Kódování znaků (nejen češtiny)

Základní terminologie

– velmi nejednotná, liší se u velkých firem a organizací, stejné pojmy se používají pro různé věci
– dále je použita terminologie z Unicode – *Character Encoding Model*
  `http://www.unicode.org/reports/tr17/tr17-5.html`
– lze používat několikaúrovňovou organizaci

Znaková sada (množina kódovaných znaků – *Coded Character Set* – CCS)
– mapování mezi množinou (abstraktních) znaků a množinou nezáporných celých čísel (kódové body – *code points*).
  – A = 41 (abstraktní znak A má kódový bod 41)
  – US-ASCII, ISO 8859-1, JIS X 0201, Unicode nebo ISO 10646-1 jsou příklady znakových sad

??? (*Character Encoding Form* – CEF)
– mapování množiny nezáporných celých čísel (prvků CCS) na množinu kódových jednotek dané šířky, např. 32bitových integerů
  – A = 00000041 (znak A namapovaný na 32bitový integer)

Kódovací schéma, kódování (*Character Encoding Scheme* – CES)
– způsob mapování kódových jednotek z CEF do posloupnosti oktetů (osmibitových bajtů)
– další používané názvy jsou *character map, charmap, character set, charset, code page*
  – A = 00 41 (pro znakovou sadu Unicode a kódovací schéma UTF-16BE)
– dále bude výhradně používán pojem „charset“, protože existuje stejně pojmenovaná příslušná třída `java.nio.charset.Charset`
– pro jednu znakovou sadu může být více charsetů
– naopak charset je často určen jen pro jednu znakovou sadu
  – např. pro znakovou sadu Unicode existují charsety
    UTF-8
    UTF-16
    UTF-32
– naopak pro několik různých asijských znakových sad (ideografická písma) existuje jedno kódovací schéma EUC
  – charsety jsou např. EUC-JP, EUC-KR, x-EUC-TW

Jeden dokument může být napsán ve více znakových sadách (Unicode, ISO 8859-2) a za použití různých charsetů (UTF-8, UCS-2, ISO-8859-2), což je nejhorší možný případ, pokud se jedná o jeden soubor.
Běžně je jeden soubor vždy jen v jednom charsetu.

Pozor:
– charsety nikdy nerozlišují tvar znaku (*glyph* – nesprávně „font“)

Characters Versus Glyphs

Glyphs	Unicode Characters
A A A A A A A A	U+0041 LATIN CAPITAL LETTER A
a a a a a a a a	U+0061 LATIN SMALL LETTER A
fi fi	U+0066 LATIN SMALL LETTER F + U+0069 LATIN SMALL LETTER I
п n ū	U+043F CYRILLIC SMALL LETTER PE
ه ٤ ھ ﻫ	U+0647 ARABIC LETTER HEH

Historie

– sedmibitový ASCII kód (US-ASCII)
  – MSB (*Most Significant Bit*) je vždy nulový
  – základ všech dalších kódování (*zero extending* – viz dále)
  – popisuje "všechny znaky z anglické klávesnice", tj. velká a malá neakcentovaná písmena latinky, číslice, interpunkci a speciální znaky
  – bez 32 řídících znaků je tedy k dispozici 128 - 32 = 96 znaků
  – malých a velkých písmen je 52, číslic 10, zbývajících znaků 34
  – stačí pouze pro čistě anglické texty

– pro mnoho jazyků nestačí jen neakcentovaná písmena a pro akcentovaná písmena už není v US-ASCII místo
– proto vznikaly osmibitové znakové sady
  – využívá se osmý bit (MSB), což dává možnost pro dalších až 128 znaků
  – čeština např. využívá áčďéěíňóřšťúůýž
    (tj. 30 dalších znaků – včetně velkých písmen)

– problémy pojmenování viz dále

– protože jsou osmibitové, není třeba používat speciální kódovací schéma
  – celé číslo znaku ze znakové sady vždy představuje 1 bajt
  – znaková sada má stejný název jako kódovací schéma a ten je stejný jako název charsetu

– existuje mnoho různých charsetů, protože 128 volných znaků nestačí pro všechny požadavky
  – k dohodě nedošlo bohužel ani mezi evropskými státy používajícími latinku
  – to znamená, že např. česko-francouzská organizace musí používat pro své dokumenty dva charsety

– dle ISO normy vznikají charsety ISO-8859-1 až ISO-8859-15
– další charsety vznikají v národních standardizačních organizacích
– kromě nich existují znakové sady závislé na platformě (proprietární formáty)
  – např. CP-1250, MacCentralEurope, které jsou většinou jako jediné na konkrétní platformě plně podporovány

