

# Foreign Languages and Non-Roman Text

---

The Web is international, yet most of the text you'll find on it is English. XML is starting to change this. XML provides full support for the double-byte Unicode character set, as well as its more compact representations. This is good news for Web authors because Unicode supports almost every character commonly used in every modern script on Earth.

In this chapter, you'll learn how international text is represented in computer applications, how XML understands text, and how you can take advantage of the software you have to read and write in languages other than English.

## Non-Roman Scripts on the Web

Although the Web is international, much of its text is in English. Because of the Web's expansiveness, however, you can still surf through Web pages in French, Spanish, Chinese, Arabic, Hebrew, Russian, Hindi, and other languages. Most of the time, though, these pages come out looking less than ideal. Figure 7-1 shows the October 1998 cover page of one of the United States Information Agency's propaganda journals, *Issues in Democracy* (<http://www.usia.gov/journals/itdhr/1098/ijdr/ijdr1098.htm>), in Russian translation viewed in an English encoding. The red Cyrillic text in the upper left is a bitmapped image file so it's legible (if you speak Russian) and so are a few words in English such as "Adobe Acrobat." However, the rest of the text is mostly a bunch of accented Roman vowels, not the Cyrillic letters they are supposed to be.



### In This Chapter

Understanding the effects of non-Roman scripts on the Web

Using scripts, character sets, fonts, and glyphs

Legacy character sets

Using the Unicode Character Set

Writing XML in Unicode



The quality of Web pages deteriorates even further when complicated, non-Western scripts like Chinese and Japanese are used. Figure 7-2 shows the home page for the Japanese translation of my book *JavaBeans* (IDG Books, 1997, <http://www.ohmsha.co.jp/data/books/contents/4-274-06271-6.htm>) viewed in an English browser. Once again the bitmapped image shows the proper Japanese (and English) text, but the rest of the text on the page looks almost like a random collection of characters except for a few recognizable English words like *JavaBeans*. The Kanji characters you're supposed to see are completely absent.



Figure 7-1: The Russian translation of the October 1998 issue of *Issues of Democracy* viewed in a Roman script

These pages look as they're intended to look if viewed with the right encoding and application software, and if the correct font is installed. Figure 7-3 shows *Issues in Democracy* viewed with the Windows 1251 encoding of Cyrillic. As you can see, the text below the picture is now readable (if you can read Russian).

You can select the encoding for a Web page from the View/Encoding menu in Netscape Navigator or Internet Explorer. In an ideal world, the Web server would tell the Web browser what encoding to use, and the Web browser would listen. It would also be nice if the Web server could send the Web browser the fonts it needed to display the page. In practice, however, you often need to select the encoding manually, even trying several to find the exact right one when more than one encoding is available for a script. For instance, a Cyrillic page might be

encoded in Windows 1251, ISO 8859-5, or KOI6-R. Picking the wrong encoding may make Cyrillic letters appear, but the words will be gibberish.



Figure 7-2: The Japanese translation of JavaBeans viewed in an English browser

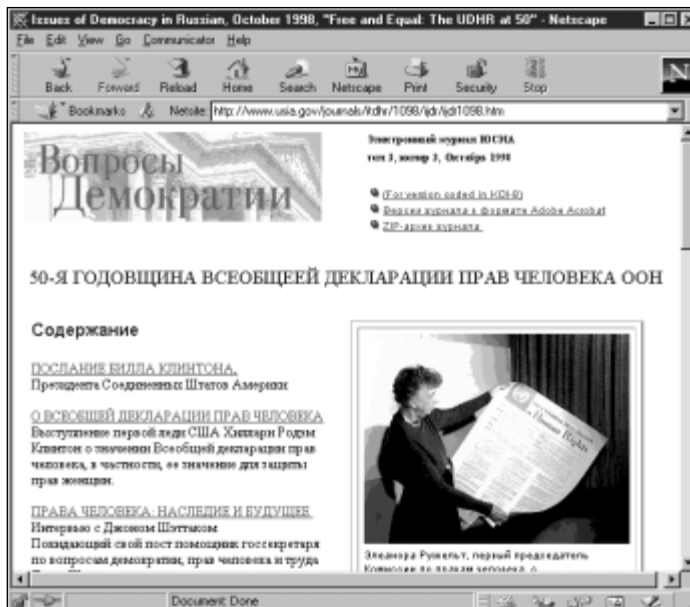


Figure 7-3: *Issues of Democracy* viewed in a Cyrillic script





Figure 7-5: The Japanese translation of JavaBeans in Kanji with the necessary fonts installed

Because each page can only have a single encoding, it is difficult to write a Web page that integrates multiple scripts, such as a French commentary on a Chinese text. For a reasons such as this the Web community needs a single, universal character set to display all characters for all computers and Web browsers. We don't have such a character set yet, but XML and Unicode get as close as is currently possible.

XML files are written in Unicode, a double-byte character set that can represent most characters in most of the world's languages. If a Web page is written in Unicode, as XML pages are, and if the browser understands Unicode, as XML browsers should, then it's not a problem for characters from different languages to be included on the same page.

Furthermore, the browser doesn't need to distinguish between different encodings like Windows 1251, ISO 8859-5, or KOI8-R. It can just assume everything's written in Unicode. As long as the double-byte set has the space to hold all of the different characters, there's no need to use more than one character set. Therefore there's no need for browsers to try to detect which character set is in use.

## Scripts, Character Sets, Fonts, and Glyphs

Most modern human languages have written forms. The set of characters used to write a language is called a *script*. A script may be a phonetic alphabet, but it doesn't have to be. For instance, Chinese, Japanese, and Korean are written with ideographic characters that represent whole words. Different languages often share scripts, sometimes with slight variations. For instance, the modern Turkish alphabet is essentially the familiar Roman alphabet with three extra letters — *ı*, *ş*, and *ı*. Chinese, Japanese, and Korean, on the other hand, share essentially the same 80,000 Han ideographs, though many characters have different meanings in the different languages.

