Character encoding
a fundamental part of software internationalisation

Lukas Elsner
Queensland University of Technology
August 29, 2014

Software constructed for the global market must be aware of regional differences, such as different languages, culture dependent formatting and customised user interfaces. Without a character encoding technique, which can handle all characters used worldwide, every software needs to add its own support for handling different character sets. Since this is a non-trivial task, the overhead to globalise and internationalise a software would be a huge impact on development time and costs. Therefore, a uniform code that can interpret all known characters is needed as the foundation of all internationalisation tasks in software development. The current solution, Unicode, solves this problem for the most part. Although not all problems were resolved by it.

Contents

1 Introduction

Computers are being used worldwide and most of the software is not constructed for a special country or region. Therefore, software developers should take this into account and develop software in a globalised and localised way. Globalised software must be independent of different languages and formats, such as date and number format. Globalised software is the basis for creating a world-ready application. Beginning with a globalised framework, software can easily be localised for every culture and language.
A fundamental task of most software is processing text. A huge amount of software on the market communicates with its user by text input and output. Therefore the software's internal representation of strings must be prepared for many different characters and should be able to handle all known character sets in a unified way. A good approach to fulfill this requirement is for the complete chain of involved software, such as operating system, integrated development environment and compiler, to handle strings in a compatible way.
The history of encoding characters is long and it seems that every time a new approach was invented, new problems arose. Most modern software systems today use Unicode, which is a unified mapping between every character and a unique number. This mapping makes it possible for every software to distinguish between different characters without any misunderstanding. Still, even Unicode has not yet solved all problems, such as reliable encoding detection, as the paper will explore.

2 History of character encoding

The history of character encoding begins long before the history of computers. The first known code-based system used for long-distance transmission of information was invented by the Greeks around 350 . Many years later, several other encoding methods were invented to transmit data. Perhaps the most famous code is the Morse code, created by Alfred Vail and Samuel Morse for their electrical telegraph system in 1837, which was the first internationally used encoding system [1]. Still, even Morse code does not fulfill all of the requirements needed to encode all known characters. At the beginning of the twentieth century, long-distance communication was conducted using telegraphic systems. In 1931, the Comité Consultatif International Télégraphique et Téléphonique (CCITT) standardised the CCITT #2, an international 58-character shifted 5-bit code [1].
Around 1950, when the first stored-program computer was invented, the need for a suitable character encoding scheme was born. Computers can only handle numbers in binary representation, but humanity requires them to do more than number crunching. One of the main input/output tasks that people require is the processing of text. Towards the end of

the 1950s, two different approaches were developed independently to define a standard in character representation on computers that would allow text processing.

- Extended Binary Coded Decimal Interchange Code (EBCDIC)
  IBM developed the 8-bit EBCDIC, which was used mainly on IBM's mainframe and midrange computers as well as on other non-IBM platforms. EBCDIC was derived from Binary-Coded Decimal (BCD), an encoding system mainly used on punch cards.

- American Standard Code for Information Interchange (ASCII)
  The American National Standards Institute (ANSI) defined a character encoding based on previously-used telegraph codes. In 1963, the first ASCII standard, ASCII-1963, was defined and used commercially [2]. ASCII is a 7-bit character encoding, designed for storing and transmitting English text. It includes 33 non-printing control and 95 printable characters. In 1967, ASCII was extended to lowercase characters and eventually became the ISO 646 standard in 1983 [1]. Table 1 shows the current mapping of plain ASCII.

