CS 580 Client-Server Programming
Spring Semester, 2009
Doc 9 Character Encodings
26 Feb, 2009

Reference

The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets (No Excuses!), Joel Spolsky, http://www.joelonsoftware.com/articles/Unicode.html

Wikipedia

Java API docs

What is a Character in a String?

65	A
66	B
67	C

It is a mapping from bits to an element in an alphabet

ASCII

A common mapping

1 - 32 map to control characters
33 - 127 map to characters in the American English alphabet with punctuation

This is an example of a block encoding
    All characters are used by the same number of bits

Variable-length coding (aside)

Use differing amounts of bits to encode characters

Use few bits for frequently occurring characters

Example: Huffman encoding

Example: First MacWrite

ASCII and 128-255

On PCs people started used 128-255 for non-standard characters

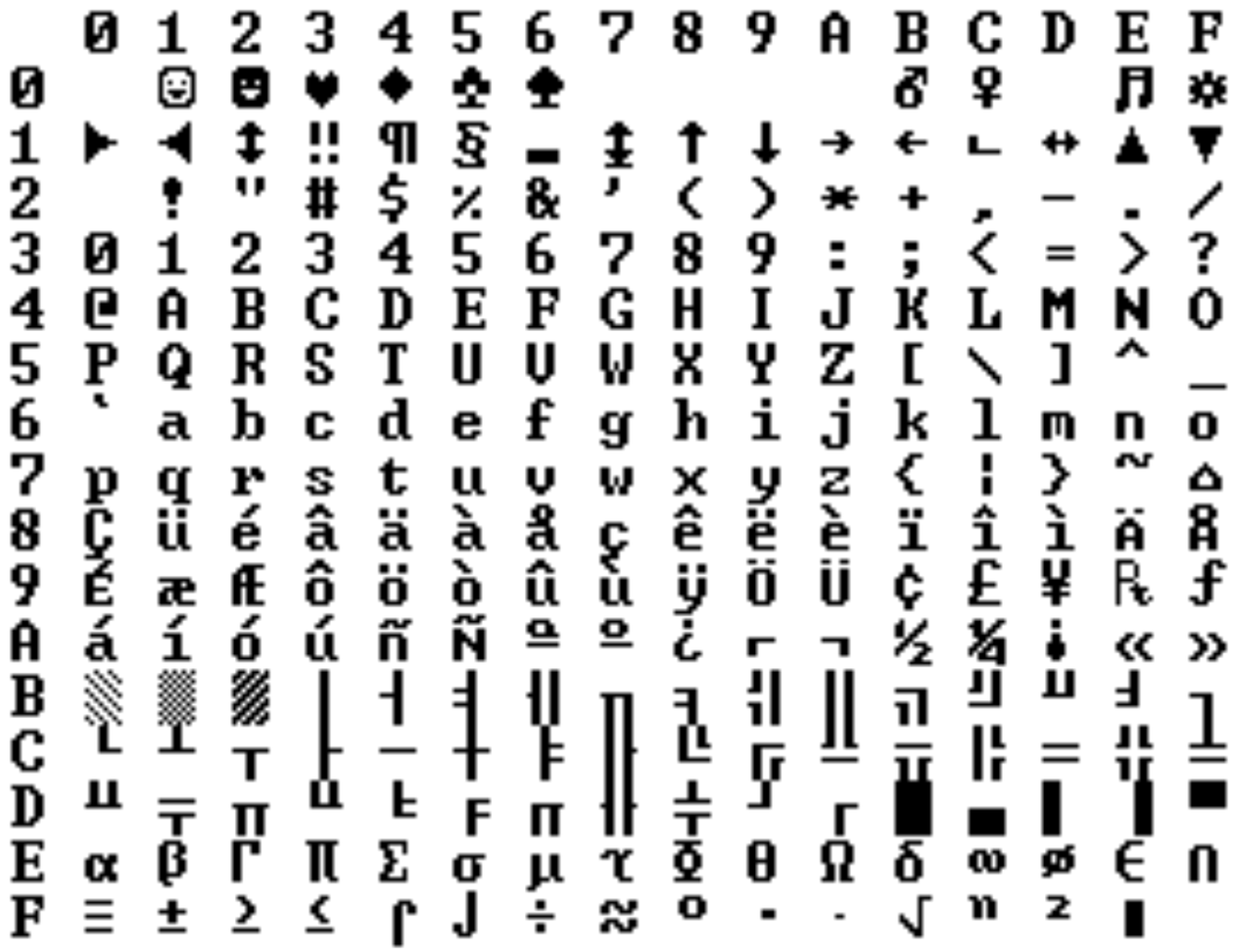
Used by non-english languages

Mappings called:

- Code pages
- OEM Code pages

1-127 same as ASCII

Code page 437 - Original IBM code page



Some DOS Code pages

437 — The original IBM PC code page
737 — Greek
775 — Estonian, Lithuanian and Latvian
850 — "Multilingual (Latin-1)" (Western European languages)
852 — "Slavic (Latin-2)" (Central and Eastern European languages)
855 — Cyrillic
857 — Turkish
858 — "Multilingual" with euro symbol
860 — Portuguese
861 — Icelandic
862 — Hebrew
863 — French Canadian
865 — Nordic
866 — Cyrillic
869 — Greek
65001 — UTF-8 Unicode

Source http://en.wikipedia.org/wiki/Code_page.
Yes the code pages were known by number and yes there were multiple code pages for the same language

OEM Pages

OEM pages supported by Windows

http://msdn.microsoft.com/en-us/goglobal/bb964655.aspx

Catalog of Character Sets and OEM pages

http://www.i18nguy.com/unicode/codepages.html

Displaying Characters

Cyrillic alphabet

а	б	в	г	д	е	ё	ж	з	и	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я
а	б	в	г	д	е	ё	ж	з	и	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я

Text mode of VGA-compatible PC graphics hardware uses 8-bit code page

Often done in graphics mode now

Originally used bit maps for characters

Fonts

Description of the visual representation of characters of an "alphabet"

Includes

- weight
- style
- width
- serif

The cat in the hat

The cat in the hat

The cat in the hat

THE CAT IN THE HAT

The cat in the hat

THE CAT IN THE HAT

THE CAT IN THE HAT

The cat in the hat

THE CAT IN THE HAT

The cat in the hat

Outline Fonts

Adobe Type 1 Fonts

TrueType

OpenType

Rather than use bitmaps for display

Use math functions to describe the outline of each character

Produces smoother characters on screen and print

Can scale the characters to different sizes

So

Code pages map 8 bit words to characters

Also need way to display the characters

Code pages and Standards

ISO/IEC 8859

- Standard code pages (8 bit) for latin alphabets
- 16 parts (pages)
- Does not cover East Asian Languages (CJK)
- 1998

Unicode

Standard to represent text of any language

Basic Idea

Code Point	Character
U+0041 (65)	A
U+0042	B

Mapping between numbers and characters

Some Terms

UCS - Universal Character Set

Standard list of all characters with code point

UTF - Unicode Transformation Format

Mapping between bits and code points

Some History

1987

Joe Becker (Xerox), Lee Collins(Apple) and Mark Davis (Apple)

Start work on a universal character set

1988 - Unicode 88

Draft calling for 16-bit character model

1991 - Unicode version 1.0.0

7161 characters

1993 - Unicode 1.1

34,233 characters

ISO/IEC 10646-1:1993 (Unicode is also an ISO standard)

2008 - Unicode 5.1

100,713 characters

Defines codespace of 1,114,112 code points

Unicode Planes

Unicode code space is divided into planes

Each plane contains 65,535 code points

Plane 0

Basic Multilingual Plane (BMP)

Many symbols (3071)

Contains almost all modern languages

Indic scripts:

Devanagari, Bengali, Gurmukhi, Gujarati, Oriya, Tamil, Telugu, Kannada, Malayalam, Sinhala

Plane 1

Supplementary Multilingual Plane (SMP)

Mostly used for historic scripts

Some musical and math symbols

Plane 2 -

Supplementary Ideographic Plane

40,000 seldom seen Han characters

Plane 3-13

Not used

Some Unicode Character Encodings

UTF-8

UTF-16

GB 18030

UTF-8

Variable-length character encoding for Unicode

Uses 1-4 bytes to represent a character

Unicode	Byte1	Byte2	Byte3	Byte4	example
0-127 U+000000-U+00007F 0xxxxxxx	0xxxxxxx				'$' U+0024 → 00100100 → 0x24
128-2,047 U+000080-U+0007FF 00000yyy xxxxxxxx	110yyyxx	10xxxxxx			'¢' U+00A2 → 11000010,10100010 → 0xC2,0xA2
2,048-65,535 U+000800-U+00FFFF yyyyyyyy xxxxxxxx	1110yyyy	10yyyyxx	10xxxxxx		'€' U+20AC → 11100010,10000010,10101100 → 0xE2,0x82,0xAC
65,536-1,114,111 U+010000-U+10FFFF 000zzzzz yyyyyyyy xxxxxxxx	11110zzz	10zzyyyy	10yyyyxx	10xxxxxx	U+10ABCD → 11110100,10001010,10101111,10001101 → 0xF4,0x8A,0xAF,0x8D