Note

The word *script* is also often used to refer to programs written in weakly typed, interpreted languages like JavaScript, Perl, and TCL. In this chapter, the word *script* always refers to the characters used to write a language and not to any sort of program.

Some languages can even be written in different scripts. Serbian and Croatian are virtually identical and are generally referred to as Serbo-Croatian. However, Serbian is written in a modified Cyrillic script, and Croatian is written in a modified Roman script. As long as a computer doesn't attempt to grasp the meaning of the words it processes, working with a script is equivalent to working with any language that can be written in that script.

Unfortunately, XML alone is not enough to read a script. For each script a computer processes, four things are required:

1. A character set for the script
2. A font for the character set
3. An input method for the character set
4. An operating system and application software that understand the character set

If any of these four elements are missing, you won't be able to work easily in the script, though XML does provide a work-around that's adequate for occasional use. If the only thing your application is missing is an input method, you'll be able to read text written in the script. You just won't be able to write in it.

### A Character Set for the Script

Computers only understand numbers. Before they can work with text, that text has to be encoded as numbers in a specified character set. For example, the popular ASCII character set encodes the capital letter 'A' as 65. The capital letter 'B' is encoded as 66. 'C' is 67, and so on.

These are semantic encodings that provide no style or font information. **C**, *C*, or even *C* are all 67. Information about how the character is drawn is stored elsewhere.

## A Font for the Character Set

A font is a collection of glyphs for a character set, generally in a specific size, face, and style. For example, **C**, *C*, and *C* are all the same character, but they are drawn with different glyphs. Nonetheless their essential meaning is the same.

Exactly how the glyphs are stored varies from system to system. They may be bitmaps or vector drawings; they may even consist of hot lead on a printing press. The form they take doesn't concern us here. The key idea is that a font tells the computer how to draw each character in the character set.

## An Input Method for the Character Set

An input method enables you to enter text. English speakers don't think much about the need for an input method for a script. We just type on our keyboards and everything's hunky-dory. The same is true in most of Europe, where all that's needed is a slightly modified keyboard with a few extra umlauts, cedillas, or thorns (depending on the country).

Radically different character sets like Cyrillic, Hebrew, Arabic, and Greek are more difficult to input. There's a finite number of keys on the keyboard, generally not enough for Arabic and Roman letters, or Roman and Greek letters. Assuming both are needed though, a keyboard can have a Greek lock key that shifts the keyboard from Roman to Greek and back. Both Greek and Roman letters can be printed on the keys in different colors. The same scheme works for Hebrew, Arabic, Cyrillic, and other non-Roman alphabetic character sets.

However, this scheme really breaks down when faced with ideographic scripts like Chinese and Japanese. Japanese keyboards can have in the ballpark of 5,000 different keys; and that's still less than 10% of the language! Syllabic, phonetic, and radical representations exist that can reduce the number of keys; but it is questionable whether a keyboard is really an appropriate means of entering text in these languages. Reliable speech and handwriting recognition have even greater potential in Asia than in the West.

Since speech and handwriting recognition still haven't reached the reliability of even a mediocre typist like myself, most input methods today are map multiple sequences of keys on the keyboard to a single character. For example, to type the Chinese character for sheep, you might hold down the Alt key and type a tilde (~), then type *yang*, then hit the enter key. The input method would then present you with a list of words that are pronounced more or less like *yang*. For example:

佯 揚 易 囑 楊 洋 瘍 羊 詳 錫 陽

You would then choose the character you wanted, `_`. The exact details of both the GUI and the transliteration system used to convert typed keys like *yang* to the ideographic characters like `_` vary from program to program, operating system to operating system, and language to language.

## Operating System and Application Software

As of this writing, the major Web browsers (Netscape Navigator and Internet Explorer) do a surprisingly good job of displaying non-Roman scripts. Provided the underlying operating system supports a given script and has the right fonts installed, a Web browser can probably display it.

MacOS 7.1 and later can handle most common scripts in the world today. However, the base operating system only supports Western European languages. Chinese, Japanese, Korean, Arabic, Hebrew, and Cyrillic are available as language kits that cost about \$100 a piece. Each provides fonts and input methods for languages written in those scripts. There's also an Indian language kit, which handles the Devanagari, Gujarati, and Gurmukhu scripts common on the Indian subcontinent. MacOS 8.5 adds optional, limited support for Unicode (which most applications don't yet take advantage of).

Windows NT 4.0 uses Unicode as its native character set. NT 4.0 does a fairly good job with Roman languages, Cyrillic, Greek, Hebrew, and a few others. The Lucida Sans Unicode font covers about 1300 of the most common of Unicode's 40,000 or so characters. Microsoft Office 97 includes Chinese, Japanese, and Korean fonts that you can install to read text in these languages. (Look in the Fareast folder in the Valupack folder on your Office CD-ROM.)

Microsoft claims Windows 2000 (previously known as NT 5.0) will also include fonts covering most of the Chinese-Japanese-Korean ideographs, as well as input methods for these scripts. However they also promised that Windows 95 would include Unicode support, and that got dropped before shipment. Consequently, I'm not holding my breath. Certainly, it would be nice if they do provide full international support in all versions of NT rather than relying on localized systems.

Microsoft's consumer operating systems, Windows 3.1, 95, and 98, do not fully support Unicode. Instead they rely on localized systems that can only handle basic English characters plus the localized script.

The major Unix variants have varying levels of support for Unicode. Solaris 2.6 supports European languages, Greek, and Cyrillic. Chinese, Japanese, and Korean are supported by localized versions using different encodings rather than Unicode. Linux has embryonic support for Unicode, which may grow to something useful in the near future.

