Vietnamese Standard Code for Information Interchange*, they represent completely different character sets and encodings. VISCII is defined in RFC 1456,* and VSCII is derived from TCVN 5712:1993 (specifically, VN2), which is a Vietnamese national standard. VSCII is also known as ISO IR 180. The differences among VISCII and VSCII are described in Chapter 3. Appendix L provides complete encoding tables for VISCII and VSCII, which better illustrate their differences.

TCVN stands for *Tiêu Chuẩn Việt Nam*, which translates into English as "Vietnamese Standard." Like CNS, GB, JIS, and KS, it represents the first portion of Vietnamese standard designations.

What Are Internationalization, Globalization, and Localization?

Internationalization—often abbreviated as I18N, composed of the initial letter "I" followed by the middle eighteen (18) letters followed by the final letter "N"—is a term that usually refers to the process of preparing software so that it is ready to be used by more than one culture, region, or locale.† Internationalization is thus what engineers do.

Globalization, similarly abbreviated as G11N, is often used synonymously with internationalization, but encompasses the business aspects, such as entering a foreign market, conducting market research, studying the competition, strategizing, becoming aware of legal restrictions, and so on.‡ Globalization is thus what companies as a whole do.

Localization—often abbreviated as L10N under the same auspices as I18N and G11N—is the process of adapting software for a specific culture, region, or locale. *Japanization*—often abbreviated as J10N—is thus a locale-specific instance of L10N. While this book does not necessarily address all of these issues, you will find information pertinent to internationalization and localization within its pages.

* *http://www.ietf.org/rfc/rfc1456.txt*

† Quiz time. Guess what CJKV6N, C10N, K11N, M17N, S32S, and V12N stand for.

‡ But, globalization should not be confused with *global domination*, which is an entirely different matter.

Internationalization and localization are different processes. For example, it is possible to develop an application that properly handles the various scripts of the world, and is thus internationalized, but provides an English-language *user interface* (UI), and is thus not localized. The converse is also possible, specifically an application with a UI that has been translated into a large number of languages, and is thus localized, but fails to properly handle the various scripts of the world, and is thus not internationalized.

In any case, market demand forces or encourages developers to embrace I18N, G11N, and L10N, because doing so results in the functionality that their customers demand, or it provides menus and documentation written in the language of the target locale. They often require special handling because many non-Latin writing systems include a large number of characters, have complex rendering issues, or both.

What Are the Multilingual and Locale Models?

There have been two basic models for internationalization: the *locale model* and the *multilingual model*. The locale model was designed to implement a set of attributes for specific locales. The user must explicitly switch from one locale to another. The character sets implemented by the locale model were specific to a given culture or region, thus locale.

The multilingual model, on the other hand, was designed or expected to go one step further by not requiring the user to flip or switch between locales. Multilingual systems thus implement a character set that includes all the characters necessary for several cultures or regions. But still, there are cases when it is impossible to correctly render characters without knowing the target locale.

For the reasons just pointed out, a combination of these two models is ideal, which is precisely what has happened in the field of software internationalization. Unicode is the ideal character set because it is truly multilingual, and it effectively bridges and spans the world's writing systems. Of course, Unicode is not perfect. But then again, nothing made by man, by definition, can be perfect. Still, no other character set spans the world's writing systems as effectively and broadly as Unicode has done. This is why it is wise to embrace Unicode, and I encourage you to do so. Embracing Unicode also requires that its data be tagged with locale attributes so that the characters behave accordingly and appropriately for each culture or region. The *Common Locale Data Repository* (CLDR) is the most complete and robust source for locale data, and will be explored in Chapter 9.[*]

Thus, both of these models for internationalization have succeeded, not by competing, but rather by combining into a single solution that has proven to work well.

What Is a Locale?

Locale is a concept for specifying the language and country or region, and is significant in that it affects the way in which an OS, application, or other software behaves. A locale,

* *http://www.unicode.org/cldr/*

as used by software, typically consists of a language identifier and a country or region identifier.

What Is Unicode?

Unicode is the first truly successful multilingual character set standard, and it is supported by three primary encoding forms, UTF-8, UTF-16, and UTF-32. Unicode is also a major focus of this book.

Conceived 20 years ago by my friend Joe Becker, Unicode has become the preferred character set and has been successful in enabling a higher level of internationalization. In other words, Unicode has trivialized many aspects of software internationalization.

How Are Unicode and ISO 10646 Related?

Make no mistake, Unicode and ISO 10646 are different standards.* The development of Unicode is managed by *The Unicode Consortium*, and that of ISO 10646 is managed by the *International Organization for Standardization* (ISO). But, what is important is that they are equivalent, or rather kept equivalent, through a process that keeps them in sync.

ISO 10646 increases its character repertoire through new versions of the standard, additionally designated by year, along with amendments. Unicode, on the other hand, does the same through new versions. It is possible to correlate Unicode and ISO 10646 by indicating the version of the former, and the year and amendments of the latter.

More detailed coverage of Unicode can be found in Chapter 3, and details of the various Unicode encoding forms can be found in Chapter 4.

What Are Row-Cell and Plane-Row-Cell?

Row-Cell is the translated form of the Japanese word 区点 (*kuten*), which literally means "ward [and] point," or more intuitively as "row [and] cell."† This notation serves as an encoding-independent method for referring to characters in most CJKV character set standards. A Row-Cell code usually consists of four decimal digits. The "Row" portion consists of a zero-padded, two-digit number with a range from 01 to 94. Likewise, the "Cell" portion also consists of a zero-padded, two-digit number with a range from 01 to 94. For example, the first character in most CJKV character set standards is represented as 01-01 in Row-Cell notation, and is more often than not a full-width "space" character, which is typically referred to as an *ideographic space*.

Bear in mind that some character sets you will encounter include or span more than a single 94×94 matrix, each of which is referred to as a *plane*. CNS 11643-2007 and JIS X

* *http://unicode.org/faq/unicode_iso.html*

† In Chinese, Row-Cell is expressed as 区位 (*qūwèi*); in Korean, as 행렬/行列 (*haengnyeol*). Note that if the "Cell" portion of "Row-Cell" is in isolation in Korean, then it is expressed instead with the hangul 열 (*yeol*), not 렬 (*nyeol*).

0213:2004 are examples of legacy character set standards that include two or more planes. Obviously, Row-Cell notation must be expanded to include the notion of plane, meaning *Plane-Row-Cell*. In Japanese, this is expressed as 面区点 (*menkuten*) and is used as the preferred notation for referencing the characters in JIS X 0213:2004.

When I provide lists of characters throughout this book, I usually include Row-Cell (or Plane-Row-Cell) codes. These are useful for future reference of these data, and so that you don't have to hunt for the codes yourself. Now that Unicode plays an important role in today's software development efforts, I also provide Unicode scalar values when appropriate.

What Is a Unicode Scalar Value?

Like Row-Cell notation, Unicode scalar values serve as an encoding-independent method of referring to specific Unicode characters, or sequences of Unicode characters. It is a notation, but instead of using decimal values such as Row-Cell notation, hexadecimal values are used due to the larger 256×256 encoding space afforded by Unicode. Still, Unicode scalar values are not tied to a specific encoding, such as UTF-8, UTF-16, nor UTF-32.

The syntax for Unicode scalar values is simple and consists of a prefix followed by four to six hexadecimal digits. The prefix is "`U+`," and the length of the hexadecimal digits varies by whether the character is in the *Basic Multilingual Plane* (BMP) or in one of the 16 supplementary planes. I prefer to think of Unicode scalar values as equivalent to UTF-32 encoding when expressed in hexadecimal notation, but zero-padded to four digits. The ASCII "space" is thus represented as `U+0020` (not `U+20`), and the last code point in Plane 16 is represented as `U+10FFFF`.

Unicode scalar values are incredibly useful. When their prefix is removed and replaced with the appropriate *Numeric Character Reference* (NCR) syntax, such as `&#x4E00;` for `U+4E00`, they become immediately usable in contexts that support HTML or XML.[*] They can also be used within angled brackets and separated with a comma to indicate standardized Unicode sequences, such as <`U+528D`, `U+E0101`>.

Unicode scalar values are used throughout this book and are provided for your convenience.

Characters Versus Glyphs: What Is the Difference?

Now here's a topic that is usually beaten to death! The term *character* is an abstract notion indicating a class of shapes declared to have the same meaning or abstract shape. The term *glyph* refers to a specific instance of a character.

[*] NCRs are SGML constructs that have carried through into SGML derivations, such as HTML and XML. I cannot guarantee that all HTML and XML implementations support NCRs, but the vast majority do.

Interestingly, more than one character can constitute a single glyph, such as the two characters *f* and *i*, which can be fused together as the single entity *ﬁ*. This ﬁ glyph is called a *ligature*. The dollar currency symbol is a good example of a character with several glyphs. There are at least four distinct glyphs for the dollar currency symbol, described and illustrated here:

- An "S" shape with a single vertical bar: $
- An "S" shape with a single broken vertical bar: $
- An "S" shape with two vertical bars: $
- An "S" shape with two broken vertical bars: $

The differences among these four glyphs are minor, but you cannot deny that they still represent the same character, specifically the dollar currency symbol. More often than not, you encounter a difference in this glyph as a difference in typeface.

