CJKV Information Processing

Ken Lunde

O'REILLY™

Beijing · Cambridge · Köln · Paris · Sebastopol · Taipei · Tokyo

CJKV Information Processing
by Ken Lunde

Published by O'Reilly & Associates, Inc., 101 Morris Street, Sebastopol, CA 95472.

Editors: Tim O'Reilly, Peter Mui, and Gigi Estabrook

Production Editors: Ken Lunde and Jane Ellin

Printing History:

January 1999: First Edition.

ISBN: 1-56592-224-7

In this chapter:
- *Multiple Writing Systems*
- *Character Set Standards*
- *Encoding Methods*
- *Input Methods*
- *Typography*
- *Basic Concepts and Terminology*

1 CJKV Information Processing Overview

A lot of mystique and intrigue surrounds how CJKV—Chinese, Japanese, Korean, and Vietnamese—text is handled on computer systems. Although I agree with there being intrigue, there is far too much mystique, in my opinion. Much of this mystery is due to a lack of information, or simply a lack of information written in a language other than Chinese, Japanese, Korean, or Vietnamese. Nevertheless, many fine folks, like you, 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. You need only 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 writing systems such as those used to represent CJKV text is a more recent phenomenon. This adaptation developed in various stages, and continues today.

There 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, writing systems
- CJKV character set standards enumerate thousands or tens of thousands of characters, which is orders of magnitude more than used in the West

* 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, *Information Processing Techniques*, beginning on page 433.

- There is no universally recognized or accepted CJKV character set standard such as ASCII for writing English—although Unicode can be considered a good first attempt
- There is no universally recognized or accepted CJKV encoding system such as ASCII encoding—again, the various Unicode encodings can be considered an attempt at accomplishing this
- There is no universally recognized or accepted input device such as the QWERTY keyboard array—although this same keyboard array, through a method of transliteration, can be 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

You will learn that the ASCII character set standard is not as universal as most people think—different flavors of ASCII exist, as do different ASCII encoding methods. You will begin to wonder why so many developers assume that everyone uses ASCII.

This chapter also includes several sections that explain and illustrate some very basic yet important computing concepts, such as notation and byte order, that relate to material in the remainder of this book. 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 previously learned.

Multiple Writing Systems

CJKV text is typically composed of a mixture of different writing systems. Japanese, as an example, is unique in that it uses four different writing systems. Others, such as Chinese and Korean, use less than four writing systems. Japanese is one of the few, if not the only, languages that exhibit this characteristic of so many writing systems 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.* The four Japanese writing systems are Latin characters, hiragana, katakana, and kanji (collectively referred to as "Chinese characters" regardless of the language). You are already familiar with Latin characters because the English language is written with these—this writing system consists of the

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

upper- and lowercase Latin alphabet, which are the characters often found on typewriter keys. Hiragana and katakana are native Japanese syllabaries (see Appendix X, *Glossary*, for a definition of “syllabary”). Both hiragana and katakana represent the same set of 108 syllables, and are collectively known as *kana*. Kanji are characters that the Japanese borrowed from China over 1,600 years ago—Chinese characters number in the thousands, and encompass meaning, reading, and shape.

Now let’s look at an example sentence composed of these four writing systems. This should serve to illustrate how the different Japanese writing systems 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.

> 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 this case there is a single abbreviation, EUC (short for *Extended Unix Code*, which refers to a locale-independent encoding method, a topic to be covered in Chapter 4, *Encoding Methods*, of this book). 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*) in Japanese.

Now let’s underline the katakana 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。

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 to be spelled out with katakana characters to closely match the borrowed words.

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 described above, 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 above sentence.

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 read *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 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 Chinese characters (called *hànzì* in Chinese, *kanji* in Japanese, *hanja* in Korean, and *chữ Hán* 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, Chinese characters appear to be more complex than the other characters in the sentence. This happens to be true most of the time. Chinese characters represent meanings, and are often called *ideographs*, *pictographs*, or *logographs*.[*] Chinese characters are also assigned one or more readings (pronunci-

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

ations), each of which is determined by context. While their readings differ depending on the language (Chinese, Japanese, Korean, or Vietnamese), Chinese characters often have the same meaning. This makes it possible for Japanese to understand (but not necessarily to pronounce) some very basic Chinese, Korean, and Vietnamese texts. Table 1-3 provides a listing of the underlined Chinese characters and compounds (words composed of two or more Chinese characters) thereof, along with their meanings and readings.