## Legacy Character Sets

Different computers in different locales use different default character sets. Most modern computers use a superset of the ASCII character set. ASCII encodes the English alphabet and the most common punctuation and whitespace characters.

In the United States, Macs use the MacRoman character set, Windows PCs use a character set called Windows ANSI, and most Unix workstations use ISO Latin-1. These are all extensions of ASCII that support additional characters like ç and ¿ that are needed for Western European languages like French and Spanish. In other locales like Japan, Greece, and Israel, computers use a still more confusing hodgepodge of character sets that mostly support ASCII plus the local language.

This doesn't work on the Internet. It's unlikely that while you're reading the *San Jose Mercury News* you'll turn the page and be confronted with several columns written in German or Chinese. However, on the Web it's entirely possible a user will follow a link and end up staring at a page of Japanese. Even if the surfer can't read Japanese it would still be nice if they saw a correct version of the language, as seen in Figure 7-5, instead of a random collection of characters like those shown in Figure 7-2.

XML addresses this problem by moving beyond small, local character sets to one large set that's supposed to encompass all scripts used in all living languages (and a few dead ones) on planet Earth. This character set is called Unicode. As previously noted, Unicode is a double-byte character set that provides representations of over 40,000 different characters in dozens of scripts and hundreds of languages. All XML processors are required to understand Unicode, even if they can't fully display it.

As you learned in Chapter 6, an XML document is divided into text and binary entities. Each text entity has an encoding. If the encoding is not explicitly specified in the entity's definition, then the default is UTF-8 — a compressed form of Unicode which leaves pure ASCII text unchanged. Thus XML files that contain nothing but the common ASCII characters may be edited with tools that are unaware of the complications of dealing with multi-byte character sets like Unicode.

## The ASCII Character Set

ASCII, the American Standard Code for Information Interchange, is one of the original character sets, and is by far the most common. It forms a sort of lowest common denominator for what a character set must support. It defines all the characters needed to write U.S. English, and essentially nothing else. The characters are encoded as the numbers 0-127. Table 7-1 presents the ASCII character set.

**Table 7-1**  
**The ASCII Character Set**

<i>Code</i>	<i>Character</i>	<i>Code</i>	<i>Char-acter</i>	<i>Code</i>	<i>Char-acter</i>	<i>Code</i>	<i>Char-acter</i>
0	null(Control-@)	32	Space	64	@	96	`
1	start of heading (Control-A)	33	!	65	A	97	a
2	start of text (Control-B)	34	"	66	B	98	b
3	end of text (Control-C)	35	#	67	C	99	c
4	end of transmis- sion (Control-D)	36	\$	68	D	100	d
5	enquiry (Control-E)	37	%	69	E	101	e
6	acknowledge (Control-F)	38	&	70	F	102	f
7	bell (Control-G)	39	'	71	G	103	g
8	backspace (Control-H)	40	(	72	H	104	h
9	tab(Control-I)	41	)	73	I	105	i
10	linefeed (Control-J)	42	*	74	J	106	j
11	vertical tab) (Control-K)	43	+	75	K	107	k
12	formfeed (Control-L)	44	,	76	L	108	l
13	carriage return (Control-M)	45	-	77	M	109	m
14	shift out (Control-N)	46	.	78	N	110	n
15	shift in (Control-O)	47	/	79	O	111	o
16	data link escape (Control-P)	48	0	80	P	112	p
17	device control 1 (Control-Q)	49	1	81	Q	113	q

<i>Code</i>	<i>Character</i>	<i>Code</i>	<i>Character</i>	<i>Code</i>	<i>Character</i>	<i>Code</i>	<i>Character</i>
18	device control 2 (Control-R)	50	2	82	R	114	r
19	device control 3 (Control-S)	51	3	83	S	115	s
20	device control 4 (Control-T)	52	4	84	T	116	t
21	negative acknowl- edge (Control-U)	53	5	85	U	117	u
22	synchronous idle (Control-V)	54	6	86	V	118	v
23	end of transmission block (Control-W)	55	7	87	W	119	w
24	cancel (Control-X)	56	8	88	X	120	x
25	end of medium (Control-Y)	57	9	89	Y	121	y
26	substitute (Control-Z)	58	:	90	Z	122	z
27	escape (Control-])	59	;	91	[	123	{
28	file separator (Control-\)	60	<	92	\	124	
29	group separator (Control-])	61	=	93	]	125	}
30	record separator (Control-^)	62	>	94	^	126	~
31	unit separator (Control-_)	63	?	95	_	127	delete

Characters 0 through 31 are non-printing control characters. They include the carriage return, the linefeed, the tab, the bell, and similar characters. Many of these are leftovers from the days of paper-based teletype terminals. For instance, carriage return used to literally mean move the carriage back to the left margin, as you'd do on a typewriter. Linefeed moved the platen up one line. Aside from the few control characters mentioned, these aren't used much anymore.

Most other character sets you're likely to encounter are supersets of ASCII. In other words, they define 0 through 127 exactly the same as ASCII, but add additional characters from 128 on up.

## The ISO Character Sets

The A in ASCII stands for American, so it shouldn't surprise you that ASCII is only adequate for writing English, and strictly American English at that. ASCII contains no *£*, *ü*, *¿*, or many other characters you might want for writing in other languages or locales.

ASCII can be extended by assigning additional characters to numbers above 128. The International Standards Organization (ISO) has defined a number of different character sets based on ASCII that add additional characters needed for other languages and locales. The most prominent such character set is ISO 8859-1, commonly called Latin-1. Latin-1 includes enough additional characters to write essentially all Western European languages. Characters 0 through 127 are the same as they are in ASCII. Characters 128 through 255 are given in Table 7-2. Again, the first 32 characters are mostly unused, non-printing control characters.

