

Unicode and Character Encodings

by Arman Gungor

Introduction

One area of litigation support that continues to grow rapidly is electronic discovery. Identifying, collecting, processing and searching in foreign languages have been major shortcomings in most commercial off-the-shelf e-discovery platforms and review tools until very recently. While several platforms have finally made the switch to support multiple languages, a number of tools in the e-discovery technician’s arsenal are still either completely ignorant of character encoding issues or provide only limited support.

Having such weak links in the e-discovery workflow, it is not uncommon to run into documents in an electronic production filled with dreaded question marks or boxes instead of foreign language symbols. Consequently, having a deep understanding of Unicode and character encodings remains to be critical in order to handle electronic documents accurately.

ASCII

Back in the day when computers were first invented, the only characters used were the unaccented English letters. They were coded using the American Standard Code for Information Interchange (ASCII), which was able to represent each printable character using a code between 32 and 127.

ASCII is a character encoding scheme based on the ordering of the English alphabet. It includes definitions for 33 non-printing control characters that are used to control devices such as printers, as well as 94 printable characters and the space character. For example, “07” makes your computer beep and “10” represents the line feed function. ASCII was the most commonly used character encoding on the Internet until 2008. It has since been surpassed by UTF-8.

A simple way of reproducing a character from its ASCII code is to open a text editor and type the ASCII code using the numeric keypad while holding down the ALT key and then releasing it. Certain control characters such as a line feed (ALT+010) or backspace (ALT+008) can also be produced using this method.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SP	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

Figure 1 Hexadecimal ASCII Table

Extensions to ASCII, ISO-8859-1 and Windows-1252

Since the number of symbols used in common natural languages exceeded the limited range of ASCII code, many extensions were proposed. As ASCII was a seven-bit code and most computers manipulated data in eight-bit bytes, many extensions used the additional 128 codes available by using all eight bits of each byte. While this helped include many languages otherwise not easily presentable in ASCII, it still was not sufficient to cover all spoken languages. Consequently, even these eight-bit extensions had to have local variants.

ISO/IEC 8859 was one of these extensions. The ISO-8859 standard consists of 15 numbered parts, such as ISO/IEC 8859-1, ISO/IEC 8859-2 etc, each of which contains characters for a different Latin alphabet. For example, Part 1 is Latin-1 Western European and Part 9 is Latin-5 Turkish.

Windows-1252, also known as "ANSI Character Encoding" was a similar 8-bit character encoding of the Latin alphabet used by default in Microsoft Windows. It is a superset of ISO 8859-1, but differs from it by using displayable characters rather than control characters in the 0x80 to 0x9F range.

One of the limitations of ANSI was the lack of ability to represent multiple languages on the same computer. For example, displaying Hebrew and Turkish on the same computer at the same time was not possible since different code pages with different interpretations of high numbers were required.

Another critical limitation was the number of characters that could be supported. While ANSI was able to encode up to 256 characters, some Asian languages contained thousands of symbols.

Unicode

Unicode is a computing standard that incorporates all reasonable writing systems in the world into a single character set. Unicode's development is coordinated by a non-profit organization called the Unicode Consortium. Unicode is a standard; it is not a character encoding. It can be implemented by different character encodings such as UTF-8, UTF-16 or UTF-32. It is widely assumed that Unicode is simply a 16-bit (double byte) code with a maximum capacity of 65,536 characters. This is simply incorrect. The latest version of Unicode, as of this writing, consists of more than 107,000 characters covering 90 scripts.

Unicode represents each letter with a code point. Code points are usually written in the form of U+262E where the numeric part is hexadecimal. For each character, a fairly typical sample rendition as well as information on how to display it (such as line breaking, hyphenation and sorting) are included.

A simple way of typing Unicode characters on your computer is to use Microsoft Global Input Method Editor (IME). For example, to type せんせい in Japanese using English letters, you can activate IME by pressing <Left Alt>+ <Shift>, switch the mode to Japanese and type "sennsei". Another simple method for converting Unicode code points to characters in Ms Word is to write the code point and to press ALT+X when the cursor is at the end of the code point. For example, U+262E that was mentioned above is translated to ☺ by Ms Word.

Unicode code points for certain characters can be found using the Character Map utility in Ms Windows (charmap.exe). Selecting a Unicode font such as "Arial Unicode MS", choosing "Unicode" as the character set and using the "Group by" drop down menu allows users to locate groups of symbols conveniently.

For example, the Pilcrow sign is listed under "Unicode Subrange\General Punctuation" as U+00B6. A comprehensive list can be obtained from the Unicode Consortium website at www.unicode.org.

The full Unicode code space supports over a million code points. The majority of characters used in today's modern languages are allocated within the first 65,536 code points, which is called the Basic Multilingual Plane (BMP). The first 128 code points within the BMP correspond to the Basic Latin ASCII characters.