Table 1-3: Sample Chinese Characters and Chinese Character Compounds

Chinese Characters	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 writing systems they use.

Table 1-4: CJKV Locales and Their Writing Systems

Locale	Writing Systems
China	Latin, zhuyin, 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 writing systems used in CJKV locales. We discuss these writing systems in greater detail in Chapter 2, *Writing Systems*.

Table 1-5: Sample CJKV Characters

Character Class	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	ㄅ ㄆ ㄇ ㄈ ㄉ ㄊ ㄋ ㄌ ㄍ ㄎ … ㄠ ㄡ ㄢ ㄣ ㄤ ㄥ ㄦ ㄧ ㄨ ㄩ
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

Table 1-5: Sample CJKV Characters (continued)

Character Class	Sample Characters
Jamo	ㄱ ㄲ ㄳ ㄴ ㄵ ㄶ ㄷ ㄸ ㄹ ㄺ … ㆅ ㆆ ㆇ ㆈ ㆉ ㆊ ㆋ ㆌ ㆍ ㆎ
Hangul	가각간갇갈갉갊감갑값 … 흽힁히힉힌힐힘힙힛힝
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 used are each of these character classes? Given an average sampling of Japanese writing, one normally finds 30 percent kanji, 60 percent hiragana, and 10 percent 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, and most Chinese texts are composed of only hanzi.* Latin characters are used the least, except in Vietnam.

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 percent 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, *Dictionaries and Dictionary Software*, they are no substitute for a human teacher.

Character Set Standards

A character set simply provides a common *bucket* of characters. You may have never thought of it this way, but the English alphabet is an example of a character

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

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 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 Chinese characters, however, character set standards play an especially vital role. They specify which Chinese characters—out of the tens of thousands in existence—are the most important to learn. The current Japanese set, called Jōyō Kanji (常用漢字 *jōyō kanji*), although advisory, limits the number of Chinese characters 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 *non-coded* 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 non-coded 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 non-coded 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 percent 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).

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

Additional incompatibility occurs because operating system developers take 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. This book documents these character sets, making it easier to grapple with such confusion.

Encoding Methods

Encoding is the process of mapping a character to a numeric value. By doing this, you create the ability to uniquely identify a character through its associated numeric value. 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 are commonly used. For example, ISO-2022-KR, EUC-KR, Johab, and Unified Hangul Code (UHC) for Korean.

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 can hold 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 seven or eight bits. Seven bits in an array allow for up to 128 unique combinations, or values; eight 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.

The first attempt to encode Chinese characters 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 two 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.