Table 7-2  
The ISO 8859-1 Latin-1 Character Set

<i>Code</i>	<i>Character</i>	<i>Code</i>	<i>Character</i>	<i>Code</i>	<i>Character</i>	<i>Code</i>	<i>Character</i>
128	Undefined	160	non-break- ing space	192	À	224	À
129	Undefined	161	ı	193	Á	225	Á
130	Bph	162	¢	194	Â	226	Â
131	Nbh	163	£	195	Ã	227	Ã
132	Undefined	164		196	Ä	228	Ä
133	Nel	165	¥	197	Å	229	Å
134	Ssa	166	ı	198	Æ	230	Æ
135	Esa	167	§	199	Ç	231	Ç
136	Hts	168	¨	200	È	232	È
137	Htj	169	©	201	É	233	É
138	Vts	170	ª	202	Ê	234	Ê
139	Pld	171	«	203	Ë	235	Ë
140	Plu	172	¬	204	Ì	236	Ì
141	Ri	173	Discretionary hyphen	205	Í	237	Í
142	ss2	174	®	206	Î	238	Î

<i>Code</i>	<i>Character</i>	<i>Code</i>	<i>Character</i>	<i>Code</i>	<i>Character</i>	<i>Code</i>	<i>Character</i>
143	ss3	175	˘	207	İ	239	ı
144	Dcs	176	°	208	Đ	240	đ
145	pu1	177	±	209	Ñ	241	ñ
146	pu2	178	²	210	Ò	242	ò
147	Sts	179	³	211	Ó	243	ó
148	Cch	180	´	212	Ô	244	ô
149	Mw	181	μ	213	Õ	245	õ
150	Spa	182	¶	214	Ö	246	ö
151	Epa	183	·	215	×	247	÷
152	Sos	184	,	216	Ø	248	ø
153	Undefined	185	¹	217	Ù	249	ù
154	Sci	186	º	218	Ú	250	ú
155	Csi	187	»	219	Û	251	ü
156	St	188	¼	220	Ü	252	ü
157	Osc	189	½	221	Ý	253	ý
158	Pm	190	¾	222	þ	254	þ
159	Apc	191	¿	223	ß	255	ÿ

Latin-1 still lacks many useful characters including those needed for Greek, Cyrillic, Chinese, and many other scripts and languages. You might think these could just be moved into the numbers from 256 up. However there's a catch. A single byte can only hold values from 0 to 255. To go beyond that, you need to use a multi-byte character set. For historical reasons most programs are written under the assumption that characters and bytes are identical, and they tend to break when faced with multi-byte character sets. Therefore, most current operating systems (Windows NT being the notable exception) use different, single-byte character sets rather than one large multi-byte set. Latin-1 is the most common such set, but other sets are needed to handle additional languages.

ISO 8859 defines ten other character sets (8859-2 through 8859-10 and 8859-15) suitable for different scripts, with four more (8859-11 through 8859-14) in active development. Table 7-3 lists the ISO character sets and the languages and scripts they can be used for. All share the same ASCII characters from 0 to 127, and then each includes additional characters from 128 to 255.

**Table 7-3**  
**The ISO Character Sets**

<i>Character Set</i>	<i>Also Known As</i>	<i>Languages</i>
ISO 8859-1	Latin-1	ASCII plus the characters required for most Western European languages including Albanian, Afrikaans, Basque, Catalan, Danish, Dutch, English, Faroese, Finnish, Flemish, Galician, German, Icelandic, Irish, Italian, Norwegian, Portuguese, Scottish, Spanish, and Swedish. However it omits the ligatures ij (Dutch), Œ (French), and German quotation marks.
ISO 8859-2	Latin-2	ASCII plus the characters required for most Central European languages including Czech, English, German, Hungarian, Polish, Romanian, Croatian, Slovak, Slovene, and Sorbian.
ISO 8859-3	Latin-3	ASCII plus the characters required for English, Esperanto, German, Maltese, and Galician.
ISO 8859-4	Latin-4	ASCII plus the characters required for the Baltic languages Latvian, Lithuanian, German, Greenlandic, and Lappish; superseded by ISO 8859-10, Latin-6
ISO 8859-5		ASCII plus Cyrillic characters required for Byelorussian, Bulgarian, Macedonian, Russian, Serbian, and Ukrainian.
ISO 8859-6		ASCII plus Arabic.
ISO 8859-7		ASCII plus Greek.
ISO 8859-8		ASCII plus Hebrew.
ISO 8859-9	Latin-5	Latin-1 except that the Turkish letters İ, İ̇, Ş, ş, Ç, ç, and Ğ take the place of the less commonly used Icelandic letters Ý, ý, Þ, þ, Ð, and ð.
ISO 8859-10	Latin-6	ASCII plus characters for the Nordic languages Lithuanian, Inuit (Greenlandic Eskimo), non-Skolt Sami (Lappish), and Icelandic.
ISO 8859-11		ASCII plus Thai.
ISO 8859-12		This may eventually be used for ASCII plus Devanagari (Hindi, Sanskrit, etc.) but no proposal is yet available.
ISO 8859-13	Latin-7	ASCII plus the Baltic Rim, particularly Latvian.
ISO 8859-14	Latin-8	ASCII plus Gaelic and Welsh.
ISO 8859-15	Latin-9, Latin-0	Essentially the same as Latin-1 but with a Euro sign instead of the international currency sign €. Furthermore, the Finnish characters Š, š, Z, z replace the uncommon symbols I, i, . And the French Œ, œ, and Ÿ characters replace the fractions 1/4, 1/2, 3/4.