However, there are some characters that have a dozen or more variant forms. Consider the kanji 辺 (*hen*, used in the common Japanese family name 渡辺 *watanabe*), which has only two variant forms that are included in JIS X 0208:1997, and are thus included in Unicode: 邉 (U+9089) and 邊 (U+908A). These are considered the traditional forms of the kanji 辺. Table 1-10 lists the additional variant forms that are included in the Adobe-Japan1-6 character collection, which will be covered in Chapter 6.

Table 1-10. Standard versus variant forms

Standard Form	Variant forms	Additional variant forms
辺	邉	邉邉邉邉邉邉邉邉邉邉邉邉邉邉
	邊	邊邊邊邊邊邊邊

Clearly, these variant forms all appear to represent that same character, but are simply different glyphs.

You will discover that CJKV character set standards do not define the glyphs for the characters contained within their specifications. Unfortunately (or, fortunately, as the case may be), many think that the glyphs that appear in these manuals are the official ones, and to some extent they become the default, or *prototypical*, glyphs for the characters.

Note, however, that the official Jōyō Kanji Table *does* define the glyph shape, at least for the 1,945 kanji contained within its specification. Interestingly, JSA at one time published two standards that did, in fact, define glyphs for characters by virtue of specifying precise bitmap patterns for every character: JIS X 9051-1984* and JIS X 9052-1983.† The glyphs

* Previously designated JIS C 6232-1984

† Previously designated JIS C 6234-1983

set forth in these two standards were designed for the JIS X 0208-1983 standard, which was current at the time. However, these glyphs have not been widely accepted in industry, mainly due to the introduction of outline fonts. It seems as though JSA has no intention of ever revising these documents, and some speculate that this may be their way of not enforcing glyphs.

The one Japanese organization that established a definitive Japanese glyph standard in Japan is the now-defunct FDPC, which is an abbreviation for *Font Development and Promotion Center* (文字フォント開発・普及センター *moji fonto kaihatsu fukyū sentā*). FDPC was a MITI (*Ministry of International Trade and Industry*—通商産業省 *tsūshō sangyō shō*)-funded organization, and has since been folded in with JSA. This government organization, with the help of members, developed a series of Japanese outline fonts called Heisei (平成 heisei) typefaces. The first two Heisei typefaces that were released were *Heisei Mincho W3* (平成明朝 W3 *heisei minchō W3*) and *Heisei Kaku (squared) Gothic W5* (平成角ゴシック W5 *heisei kaku goshikku W5*). In fact, the standard Japanese typeface used in the production of the first edition of this book was Heisei Mincho W3. A total of seven weights of both designs were produced, ranging from 3 (W3) to 9 (W9). Two weights of *Heisei Maru (rounded) Gothic* (平成丸ゴシック *heisei maru goshikku*), specifically 4 and 8, also were developed. The Heisei typefaces have become somewhat commonplace in the Japanese market.

Stability versus correctness

I have learned that changes to prototypical glyphs are inevitable and unavoidable. There are two forces or notions at work. One is *stability*, and the other is the notion of *correctness*. Unfortunately, the notion of correctness can—and ultimately will—change over time. Languages and their writing systems change, which is part of their nature. In Japan, the first series of prototypical glyph changes took place in 1983, when JIS X 0208-1983 was published. Bear in mind that the first version of the standard was published in 1978. Somewhat subtle prototypical glyph changes took place when it was revised in 1990 and designated JIS X 0208-1990. Stability ruled the day when the 1997 revision, designated JIS X 0208:1997, was published, along with its extension in 2000, designated JIS X 0213:2000. A new set of 1,022 kanji, called *NLC Kanji* (表外漢字 *hyōgai kanji*), introduced a new notion of correctness, and directly influenced prototypical glyph changes that were introduced in 2004, as a revision to the existing standard, designated JIS X 0213:2004. I have found it interesting that some of these prototypical glyph changes have caused the glyphs for some characters to come full circle, meaning that they reverted to their original forms as found in the 1978 version of the standard. Table 1-11 provides but one example of a JIS X 0208:1997 kanji that came full circle—specifically 36-52. Its Unicode code point, regardless of glyph, is U+8FBB.

Table 1-11. Prototypical glyph changes over time—Japan

Code point	1978	1983	1990	1997	2000	2004
36-52	辻	辻	辻	辻	辻	辻

China takes glyph issues very seriously and expended the effort to develop a series of standards, published in a single manual entitled *32×32 Dot Matrix Font Set and Data Set of Chinese Ideograms for Information Interchange* (信息交换用汉字 32×32 点阵字模集及数据集 *xìnxī jiāohuàn yòng hànzì 32×32 diǎnzhèn zìmújí jí shújùjí*). This explicitly defined glyphs for the GB 2312-80 character set standard in various typeface styles. These standards are listed in Table 1-12.

Table 1-12. Chinese glyph standards

Standard	Pages	Title (in English)
GB 6345.1-86	1–27	*32×32 Dot Matrix Font Set of Chinese Ideograms for Information Interchange*
GB 6345.2-86	28–31	*32×32 Dot Matrix Font Data Set of Chinese Ideograms for Information Interchange*
GB 12034-89	32–55	*32×32 Dot Matrix Fangsongti Font Set and Data Set of Chinese Ideograms for Information Interchange*
GB 12035-89	56–79	*32×32 Dot Matrix Kaiti Font Set and Data Set of Chinese Ideograms for Information Interchange*
GB 12036-89	80–103	*32×32 Dot Matrix Heiti Font Set and Data Set of Chinese Ideograms for Information Interchange*

Song (specified in GB 6345.1-86), Fangsong, Kai, and Hei are the most common typeface styles used in Chinese. When the number of available pixels is reduced, it is impossible to completely represent all of an ideograph's strokes. These standards are useful because they establish bitmapped patterns that offer a compromise between accuracy and legibility. The GB 16794.1-1997 standard (信息技术—通用多八位编码字符集 48 点阵字形 *xìnxī jìshù—tōngyòng duōbāwèi biānmǎ zìfùjí 48 diǎnzhèn zìxíng*) is similar to the GB standards listed in Table 1-12, but covers the complete GBK character set and provides 48×48 bitmapped patterns for every character. An older set of GB standards, GB 5007.1-85 (信息交换用汉字 24×24 点阵字模集 *xìnxī jiāohuàn yòng hànzì 24×24 diǎnzhèn zìmújí*) and GB 5007.2-85 (信息交换用汉字 24×24 点阵字模数据集 *xìnxī jiāohuàn yòng hànzì 24×24 diǎnzhèn zìmú shújùjí*), provided 24×24 bitmapped patterns for a single design, and obviously covered GB 2312-80, not GBK, given that they were published in 1985, long before GBK was developed.

Exactly how the terms *character* and *glyph* are defined can differ depending on the source. Table 1-13 provides the ISO and The Unicode Consortium definitions for the terms *abstract character*, *character*, *glyph*, *glyph image*, and *grapheme*.

Table 1-13. *Abstract character, character, glyph, glyph image, and grapheme definitions*

Terminology	ISO	Unicode[a]
Abstract character	n/a	A unit of information used for the organization, control, or representation of textual data. (See definition D7 in Section 3.4, Characters and Encoding.)
Character	A member of a set of elements used for the organization, control, or representation of data.[b] An atom of information with an individual meaning, defined by a character repertoire.[c]	(1) The smallest component of written language that has semantic value; refers to the abstract meaning and/or shape, rather than a specific shape (see also *glyph*), though in code tables some form of visual representation is essential for the reader's understanding. (2) Synonym for abstract character. (3) The basic unit of encoding for the Unicode character encoding. (4) The English name for the ideographic written elements of Chinese origin. [See *ideograph* (2).]
Glyph	A recognizable abstract graphical symbol which is independent of any specific design.[c]	(1) An abstract form that represents one or more glyph images. (2) A synonym for *glyph image*. In displaying Unicode character data, one or more glyphs may be selected to depict a particular character. These glyphs are selected by a rendering engine during composition and layout processing. (See also *character*.)
Glyph image	An image of a glyph, as obtained from a glyph representation displayed on a presentation surface.[c]	The actual, concrete image of a glyph representation having been rasterized or otherwise imaged onto some display surface.
Grapheme	n/a	(1) A minimally distinctive unit of writing in the context of a particular writing system. For example, ‹b› and ‹d› are distinct graphemes in English writing systems because there exist distinct words like *big* and *dig*. Conversely, a lowercase italiform letter *a* and a lowercase Roman letter a are not distinct graphemes because no word is distinguished on the basis of these two different forms. (2) What a user thinks of as a character.

a. *The Unicode Standard, Version 5.0* (Addison-Wesley, 2006)

b. ISO 10646:2003

c. ISO 9541-1:1991

As usual, the standards—in this case ISO 10646 and Unicode—provide clear and precise definitions for otherwise complex and controversial terms. Throughout this book, I use the terms *character* and *glyph* very carefully.

What Is the Difference Between Typeface and Font?