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0x00	000	NUL	32	0x20	040	SP	64	0x40	100	@	96	0x60	140	‘
1	0x01	001	SOH	33	0x21	041	!	65	0x41	101	A	97	0x61	141	a
2	0x02	002	STX	34	0x22	042	"'	66	0x42	102	B	98	0x62	142	b
3	0x03	003	ETX	35	0x23	043	#	67	0x43	103	C	99	0x63	143	c
4	0x04	004	EOT	36	0x24	044	$	68	0x44	104	D	100	0x64	144	d
5	0x05	005	ENQ	37	0x25	045	%	69	0x45	105	E	101	0x65	145	e
6	0x06	006	ACK	38	0x26	046	&	70	0x46	106	F	102	0x66	146	f
7	0x07	007	BEL	39	0x27	047	'	71	0x47	107	G	103	0x67	147	g
8	0x08	010	BS	40	0x28	050	(	72	0x48	110	H	104	0x68	150	h
9	0x09	011	TAB	41	0x29	051	)	73	0x49	111	I	105	0x69	151	i
10	0x0A	012	LF	42	0x2A	052	*	74	0x4A	112	J	106	0x6A	152	j
11	0x0B	013	VT	43	0x2B	053	+	75	0x4B	113	K	107	0x6B	153	k
12	0x0C	014	FF	44	0x2C	054	,	76	0x4C	114	L	108	0x6C	154	l
13	0x0D	015	CR	45	0x2D	055	-	77	0x4D	115	M	109	0x6D	155	m
14	0x0E	016	SO	46	0x2E	056	.	78	0x4E	116	N	110	0x6E	156	n
15	0x0F	017	SI	47	0x2F	057	/	79	0x4F	117	O	111	0x6F	157	o
16	0x10	020	DLE	48	0x30	060	0	80	0x50	120	P	112	0x70	160	p
17	0x11	021	DC1	49	0x31	061	1	81	0x51	121	Q	113	0x71	161	q
18	0x12	022	DC2	50	0x32	062	2	82	0x52	122	R	114	0x72	162	r
19	0x13	023	DC3	51	0x33	063	3	83	0x53	123	S	115	0x73	163	s
20	0x14	024	DC4	52	0x34	064	4	84	0x54	124	T	116	0x74	164	t
21	0x15	025	NAK	53	0x35	065	5	85	0x55	125	U	117	0x75	165	u
22	0x16	026	SYN	54	0x36	066	6	86	0x56	126	V	118	0x76	166	v
23	0x17	027	ETB	55	0x37	067	7	87	0x57	127	W	119	0x77	167	w
24	0x18	030	CAN	56	0x38	070	8	88	0x58	130	X	120	0x78	170	x
25	0x19	031	EM	57	0x39	071	9	89	0x59	131	Y	121	0x79	171	y
26	0x1A	032	SUB	58	0x3A	072	:	90	0x5A	132	Z	122	0x7A	172	z
27	0x1B	033	ESC	59	0x3B	073	;	91	0x5B	133	[	123	0x7B	173	{
28	0x1C	034	FS	60	0x3C	074	"<	92	0x5C	134		124	0x7C	174	∪
29	0x1D	035	GS	61	0x3D	075	=	93	0x5D	135	]	125	0x7D	175	}
30	0x1E	036	RS	62	0x3E	076	">	94	0x5E	136	^	126	0x7E	176	"
31	0x1F	037	US	63	0x3F	077	?	95	0x5F	137	_	127	0x7F	177	DEL

Table 1: ASCII table [3]

In many countries, the original ASCII encoding was modified to fulfill particular needs. In order to create a localised version of the original ASCII, certain characters were replaced. For example, the currency symbol, which differs between countries, was substituted with a character that represents the local currency. Some examples of modified ASCII are NF Z62010 in France, BS 4730 in United Kingdom and JIS C-6220 in Japan [1].

Besides creating those modifications for a particular country, there were so called “Extended-ASCII” character sets, which used the eighth bit and could take advantage of 128 more characters. A widespread version of this is the codepage cp-125x from Microsoft, which later became a slightly different version known as the ISO-8859-x standard [4].

An additional approach in the direction of internationalised software was the ability to use multiple code pages in one document. The ISO 2022 standard was developed as a general framework to switch between code pages using special control codes. This complex standard, which is not compatible with the Windows code pages, has never been implemented

completely and was only used for East Asian languages [4]. The ISO 2002 standard is a stateful encoding system, which means that a control code within the document can change the interpretation of a character code. One disadvantage of switching between code pages within one document is that the document must be read from the beginning. Random access at a particular position is not possible and may lead to misinterpretation of character codes. Therefore, a new and more uniform approach was needed.
The development of a universal character set, called Unicode, can be traced to late 1987, when engineers of Apple and Xerox began discussing this new encoding approach. Joe Becker, Lee Collins and Mark Davis started to investigate a universal character encoding system. They aimed for three main goals:

- **Universal**
  The needs of real world languages must be addressed
- **Uniform**
  Fixed-width codes, to preserve efficient access
- **Unique**
  One bit sequence must only have one interpretation into a character code

In fall 1988, the work on the first Unicode database began The result was the first Unicode database between 1988 and 1990, which mainly preserved the mappings of ASCII and ISO-8859 standards, to maintain backward compatibility. [5].