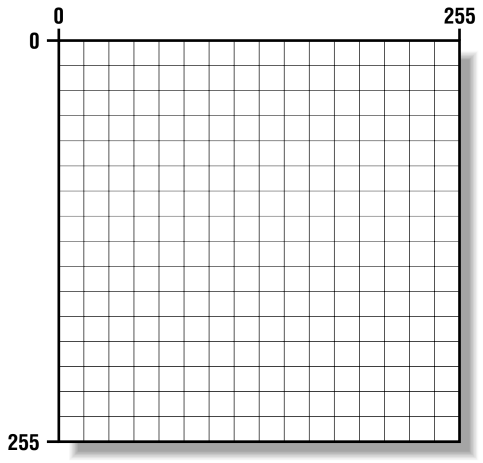


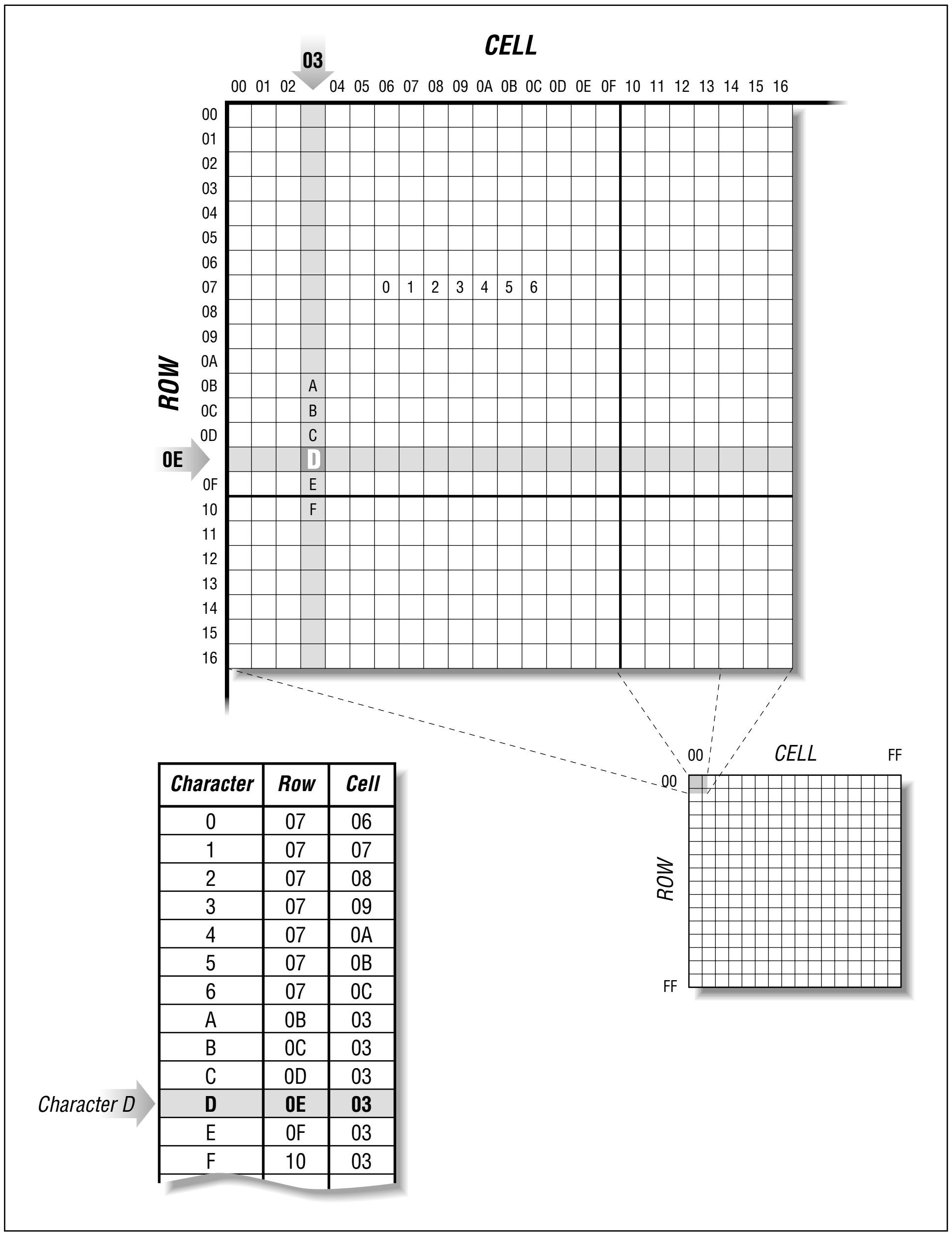
Figure 1-1: 256×256 encoding matrix

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 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. However, languages with large character sets—those spoken in the CJKV locales—require two bytes to encode characters. A mixed one- and two-byte character stream efficiently represents a mixture of English and Chinese text.

Along with discussions about character sets and encodings, you will encounter the terms "row" and "cell" again and again in this book. These refer to the axes of a matrix used to hold and encode characters. A matrix is composed of rows, and a row is made up of cells. The first byte of the character specifies the row, and the second byte specifies the cell within the row. Figure 1-2 illustrates a matrix and how characters' positions correspond to row and cell values.

* *Code space* refers to the area within the (usual) 256×256 encoding matrix that can be used for encoding characters. Most of the figures in Chapter 4 and Appendix D, *Vendor Encoding Methods*, illustrate code spaces that fall within this 256×256 matrix.


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.

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 *conversion dictionaries*.

Most CJKV text is typically input in two stages:

1. The user types raw keyboard input, which the computer interprets using the input method and the conversion dictionary to display a list of *candidate* characters (*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 is handled on your computer depends greatly on the quality (and vintage) of the input software you are using.