The term *typeface* refers to the design characteristics of a collection of glyphs, and is often comprised of multiple fonts. Thus, a *font* refers to a single instance of a typeface, such as a specific style, relative weight, relative width, or other design attributes, and in the case of bitmapped fonts, point size. This is why the commonly used term *outline font* is somewhat of a misnomer—the mathematical outlines are, by definition, scalable, which means that they are not specific to a point size. A better term is *outline font instance*. But, I digress.

Western typography commonly uses serif, sans serif, and script typeface styles. Table 1-14 lists the common CJKV typeface styles, along with correspondences across locales.

Table 1-14. Western versus CJKV typeface styles

Western	Chinese[a]	Japanese	Korean
Serif[b]	Ming (明體 *míngtǐ*) Song (宋体 *sòngtǐ*)	Mincho (明朝体 *minchōtai*)	Batang (바탕 *batang*)[c]
Sans serif	Hei (黑体 *hēitǐ*)	Gothic (ゴシック体 *goshikkutai*)	Dotum (돋움 *dotum*)[d]
Script	Kai (楷体 *kǎitǐ*)	Kaisho (楷書体 *kaishotai*) Gyosho (行書体 *gyōshotai*) Sosho (草書体 *sōshotai*)	Haeseo (해서체/楷書體 *haeseoche*) Haengseo (행서체/行書體 *haengseoche*) Choseo (초서체/草書體 *choseoche*)
Other	Fangsong (仿宋体 *fǎngsòngtǐ*)	Kyokasho (教科書体 *kyōkashotai*)	

a. Replace 体 with 體 in these typeface style names for Traditional Chinese.

b. The convention has been that Ming is used for Traditional Chinese, and Song is used for Simplified Chinese.

c. In the mid-1990s, the Korean Ministry of Culture specified this term, replacing *Myeongjo* (명조체/明朝體 *myeongjoche*).

d. In the mid-1990s, the Korean Ministry of Culture specified this term, replacing *Gothic* (고딕체/고딕體 *godikche*).

Table 1-14 by no means constitutes a complete list of CJKV typeface styles—there are numerous typeface styles for hangul, for example. To provide a sample of typeface variation within a locale, consider the four basic typeface styles used for Chinese, as illustrated in Table 1-15.

Table 1-15. Chinese typeface styles—examples

Song	Hei	Kai	Fangsong
中文简体字	**中文简体字**	中文简体字	中文简体字

Clearly, the glyphs shown in Table 1-15 represent the same characters, in that they convey identical meanings, and differ only in that their glyphs are different by virtue of having different typeface designs.

What Are Half- and Full-Width Characters?

The terms *half-* and *full-width* refer to the relative glyph size of characters. These are referred to as *hankaku* (半角 *hankaku*) and *zenkaku* (全角 *zenkaku*), respectively, in

Japanese.[*] Half-width is relative to full-width. Full-width refers to the glyph size of standard CJKV characters, such as zhuyin, kana, hangul syllables, and ideographs. Latin characters, which appear to take up approximately half the display width of CJKV characters, are considered to be half-width by this standard. The very first Japanese characters to be processed on computer systems were half-width katakana. They have the same approximate display width as Latin characters. There are now full-width Latin and katakana characters. Table 1-16 shows the difference in display width between half- and full-width characters (the katakana character used as the example is pronounced *ka*).

Table 1-16. Half- and full-width characters

Width	Katakana	Latin
Half	ｶｶｶｶｶ	12345
Full	カカカカカ	１２３４５

As you can see, full-width characters occupy twice the display width as their half-width versions. At one point in time there was a clear-cut relationship between the display width of a glyph and the number of bytes used to encode it (the *encoding length*)—the number of bytes simply determined the display width. Half-width katakana characters were originally encoded with one byte. Full-width characters were encoded with two bytes. Now that there is a much richer choice of encoding methods available, this relationship no longer holds true. Table 1-17 lists several popular encoding methods, along with the number of bytes required to represent half- and full-width characters.

Table 1-17. Half- and full-width character representations—Japanese

Width	Script	ASCII	ISO-2022-JP	Shift-JIS	EUC-JP	UTF-8	UTF-16
Full	Katakana	n/a	2 bytes	2 bytes	2 bytes	3 bytes	16 bits
	Latin	n/a	2 bytes	2 bytes	2 bytes	3 bytes	16 bits
Half	Katakana	n/a	1 byte	1 byte	2 bytes	3 bytes	16 bits
	Latin	1 byte	1 byte	1 byte	1 byte	1 byte	16 bits

I should also point out that in some circumstances, half-width may also mean not full-width, and can refer to characters that are intended to be proportional.

* In Chinese, specifically in Taiwan, these terms are 半形 (*bànxíng*) and 全形 (*quánxíng*), respectively. But, in China, they are the same as those used for Japanese, specifically 半角 (*bànjiǎo*) and 全角 (*quánjiǎo*), respectively. In Korean, these terms are 반각/半角 (*bangak*) and 전각/全角 (*jeongak*), respectively.

Latin Versus Roman Characters

To this day, many people debate whether the 26 letters of the English alphabet should be referred to as *Roman* or *Latin* characters. While some standards, such as those published by ISO, prefer the term Latin, other standards prefer the term Roman. In this book, I prefer to use Latin over Roman, and use it consistently, but readers are advised that they should treat both terms synonymously.

When speaking of typeface designs, the use of the term Roman is used in contrast with the term *italic*, and refers to the upright or nonitalic letterforms.

What Is a Diacritic Mark?

A diacritic mark is an attachment to a character that typically serves as an accent, or indicates tone or other linguistic information. Diacritic marks are important in the scope of this book, because many transliteration and Romanization systems make use of them. Japanese long vowels, for example, are indicated through the use of the diacritic mark called a *macron*. Vietnamese makes extensive use of multiple diacritic marks, meaning that some characters can include more than one diacritic mark. Many non-Latin scripts, such as Japanese hiragana and katakana, also use diacritic marks. In the case of Japanese kana, one such diacritic mark serves to indicate the voicing of consonants.

What Is Notation?

The term notation refers to a method of representing units. A given distance, whether expressed in miles or kilometers, is, after all, the same physical distance. In computer science, common notations for representing the value of bit arrays, bytes, and larger units are listed in Table 1-18, and all correspond to a different numeric base.

Table 1-18. Decimal 100 in common notations

Notation	Base	Value range	Example
Binary	2	`0` and `1`	`01100100`
Octal	8	`0`–`7`	`144`
Decimal	10	`0`–`9`	`100`
Hexadecimal	16	`0`–`9` and `A`–`F`	`64`

While the numbers in the Example column all have the same underlying value, specifically `100` (in decimal), they have been expressed using different notations, and thus take on a different form. Most people—that is, non-nerds—think in decimal notation, because that is what we were taught. However, computers—and some nerds—process information using binary notation.* As discussed previously, computers process bits, which have two possible values. In the next section, you will learn that hexadecimal notation does, however, have distinct advantages when dealing with computers.

What Is an Octet?

We have already discussed the terms *bits* and *bytes*. But what about the term *octet*? At a glance, you can tell it has something to do with the number eight. An octet represents eight bits, and is thus an eight-bit byte, as opposed to a seven-bit one. This becomes confusing when dealing with 16-bit encodings. Sixteen bits can be broken down into two eight-bit bytes, or two octets. Thirty-two bits, likewise, can be broken down into four eight-bit bytes, or four octets.

Given 16 bits in a row:

```
0110010001011111
```

this string of bits can be broken down into two eight-bit units, specifically octets (bytes):

```
01100100
01011111
```

The first eight-bit unit represents decimal 100 (0x64), and the second eight-bit unit represents decimal 95 (0x5F). All 16 bits together as a single unit are usually equal to 25,695 in decimal, or <645F> in hexadecimal (it may be different depending on a computer's specific architecture). Divide 25,695 by 256 to get the first byte's value as a decimal octet, which results in 100 in this case; the remainder from this division is the value of the second byte, which, in this case, is 95. Table 1-19 lists representations of two octets (bytes), along with their 16-bit unit equivalent. This is done for you in the four different notations.

Table 1-19. Octets and 16-bit units in various notations

Notation	First octet	Second octet	16-bit unit
Binary	01100100	01011111	01100100 01011111
Octal	144	137	62137
Decimal	100	95	25,695
Hexadecimal	64	5F	64 5F

Note how going from two octets to a 16-bit unit is a simple matter of concatenation in the case of binary and hexadecimal notation. This is not true with decimal notation, which requires multiplication of the first octet by 256, followed by the addition of the second octet. Thus, the ease of converting between different representations—octets versus 16-bit units—depends on the notation that you are using. Of course, string concatenation

* Now that I think about it, the *Bynars*, featured in the *Star Trek: The Next Generation* episode entitled "11001001," represent an entire race that processes information in binary form.

is easier than two mathematical operations. This is precisely why hexadecimal notation is used very frequently in computer science and software development.

In some cases, the order in which byte concatenation takes place matters, such as when the byte order (also known as *endianness*) differs depending on the underlying computing architecture. Guess what the next section is about?

What Are Little- and Big-Endian?