3 Encoding characters today

Almost every computer today uses Unicode encoding as its default character encoding system. Unicode is a unique mapping between a character and a positive integer. There are a couple of prerequisites to understand how Unicode works. One of the most important things to know is that Unicode itself does not specify how encoding has to be done. Another important factor, which should not to be underestimated, is the definition of the word "character". This chapter will define "character" and give a brief introduction into the mappings of Unicode as well as the multiple possibilities to encode Unicode into a binary representation.

3.1 Characters

When talking about characters many people might think of a particular letter in their language, which is correct, but a character can be even more than a simple 'a'. It can also be an accented 'a', like 'á' or 'ä'. A great number of accented characters are used in languages

using the Latin alphabet, so a unique representation must be designated for each one. A character can also be a symbol which many people have not seen before in their life. For example the Mongolian letter 'todo pa' (ᠮ) or the Thai character 'no nen' (ณ). Characters can also be ligatures, which are composed letters containing two or more characters. One well known ligature is the Latin 'fi'. Another one is the Latin lowercase 'ij' which can be explicitly typed as one character 'ij'. Depending of the font, there might not be any difference in the rendering. Using the definition of character given above, there are thousands of characters that must be considered by a unified encoding system. This system must also be able to add more characters in the future.

3.2 Unicode

Unicode is divided in 17 so called planes. Each plane can hold up to 65,536 different characters. Thus, Unicode can address 1,114,112 different characters and its address space length is 21-bit. Each of these addresses is called a code point. In June 2014, the Unicode consortium released the Unicode standard 7.0, which addresses a total of 123 scripts with 113,021 characters. [6] To refer to a particular Unicode character, the general writing of 'U+' followed by the code points hexadecimal number was established. This hexadecimal number is the concatenation of plane number and the address within the plane, where the plane number can be ignored when zero. For example, to reference character 3603 in plane 0, we can just write 'U+0e13'.

3.3 Unicode Transformation Format

The Unicode Transformation Format (UTF) refers to several types of character encodings which are used to transform Unicode characters into binary representations. If a computer encodes characters in Unicode, it needs to use one of Unicode's character encoding formats, UTF-8, UTF-16 or UTF-32. While UTF-16 and UTF-32 can be subdivided into the two byte orders 'big endian' and 'little endian', UTF-8, by design, will always have the same byte order. UTF-8 and UTF-16 are variable-width encodings which means, that one particular character will be represented as one to four bytes in UTF-8 and as two or four bytes in UTF-16. UTF-32 has a fixed-width length of four bytes. UTF-8 is very widespread in the World Wide Web and is the default encoding in most of common Linux distributions. Microsoft Windows uses UTF-16 for its internal data representation. UTF-32 is not commonly used, because of its huge data overhead. UTF-32 is usually only used when it is necessary to quickly and randomly access the n-th code point of a document, because this is only possible with a fixed-width format. Table 2 shows the translation of a code point into a bit sequence

when using UTF-8.

Bits of code point	First code point	Last code point	Bytes in sequence	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
7	U+0000	U+07FF	1	0xxxxxxx					
11	U+0080	U+FFFF	2	110xxxxx	10xxxxxx				
16	U+0800	U+FFFF	3	1110xxxx	10xxxxxx	10xxxxxx			
21	U+10000	U+1FFFFF	4	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx		
26	U+200000	U+3FFFFFF	5	111110xx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	
31	U+4000000	U+7FFFFFFF	6	1111110x	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx

Table 2: Scheme of UTF-8 encoding [7]

With the help of Table 2, it is easy to convert a Unicode code point into UTF-8 binary data. To encode the Thai character 'no nen', the following has to be done: The Unicode code point is 'U+0e13', which is '111000010011' in binary representation. 'U+0e13' is between 'U+0800' and 'U+FFFF'. Thus, we need three bytes to encode this character. The result would be: '11100000 10111000 10010011'. The UTF-16 encoding is a bit more complicated. However, when encoding a character of plane 0, the encoding is numerically equal to the corresponding code point. Thus, 'U+0e13' is represented as '00001110 00010011' in a big endian system and as '00010011 00001110' in a little endian system.