– celkově se odhaduje, že existují stovky těchto charsetů
– pokud kódují znaky latinky, je v naprosté většině prvních 128 znaků totožných s US-ASCII (*zero-extending*)
  – existují výjimky, např. EBCDIC (*Extended Binary-Coded Decimal Interchange Code*), který popisuje jen neakcentované znaky, ale ty jsou uloženy i v horních 128 pozicích

– některé aplikace označují osmibitové charsety jako ANSI

– společné označení pro sedmi a osmibitové charsety je **SBCS** (*Single-Byte Character Set)*

Současnost

– s rozvojem Internetu zvýšená potřeba výměny dokumentů mezi různými platformami a různými národními jazyky
– řešením je 16bitová znaková sada
– do její přípravy se bohužel najednou pouštějí dvě různé organizace
  a) ISO (*International Organization for Standardization*) od 1989, znaková sada ISO/IEC 10646, ve zkratce **UCS** (*Universal Character Set*)
  b) Unicode (*Unicode Consortium*) od 1990, znaková sada Unicode, ve zkratce **Unicode**

– naštěstí se obě organizace domlouvají a od roku 1991 pracují na sjednocení obou znakových sad, což bylo dokončeno v 1993
  – v současnosti obě znakové sady nejsou absolutně totožné, liší se např. v drobných specifikacích jednotlivých znaků, částečně odlišné terminologii apod.,
  – číselné hodnoty (*code points*) jednotlivých znaků jsou identické

– z běžného pohledu nemá význam obě sady rozlišovat a běžně se považují za synonyma a používá se společné označení Unicode
  – velmi dobře zdokumentovaná znaková sada viz `www.unicode.org`

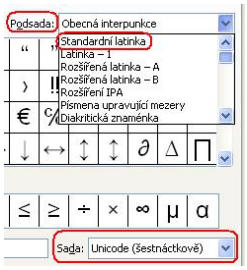
– *code points* jednotlivých znaků se označují jako U+hexa_číslo, např. znak A je U+0041
  – prvních 128 znaků je stejných jako u US-ASCII (*zero extending*)

bitů	charset	binárně	hexa	znak
7	US-ASCII	1000001	41	A
8	ISO-8859-2	01000001	41	A
16	UTF-16, UCS-2	00000000 01000001	41	A
32	UTF-32, UCS-4	00000000 00000000 00000000 01000001	41	A

Historie Unicode

verze 1. 1991 28292 znaků
verze 2. 1996 38869 znaků -- rozšíření na 32 bitů (resp. 21 bitů)
verze 3. 2000 49170 znaků
verze 4. 2003 96248 znaků
verze 5. 2004 připravovaná verze

– 16 bitová znaková sada umožňuje použít asi 65 tisíc znaků (U+0000 až U+FFFF)
  – ty jsou ve skupině označené jako **BMP** (*Basic Multilingual Plane*)
  – BMP obsahuje všechny znaky používané v Evropě a Americe plus základní ideografická písma čínštiny, japonštiny a korejštiny (HAN písmo)
  – samotné BMP je vnitřně děleno do bloků, kde např. ASCII je v *Basic Latin block* (dělení je občas používáno ve fontech)



– od Unicode verze 2.0 přestal rozsah 16 bitů dostačovat a Unicode přešlo na 32 bitů
  – z nich ale využívá jen 21 bitů, tj. U+000000 až U+10FFFF
  – to znamená celkem 16 skupin (sfér – *plane*), každá o velikosti asi 65 tisíc znaků, dohromady asi 1 milion znaků
  – první skupina v pořadí je BMP (U+0000 až U+FFFF)
  – další jsou v rozsahu U+10000 až U+10FFFF a běžně se téměř nikdy (v našich zeměpisných šířkách) nepoužívají
  – znaky v těchto skupinách se nazývají *supplementary characters*
  – jak Unicode, tak i ISO neplánují ani v budoucnu překročit počet 21 bitů

Problém pořadí bajtů

– ukládáme-li do paměti (do souboru) vícebajtové entity, je třeba rozlišovat pořadí bajtů
  – A = 0041
  – *little-endian* LE (vyšší řády na vyšší adrese) 41 00
  – *big-endian* BE (vyšší řády na nižší adrese) 00 41
  – toto platí i pro např. čtyřbajtové entity 00 00 00 41 či 41 00 00 00
  – způsob ukládání *little-* nebo *big-endian* závisí na platformě (Windows LE), procesoru (Intel LE, Motorola BE), programovacím jazyce (Java vždy BE), aplikaci, ...