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)
- Native-script input (zhuyin for Chinese, 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 percent 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 Chinese characters be input one at a time? While single Chinese-character 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 Chinese character
- Chinese character compound
- Chinese character phrase

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

Table 1-6: Single Chinese Character Input—Japanese

Character	Reading	Chinese Characters with Identical Readings
漢	`K A N`	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	`J I`	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 Chinese characters with those readings, so you may have to wade through a long list of candidate Chinese characters before you find the correct one. A more efficient way is to input them as one unit, called a Chinese character compound. This produces a much shorter list of candidates from which to choose. Table 1-7 illustrates the two Chinese characters input as a compound, along with candidate compounds with the same reading.

Table 1-7: Chinese Character 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 Chinese character compounds is much shorter in this case. There is an even higher-level input unit called a Chinese character phrase. This is similar to inputting two or more Chinese characters as a single compound, but adds another element, similar to a preposition in English, that makes the whole string into a phrase. An example of a Chinese character phrase is 漢字は, which means "the Chinese character" in Japanese. Because Chinese-language text is composed solely of hanzi, Chinese character phrase applies 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 Chinese character phrases, 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 writing system, hangul, is composed. Whether input is by a QWERTY or a Korean keyboard array, Korean input involves the entering of hangul elements called *jamo*. As the jamo are input, the operating system or input software attempts to compose hangul using an automaton. Because of how hangul are composed of jamo, the user may have up to three alternatives for deleting characters:

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

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

Typography

CJKV text can usually be written or set in one of two orientations: left to right, top to bottom (horizontal setting, as in this book); and top to bottom, right to left (vertical setting). Chapter 7, *Typography*, provides plenty of examples of horizontal versus vertical writing. Vertical writing orientation, more often than not, causes problems with Western-language software. 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-degree clockwise rotation) or a different position within the *em-square*.[*] Chapter 7 provides some sample text set both horizontally and vertically, and illustrates some characters that require special treatment.

In addition to the two writing directions for CJKV text, there are other special text formatting considerations, such as special rules for wrapping characters at the ends of lines, special justification, metrics adjustment, and a way to annotate characters.

Basic Concepts and Terminology

Now I'll define some basic concepts which will help carry you through this entire book. These concepts are posed as questions. After all, these are 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 Those 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 confusing) ones are explained in the following section, but when in doubt, consult Appendix X.

What is the difference between GB and GB/T?

Most references to "GB" mean the GB 2312-80 character set standard, which represents the most widely implemented character set for Chinese.

GB stands for "Guo Biao" (国标 *guóbiāo*), which is short for "Guojia Biaozhun" (国家标准 *guójiā biāozhǔn*), and 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." Yet another myth to blow out of the water. The "T" in "GB/T" actually stands for "Tui" (推 *tuī*), which is short for "Tuijian" (推荐 *tuījiàn*), and means "recommended." It

[*] 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.

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.”

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 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 symbol for JIS. 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 JIS manuals. Context should usually make its meaning clear. The term 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 JIS 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 *really* 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.

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

* There are even JIS standards for manufacturing toilet paper!

Table 1-8: JIS Standard Designation Changes (continued)

JIS "C" Series	**JIS "X" Series**
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 a decade 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 *hangug gongeob gyugyeog*). All Korean character set standard designations begin with "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 1005-1
KS C 5861	KS X 2901
KS C 5715	KS X 5002

Because these changes are very recent, it may take years until they are reflected in software and documentation.

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

Are VISCII and VSCII identical? What about TCVN?

While 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, *Character Set Standards*, beginning on page 78. Appendix S, *Single-Byte Code Tables*, provides complete encoding tables for VISCII and VSCII, which better illustrate their differences.

TCVN stands for *Tiêu Chuẩn Việt Nam*, which means "Vietnamese Standard" in Vietnamese. Like GB, JIS, and KS, it represents the first portion of Vietnamese standard designations.

What Are Internationalization and Localization?

Internationalization (often abbreviated as I18N—the initial letter "I" followed by the middle 18 letters followed by the final letter "N") is a blanket term referring to the process of preparing software so that it can be used by more than one culture, region, or locale.* Localization (often abbreviated as L10N) is the process of adapting software to one specific culture, region, or locale. Japanization (often abbreviated as J10N) is a specific instance of L10N. While this book does not necessarily address all of these issues, you will find information pertinent to internationalization and localization.