3.4 Unicode in modern software

A Unicode enabled operating system should meet the following criteria:

- File names can contain Unicode characters.
- System software can handle Unicode in file names, command line parameters, etc.
- User applications, such as text editors, are able to support Unicode data.

Nowadays, nearly all modern operating systems support Unicode. Of course there are many different implementations, depending of the operating system's history and the developer's decisions. Linux uses UTF-8 as its default internal data format, while Microsoft decided to store data with the UTF-16 format. [8] To support legacy software which cannot handle Unicode, an operating system can provide an Application Programming Interface (API) to convert between different encodings. If, for example, a Microsoft Windows application has to handle legacy code page data, it can make use of the Windows API functions MultiByteToWideChar [9] and WideCharToMultiByte [10] to convert from Unicode to legacy code pages and back.

However, this can lead to loss of information and should only be used when absolutely necessary. [11] To write internationalised software, it is very important that the used programming language and its compiler can handle Unicode encoded data.
Fortunately, most of those development tools have full support for this today. Java and .NET use a UTF-16 representation for every string or character object. Google's Go programming language works with UTF-8 internally and finally, the core libraries of C and C++ have added wide-string manipulation functions in addition to the legacy 'char*' functions. However, the internal data representation is compiler dependent and can vary from system to system. Thus, the Unicode support is the programmer's task in these two languages. [12] However, there are third party libraries, such as International Components for Unicode (ICU), which are adding native Unicode support to C and C++. ICU also is part of the Java Software Development Kit (SDK) to support Unicode for Java programmers in a transparent way. [13]

4 Technical issues

With or without Unicode, there are several problems for programmers and end users. Even as Unicode appears as the globalised character encoding scheme, it is not perfect. The use and implementation of Unicode can lead to unexpected outcomes.

4.1 Detecting a file encoding

Detecting the encoding used for a file can be a difficult task. ASCII, ISO-8859 and Windows code pages do not prepend magic numbers for identification. UTF-8 and UTF-16 may or may not include a byte order mask at the beginning of the document. Therefore, there is no pre-determined way of revealing the right encoding with only the file as the information source. The only way to determine the encoding is to guess, which might lead into unexpected representations of some characters. If the file only contains characters in the ASCII range (0-127), then guessing may be an easy task. On the other hand, if there are characters with the eighth bit set, there is no indication of which encoding to assume. If, for example, the file contains 'D0 AE', there are at least four different ways to interpret it:

- Assuming 8-bit ANSI (Windows cp-1252), there are two characters: 'U+00D0' and 'U+00AE', or "Ð®"
- Assuming UTF-8, there is only one character 'U+042E', or "Ю"
- Assuming UTF-16 in big endian, it would be 'U+D0AE', or "킮"
- Assuming UTF-16 in little endian, it would be 'U+AED0', or "껐"

All of these decisions could make sense, so there is no way to know which one is correct. [14] A real world example for a faulty detection algorithm is the 'Bush Hid The Facts'-Bug. This bug, which was often used as a hoax in online forums, can be reproduces on every Windows XP or Windows NT/2000 version. A text file, created with notepad.exe, containing only the text "bush hid the facts" will show nonsense squares and Chinese characters after reopening. [15]

4.2 Fonts

A font is the essential software needed to render and display a character in an appropriate way. If a font does not support the needed Unicode character set, it is not able to present the characters to the user. It may chose the replacement character 'U+FFFD', if it has no glyph available for the given code point. This special character can look like the following: □ ☒ □ ?.