These sets often overlap. Several languages, most notably English and German, can be written in more than one of the character sets. To some extent the different sets are designed to allow different combinations of languages. For instance Latin-1 can combine most Western languages and Icelandic whereas Latin-5 combines most Western languages with Turkish instead of Icelandic. Thus if you needed a document in English, French, and Icelandic, you'd use Latin-1. However a document containing English, French, and Turkish would use Latin-5. However, a document that required English, Hebrew, and Turkish, would have to use Unicode since no single-byte character set handles all three languages and scripts.

A single-byte set is insufficient for Chinese, Japanese, and Korean. These languages have more than 256 characters apiece, so they must use multi-byte character sets.

## The MacRoman Character Set

The MacOS predates Latin-1 by several years. (The ISO 8859-1 standard was first adopted in 1987. The first Mac was released in 1984.) Unfortunately this means that Apple had to define its own extended character set called MacRoman. MacRoman has most of the same extended characters as Latin-1 (except for the Icelandic letters þ, þ, and ð) but the characters are assigned to different numbers. MacRoman is the same as ASCII and Latin-1 in the codes though the first 127 characters. This is one reason text files that use extended characters often look funny when moved from a PC to a Mac or vice versa. Table 7-4 lists the upper half of the MacRoman character set.

Table 7-4  
The MacRoman Character Set

<i>Code</i>	<i>Character</i>	<i>Code</i>	<i>Character</i>	<i>Code</i>	<i>Character</i>	<i>Code</i>	<i>Character</i>
128	Â	160	†	192	ı	224	‡
129	Å	161	°	193	ı	225	·
130	Ç	162	¢	194	¬	226	,
131	É	163	£	195	√	227	„
132	Ñ	164	§	196	f	228	‰
133	Ö	165	·	197	~	229	Â
134	Û	166	¶	198	Δ	230	Ê
135	Á	167	ß	199	«	231	Á
136	À	168	®	200	»	232	È

*Continued*

Table 7-4 (continued)

<i>Code</i>	<i>Character</i>	<i>Code</i>	<i>Character</i>	<i>Code</i>	<i>Character</i>	<i>Code</i>	<i>Character</i>
137	Â	169	©	201	...	233	È
138	Ä	170	™	202	non-break- ing space	234	Í
139	Å	171	´	203	À	235	Î
140	Å	172	¨	204	Ã	236	Ï
141	Ç	173	≠	205	Ö	237	Ì
142	É	174	Æ	206	Œ	238	Ï
143	È	175	Ø	207	Œ	239	Ó
144	Ê	176	∞	208	-	240	Ô
145	Ë	177	±	209	_	241	Apple
146	Í	178	≤	210	"	242	Ò
147	Ì	179	≥	211	"	243	Ú
148	ì	180	¥	212	´	244	Û
149	ï	181	μ	213	´	245	´
150	ñ	182	ð	214	÷	246	^
151	ó	183	∑	215	◇	247	~
152	ò	184	∏	216	ÿ	248	-
153	ô	185	∏	217	ÿ	249	˘
154	ö	186	ƒ	218	/	250	·
155	õ	187	ª	219		251	°
156	ú	188	°	220	‹	252	,
157	Û	189	Ω	221	›	253	"
158	Û	190	Æ	222	fi	254	˙
159	Ü	191	Ø	223	fl	255	˘

## The Windows ANSI Character Set

The first version of Windows to achieve widespread adoption followed the Mac by a few years, so it was able to adopt the Latin-1 character set. However, it replaced the non-printing control characters between 130 and 159 with more printing characters to stretch the available range a little further. This modified version of Latin-1 is generally called "Windows ANSI." Table 7-5 lists the Windows ANSI characters.

**Table 7-5**  
**The Windows ANSI Character Set**

<i>Code</i>	<i>Character</i>	<i>Code</i>	<i>Character</i>	<i>Code</i>	<i>Character</i>	<i>Code</i>	<i>Character</i>
128	Undefined	136	^	144	Undefined	152	~
129	Undefined	137	‰	145	'	153	™
130	,	138	§	146	'	154	§
131	□	139	‹	147	“	155	›
132	”	140	Œ	148	”	156	Œ
133	...	141	Undefined	149	•	157	Undefined
134	†	142	Undefined	150	–	158	Undefined
135	‡	143	Undefined	151	–	159	ÿ

## The Unicode Character Set

Using different character sets for different scripts and languages works well enough as long as:

1. You don't need to work in more than one script at once.
2. You never trade files with anyone using a different character set.

Since Macs and PCs use different character sets, more people fail these criteria than not. Obviously what is needed is a single character set that everyone agrees on and that encodes all characters in all the world's scripts. Creating such a set is difficult. It requires a detailed understanding of hundreds of languages and their scripts. Getting software developers to agree to use that set once it's been created is even harder. Nonetheless work is ongoing to create exactly such a set called Unicode, and the major vendors (Microsoft, Apple, IBM, Sun, Be, and many others) are slowly moving toward complying with it. XML specifies Unicode as its default character set.

Unicode encodes each character as a two-byte unsigned number with a value between 0 and 65,535. Currently a few more than 40,000 different Unicode characters are defined. The remaining 25,000 spaces are reserved for future extensions. About 20,000 of the characters are used for the Han ideographs and another 11,000 or so are used for the Korean Hangul syllables. The remainder of the characters encodes most of the rest of the world's languages. Unicode characters 0 through 255 are identical to Latin-1 characters 0 through 255.

I'd love to show you a table of all the characters in Unicode, but if I did this book would consist entirely of this table and not much else. If you need to know more about the specific encodings of the different characters in Unicode, get a copy of

*The Unicode Standard* (second edition, ISBN 0-201-48346-9, from Addison-Wesley). This 950-page book includes the complete Unicode 2.0 specification, including character charts for all the different characters defined in Unicode 2.0. You can also find information online at the Unicode Consortium Web site at <http://www.unicode.org/> and <http://charts.unicode.org/>. Table 7-6 lists the different scripts encoded by Unicode which should give you some idea of Unicode's versatility. The characters of each script are generally encoded in a consecutive sub-range (block) of the 65,536 code points in Unicode. Most languages can be written with the characters in one of these blocks (for example, Russian can be written with the Cyrillic block) though some languages like Croatian or Turkish may need to mix and match characters from the first four Latin blocks.