There are two basic computer architectures when it comes to the issue of byte order: little-endian and big-endian. That is, the order in which the bytes of larger-than-byte storage units—such as integers, floats, doubles, and so on—appear.* One-byte storage units, such as *char* in C/C++, do not need this special treatment. That is, unless your particular machine or implementation represents them with more than one byte. The following is a synopsis of little- and big-endian architectures:

- Little-endian machines use computing architectures supported by Vax and Intel processors. Historically, MS-DOS and Windows machines are little-endian.
- Big-endian machines use computing architectures supported by Motorola and Sun processors. Historically, Mac OS and most Unix workstations are big-endian. Big-endian is also known as *network byte order*.

Linux, along with Apple's rather recent switch to Intel processors, has blurred this distinction, to the point that platform or OS are no longer clear indicators of whether little- or big-endian byte order is used. In fact, until the 1970s, virtually all processors used big-endian byte order. The introduction of microprocessors with their (initially) simple logic circuits and use of byte-level computations led to the little-endian approach. So, from the viewpoint of history, the mainframe versus microprocessor distinction gave birth to byte order differences. I should also point out that runtime detection of byte order is much more robust and reliable than guessing based on OS.

Table 1-20 provides an example two-byte value as encoded on little- and big-endian machines.

Table 1-20. Little- and big-endian representation

Notation	High byte	Low byte	Little-endian	Big-endian
Binary	01100100	01011111	01011111 01100100	01100100 01011111
Hexadecimal	64	5F	5F 64	64 5F

A four-byte example, such as 0x64, 0x5F, 0x7E, and 0xA1, becomes <A1 7E 5F 64> on little-endian machines, and <64 5F 7E A1> on big-endian machines. Note how the bytes

* A derivation of little- and big-endian came from *Gulliver's Travels*, in which there were civil wars fought over which end of a boiled egg to crack.

themselves—not the underlying bits of each byte—are reversed depending on endianness. This is precisely why endianness is also referred to as byte order. The term *endian* is used to describe what impact the byte at the end has on the overall value. The UTF-16 value for the ASCII "space" character, U+0020, is <00 20> for big-endian machines and <20 00> for little-endian ones.

Now that you understand the concept of endianness, the real question that needs answering is when endianness matters. Please keep reading….

What Are Multiple-Byte and Wide Characters?

If you have ever read comprehensive books and materials about ANSI C, you more than likely came across the terms multiple-byte and wide characters. Those documents typically don't do those terms justice, in that they are not fully explained. Here you'll get a definitive answer.

When dealing with encoding methods that are processed on a per-byte basis, endianness or byte order is irrelevant. The bytes that compose each character have only one order, regardless of the underlying architecture. These encoding methods support what are known as multiple-byte characters. In other words, these encoding methods use the byte as their code unit.

So, what encoding methods support multiple-byte characters? Table 1-21 provides an incomplete yet informative list of encoding methods that support multiple-byte characters. Some encoding methods are tied to a specific locale, and some are tied to CJKV locales in general.

Table 1-21. Multiple-byte characters—encoding methods

Encoding	Encoding length	Locale
ASCII	One-byte	n/a
ISO-2022	One- and two-byte	CJKV
EUC	One- through four-byte, depending on locale	CJKV
GBK	One- and two-byte	China
GB 18030	One-, two-, and four-byte	China
Big Five	One- and two-byte	Taiwan and Hong Kong
Shift-JIS	One- and two-byte	Japan
Johab	One- and two-byte	Korea
UHC	One- and two-byte	Korea
UTF-8	One- through four-byte	n/a

There are some encodings that require special byte order treatment, and thus cannot be treated on a per-byte basis. These encodings use what are known as wide characters, and almost always provide a facility for indicating the byte order. Table 1-22 lists some encoding methods that make use of wide characters, all of which are encoding methods for Unicode.

Table 1-22. Wide characters—encoding methods

Encoding	Encoding length
UCS-2	16-bit fixed
UTF-16	16-bit variable-length
UTF-32	32-bit fixed

Sometimes the encodings listed in Table 1-22 are recommended to use the *Byte Order Mark* (BOM) at the beginning of a file to explicitly indicate the byte order. The BOM is covered in greater detail in Chapter 4.

It is with endianness or byte order that we can more easily distinguish multiple-byte from wide characters. Multiple-byte characters have the same byte order, regardless of the underlying processor architecture. The byte order of wide characters is determined by the underlying processor architecture and must be flagged or indicated in the data itself.

Advice to Readers

This chapter serves as an introduction to the rest of this book, and is meant to whet your appetite for what lies ahead in the pages that follow. When reading the chapters of this book, I suggest that you focus on the sections that cover or relate to Unicode, because they are likely to be of immediate value and benefit. Information about legacy character sets and encodings is still of great value because it relates to Unicode, often directly, and also serves to chronicle how we got to where we are today.

In any case, I hope that you enjoy reading this book as much as I enjoyed writing the words that are now captured within its pages.

CHAPTER 2

Writing Systems and Scripts

Reading the introductory chapter provided you with a taste of what you can expect to learn about CJKV information processing in this book. Let's begin the journey with a thorough description of the various CJKV writing systems that serve as the basis for the characters set standards that will be covered in Chapter 3.

Mind you, we have already touched upon this subject, though briefly, in the introductory chapter, but there is a lot more to learn! After reading this chapter, you should have a firm grasp of the types of characters, or character classes, used to write CJKV text, specifically the following:

- Latin characters—including transliteration and romanization systems
- Zhuyin—also called *bopomofo*
- Kana—*hiragana* and *katakana*
- Hangul syllables—including *jamo*, the elements from which they're made
- Ideographs—originating in China
- Non-Chinese ideographs—originating in Japan, Korea, and Vietnam

Knowing that each of these character classes exhibits its own special characteristics and often has locale-specific usages is important to grasp. This information is absolutely crucial for understanding discussions elsewhere in this book. After all, many of the problems and issues that caused you to buy this book are the result of the complexities of these writing systems. This is not a bad thing: the complexities and challenges that we face are what make our lives interesting, and to some extent, unique from one another.

Latin Characters, Transliteration, and Romanization

Latin characters (拉丁字母 *lādīng zìmǔ* in Chinese, ラテン文字 *raten moji* or ローマ字 *rōmaji* in Japanese, 로마자 *romaja* in Korean, and *Quốc ngữ*/國語 in Vietnamese) used in the context of CJKV text are the same as those used in Western text, specifically the 52 upper- and lowercase letters of the Latin alphabet, sometimes decorated with accents to indicate length, tone, or other phonetic attributes, and sometimes set with full-width metrics. Also included are the 10 numerals 0 through 9. Accented characters, usually vowels, are often required for transliteration or Romanization purposes. Table 2-1 lists the basic set of Latin characters.

Table 2-1. Latin characters

Character class	Characters
Lowercase	abcdefghijklmnopqrstuvwxyz
Uppercase	ABCDEFGHIJKLMNOPQRSTUVWXYZ
Numerals	0123456789

There is really nothing special about these characters. Latin characters are most often used in tables (numerals), in abbreviations and acronyms (alphabet), or for transcription or transliteration purposes, sometimes with accented characters to account for tones or other phonetic attributes.

Transliteration systems are distinguished from Romanization systems in that they are not the primary way to write a language, and serve as a prounounciation aid for those who are not familiar with the primary scripts of the language.

Commonly used transliteration systems for CJKV text that use characters beyond the standard set of Latin characters illustrated in Table 2-1 include *Pinyin* (Chinese), *Hepburn* (Japanese), *Kunrei* (Japanese), and *Ministry of Education* (Korean). These and other CJKV transliteration systems are covered in the following sections. *Quốc ngữ*, on the other hand, is a Romanization system, because it is the accepted way to write Vietnamese.

Special cases of transliteration are covered in Chapter 9, specifically in the section entitled "Special Transliteration Considerations."

Chinese Transliteration Methods

Chinese uses two primary transliteration methods: *Pinyin* (拼音 *pīnyīn*) and *Wade-Giles* (韋氏 *wéishì*). There is also the *Yale* method, which is not covered in this book. There are many similarities between these two transliteration methods; they mainly differ in where they are used. Pinyin is used in China, whereas Wade-Giles is popular in Taiwan. Historically speaking, Wade-Giles was the original Chinese transliteration system recognized during the nineteenth century.

Table 2-2 lists the consonant sounds as transliterated by Pinyin and Wade-Giles—zhuyin/bopomofo symbols, also a transliteration system, and described later in this chapter, are included for the purpose of cross-reference.

Table 2-2. Chinese transliteration—consonants

Zhuyin/bopomofo	Pinyin	Wade-Giles
ㄅ	B	P
ㄆ	P	P’
ㄇ	M	M
ㄈ	F	F
ㄉ	D	T
ㄊ	T	T’
ㄋ	N	N
ㄌ	L	L
ㄍ	G	K
ㄎ	K	K’
ㄏ	H	H
ㄐ	J	CH[a]
ㄑ	Q	CH’[a]
ㄒ	X	HS[a]
ㄓ	ZH	CH
ㄔ	CH	CH’
ㄕ	SH	SH
ㄖ	R	J
ㄗ	Z	TS
ㄘ	C	TS’
ㄙ	S	S

a. Only before *i* or *ü*

Table 2-3 lists the vowel sounds as transliterated by Pinyin—zhuyin/bopomofo are again included for reference. Note that this table is constructed as a matrix that indicates what zhuyin vowel combinations are possible and how they are transliterated. The two axes themselves serve to indicate the transliterations for single zhuyin vowels.