Either way, I18N or L10N are often desired by users because they provide menus and documentation written in the language of the target locale. They often require special character set handling because so many non-Latin character sets require more than one byte to represent all their characters.

What Are the Multilingual and Locale Models?

There are two basic models for internationalization: the *locale model* and the *multilingual model*. The locale model implements 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 are specific to a given culture, region, or locale.

The multilingual model goes one step further by not requiring you to flip between locales—multilingual systems use a character set that contains all the characters

* Quiz time. Guess what CJKV6N, C10N, G11N, K11N, M17N, S32S, and V12N stand for?

necessary for several cultures or regions. But still, there are cases when it is impossible to correctly render characters without knowing the target locale.

What Is Row-Cell?

Row-Cell is the translated form of the Japanese word 区点 (*kuten*), which literally means "ward [and] point" (or, more intuitively, "row [and] cell").* This idea serves as an encoding-independent method for referring to characters in CJKV character set standards. A Row-Cell code usually consists of four decimal digits—the "Row" portion consists of a two-digit number with a range from 1 to 94; likewise, the "Cell" portion also consists of a two-digit number with a range from 1 to 94. For example, the first character in most CJKV character set standards is 01-01 in Row-Cell notation, and is more often than not a "space" character.

When I provide lists of characters throughout this book, I usually include Row-Cell codes. These are useful for future reference of these data (so that you don't have to hunt for the codes yourself!).

Characters and Glyphs—What Is the Difference?

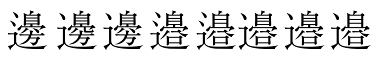
Now here's a topic that is usually beaten to death! The term "character" is an abstract notion indicating a class of shapes declared to have the same meaning or form. The term "glyph" is a specific instance of a character. Sometimes, more than one character can constitute a single glyph, such as the two characters f and i, which can be fused together as the single entity "fi." This glyph "fi" is called a *ligature*. The dollar sign is a good example of a character with several glyphs. There are at least four glyphs for the dollar sign, listed as follows:

- 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: $

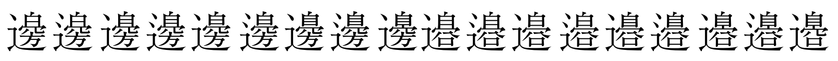
The differences among these four glyphs are minor, but you cannot deny that they still represent the same character, specifically the "dollar sign." Quite often you see a difference in glyph as a difference in typeface. However, there are some characters that have a dozen or more variant forms. Consider the kanji 辺 (*hen*, used in the Japanese family name 渡辺 *watanabe*), which has only two variant forms that are generally available in Japanese fonts (including Unicode-based fonts): 邊 and

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

邉. DTP center Biblos, a developer of "Gaiji" fonts, offers fonts that provide the following eight additional variant forms of this kanji:



Enfour Media, another developer of "Gaiji" fonts, offers fonts that provide the following 18 variant forms of this kanji:



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

You will find that CJKV character set standards do not define the glyph for the characters contained within their pages. Unfortunately (or, fortunately, as the case may be), many think that the glyphs that appear in these manuals are the official ones. Note, however, that the official Jōyō Kanji Table *does* define the glyph shape, at least for the 1,945 kanji contained within. JSA published two manuals that do, in fact, define glyph shapes: JIS X 9051-1984* and JIS X 9052-1983.† They were designed for the JIS X 0208-1983 standard. However, these glyphs have not been widely accepted in industry. It seems as though JSA has no intention of ever revising these documents—this may be their way of not enforcing glyphs.

The one Japanese organization that had a chance in establishing a definitive Japanese glyph standard in Japan is called FDPC, which is short 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 developing members, developed a series of Japanese outline fonts called "Heisei" (平成 *heisei*) typefaces. The first two Heisei typefaces that were released are 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 this book is Heisei Mincho W3. A total of seven weights of both designs have been produced, weights 3 (W3) through 9 (W9). Two weights of Heisei Maru (rounded) Gothic (平成丸ゴシック *heisei maru goshikku*), 4 and 8, have also been developed. The Heisei typefaces have become somewhat commonplace in the Japanese market.