**Table 7-6**  
**Unicode Script Blocks**

<i>Script</i>	<i>Range</i>	<i>Purpose</i>
Basic Latin	0-127	ASCII, American English.
Latin-1 Supplement	126-255	Upper half of ISO Latin-1, in conjunction with the Basic Latin block can handle Danish, Dutch, English, Faroese, Flemish, German, Hawaiian, Icelandic, Indonesian, Irish, Italian, Norwegian, Portuguese, Spanish, Swahili, and Swedish.
Latin Extended-A	256-383	This block adds the characters from the ISO 8859 sets Latin-2, Latin-3, Latin-4, and Latin-5 not already found in the Basic Latin and Latin-1 blocks. In conjunction with those blocks, this block can encode Afrikaans, Breton, Basque, Catalan, Czech, Esperanto, Estonian, French, Frisian, Greenlandic, Hungarian, Latvian, Lithuanian, Maltese, Polish, Provençal, Rhaeto-Romanic, Romanian, Romany, Slovak, Slovenian, Sorbian, Turkish, and Welsh.
Latin Extended-B	383-591	Mostly characters needed to extend the Latin script to handle languages not traditionally written in this script; includes many African languages, Croatian digraphs to match Serbian Cyrillic letters, the Pinyin transcription of Chinese, and the Sami characters from Latin-10.
IPA Extensions	592-687	The International Phonetic Alphabet.
Spacing Modifier Letters	686-767	Small symbols that somehow change (generally phonetically) the previous letter.
Combining Diacritical Marks	766-879	Diacritical marks like ~, ' , and _ that will somehow be combined with the previous character (most commonly, be placed on top of) rather than drawn as a separate character.

<i>Script</i>	<i>Range</i>	<i>Purpose</i>
Greek	880-1023	Modern Greek, based on ISO 8859-7; also provides characters for Coptic.
Cyrillic	1024-1279	Russian and most other Slavic languages (Ukrainian, Byelorussian, and so forth), and many non-Slavic languages of the former Soviet Union (Azerbaijani, Ossetian, Kabardian, Chechen, Tajik, and so forth); based on ISO 8859-5. A few languages (Kurdish, Abkhazian) require both Latin and Cyrillic characters
Armenian	1326-1423	Armenian
Hebrew	1424-1535	Hebrew (classical and modern), Yiddish, Judezmo, early Aramaic.
Arabic	1536-1791	Arabic, Persian, Pashto, Sindhi, Kurdish, and classical Turkish.
Devanagari	2304-2431	Sanskrit, Hindi, Nepali, and other languages of the Indian subcontinent including Awadhi, Bagheli, Bhatneri, Bhili, Bihari, Braj Bhasha, Chhattisgarhi, Garhwali, Gondi, Harauti, Ho, Jaipuri, Kachchhi, Kanauji, Konkani, Kului, Kumaoni, Kurku, Kurukh, Marwari, Mundari, Newari, Palpa, and Santali.
Bengali	2432-2559	A North Indian script used in India's West Bengal state and Bangladesh; used for Bengali, Assamese, Daphla, Garo, Hallam, Khasi, Manipuri, Mizo, Naga, Munda, Rian, Santali.
Gurmukhi	2560-2687	Punjabi
Gujarati	2686-2815	Gujarati
Oriya	2816-2943	Oriya, Khondi, Santali.
Tamil	2944-3071	Tamil and Badaga, used in south India, Sri Lanka, Singapore, and parts of Malaysia.
Telugu	3072-3199	Telugu, Gondi, Lambadi.
Kannada	3200-3327	Kannada, Tulu.
Malalayam	3326-3455	Malalayam
Thai	3584-3711	Thai, Kuy, Lavna, Pali.
Lao	3712-3839	Lao
Tibetan	3840-4031	Himalayan languages including Tibetan, Ladakhi, and Lahuli.

*Continued*

Table 7-6 (continued)

<i>Script</i>	<i>Range</i>	<i>Purpose</i>
Georgian	4256-4351	Georgian, the language of the former Soviet Republic of Georgian on the Black Sea.
Hangul Jamo	4352-4607	The alphabetic components of the Korean Hangul syllabary.
Latin Extended Additional	7680-7935	Normal Latin letters like E and Y combined with diacritical marks, rarely used except for Vietnamese vowels
Greek Extended	7936-8191	Greek letters combined with diacritical marks; used in Polytonic and classical Greek.
General Punctuation	8192-8303	Assorted punctuation marks.
Superscripts and Subscripts	8304-8351	Common subscripts and superscripts.
Currency Symbols	8352-8399	Currency symbols not already present in other blocks.
Combining Marks for Symbols	8400-8447	Used to make a diacritical mark span two or more characters.
Letter like Symbols	8446-8527	Symbols that look like letters such as ™ and _.
Number Forms	8526-8591	Fractions and Roman numerals.
Arrows	8592-8703	Arrows
Mathematical Operators	8704-8959	Mathematical operators that don't already appear in other blocks.
Miscellaneous Technical	8960-9039	Cropping marks, bracket notation from quantum mechanics, symbols needed for the APL programming language, and assorted other technical symbols.
Control Pictures	9216-9279	Pictures of the ASCII control characters; generally used in debugging and network-packet sniffing.
Optical Character Recognition	9280-9311	OCR-A and the MICR (magnetic ink character recognition) symbols on printed checks.
Enclosed alphanumerics	9312-9471	Letters and numbers in circles and parentheses.
Box Drawing	9472-9599	Characters for drawing boxes on monospaced terminals.
Block Elements	9600-9631	Monospaced terminal graphics as used in DOS and elsewhere.
Geometric Shapes	9632-9727	Squares, diamonds, triangles, and the like.