Table 2-3. Chinese transliteration—vowels

	ㄧ **I**	ㄨ **U**	ㄩ **Ü**
ㄚ **A**	ㄧㄚ IA	ㄨㄚ UA	
ㄛ **O**		ㄨㄛ UO	

Table 2-3. Chinese transliteration—vowels

	ㄧ **I**	ㄨ **U**	ㄩ **Ü**
ㄜ **E**	ㄧㄝ IE		ㄩㄝ ÜE
ㄞ **AI**		ㄨㄞ UAI	
ㄟ **EI**		ㄨㄟ UEI	
ㄠ **AO**	ㄧㄠ IAO		
ㄡ **OU**	ㄧㄡ IOU		
ㄢ **AN**	ㄧㄢ IAN	ㄨㄢ UAN	ㄩㄢ ÜAN
ㄣ **EN**	ㄧㄣ IN	ㄨㄣ UEN	ㄩㄣ ÜN
ㄤ **ANG**	ㄧㄤ IANG	ㄨㄤ UANG	
ㄥ **ENG**	ㄧㄥ ING	ㄨㄥ UENG or ONG	ㄩㄥ IONG

The zhuyin character ㄦ, which deserves separate treatment from the others, is usually transliterated *er*.

It is sometimes necessary to use an apostrophe to separate the Pinyin readings of individual hanzi when the result can be ambiguous. Consider the transliterations for the words 先 and 西安, which are *xiān* and *xī’ān*, respectively. Note the use of the apostrophe to distinguish them.

More details about the zhuyin characters themselves appear later in this chapter. Po-Han Lin (林伯翰 *lín bóhàn*) has developed a Java applet that can convert between the Pinyin, Wade-Giles, and Yale transliteration systems.[*] He also provides additional details about Chinese transliteration.[†]

Chinese tone marks

Also of interest is how tone marks are rendered when transliterating Chinese text. Basically, there are two systems for indicating tone. One system, which requires the use of special fonts, employs diacritic marks that serve to indicate tone. The other system uses the numerals 1 through 4 immediately after each hanzi transliteration—no special fonts are required. Pinyin transliteration generally uses diacritic marks, but Wade-Giles uses numerals.

Table 2-4 lists the names of the Chinese tone marks, along with an example hanzi for each. Note that there are cases in which no tone is required.

* *http://www.edepot.com/java.html*

† *http://www.edepot.com/taoroman.html*

Table 2-4. Chinese tone mark examples

Tone	Tone name	Number[a]	Example	Meaning
None	轻声/輕聲 (*qīngshēng*)	None	*ma* (吗)	Question particle
Flat	阴平/陰平 (*yīnpíng*)	1	*ma1* or *mā* (妈)	mother
Rising	阳平/陽平 (*yángpíng*)	2	*ma2* or *má* (麻)	hemp, flax
Falling-Rising	上声/上聲 (*shǎngshēng*)	3	*ma3* or *mǎ* (马)	horse
Falling	去声/去聲 (*qùshēng*)	4	*ma4* or *mà* (骂)	cursing, swearing

a. Microsoft's Pinyin input method uses the numeral 5 to indicate no tone.

It is also common to find reference works in which Pinyin readings have no tone marks at all—that is, no numerals and no diacritic marks. I have observed that tone marks can be effectively omitted when the corresponding hanzi are in proximity, such as on the same page; the hanzi themselves can be used to remove any ambiguity that arises from no indication of tones. Pinyin readings provided throughout this book use diacritic marks to indicate tone.

Japanese Transliteration Methods

There are four Japanese transliteration systems worth exploring in the context of this book:

The Hepburn system (ヘボン式 *hebon shiki*)
: Popularized by James Curtis Hepburn, an American missionary, in 1887 in the third edition of his dictionary, this is considered the most widely used system. This transliteration system was developed two years earlier, in 1885, by the *Roman Character Association* (羅馬字会 *rōmajikai*).

The Kunrei system (訓令式 *kunrei shiki*)
: Developed in 1937, this is considered the official transliteration system by the Japanese government.

The Nippon system (日本式 *nippon shiki*)
: Developed by Aikitsu Tanakadate (田中館愛橘 *tanakadate aikitsu*) in 1881—nearly identical to the Kunrei system, but the least used.

The Word Processor system (ワープロ式 *wāpuro shiki*)
: Developed in a somewhat *ad hoc* fashion over recent years by Japanese word processor and input method manufacturers. Whereas the other three transliteraton systems are largely phonemic, the Word Processor system more closely adheres to a one-to-one transcription of the kana.

The Japanese transliterations in this book adhere to the Hepburn system. Because the Word Processor system allows for a wide variety of transliteration possibilities, which is the nature of input methods, it is thus a topic of discussion in Chapter 5.

Table 2-5 lists the basic kana syllables (shown here and in other tables of this section using hiragana), transliterated according to the three transliteration systems. Those that are transliterated differently in the three systems have been highlighted for easy differentiation. Table 2-18 provides similar information, but presented in a different manner.

Table 2-5. Single syllable Japanese transliteration

Kana	Hepburn	Kunrei	Nippon
あ	A	A	A
い	I	I	I
う	U	U	U
え	E	E	E
お	O	O	O
か	KA	KA	KA
が	GA	GA	GA
き	KI	KI	KI
ぎ	GI	GI	GI
く	KU	KU	KU
ぐ	GU	GU	GU
け	KE	KE	KE
げ	GE	GE	GE
こ	KO	KO	KO
ご	GO	GO	GO
さ	SA	SA	SA
ざ	ZA	ZA	ZA
し	SHI	SI	SI
じ	JI	ZI	ZI
す	SU	SU	SU
ず	ZU	ZU	ZU
せ	SE	SE	SE
ぜ	ZE	ZE	ZE

Table 2-5. Single syllable Japanese transliteration

Kana	Hepburn	Kunrei	Nippon
そ	SO	SO	SO
ぞ	ZO	ZO	ZO
た	TA	TA	TA
だ	DA	DA	DA
ち	CHI	TI	TI
ぢ	JI	ZI	DI
つ	TSU	TU	TU
づ	ZU	ZU	DU
て	TE	TE	TE
で	DE	DE	DE
と	TO	TO	TO
ど	DO	DO	DO
な	NA	NA	NA
に	NI	NI	NI
ぬ	NU	NU	NU
ね	NE	NE	NE
の	NO	NO	NO
は	HA	HA	HA
ば	BA	BA	BA
ぱ	PA	PA	PA
ひ	HI	HI	HI
び	BI	BI	BI
ぴ	PI	PI	PI
ふ	FU	HU	HU
ぶ	BU	BU	BU
ぷ	PU	PU	PU
へ	HE	HE	HE

Table 2-5. Single syllable Japanese transliteration

Kana	Hepburn	Kunrei	Nippon
べ	BE	BE	BE
ぺ	PE	PE	PE
ほ	HO	HO	HO
ぼ	BO	BO	BO
ぽ	PO	PO	PO
ま	MA	MA	MA
み	MI	MI	MI
む	MU	MU	MU
め	ME	ME	ME
も	MO	MO	MO
や	YA	YA	YA
ゆ	YU	YU	YU
よ	YO	YO	YO
ら	RA	RA	RA
り	RI	RI	RI
る	RU	RU	RU
れ	RE	RE	RE
ろ	RO	RO	RO
わ	WA	WA	WA
ゐ	WI	WI	WI
ゑ	WE	WE	WE
を	O	O	WO
ん	N or M[a]	N	N

a. An *m* was once used before the consonants *b*, *p*, or *m*—an *n* is now used in all contexts.

Table 2-6 lists what are considered to be the palatalized syllables—although they signify a single syllable, they are represented with two kana characters. Those that are different in the three transliteration systems are highlighted.

Table 2-6. Japanese transliteration—palatalized syllables

Kana	Hepburn	Kunrei	Nippon
きゃ	KYA	KYA	KYA
ぎゃ	GYA	GYA	GYA
きゅ	KYU	KYU	KYU
ぎゅ	GYU	GYU	GYU
きょ	KYO	KYO	KYO
ぎょ	GYO	GYO	GYO
しゃ	SHA	SYA	SYA
じゃ	JA	ZYA	ZYA
しゅ	SHU	SYU	SYU
じゅ	JU	ZYU	ZYU
しょ	SHO	SYO	SYO
じょ	JO	ZYO	ZYO
ちゃ	CHA	TYA	TYA
ぢゃ	JA	ZYA	DYA
ちゅ	CHU	TYU	TYU
ぢゅ	JU	ZYU	DYU
ちょ	CHO	TYO	TYO
ぢょ	JO	ZYO	DYO
にゃ	NYA	NYA	NYA
にゅ	NYU	NYU	NYU
にょ	NYO	NYO	NYO
みゃ	MYA	MYA	MYA
みゅ	MYU	MYU	MYU
みょ	MYO	MYO	MYO
ひゃ	HYA	HYA	HYA
びゃ	BYA	BYA	BYA
ぴゃ	PYA	PYA	PYA

Table 2-6. Japanese transliteration—palatalized syllables

Kana	Hepburn	Kunrei	Nippon
ひゅ	HYU	HYU	HYU
びゅ	BYU	BYU	BYU
ぴゅ	PYU	PYU	PYU
ひょ	HYO	HYO	HYO
びょ	BYO	BYO	BYO
ぴょ	PYO	PYO	PYO
りゃ	RYA	RYA	RYA
りゅ	RYU	RYU	RYU
りょ	RYO	RYO	RYO

Table 2-7 lists what are considered to be long (or doubled) vowels. The first five rows are hiragana, and the last five are katakana. Note that only the long hiragana *i*—written いい, and transliterated *ii*—is common to all three systems, and that the Kunrei and Nippon systems are identical in this regard.