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*

* Previously designated JIS C 6232-1984.

† Previously designated JIS C 6234-1983.

jí shūjùjí), that explicitly define glyphs for the GB 2312-80 character set standard in various typeface styles. These standards are listed in Table 1-10.

Table 1-10: Chinese Glyph Standards

Standard	Page Numbers in Manual	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*

Songti (specified in GB 6345.1-86), Fangsongti, Kaiti, and Heiti 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 a Chinese character's strokes. These standards are useful because they establish bitmapped patterns that offer a compromise between accuracy and legibility. The recent GB 16794.1-1997 standard (信息技术—通用多八位编码字符集48点阵字形 *xīnxī jìshù—tōngyòng duōbāwèi biānmǎ zìfújí 48 diǎnzhèn zìxíng*) is similar to the GB standards listed in Table 1-10, but covers the complete GBK character set and provides 48×48 bitmapped patterns. 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 shūjùjí*), provided 24×24 bitmapped patterns for a single design.

Exactly how character and glyph are defined can differ depending on the source. Table 1-11 provides the ISO (International Organization for Standardization) and The Unicode Consortium definitions for the terms character, glyph, and glyph image.

What Is the Difference Between Typeface and Font?

The term *typeface* refers to the printed style of a glyph or character set. A *font*, on the other hand, refers to a single instance of a typeface, such as a specific point size. This is why the commonly used term *outline font* is a misnomer—the outlines are scalable, which means that they are not specific to any one point size. A better term is *outline font instance*.

Table 1-11: Character, Glyph, and Glyph Image Definitions—ISO and Unicode

Terminology	ISO	Unicode[a]
Character	A member of a set of elements used for the organisation, 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 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.
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*.
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.

[a] *The Unicode Standard, Version 2.0* (Addison-Wesley, 1996).
[b] ISO 10646-1:1993.
[c] ISO 9541-1:1991.

Western typography commonly uses serif, sans serif, and script typeface styles. Table 1-12 lists the common CJKV typeface styles, along with correspondences across locales.

Table 1-12: CJKV Typeface Styles

Western	Chinese[a]	Japanese	Korean
Serif	Song (宋体 *sòngtǐ*)	Mincho (明朝体 *minchōtai*)	Myeongjo (명조체/明朝體 *myeongjoce*)
Sans serif	Hei (黑体 *hēitǐ*)	Gothic (ゴシック体 *goshikkutai*)	Gothic (고딕체/고딕體 *godigce*)
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 *haeseoce*) Haengseo (행서체/行書體 *haengseoce*) Choseo (초서체/草書體 *coseoce*)
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.

Table 1-12 by no means constitutes a complete list of CJKV typeface styles—there are numerous typeface styles for hangul, for example.

What Are Half- and Full-Width Characters?

The terms half- and full-width refer to the relative glyph size of characters. These terms 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, and Chinese characters. 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 1-13 shows the difference in display width between half- and full-width characters (the katakana character used as the example is read *ka*).

Table 1-13: Half- and Full-Width Characters

	Katakana	**Latin**
Half-width	ｶｶｶｶｶ	12345
Full-width	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 display width of a glyph and 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 ones were encoded with two bytes. Now that there is a much richer choice of encoding methods available, this relationship no longer holds true. Table 1-14 lists several popular encoding methods, along with the number of bytes required to represent half- and full-width characters.

Table 1-14: Half- and Full-Width Character Representations

	ASCII	**ISO-2022-JP**	**Shift-JIS**	**EUC-JP**	**ISO 10646-1:1993**
Full-width					
Katakana	…	2 bytes	2 bytes	2 bytes	2 or 4 bytes
Latin	…	2 bytes	2 bytes	2 bytes	2 or 4 bytes
Half-width					
Katakana	…	1 byte	1 byte	2 bytes	2 or 4 bytes
Latin	1 byte	1 byte	1 byte	1 byte	2 or 4 bytes

* In Chinese, these terms are 半形 (*bànxíng*) and 全形 (*quánxíng*), respectively. In Korean, perhaps 반각/半角 (*bangag*) and 전각/全角 (*jeongag*), respectively.

Latin Versus Roman Characters

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, others prefer the term Roman. This book will prefer the term Latin over Roman. Readers of this book should treat both terms synonymously.