<i>Script</i>	<i>Range</i>	<i>Purpose</i>
Miscellaneous Symbols	9726-9983	Cards, chess, astrology, and more.
Dingbats	9984-10175	The Zapf Dingbat characters.
CJK Symbols and Punctuation	12286-12351	Symbols and punctuation used in Chinese, Japanese, and Korean.
Hiragana	12352-12447	A cursive syllabary for Japanese
Katakana	12446-12543	A non-cursive syllabary used to write words imported from the West in Japanese, especially modern words like “keyboard”.
Bopomofo	12544-12591	A phonetic alphabet for Chinese used primarily for teaching.
Hangul Compatibility Jamo	12592-12687	Korean characters needed for compatibility with the KSC 5601 encoding.
Kanbun	12686-12703	Marks used in Japanese to indicate the reading order of classical Chinese.
Enclosed CJK Letters and Months	12800-13055	Hangul and Katakana characters enclosed in circles and parentheses.
CJK Compatibility	13056-13311	Characters needed only to encode KSC 5601 and CNS 11643.
CJK Unified Ideographs	19966-40959	The Han ideographs used for Chinese, Japanese, and Korean.
Hangul Syllables	44032-55203	A Korean syllabary.
Surrogates	55296-57343	Currently unused, but will eventually allow the extension of Unicode to over one million different characters.
Private Use	57344-63743	Software developers can include their custom characters here; not compatible across implementations.
CJK Compatibility Ideographs	63744-64255	A few extra Han ideographs needed only to maintain compatibility with existing standards like KSC 5601.
Alphabetic Presentation Forms	64256-64335	Ligatures and variants sometimes used in Latin, Armenian, and Hebrew.
Arabic Presentation Forms	64336-65023	Variants of assorted Arabic characters.

*Continued*

Table 7-6 (continued)

<i>Script</i>	<i>Range</i>	<i>Purpose</i>
Combining Half Marks	65056-65071	Combining multiple diacritical marks into a single diacritical mark that spans multiple characters.
CJK Compatibility Forms	65072-65103	Mostly vertical variants of Han ideographs used in Taiwan.
Small Form Variants	65104-65135	Smaller version of ASCII punctuation mostly used in Taiwan.
Additional Arabic Presentation Forms	65136-65279	More variants of assorted Arabic characters.
Half-width and Full-width Forms	65280-65519	Characters that allow conversion between different Chinese and Japanese encodings of the same characters.
Specials	65520-65535	The byte order mark and the zero-width, no breaking space often used to start Unicode files.

## UTF 8

Since Unicode uses two bytes for each character, files of English text are about twice as large in Unicode as they would be in ASCII or Latin-1. UTF-8 is a compressed version of Unicode that uses only a single byte for the most common characters, that is the ASCII characters 0-127, at the expense of having to use three bytes for the less common characters, particularly the Hangul syllables and Han ideographs. If you're writing mostly in English, UTF-8 can reduce your file sizes by as much as 50 percent. On the other hand if you're writing mostly in Chinese, Korean, or Japanese, UTF-8 can *increase* your file size by as much as 50 percent — so it should be used with caution. UTF-8 has mostly no effect on non-Roman, non-CJK scripts like Greek, Arabic, Cyrillic, and Hebrew.

XML processors assume text data is in the UTF-8 format unless told otherwise. This means they can read ASCII files, but other formats like MacRoman or Latin-1 cause them trouble. You'll learn how to fix this problem shortly.

## The Universal Character System

Unicode has been criticized for not encompassing enough, especially in regard to East Asian languages. It only defines about 20,000 of the 80,000 Han ideographs used amongst Chinese, Japanese, Korean, and historical Vietnamese. (Modern Vietnamese uses a Roman alphabet.)

UCS (Universal Character System), also known as ISO 10646, uses four bytes per character (more precisely, 31 bits) to provide space for over two billion different

characters. This easily covers every character ever used in any language in any script on the planet Earth. Among other things this enables a full set of characters to be assigned to each language so that the French “e” is not the same as the English “e” is not the same as the German “e,” and so on.

Like Unicode, UCS defines a number of different variants and compressed forms. Pure Unicode is sometimes referred to as UCS-2, which is two-byte UCS. UTF-16 is a special encoding that maps some of the UCS characters into byte strings of varying length in such a fashion that Unicode (UCS-2) data is unchanged.

At this point, the advantage of UCS over Unicode is mostly theoretical. The only characters that have actually been defined in UCS are precisely those already in Unicode. However, it does provide more room for future expansion.

## How to Write XML in Unicode

Unicode is the native character set of XML, and XML browsers will probably do a pretty good job of displaying it, at least to the limits of the available fonts. Nonetheless, there simply aren't many if any text editors that support the full range of Unicode. Consequently, you'll probably have to tackle this problem in one of a couple of ways:

1. Write in a localized character set like Latin-3; then convert your file to Unicode.
2. Include Unicode character references in the text that numerically identify particular characters.

The first option is preferable when you've got a large amount of text to enter in essentially one script, or one script plus ASCII. The second works best when you need to mix small portions of multiple scripts into your document.

## Inserting Characters in XML Files with Character References

Every Unicode character is a number between 0 and 65,535. If you do not have a text editor that can write in Unicode, you can always use a character reference to insert the character in your XML file instead.