4.3 Gmail and a globalised world

In 2012, the Internet Engineering Task Force (IETF) created a new standard for email addresses, to support non-Latin and accented Latin characters. Google is the first email provider who implemented this feature for its customers. [16] However, being able to use such special characters in email addresses and domains, can lead to a high security risk. The Myanmar letter Wa ('U+101D', or 'ဝ'), the Gujarati digit zero ('U+AE6', or '૦') and the Greek small letter omicron ('U+03BF', or 'ο') are all very similar to the plain ASCII letter 'o'. It might not be a direct technical issue, but in a world of spam and scam, it is important to understand how this can be used for nefarious purposes. For example, getting an email from 'a-gοοd-friend@dοmain.com.au', instead of 'a-good-friend@domain.com.au', and responding to it with sensitive information can result in numerous negative outcomes. [17]

4.4 This document

To illustrate the shortcomings of Unicode, we will examine some of the difficulties of compiling this document. It was created with vim [18], a Unicode-compatible text editor on a Gentoo Linux [19] using en_US.UTF-8 as its default locale. To compile the LaTeX document, the XeLaTeX-compiler [20] was used, which has native Unicode support. Typing a special Unicode character is easy. Just by pressing <Ctrl+Shift+u>, followed by the code point's hex-code and <Enter>. The plain text editor displayed every character correctly, but the compiled PDF-document usually displayed only a blank, or a replacement character, for the non-Latin symbols. This occurs, because the LaTeXcompiler assumes only one predefined language per document, which is English in this case. There are two common ways to

resolve this problem. The 'babel' package has support for switching languages on the fly with help of special commands. The other simple way is to define multiple fonts in the preamble of the document, which could be used explicitly for special characters. This was the more reliable solution for this document, because only single foreign characters were used. For this task the Noto font package from Google [21], which has the goal to support every Unicode character, was very helpful.

5 Future of character encoding

At first glance, Unicode looks like the solution to all character encoding problems. But as we have seen, there are still many problems left to resolve. Of course, the standard has to be maintained. New characters have to be added, and obsolete ones have to be deleted. It also would be very nice if there was one unique encoding scheme, which satisfies all needs and requirements.
It is very likely that more fonts, operating systems and user software will support Unicode in the future, which definitely seems to be the right approach in a globalised and localised software world. However, in the current situation it does not look like there is an easy way to solve all the problems.

6 Conclusion

We have seen that humanity began encoding characters long before there were any thoughts on globalisation or localisation of software. This simple looking task gained more and more complexity in a world where computers and the interaction between them and humans evolved into an indispensable task. The development of Unicode at the beginning of this globalised market was a very important step in the direction of software localisation without reinventing basic encoding schemes for every new software. On the other hand, we have noted that there are several technical and non-technical problems left. Languages are complex and dynamic structures and can not easily be mapped into a simple computer scheme. Although there is no need for another, better encoding standard, Unicode has to be maintained and further developed to satisfy the needs of all globalised and localised software development aspects.

Glossary

ANSI	The American National Standards Institute is a non-profit organisation which coordinates worldwide used standards	4, 9
API	An Application Programming Interface is a specification of a software package describing operations inputs and outputs	8
ASCII	American Standard Code for Information Interchange is a standardised 7-bit character encoding based on the English alphabet	4–6, 9, 10
BCD	Binary-Coded Decimal are non standardised 6-bit character encodings mainly used on punch cards	4
BCE	abbreviation for Before the Common Era	3
CCITT	The Comité Consultatif International Télégraphique et Téléphonique is a French organisation which coordinates standards for telecommunications. It was renamed to ITU-T in 1993	3
code point	numerical value to refer to a specific position within the Unicode code space	7, 8, 10
cp-125x	series of 8-bit character encodings as extension on top of ASCII, used in Microsoft Windows operating systems	5, 9
EBCDIC	Extended Binary Coded Decimal Interchange Code is an 8-bit character encoding from IBM derived from BCD	4
Endianness	convention used to determine the direction of stored bytes in computer memory	7–9
glyph	elemental symbol within a font, for representing a readable character	10
ICU	The International Components for Unicode are software libraries for supporting internationalisation	9
IETF	The Internet Engineering Task Force is an open standards organisation which develops Internet standards	10