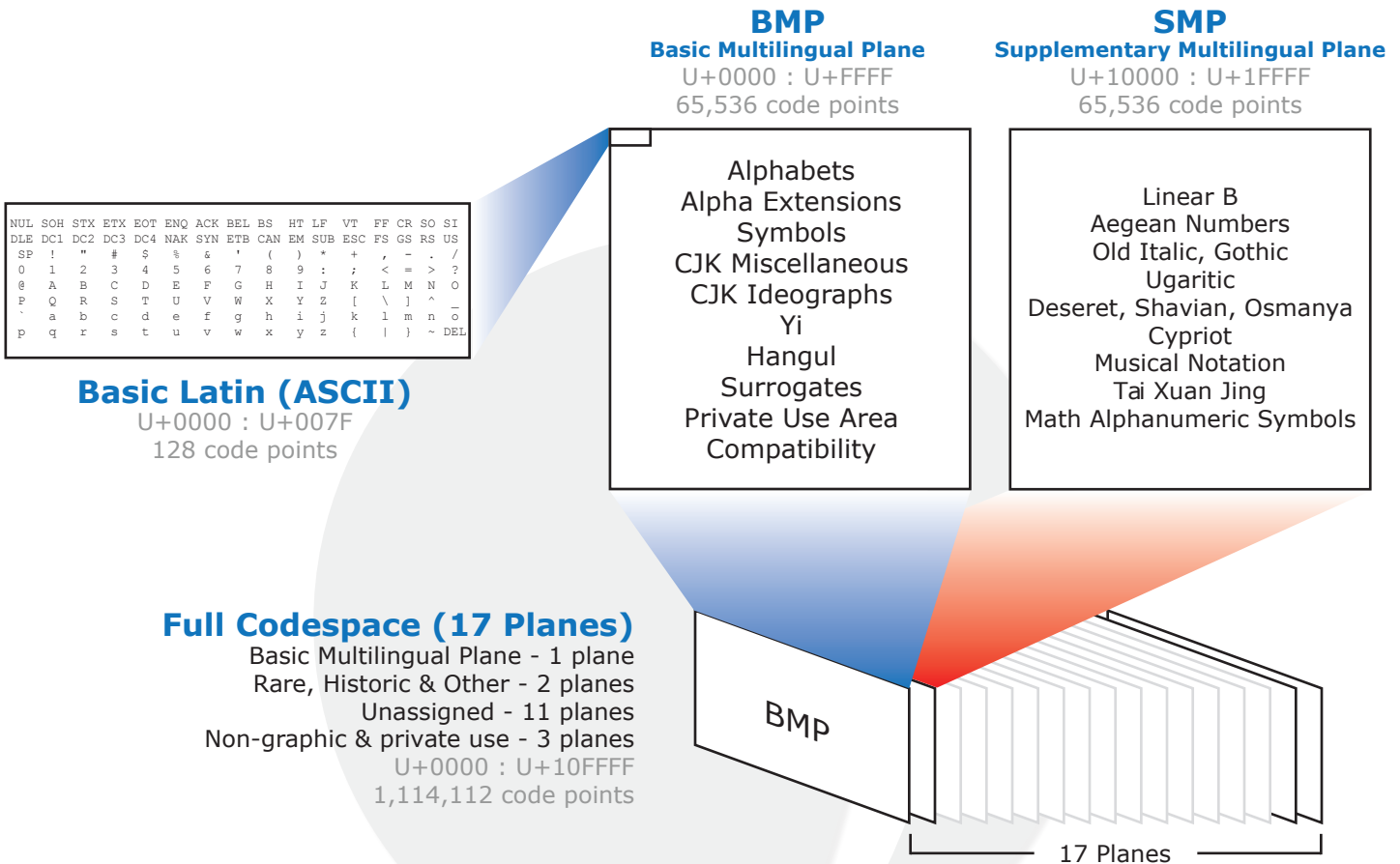


Figure 2 Unicode Allocation

Character Encodings

Unicode Transformation Format (UTF) and Universal Character Set (UCS) encodings are the two mapping methods used in Unicode implementation. These encodings map the Unicode code points to bytes so that they can be stored in memory. Some of the common UTF encodings are UTF-1, UTF-7, UTF-8, UTF-EBCDIC, UTF-16 and UTF-32.

UTF-8 is a Unicode encoding that uses 8-bit bytes to store code points in memory (similarly, UTF-7 contains 7 bits in one code value). In UTF-8, every code point from 0-127 is stored in a single byte, making the encoding very compact for standard Latin characters. Code points 128 and above are stored using 2 or more, up to 6, bytes. In other words, encoded English text looks the same in UTF-8 as it would in ASCII. For example, the string "TEXT" is represented as U+0054 U+0045 U+0058 U+0054 using Unicode code points. The UTF-8 encoded version of the same string would be 54 45 58 54, which are the ASCII representations of these characters (see Figure 1).