A Unicode character reference consists of the two characters `&#` followed by the character code, followed by a semicolon. For instance, the Greek letter  $\pi$  has Unicode value 960 so it may be inserted in an XML file as `&#960;`. The Cyrillic character  $\Uparrow$  has Unicode value 1206 so it can be included in an XML file with the character reference `&#1206;`

Unicode character references may also be specified in hexadecimal (base 16). Although most people are more comfortable with decimal numbers, the Unicode

Specification gives character values as two-byte hexadecimal numbers. It's often easier to use hex values directly rather than converting them to decimal.

All you need to do is include an `x` after the `&#` to signify that you're using a hexadecimal value. For example,  $\pi$  has hexadecimal value `3C0` so it may be inserted in an XML file as `&#x03C0;`. The Cyrillic character `У` has hexadecimal value `4B6` so it can be included in an XML file with the escape sequence `&#x04B6;`. Because two bytes always produce exactly four hexadecimal digits, it's customary (though not required) to include leading zeros in hexadecimal character references so they are rounded out to four digits.

Unicode character references, both hexadecimal and decimal, may be used to embed characters that would otherwise be interpreted as markup. For instance, the ampersand (`&`) is encoded as `&#38;` or `&#x0026;`. The less-than sign (`<`) is encoded as `&#60;` or `&#x003C;`.

## Converting to and from Unicode

Application software that exports XML files, such as Adobe Framemaker, handles the conversion to Unicode or UTF-8 automatically. Otherwise you'll need to use a conversion tool. Sun's freely available Java Development Kit (JDK) includes a simple command-line utility called `native2ascii` that converts between many common and uncommon localized character sets and Unicode.

For example, the following command converts a text file named `myfile.txt` from the platform's default encoding to Unicode

```
C:\> native2ascii myfile.txt myfile.uni
```

You can specify other encodings with the `-encoding` option:

```
C:> native2ascii -encoding Big5 chinese.txt chinese.uni
```

You can also reverse the process to go from Unicode to a local encoding with the `-reverse` option:

```
C:> native2ascii -encoding Big5 -reverse chinese.uni chinese.txt
```

If the output file name is left off, the converted file is printed out.

The `native2ascii` program also processes Java-style Unicode escapes, which are characters embedded as `\u09E3`. These are not in the same format as XML numeric character references, though they're similar. If you convert to Unicode using `native2ascii`, you can still use XML character references — the viewer will still recognize them.

## How to Write XML in Other Character Sets

Unless told otherwise, an XML processor assumes that text entity characters are encoded in UTF-8. Since UTF-8 includes ASCII as a subset, ASCII text is easily parsed by XML processors as well.

The only character set other than UTF-8 that an XML processor is required to understand is raw Unicode. If you cannot convert your text into either UTF-8 or raw Unicode, you can leave the text in its native character set and tell the XML processor which set that is. This should be a last resort, though, because there's no guarantee an arbitrary XML processor can process other encodings. Nonetheless Netscape Navigator and Internet Explorer both do a pretty good job of interpreting the common character sets.

To warn the XML processor that you're using a non-Unicode encoding, you include an `encoding` attribute in the XML declaration at the start of the file. For example, to specify that the entire document uses Latin-1 by default (unless overridden by another processing instruction in a nested entity) you would use this XML declaration:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
```

You can also include the encoding declaration as part of a separate processing instruction after the XML declaration but before any character data appears.

```
<?xml encoding="ISO-8859-1"?>
```

Table 7-7 lists the official names of the most common character sets used today, as they would be given in XML encoding attributes. For encodings not found in this list, consult the official list maintained by the Internet Assigned Numbers Authority (IANA) at <http://www.isi.edu/in-notes/iana/assignments/character-sets>.

**Table 7-7**  
**Names of Common Character Sets**

<i>Character Set Name</i>	<i>Languages/Countries</i>
US-ASCII	English
UTF-8	Compressed Unicode
UTF-16	Compressed UCS
ISO-10646-UCS-2	Raw Unicode
ISO-10646-UCS-4	Raw UCS

*Continued*

Table 7-7 (continued)

<i>Character Set Name</i>	<i>Languages/Countries</i>
ISO-8859-1	Latin-1, Western Europe
ISO-8859-2	Latin-2, Eastern Europe
ISO-8859-3	Latin-3, Southern Europe
ISO-8859-4	Latin-4, Northern Europe
ISO-8859-5	ASCII plus Cyrillic
ISO-8859-6	ASCII plus Arabic
ISO-8859-7	ASCII plus Greek
ISO-8859-8	ASCII plus Hebrew
ISO-8859-9	Latin-5, Turkish
ISO-8859-10	Latin-6, ASCII plus the Nordic languages
ISO-8859-11	ASCII plus Thai
ISO-8859-13	Latin-7, ASCII plus the Baltic Rim languages, particularly Latvian
ISO-8859-14	Latin-8, ASCII plus Gaelic and Welsh
ISO-8859-15	Latin-9, Latin-0; Western Europe
ISO-2022-JP	Japanese
Shift_JIS	Japanese, Windows
EUC-JP	Japanese, Unix
Big5	Chinese, Taiwan
GB2312	Chinese, mainland China
KOI6-R	Russian
ISO-2022-KR	Korean
EUC-KR	Korean, Unix
ISO-2022-CN	Chinese

## Summary

In this chapter you learned:

- ♦ Web pages should identify the encoding they use.
- ♦ What a script is, how it relates to languages, and the four things a script requires.
- ♦ How scripts are used in computers with character sets, fonts, glyphs, and input methods.
- ♦ What character sets are commonly used on different platforms and that most are based on ASCII.
- ♦ How to write XML in Unicode without a Unicode editor (write the document in ASCII and include Unicode character references).
- ♦ When writing XML in other encodings, include an `encoding` attribute in the XML declaration.

In the next chapter, you'll begin exploring DTDs and how they enable you to define and enforce a vocabulary, syntax, and grammar for your documents.