When speaking of typeface designs, the use of the term Roman is used in contrast with the term italic.

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 distance. In computer science, common notations for representing the value of bytes are listed in Table 1-15, and all correspond to a different numeric base.

Table 1-15: Decimal 100 in Common Notations

Notation	Base	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, they have been expressed using different notations, and thus take on a different form. Most people (that is, non-nerds) think in decimal notation; however, computers (and some nerds) process information using binary notation (as discussed above, computers process bits, which have two possible values). Below you will find 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 an eight-bit byte. This becomes confusing when dealing with 16-bit encodings. 16 bits can be broken down into two eight-bit bytes, or two octets. 32 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 bits represent 100 (0x64), and the second 95 (0x5F). All 16 bits together as one unit are usually equal to 25695 in decimal or 0x645F in hexadecimal—it may be different depending on a computer's specific architecture. Divide 25695 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 1-16 lists representations of two octets (bytes) and their 16-bit unit equivalent. This is done for you in different notations.

Table 1-16: Octets and 16-Bit Units in Various Notations

Notation	First Octet	Second Octet	16-Bit Unit
Binary	`01100100`	`01011111`	`0110010001011111`
Octal	`144`	`137`	`62137`
Decimal	`100`	`95`	`25695`
Hexadecimal	`64`	`5F`	`645F`

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. Not so with decimal notation, which requires multiplication of the first octet by 256, then addition of the second octet. The ease of going between different representations (octets versus 16-bit units) depends on the notation that you are using. Of course, string concatenation is easier than two mathematical processes. This is why hexadecimal is used so frequently in computers.

In some cases, the order in which byte concatenation takes place matters, such as when the byte order (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 multiple-byte storage units (such as integers, floats, doubles, and so on) appear.[*] One-byte storage units, such as char, do not need this special treatment (that is,

[*] 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.

unless your particular machine or implementation represents them with more than one byte).

- Little endian machines use computing architectures supported by Vax and Intel processors. This typically means that MS-DOS and Windows machines are little endian.
- Big endian machines use computing architectures supported by Motorola and Sun processors. This typically means MacOS and most Unix workstations. Big endian is also known as "network byte order."

Table 1-17 provides an example two-byte value as encoded on little and big endian machines.

Table 1-17: Little and Big Endian Representation

Notation	High Byte	Low Byte	Little Endian	Big Endian
Binary	`01100100`	`01011111`	`0101111101100100`	`0110010001011111`
Hexadecimal	`64`	`5F`	`5F64`	`645F`

A four-byte example, such as 0x64, 0x5F, 0x7E, and 0xA1, becomes 0xA17E5F64 on little endian machines, and 0x645F7EA1 on big endian machines. Note how the bytes 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 Unicode value for a "space" character is 0x0020 for big-endian, and 0x2000 for little-endian.

Now that you understand the concept of endianness, the real question that needs answering is when endianness matters. 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 don't do those terms justice. Here you'll get a definitive answer.

When dealing with encodings that are processed on a per-byte basis, endianness is irrelevant. These encodings support what are known as *multiple-byte* characters. So, what encodings are these? Table 1-18 provides an incomplete yet informative list of these encodings.

There are some encodings that require special endian treatment, and cannot be treated on a per-byte basis. These encodings include what are known as *wide* characters, and almost always provide a facility for indicating the byte order. Table 1-19 lists some encodings that use wide characters.

Table 1-18: Multiple-Byte Character Encodings

Encoding	Encoding Length	Locale
ASCII	one-byte	*not applicable*
ISO-2022	one- and two-byte	CJKV
EUC	one- through four-byte, depending on locale	CJKV
GBK	one- and two-byte	China
Big Five	one- and two-byte	Taiwan
Big Five Plus	one- and two-byte	Taiwan
Shift-JIS	one- and two-byte	Japan
Johab	one- and two-byte	Korea
UHC	one- and two-byte	Korea
UTF-8	one- through six-byte	*not applicable*

Table 1-19: Wide Character Encodings

Encoding	Encoding Length
UCS-2	16-bit fixed
UCS-4	32-bit fixed
UTF-16	16-bit variable-length
Unicode Version 2.0	*Same as UTF-16*

It is with endianness 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.