UTF-8 has quickly become the de facto encoding standard for interchange of Unicode text over the internet.

UTF-16 is another variable-width encoding that uses 16 bits in each code value. It can be considered a compromise as it uses 2 bytes most of the time and expands to 4 bytes per character as necessary in order to represent characters outside of the Basic Multilingual Plane (BMP).

Unfortunately, even though Unicode is not an encoding, it is incorrectly used interchangeably with UTF-16 encoding. For example, members of the *System.Text.Encoding* class in .NET are as follows:

```
System.Text.ASCIIEncoding  
System.Text.UnicodeEncoding  
System.Text.UTF32Encoding  
System.Text.UTF7Encoding  
System.Text.UTF8Encoding
```

What is implied by *System.Text.UnicodeEncoding* is actually what should have been *System.Text.UTF16Encoding*. Similarly, Ms Notepad allows users to specify the encoding while saving a text document. Provided options are ANSI, Unicode and UTF-8. What's meant by Unicode is actually UTF-16 encoding.

Unicode Byte Order Mark (BOM) and Endianness

In computer architecture, endianness is the byte or bit order used in representing data while storing it in memory or transferring it over a network. Endianness is generally dictated by hardware. For example, architectures such as x86, Z80 and VAX use the little-endian convention, which is least significant byte (LSB) first, while some others such as PowerPC or SPARC use the opposite order, big-endian (most significant byte first). This also applies to the way Unicode data is encoded. For example, UTF-16 encoding of the string "TEXT" mentioned above would be 0054 0045 0058 0054 in little-endian format. However, it is possible to reverse the byte order and represent the same Unicode data as 5400 4500 5800 5400 using UTF-16 big-endian.

Since encoded Unicode text can be ordered two different ways, we are forced to include a Byte Order Mark (BOM) at the beginning of every encoded Unicode string. The order of the BOM (FE FF or FF FE for UTF-16) indicates whether little-endian or big-endian order is used. Since UTF-8 is byte oriented, it does not require using a BOM. However, an initial BOM might be useful in identifying the data stream as UTF-8. Typical BOM values for different encodings are as follows:

Bytes	Encoding Form
00 00 FE FF	UTF-32, big-endian
FF FE 00 00	UTF-32, little-endian
FE FF	UTF-16, big-endian
FF FE	UTF-16, little-endian
EF BB BF	UTF-8

Figure 3 Byte Order Marks for Some Unicode Encodings

The UTF-8 BOM is frequently displayed as "ï»¿" in ISO-8859-1 by tools that are not prepared to handle UTF-8.

Strings are meaningful when accompanied by their encoding information. When working with a string in memory, it is important to know how it is encoded in order to correctly interpret and display it. Most problems regarding documents that look like gibberish are caused by programmers or technicians who fail to preserve the encoding information a string uses. When transforming byte strings to Unicode, we are in effect decoding our data. If we fail to provide the correct character encoding to decode from, we will inevitably end up with garbled data.

Conclusion

Understanding the key concepts underlying Unicode and character encodings is a fundamental step for the litigation support specialist. Almost every law firm or vendor has to deal with data originating from foreign countries. If your processes are unable to collect, store and interpret Unicode data, there is a good chance that this will become a major problem very quickly. Spending some time on the Unicode Consortium Website to get familiar with the code charts, getting a copy of the printed Unicode Standard and attending one of the semiannual Unicode conferences are good starting points.





About Meridian Discovery, LLC

Meridian Discovery is a top-tier provider of evidence lifecycle management, forensic collection, electronic data discovery and hosting services. Our service spectrum covers the management, identification, preservation, collection, processing, review, analysis and production of electronically stored information. We combine the power of our talented staff and true understanding of the litigation environment with our well designed workflow, technology and passion in providing exceptional service to our customers. Learn more about Meridian Discovery at www.meridiandiscovery.com.

This article in whole or in part may not be duplicated, reproduced, stored in a retrieval system or retransmitted without prior written permission of Meridian Discovery, LLC. All opinions and estimates herein constitute our judgement as of this date and are subject to change without notice. Product names mentioned herein may be trademarks and/or registered trademarks of their respective companies.

©2010 Meridian Discovery, LLC. All Rights Reserved.

Corporate Headquarters:

611 Wilshire Blvd, Ste 315
Los Angeles, CA, 90017
Office: 213.908.2188
Fax: 1 888.452.4982
Email: info@meridiandiscovery.com
www.meridiandiscovery.com