Table 2-7. Japanese transliteration—long vowels

Kana	Hepburn	Kunrei	Nippon
ああ	Ā	Â	Â
いい	II	II	II
うう	Ū	Û	Û
ええ	Ē	Ê	Ê
えい	EI	EI	EI
おう	Ō	Ô	Ô
アー	Ā	Â	Â
イー	Ī	Î	Î
ウー	Ū	Û	Û
エー	Ē	Ê	Ê
オー	Ō	Ô	Ô

The only difference among these systems' long vowel transliterations is the use of a macron (Hepburn) versus a circumflex (Kunrei and Nippon). Almost all Latin fonts include circumflexed vowels, but those with macroned vowels are still relatively rare.

Finally, Table 2-8 shows some examples of how to transliterate Japanese double consonants, all of which use a small つ or ツ (*tsu*).

Table 2-8. Japanese transliteration—double consonants

Example	Transliteration
かっこ	*kakko*
いっしょ	*issho*
ふっそ	*fusso*
ねっちゅう	*netchū*
しって	*shitte*
ビット	*bitto*
ベッド	*beddo*
バッハ	*bahha*

Korean Transliteration Methods

There are now four generally accepted methods for transliterating Korean text: *The Revised Romanization of Korean*[*] (국어의 로마자 표기법/國語의 로마字 表記法 *gugeoui romaja pyogibeop*), established on July 7, 2000; *Ministry of Education* (문교부/文教部 *mungyobu*, derived from and sometimes referred to as *McCune-Reischauer*), established on January 13, 1984;[†] *Korean Language Society* (한글 학회/한글 學會 *hangeul hakhoe*), established on February 21, 1984;[‡] and ISO/TR 11941:1996 (*Information Documentation—Transliteration of Korean Script into Latin Characters*), established in 1996. The transliterated Korean text in this book adheres to the RRK transliteration method because it represents the official way in which Korean text is transliterated, at least in South Korea.[§] Other transliteration methods, not covered in this book, include Yale, Lukoff, and Horne.

Table 2-9 lists the jamo that represent consonants, along with their representation in these three transliteration methods. Also included are the ISO/TR 11941:1996 transliterations when these jamo serve as the final consonant of a syllable. ISO/TR 11941:1996 Method 1

* *http://www.mct.go.kr/english/roman/roman.jsp*

† *http://www.hangeul.or.kr/24_1.htm*

‡ *http://www.hangeul.or.kr/hnp/hanroma.hwp*

§ Notable exceptions include words such as *hangul*, which should really be transliterated as *hangeul*.

is used for North Korea (DPRK), and Method 2 is used for South Korea (ROK). Uppercase is used solely for clarity.

Table 2-9. Korean transliteration—consonants

Jamo	RRK[a]	MOE	KLS	ISO (DPRK)	Final	ISO (ROK)	Final
ㄱ	G/K[b]—G	K/G	G	K	K	G	G
ㄴ	N	N	N	N	N	N	N
ㄷ	D/T[c]—D	T/D	D	T	T	D	D
ㄹ	R/L[d]—L	R/L	L	R	L	R	L
ㅁ	M	M	M	M	M	M	M
ㅂ	B/P[e]—B	P/B	B	P	P	B	B
ㅅ	S	S/SH	S	S	S	S	S
ㅇ	None/NG	None/NG	None/NG	None	NG	None	NG
ㅈ	J	CH/J	J	C	C	J	J
ㅊ	CH	CH’	CH	CH	CH	C	C
ㅋ	K	K’	K	KH	KH	K	K
ㅌ	T	T’	T	TH	TH	T	T
ㅍ	P	P’	P	PH	PH	P	P
ㅎ	H	H	H	H	H	H	H
ㄲ	KK	KK	GG	KK	KK	GG	GG
ㄸ	TT	TT	DD	TT	n/a	DD	n/a
ㅃ	PP	PP	BB	PP	n/a	BB	n/a
ㅆ	SS	SS	SS	SS	SS	SS	SS
ㅉ	JJ	TCH	JJ	CC	n/a	JJ	n/a

a. When Clause 8 of this system is invoked, the character shown after the dash shall be used.

b. *G* is used before vowels, and *K* is used when followed by another consonant or to form the final sound of a word.

c. *D* is used before vowels, and *T* is used when followed by another consonant or to form the final sound of a word.

d. *R* is used before vowels, and *L* is used when followed by another consonant or to form the final sound of a word.

e. *B* is used before vowels, and *P* is used when followed by another consonant or to form the final sound of a word.

Note that some of the double jamo do not occur at the end of syllables. Also, some of these transliteration methods, most notably the Ministry of Education system, have a number of rules that dictate how to transliterate certain jamo depending on their context.

This context dependency arises because the MOE and RRK (without Clause 8 invoked) are transcription systems and not transliteration systems, and because the hangul script is morphophonemic (represents the underlying root forms) and not phonemic (represents the actual sounds). If a one-to-one correspondence and round-trip conversion are desired, invoking Clause 8 of the RRK system accomplishes both. ICU's Hangul-Latin transliteration function does this.*

For example, the ㄱ jamo is transliterated as *k* when voiceless (such as at the beginning of a word or not between two voiced sounds), and as *g* when between two voiced sounds (such as between two vowels, or between a vowel and a voiced consonant). Thus, ㄱ in 강물 is pronounced *k* because it is voiceless, but the same ㄱ in 한강 is pronounced *g* because it is between a voiced consonant (ㄴ) and a vowel (ㅏ), and becomes voiced itself.

Clause 8 of RRK states that when it is necessary to convert transliterated Korean back into hangul for special purposes, such as for academic papers, transliteration is done according to hangul spelling and not by pronunciation. The jamo ㄱ, ㄷ, ㅂ, and ㄹ are thus always written as *g*, *d*, *b*, and *l*. Furthermore, when ㅇ has no phonetic value, it is replaced by a hyphen, which may also be used to separate or distinguish syllables.

ISO/TR 11941:1996 also defines transliterations for compound consonant jamo that appear only at the end of hangul syllables, all of which are listed in Table 2-10.

Table 2-10. ISO/TR 11941:1996 compound jamo transliteration

Jamo	DPRK	ROK
ㄳ	KS	GS
ㄵ	NJ	NJ
ㄶ	NH	NH
ㄺ	LK	LG
ㄻ	LM	LM
ㄼ	LP	LB
ㄽ	LS	LS
ㄾ	LTH	LT
ㄿ	LPH	LP
ㅀ	LH	LH
ㅄ	PS	BS

* *http://icu-project.org/userguide/Transform.html*

Table 2-11 lists the jamo that represent vowels and diphthongs, along with their representations in the three transliteration methods. Again, uppercase is used for clarity, and differences have been highlighted.

Table 2-11. Korean transliteration—vowels

Jamo	RRK	MOE	KLS	ISO (DPRK and ROK)
ㅏ	A	A	A	A
ㅑ	YA	YA	YA	YA
ㅓ	EO	Ŏ	EO	EO
ㅕ	YEO	YŎ	YEO	YEO
ㅗ	O	O	O	O
ㅛ	YO	YO	YO	YO
ㅜ	U	U	U	U
ㅠ	YU	YU	YU	YU
ㅡ	EU	Ŭ	EU	EU
ㅣ	I	I	I	I
ㅐ	AE	AE	AE	AE
ㅒ	YAE	YAE	YAE	YAE
ㅔ	E	E	E	E
ㅖ	YE	YE	YE	YE
ㅘ	WA	WA	WA	WA
ㅙ	WAE	WAE	WAE	WAE
ㅚ	OE	OE	OE	OE
ㅝ	WO	WŎ	WEO	WEO
ㅞ	WE	WE	WE	WE
ㅟ	WI	WI	WI	WI
ㅢ	UI	ŬI	EUI	YI

Note that the ISO/TR 11941:1996 transliteration method is identical for both North and South Korea (DPRK and ROK, respectively).

As with most transliteration methods, there are countless exceptions and special cases. Tables 2-9 and 2-11 provide only the basic transliterations for jamo. It is when you start combining consonants and vowels that exceptions and special cases become an issue. In fact, a common exception is the transliteration of the hangul used for the Korean surname "Lee." I suggest that you try Younghong Cho's *Korean Transliteration Tools*.[*]

Vietnamese Romanization Methods

Writing Vietnamese using Latin characters—called *Quốc ngữ* (國語)—is considered the most acceptable method for expressing Vietnamese today. As a result, Quốc ngữ is not considered a transliteration method. As with using Latin characters to represent Chinese, Japanese, and Korean text, it is the currently acceptable means to express Vietnamese in writing. Quốc ngữ is thus a Romanization system.