ASCII encoding is identical to the UTF-8 encoding of same characters

Table is from http://en.wikipedia.org/wiki/UTF-8

Note

Some bit patterns are used to indicate a character needs multi-bytes to encode

So we can only encode 127 character with one byte

UTF-16 & UCS-2

UCS-2
UCS-2BE (Big Endian)
UCS-2LE (Little Endian)
Fix-Length Character encoding
Uses 2-bytes

UTF-16
UTF-16 BE (Big Endian)
UTF-16 LE (Little Endian)
Variable-Length Character encoding
Uses 2-bytes words

BOM - Byte Order Mark

Use to indicate Big or Little Endian in UTF-16 and UTC-2

Zero-Width No-Break Space - U+FEFF

FE FF for Bid Endian

FF FE for Little Endian

Required for UTC-2

Recommended for UTF-16

Placed before characters

UTF-16 & UTC-2 Uses

UTF-16	UTC-2
Windows 2000-Vista	Java before Java 5.0
Mac OS X	Windows before Windows 2000
Qualcomm BREW	Symbian OS
Java	Sony Ericsson UIQ handsets
.NET	Python

Current version of Python use UTC-4 for plane 1 & 2 characters

How many Encodings are There?

Many more than you would like

Java Required Encodings

Charset	Description
US-ASCII	Seven-bit ASCII, a.k.a. ISO646-US, a.k.a. the Basic Latin block of the Unicode character set
ISO-8859-1	ISO Latin Alphabet No. 1, a.k.a. ISO-LATIN-1
UTF-8	Eight-bit UCS Transformation Format
UTF-16BE	Sixteen-bit UCS Transformation Format, big-endian byte order
UTF-16LE	Sixteen-bit UCS Transformation Format, little-endian byte order
UTF-16	Sixteen-bit UCS Transformation Format, byte order identified by an optional byte-order mark

java.nio.charset.Charset.availableCharsets()