– proč to potřebujeme?
  – většina současných souborových systémů pracuje s bajty (pro správnou serializaci znaků do bajtového proudu)

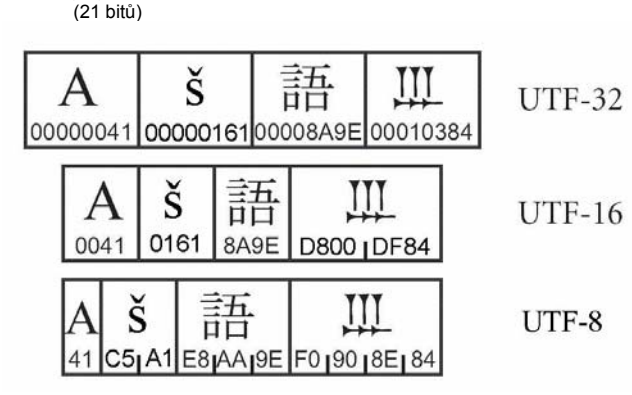
Problém kódovacích schémat

– znak přesahuje jeden bajt – je možné používat více kódovacích schémat
  – pro detailní pochopení je vhodné rozlišovat Unicode a ISO 10646 (pro praktické použití ne)
  – Unicode samo o sobě nepopisuje mechanismy pro identifikaci proudu bajtů

– ISO 10646 má dva základní charsety
  – UCS-2 (2 bajty) stačí právě pro BMP, ale pro nic víc!
  – UCS-4 (4 bajty) pro celý rozsah
  UCS = *Universal Multiple-Octet Character Set*

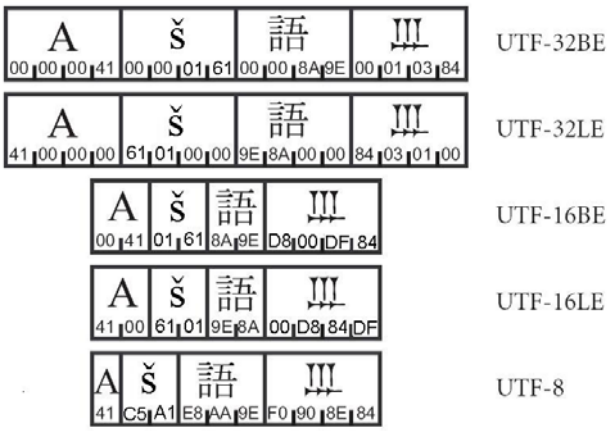
  – dolní polovina UCS-4 je identická s UCS-2
  – ISO 10646 využívá i možností UTF-x

– Unicode má 7 platných charsetů:
  UTF-8, UTF-16, UTF-16BE, UTF-16LE, UTF-32, UTF-32BE, UTF-32LE
  – **UTF** = *Unicode* (nebo *UCS*) *Transformation Format*
  – UTF-x se používají i pro ISO 10646
  – jakýkoliv UTF-x může sloužit pro uložení celého rozsahu Unicode (21 bitů)



– obecný omyl je že UCS-2 = UTF-16
  – rovnost platí pro všechny znaky z BMP (proto se zaměňují)
  – UCS-2 nemůže popsat znaky z oblastí nad BMP (*supplementary characters*) a nemůže využívat **zástupné páry** (*surrogate pairs*) jako UTF-16 (viz dále)

– UTF-xy (ne UTF-8) mohou přímo určit pořadí bajtů – LE nebo BE



Značka bajtového pořadí

– UTF-x mohou pro identifikaci pořadí bajtů využívat počáteční značku bajtového pořadí
  – *initial byte order mark* (**BOM**), která je umístěna na samém začátku souboru
  – pro tento účel definuje Unicode dva kódové body
    ▪ U+FEFF = ZERO WIDTH NO-BREAK SPACE (*byte order mark*) (pevná mezera nulové délky)
    ▪ U+FFFE = not a character code

– pro UTF-16 pro BE má tvar FE FF a pro LE pak FF FE
– je-li načtena správně, pevná mezera nulové délky se ze své podstaty nemůže zobrazit
  – při nesprávném načtení (záměna BE za LE nebo naopak) se opět nezobrazí, protože se jedná o neplatný znak
  – k nesprávnému načtení by ale principiálně nemělo dojít

– BOM se ale někdy nemusí nebo nesmí vyskytovat, což je dáno charsetem

charset	BOM	hodnota	
UTF-8	volitelný	EF BB BF	
UTF-16	doporučený	FE FF nebo FF FE	
UTF-16LE	zakázaný	FF FE	na BOM se nebere zřetel, tj. i opačný FE FF musí být ignorován
UTF-16BE	zakázaný	FE FF	na BOM se nebere zřetel, tj. i opačný FF FE musí být ignorován
UTF-32	doporučený	00 00 FE FF nebo FF FE 00 00	
UTF-32LE	zakázaný	FF FE 00 00	na BOM se nebere zřetel
UTF-32BE	zakázaný	00 00 FE FF	na BOM se nebere zřetel