ISO-8859	series of fifteen 8-bit character encodings as extension on top of ASCII	5, 6, 9
magic number	numerical or text value to identify file formats and protocols	9
plane	continuous group of 65,536 code points	7, 8
punch card	a piece of stiff paper, containing either a program or data represented by the presence or absence of holes	4
scam	the act of swindling by some fraudulent scheme	10
SDK	A Software Development Kit is a collection of software development tools for creating applications on top of a specific framework	9
spam	unsolicited, undesired, or illegal email messages	10
UTF	A Unicode Transformation Format is a mapping method for representing Unicode code points as binary value	7–10

List of Tables

References

[1] Haralambous, Y. and Horne, P.S., Fonts & Encodings, ser. O'Reilly Series. O'Reilly Media, 2007. [Online]. Available: http://books.google.com.au/books?id=qrElYgVLDwYC

[2] M. Brandle. 1963: The debut of ASCII. 14.08.2014. [Online]. Available: http://edition.cnn.com/TECH/computing/9907/06/1963.idg/

[3] ASCII - Tabelle für LaTeX. 27.08.2014. [Online]. Available: http://www.undertec.de/blog/2008/12/ascii-tabelle-fur-latex.html

[4] Korpela, J., Unicode Explained, ser. Internationalize documents, programs, and web

sites. O'Reilly Media, 2006. [Online]. Available: http://books.google.com.au/books?id=PcWU2yxc8WkC

[5] Summary Narrative. 16.08.2014. [Online]. Available: http://www.unicode.org/history/summary.html

[6] Unicode 7.0.0. 28.08.2014. [Online]. Available: http://www.unicode.org/versions/Unicode7.0.0/

[7] UTF-8. 27.08.2014. [Online]. Available: http://en.wikipedia.org/wiki/Utf8

[8] Michał Kosmulski. Introduction to Unicode — Using Unicode in Linux. 14.08.2014. [Online]. Available: http://michal.kosmulski.org/computing/articles/linux-unicode.html

[9] MultiByteToWideChar function. 27.08.2014. [Online]. Available: http://msdn.microsoft.com/en-us/library/windows/desktop/dd319072(v=vs.85).aspx

[10] WideCharToMultiByte function. 27.08.2014. [Online]. Available: http://msdn.microsoft.com/en-us/library/windows/desktop/dd374130(v=vs.85).aspx

[11] Microsoft. Code Pages. 14.08.2014. [Online]. Available: http://msdn.microsoft.com/en-us/library/dd317752.aspx

[12] Unicode Support in Various Programming Languages. 16.08.2014. [Online]. Available: http://stackoverflow.com/questions/1036585/unicode-support-in-various-programming-languages

[13] ICU - International Components for Unicode. 27.08.2014. [Online]. Available: http://site.icu-project.org/

[14] Chen, Raymond. The Notepad file encoding problem. 14.08.2014. [Online]. Available: http://blogs.msdn.com/b/oldnewthing/archive/2007/04/17/2158334.aspx

[15] B. M. Christensen. Bush Hid The Facts - Notepad Conspiracy Claim. 28.08.2014. [Online]. Available: http://www.hoax-slayer.com/bush-hid-the-facts-notepad.html

[16] P. Chaparro Monferrer. A first step toward more global email. 14.08.2014. [Online]. Available: http://googleblog.blogspot.com.au/2014/08/a-first-step-toward-more-global-email.html

[17] M. Risher. Protecting Gmail in a global world. 27.08.2014. [Online]. Available: http://googleonlinesecurity.blogspot.com.au/2014/08/protecting-gmail-in-global-world.html

[18] Vim. Vim the editor. 27.08.2014. [Online]. Available: http://www.vim.org/

[19] gentoo linux. gentoo linux. 27.08.2014. [Online]. Available: https://www.gentoo.org/

[20] XeLaTeX. The XeLaTeX wiki/resource main page. 27.08.2014. [Online]. Available: http://www.xelatex.org/

[21] Beautiful and free fonts for all languages. 27.08.2014. [Online]. Available: https://www.google.com/get/noto