Available in My JRE

{Big5=Big5, Big5-HKSCS=Big5-HKSCS, EUC-JP=EUC-JP, EUC-KR=EUC-KR, GB18030=GB18030, GB2312=GB2312, GBK=GBK, IBM-Thai=IBM-Thai, IBM00858=IBM00858, IBM01140=IBM01140, IBM01141=IBM01141, IBM01142=IBM01142, IBM01143=IBM01143, IBM01144=IBM01144, IBM01145=IBM01145, IBM01146=IBM01146, IBM01147=IBM01147, IBM01148=IBM01148, IBM01149=IBM01149, IBM037=IBM037, IBM1026=IBM1026, IBM1047=IBM1047, IBM273=IBM273, IBM277=IBM277, IBM278=IBM278, IBM280=IBM280, IBM284=IBM284, IBM285=IBM285, IBM297=IBM297, IBM420=IBM420, IBM424=IBM424, IBM437=IBM437, IBM500=IBM500, IBM775=IBM775, IBM850=IBM850, IBM852=IBM852, IBM855=IBM855, IBM857=IBM857, IBM860=IBM860, IBM861=IBM861, IBM862=IBM862, IBM863=IBM863, IBM864=IBM864, IBM865=IBM865, IBM866=IBM866, IBM868=IBM868, IBM869=IBM869, IBM870=IBM870, IBM871=IBM871, IBM918=IBM918, ISO-2022-CN=ISO-2022-CN, ISO-2022-JP=ISO-2022-JP, ISO-2022-KR=ISO-2022-KR, ISO-8859-1=ISO-8859-1, ISO-8859-13=ISO-8859-13, ISO-8859-15=ISO-8859-15, ISO-8859-2=ISO-8859-2, ISO-8859-3=ISO-8859-3, ISO-8859-4=ISO-8859-4, ISO-8859-5=ISO-8859-5, ISO-8859-6=ISO-8859-6, ISO-8859-7=ISO-8859-7, ISO-8859-8=ISO-8859-8, ISO-8859-9=ISO-8859-9, JIS_X0201=JIS_X0201, JIS_X0212-1990=JIS_X0212-1990, KOI8-R=KOI8-R, MacRoman=MacRoman, Shift_JIS=Shift_JIS, TIS-620=TIS-620, US-ASCII=US-ASCII, UTF-16=UTF-16, UTF-16BE=UTF-16BE, UTF-16LE=UTF-16LE, UTF-8=UTF-8, windows-1250=windows-1250, windows-1251=windows-1251, windows-1252=windows-1252, windows-1253=windows-1253, windows-1254=windows-1254, windows-1255=windows-1255, windows-1256=windows-1256, windows-1257=windows-1257, windows-1258=windows-1258, windows-31j=windows-31j, x-Big5-Solaris=x-Big5-Solaris, x-euc-jp-linux=x-euc-jp-linux, x-EUC-TW=x-EUC-TW, x-eucJP-Open=x-eucJP-Open, x-IBM1006=x-IBM1006, x-IBM1025=x-IBM1025, x-IBM1046=x-IBM1046, x-IBM1097=x-IBM1097, x-IBM1098=x-IBM1098, x-IBM1112=x-IBM1112, x-IBM1122=x-IBM1122, x-IBM1123=x-IBM1123, x-IBM1124=x-IBM1124, x-IBM1381=x-IBM1381, x-IBM1383=x-IBM1383, x-IBM33722=x-IBM33722, x-IBM737=x-IBM737, x-IBM834=x-IBM834, x-IBM856=x-IBM856, x-IBM874=x-IBM874, x-IBM875=x-IBM875, x-IBM921=x-IBM921, x-IBM922=x-IBM922, x-IBM930=x-IBM930, x-IBM933=x-IBM933, x-IBM935=x-IBM935, x-IBM937=x-IBM937, x-IBM939=x-IBM939, x-IBM942=x-IBM942, x-IBM942C=x-IBM942C, x-IBM943=x-IBM943, x-IBM943C=x-IBM943C, x-IBM948=x-IBM948, x-IBM949=x-IBM949, x-IBM949C=x-IBM949C, x-IBM950=x-IBM950, x-IBM964=x-IBM964, x-IBM970=x-IBM970, x-ISCII91=x-ISCII91, x-ISO-2022-CN-CNS=x-ISO-2022-CN-CNS, x-ISO-2022-CN-GB=x-ISO-2022-CN-GB, x-iso-8859-11=x-iso-8859-11, x-JIS0208=x-JIS0208, x-JISAutoDetect=x-JISAutoDetect, x-Johab=x-Johab, x-MacArabic=x-MacArabic, x-MacCentralEurope=x-MacCentralEurope, x-MacCroatian=x-MacCroatian, x-MacCyrillic=x-MacCyrillic, x-MacDingbat=x-MacDingbat, x-MacGreek=x-MacGreek, x-MacHebrew=x-MacHebrew, x-MacIceland=x-MacIceland, x-MacRomania=x-MacRomania, x-MacSymbol=x-MacSymbol, x-MacThai=x-MacThai, x-MacTurkish=x-MacTurkish, x-MacUkraine=x-MacUkraine, x-MS950-HKSCS=x-MS950-HKSCS, x-mswin-936=x-mswin-936, x-PCK=x-PCK, x-windows-50220=x-windows-50220, x-windows-50221=x-windows-50221, x-windows-874=x-windows-874, x-windows-949=x-windows-949, x-windows-950=x-windows-950, x-windows-iso2022jp=x-windows-iso2022jp}

How do we know the Encoding Used

In HTML there is a way to specify it

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
```

If it is not specified Web browsers guess

IE does a byte frequency analysis

How do we know the Encoding Used

In network protocols you have to specify the encoding

Single Most Important Fact About Encodings

There Ain't No Such Thing As Plain Text.

It does not make sense to have a string without knowing what encoding it uses

Java Strings

Some String Constructors

String(byte[] bytes, String charsetName)

Constructs a new String by decoding the specified array of bytes using the specified charset.

String(int[] codePoints, int offset, int count)

Allocates a new String that contains characters from a subarray of the Unicode code point array argument.

String Methods

char charAt(int index)

Returns the char value at the specified index.

int codePointAt(int index)

Returns the character (Unicode code point) at the specified index.

Java Streams

Read/Write bytes
Does not know about charsets

Can be used to read Unicode character/other encodings

But your code must convert using correct encoding

Java Reader/Writers

Handle character encodings

But you have to tell it which encoding

InputStreamReader Constructor

InputStreamReader(InputStream in)

Creates an InputStreamReader that uses the default charset.

InputStreamReader(InputStream in, Charset cs)

Creates an InputStreamReader that uses the given charset.

Default Encoding in My JRE

java.nio.charset.Charset.defaultCharset()
MacRoman

Don't forget

Networks only deal with bytes

Some protocols are binary so deal only in bytes

Some protocols deal with both binary and text