This writing system is based on Latin script, but is decorated with additional characters and many diacritic marks. This complexity serves to account for the very rich Vietnamese sound system, complete with tones.

In addition to the upper- and lowercase English alphabet, Quốc ngữ requires two additional consonants and 12 additional base characters (that is, characters that do not indicate tone), as shown in Table 2-12.

Table 2-12. Additional Quốc ngữ consonants and base characters

Character class	Consonants	Base characters
Lowercase	đ	ăâêôơư
Uppercase	Đ	ĂÂÊÔƠƯ

The modifiers that are used for the base vowels, in the order shown in Table 2-12, are called *breve* or *short* (*trăng* or *mũ ngược* in Vietnamese), *circumflex* (*mũ* in Vietnamese), and *horn* (*móc* or *râu* in Vietnamese).

While these additional base characters include diacritic marks and other attachments, they do not indicate tone. There are six tones in Vietnamese, five of which are written with a tone mark. Every Vietnamese word must have a tone. The diacritic marks for these six tones are shown in Table 2-13, along with their names.

Table 2-13. The six Vietnamese tones

Tone mark	Name in Vietnamese	Name in English
none	Không dấu	none
◌̀	Huyền	Grave

* *http://www.sori.org/hangul/conv2kr.cgi*

Table 2-13. The six Vietnamese tones

Tone mark	Name in Vietnamese	Name in English
◌̉	Hỏi	Hook above, curl, or *hoi*
◌̃	Ngã	Tilde
◌́	Sắc	Acute
◌̣	Nặng	Dot below, underdot, or *nang*

All of the diacritic-annotated characters that are required for the Quốc ngữ writing system, which are combinations of base characters plus tones, are provided in Table 2-14.

Table 2-14. Quốc ngữ base characters and tone marks

		Base characters											
		a A	ă Ă	â Â	e E	ê Ê	i I	o O	ô Ô	ơ Ơ	u U	ư Ư	y Y
Tone marks	◌̀	à À	ằ Ằ	ầ Ầ	è È	ề Ề	ì Ì	ò Ò	ồ Ồ	ờ Ờ	ù Ù	ừ Ừ	ỳ Ỳ
	◌̉	ả Ả	ẳ Ẳ	ẩ Ẩ	ẻ Ẻ	ể Ể	ỉ Ỉ	ỏ Ỏ	ổ Ổ	ở Ở	ủ Ủ	ử Ử	ỷ Ỷ
	◌̃	ã Ã	ẵ Ẵ	ẫ Ẫ	ẽ Ẽ	ễ Ễ	ĩ Ĩ	õ Õ	ỗ Ỗ	ỡ Ỡ	ũ Ũ	ữ Ữ	ỹ Ỹ
	◌́	á Á	ắ Ắ	ấ Ấ	é É	ế Ế	í Í	ó Ó	ố Ố	ớ Ớ	ú Ú	ứ Ứ	ý Ý
	◌̣	ạ Ạ	ặ Ặ	ậ Ậ	ẹ Ẹ	ệ Ệ	ị Ị	ọ Ọ	ộ Ộ	ợ Ợ	ụ Ụ	ự Ự	ỵ Ỵ

In summary, Quốc ngữ requires 134 additional characters beyond the English alphabet. Fourteen are additional base characters (see Table 2-12), and the remaining 120 include diacritic marks that indicate tone (see Table 2-14). Although the U+1E*xx* block of Unicode provides the additional characters necessary for Vietnamese in precomposed form, they can still be represented as sequences that are composed of a base character followed by one or more diacritic marks. Windows Code Page 1258 includes some, but not all, of these precomposed characters.

ASCII-based Vietnamese transliteration methods

When only the ASCII character set is available, it is still possible to represent Vietnamese text using well-established systems. The two most common ASCII-based transliteration methods are called *VIetnamese Quoted-Readable* (VIQR) and *VSCII MNEMonic* (VSCII-MNEM). The VIQR system is documented in RFC 1456.* Table 2-15 illustrates how Quốc ngữ base characters and tones are represented in these two systems.

* *http://www.ietf.org/rfc/rfc1456.txt*

Table 2-15. VIQR and VSCII-MNEM transliteration methods

	Quốc ngữ		VIQR		VSCII-MNEM	
Base characters	ă	Ă	a(	A(	a<	A<
	â	Â	a^	A^	a>	A>
	ê	Ê	e^	E^	e>	E>
	ô	Ô	o^	O^	o>	O>
	ơ	Ơ	o+	O+	o*	O*
	ư	Ư	u+	U+	u*	U*
	đ	Đ	dd	DD	dd	DD
Tones	à	À	a`	A`	a!	A!
	ả	Ả	a?	A?	a?	A?
	ã	Ã	a~	A~	a"	A"
	á	Á	a'	A'	a'	A'
	ạ	Ạ	a.	A.	a.	A.

Table 2-16 illustrates how base characters and tones are combined in each system. Note how the base character's ASCII-based annotation comes before the ASCII-based tone mark.

Table 2-16. Base character plus tones using VIQR and VSCII-MNEM methods

Quốc ngữ		VIQR		VSCII-MNEM	
ờ	Ờ	o+`	O+`	o*!	O*!
ở	Ở	o+?	O+?	o*?	O*?
ỡ	Ỡ	o+~	O+~	o*"	O*"
ớ	Ớ	o+'	O+'	o*'	O*'
ợ	Ợ	o+.	O+.	o*.	O*.

Zhuyin/Bopomofo

Zhuyin, developed in the early part of the 20th century, is a method for transcribing Chinese text using ideograph elements for their reading value. In other words, it is a transliteration system. It is also known as the *National Phonetic System* (注音符号 *zhùyīn fúhào*) or *bopomofo*. The name *bopomofo* is derived from the readings of the first four characters in the character set: *b*, *p*, *m*, and *f*. There are a total of 37 characters (representing 21 consonants and 16 vowels), along with five symbols to indicate tone (one of which has no glyph) in the zhuyin character set.

Table 2-17 illustrates each of the zhuyin characters, along with the ideograph from which they were derived, and their reading. Those that represent vowels are at the end of the table.

Table 2-17. Zhuyin characters

Zhuyin	Ideograph	Reading—Pinyin
ㄅ	勹	B
ㄆ	攵	P
ㄇ	𠘨	M
ㄈ	匚	F
ㄉ	𠚣	D
ㄊ	𠫓	T
ㄋ	乃	N
ㄌ	力	L
ㄍ	巜	G
ㄎ	丂	K
ㄏ	厂	H
ㄐ	丩	J
ㄑ	𡿨	Q
ㄒ	丅	X
ㄓ	㞢	ZH
ㄔ	彳	CH
ㄕ	尸	SH
ㄖ	日	R
ㄗ	卩	Z
ㄘ	𠀁	C
ㄙ	厶	S
ㄚ	丫	A

Table 2-17. Zhuyin characters

Zhuyin	Ideograph	Reading—Pinyin
ㄛ	𠀀	O
ㄜ	左	E
ㄝ	也	EI
ㄞ	𠀅	AI
ㄟ	乁	EI
ㄠ	幺	AO
ㄡ	又	OU
ㄢ	𢎘	AN
ㄣ	𠃑	EN
ㄤ	尢	ANG
ㄥ	𠃋	ENG
ㄦ	儿	ER
ㄧ or 一	丨	I
ㄨ	㐅	U
ㄩ	凵	IU

The zhuyin character set is included in character sets developed in China (GB 2312-80 and GB/T 12345-90, Row 8) and Taiwan (CNS 11643-2007, Plane 1, Row 5). This set of characters is identical across these two Chinese locales, with one exception, which is indicated in Table 2-17 with two different characters: "ㄧ" is used in China, and "一" is used in Taiwan.

Kana

The most frequently used script found in Japanese text is *kana*. It is a collective term for two closely related scripts, as follows:

- Hiragana
- Katakana

Although one would expect to find kana characters only in Japanese character sets, they are, in fact, part of some Chinese and Korean character sets, in particular GB 2312-80 and KS X 1001:2004. In fact, kana are encoded at the same code points in the case of GB 2312-80! Why in the world would Chinese and Korean character sets include kana? Most likely for the purposes of creating Japanese text using a Chinese or Korean character set.[*] After all, many of the ideographs are common across these locales.

The following sections provide detailed information about kana, along with how they were derived from ideographs.

Hiragana

Hiragana (平仮名 *hiragana*) are characters that represent sounds, specifically syllables. A syllable is generally composed of a consonant plus a vowel—sometimes a single vowel will do. In Japanese, there are five vowels: *a*, *i*, *u*, *e*, and *o*; and 14 basic consonants: *k*, *s*, *t*, *n*, *h*, *m*, *y*, *r*, *w*, *g*, *z*, *d*, *b*, and *p*. It is important to understand that hiragana is a syllabary, not an alphabet: you cannot decompose a hiragana character into a part that represents the vowel and a part that represents the consonant. Hiragana (and katakana, covered in the next section) is one of the only true syllabaries still in common use today. Table 2-18 illustrates a matrix containing the basic and extended hiragana syllabary.

Table 2-18. The hiragana syllabary

		K	S	T	N	H	M	Y	R	W	G	Z	D	B	P
A	あ	か	さ	た	な	は	ま	や	ら	わ	が	ざ	だ	ば	ぱ
I	い	き	し	ち	に	ひ	み		り	ゐ	ぎ	じ	ぢ	び	ぴ
U	う	く	す	つ	ぬ	ふ	む	ゆ	る		ぐ	ず	づ	ぶ	ぷ
E	え	け	せ	て	ね	へ	め		れ	ゑ	げ	ぜ	で	べ	ぺ
O	お	こ	そ	と	の	ほ	も	よ	ろ	を	ご	ぞ	ど	ぼ	ぽ
N	ん														

The following are some notes to accompany Table 2-18:

- Several hiragana have smaller versions, and are as follows (the standard version is in parentheses): ぁ (あ), ぃ (い), ぅ (う), ぇ (え), ぉ (お), っ (つ), ゃ (や), ゅ (ゆ), ょ (よ), and ゎ (わ).
- Two hiragana, ゐ and ゑ, are no longer commonly used.
- The hiragana を is read as *o*, not *wo*.
- The hiragana ん is considered an independent syllable and is pronounced approximately *ng*.

[*] There is, however, one fatal flaw in the Chinese and Korean implementations of kana. They omitted five symbols used with kana, all of which are encoded in row 1 of JIS X 0208:1997 (Unicode code points also provided): ヽ (01-19; U+30FD), ヾ (01-20; U+30FE), ゝ (01-21; U+309D), ゞ (01-22; U+309E), and ー (01-28; U+30FC).

Notice that some cells do not contain any characters. These sounds are no longer used in Japanese, and thus no longer need a character to represent them. Also, the first block of characters is set in a 5×10 matrix. This is sometimes referred to as the *50 Sounds Table* (50 音表 *gojūon hyō*), so named because it has a capacity of 50 cells. The other blocks of characters are the same as those in the first block, but with diacritic marks.

Diacritic marks serve to annotate characters with additional information—usually a changed pronunciation. In the West you commonly see accented characters such as *á*, *à*, *â*, *ä*, *ã*, and *å*. The accents are called *diacritic marks*.

In Japanese there are two diacritic marks: *dakuten* (also called *voiced* and *nigori*) and *handakuten* (also called *semi-voiced* and *maru*). The dakuten (濁点 *dakuten*) appears as two short diagonal strokes (゛) in the upper-right corner of some kana characters. The dakuten serves to voice the consonant portion of the kana character to which it is attached.[*] Examples of voiceless consonants include *k*, *s*, and *t*. Their voiced counterparts are *g*, *z*, and *d*, respectively. Hiragana *ka* (か) becomes *ga* (が) with the addition of the dakuten. The *b* sound is a special voiced version of a voiced *h* in Japanese.

The handakuten (半濁点 *handakuten*) appears as a small open circle (゜) in the upper-right corner of kana characters that begin with the *h* consonant. It transforms this *h* sound into a *p* sound.

Hiragana were derived by cursively writing kanji, but no longer carry the meaning of the kanji from which they were derived. Table 2-22 lists the kanji from which the basic hiragana characters were derived.

In modern Japanese, hiragana are used to write grammatical words, inflectional endings for verbs and adjectives, and some nouns.[†] They can also be used as a fallback (read "crutch") in case you forget how to write a kanji—the hiragana that represent the reading of a kanji are used in this case. In summary, hiragana are used to write some native Japanese words.

Table 2-19 enumerates the hiragana characters that are included in the JIS X 0208:1997 and JIS X 0213:2004 character set standards. For both character set standards, all of these characters are in Row 4.

* *Voicing* is a linguistic term referring to the vibration of the vocal bands while articulating a sound.

† Prior to the Japanese writing system reforms that took place after World War II, hiragana and katakana were used interchangeably, and many legal documents used katakana for inflectional endings and for purposes now used exclusively by hiragana.

Table 2-19. Hiragana characters in JIS standards

Standard	Characters
JIS X 0208:1997	ぁあぃいぅうぇえぉおかがきぎくぐけげこごさざしじすずせぜそぞただちぢっつづてでとどなにぬねのはばぱひびぴふぶぷへべぺほぼぽまみむめもゃやゅゆょよらりるれろゎわゐゑをん
JIS X 0213:2004	ゔゕゖか゚き゚く゚け゚こ゚

Note how these characters have a cursive or calligraphic look to them (cursive and calligraphic refer to a smoother, handwritten style of characters). Keep these shapes in mind while we move on to katakana.

Katakana

Katakana (片仮名 *katakana*), like hiragana, is a syllabary, and with minor exceptions, they represent the same set of sounds as hiragana. Their modern usage, however, differs from hiragana. Where hiragana are used to write native Japanese words, katakana are primarily used to write words of foreign origin, called *gairaigo* (外来語 *gairaigo*), to write onomatopoeic words,* to express "scientific" names of plants and animals, or for emphasis—similar to the use of italics to represent foreign words and to express emphasis in English. For example, the Japanese word for *bread* is written パン and is pronounced *pan*. It was borrowed from the Portuguese word *pão*, which is pronounced sort of like *pown*. Katakana are also used to write foreign names. Table 2-20 illustrates the basic and extended katakana syllabary.

Table 2-20. The katakana syllabary

		K	S	T	N	H	M	Y	R	W	G	Z	D	B	P
A	ア	カ	サ	タ	ナ	ハ	マ	ヤ	ラ	ワ	ガ	ザ	ダ	バ	パ
I	イ	キ	シ	チ	ニ	ヒ	ミ		リ	ヰ	ギ	ジ	ヂ	ビ	ピ
U	ウ	ク	ス	ツ	ヌ	フ	ム	ユ	ル		グ	ズ	ヅ	ブ	プ
E	エ	ケ	セ	テ	ネ	ヘ	メ		レ	ヱ	ゲ	ゼ	デ	ベ	ペ
O	オ	コ	ソ	ト	ノ	ホ	モ	ヨ	ロ	ヲ	ゴ	ゾ	ド	ボ	ポ
N	ン														

* *Onomatopoeic* refers to words that serve to describe a sound, such as *buzz* or *hiss* in English. In Japanese, for example, ブクブク (*bukubuku*) represents the sound of a balloon expanding.

The following are some notes to accompany Table 2-20:

- Several katakana have smaller versions, and are as follows (the standard version is in parentheses): ァ (ア), ィ (イ), ゥ (ウ), ェ (エ), ォ (オ), ヵ (カ), ヶ (ケ), ッ (ツ), ャ (ヤ), ュ (ユ), ョ (ヨ), and ヮ (ワ).
- Two katakana, ヰ and ヱ, are no longer commonly used.
- The katakana ヲ is read as *o*, not *wo*.
- The katakana ン is considered an independent syllable, and is pronounced approximately *ng*.

Katakana were derived by extracting a single portion of a whole kanji, and, like hiragana, no longer carry the meaning of the kanji from which they were derived. If you compare several of these characters to some kanji, you may recognize common shapes. Table 2-20 lists the basic katakana characters, along with the kanji from which they were derived.

Table 2-21 enumerates the katakana characters that are included in the JIS X 0208:1997 and JIS X 0213:2004 character set standards. As shown in the table, those in JIS X 0208:1997 are in Row 5, and those in JIS X 0213:2004 are in Plane 1, but spread across Rows 5 through 7.

Table 2-21. Katakana characters in JIS standards

Standard	Row	Characters
JIS X 0208:1997	5	ァアィイゥウェエォオカガキギクグケゲコゴサザシジスズセゼソゾタダチヂッツヅテデトドナニヌネノハバパヒビピフブプヘベペホボポマミムメモャヤュユョヨラリルレロヮワヰヱヲンヴヵヶ
JIS X 0213:2004	5	カ゚キ゚ク゚ケ゚コ゚セ゚ツ゚ト゚
	6	ㇰㇱㇲㇳㇴㇵㇶㇷㇸㇹㇷ゚ㇺㇻㇼㇽㇾㇿ
	7	ヷヸヹヺ

Katakana, unlike hiragana, have a squared, more rigid feel to them. Structurally speaking, they are quite similar in appearance to kanji, which we discuss later.

The Development of Kana

You already know that kana were derived from kanji, and Table 2-22 provides a complete listing of kana characters, along with the kanji from which they were derived.

Table 2-22. The ideographs from which kana were derived

Katakana		Ideograph		Hiragana
ア	阿		安	あ
イ	伊		以	い
ウ		宇		う
エ	江		衣	え
オ		於		お
カ		加		か
キ		幾		き
ク		久		く
ケ	介		計	け
コ		己		こ
サ	散		左	さ
シ		之		し
ス	須		寸	す
セ		世		せ
ソ		曽		そ
タ	多		太	た
チ	千		知	ち
ツ		川		つ
テ		天		て
ト		止		と
ナ		奈		な
ニ	二		仁	に
ヌ		奴		ぬ
ネ		祢		ね
ノ		乃		の
ハ	八		波	は
ヒ		比		ひ