UTF-8

– bylo vytvořeno proto, aby se znaky Unicode daly zakódovat posloupností bajtů, protože s bajty umí pracovat každá aplikace a každý souborový systém
– vzniká v 1992
– staré a nepoužívané označení je UTF-2
– je popsáno v RFC 2279, v Amendment 2 v ISO 10646-1 a též v Unicode Standard, tzn. je to obecně rozšířený a přijímaný formát
  – např. řetězce v Javě v `.class` souborech jsou v UTF-8

– ze zmíněných 7 charsetů Unicode se (v našich zeměpisných šířkách) často používá právě UTF-8
– výhody
  – pro texty využívající jen znaky anglické abecedy je UTF-8 totožná s US-ASCII
    ▪ využívá se jen jeden bajt na jeden znak
    ▪ s US-ASCII umí pracovat každá aplikace
  – pro akcentované znaky se využívají dva bajty

– soubory kódované v UTF-16 zabírají pro většinu textů psaných latinkou (nejen angličtinou) zbytečně dvojnásobné místo
  – zkoumáme-li četnost výskytu akcentovaných znaků v českém textu, zjistíme, že mají asi 10% výskyt

"Například v tomto odstavci napsaném zcela prokazatelně češtinou s plným využitím akcentů je z celkových 129 písmen 114 písmen bez akcentů a jen 15 písmen s akcenty."

  – započítáme-li do předchozího příkladu i mezery, interpunkci a číslice (všechny jsou z ASCII) dostaneme celkově 164 znaků a z toho 15 akcentovaných

– základní nevýhoda UTF-8 – znaky nemají stejnou délku => není možné skočit přímo na určitý znak („přeskoč prvních 20 znaků“)
  – pravděpodobnost "omylu" (považování poloviny znaku za celý znak) je omezena principem kódovacího schématu

znak Unicode	max. význ. bitů	bajty UTF-8
U+0000 až U+007F	7	0xxx xxxx
U+0080 až U+07FF	11	110x xxxx 10yy yyyy
U+0800 až U+FFFF	16	1110 xxxx 10yy yyyy 10zz zzzz
U+10000 až U+10FFFF	21	1111 0xxx 10yy yyyy 10zz zzzz 10ss ssss

– princip čtení
  – má-li bajt nastaveno MSB, pak počet jedničkových bitů za ním udává počet následujících bajtů znaku, které vždy začínají bity 10
  – "trefíme-li" se náhodně doprostřed znaku, poznáme to podle začátku 10 a pak je nutné přeskočit všechny následující bajty začínající 10

Poznámka:
– principiálně je možné zakódovat pomocí UTF-8 až 31 bitů
1111 110x 10yy yyyy 10zz zzzz 10ss ssss 10tt tttt 10uu uuuu
  – ale protože Unicode končí na významových 21 bitech, se tento způsob nevyužívá
  – nejširší znak v UTF-8 má tedy 4 bajty

– u UTF-8 je z principu zbytečná značka bajtového pořadí (BOM)
  – specifikace říká, že není ani vyžadována, ani doporučována, ale pokud je použita, nesmí vadit
  – mnoho aplikací ale BOM vyžaduje a vytváří a bez BOM pracují chybně, např. nejsou schopné automaticky rozpoznat charset
  – některé (např. SciTe) dokonce považují BOM za nezbytnou část a označení “UTF-8“ znamená “UTF-8 + BOM“ a pro UTF-8 bez BOM používají označení "UTF-8 Cookie"

– praktický příklad

znak	Unicode	Unicode bitově	UTF-8 bitově	UTF-8 bajtově
D	U+0044	0000 0000 0*100 0100*	0*100 0100*	44
á	U+00E1	0000 0**000 11***10 0001*	110**0 0011** 1*010 0001*	C3 A1
š	U+0161	0000 0**001 01***10 0001*	110**0 0101** 1*010 0001*	C5 A1
a	U+0061	0000 0000 0*110 0001*	0*110 0001*	61
BOM	U+FEFF	1111 1110 1111 1111	1110 1111 1011 1011 1011 1111	EF BB BF
	U+FFFE	1111 1111 1111 1110	1110 1111 1011 1111 1011 1110	EF BF BE

– jediná správná značka bajtového pořadí je v UTF-8 EF BB BF
  – posloupnost EF BF BE vznikla zakódováním neexistujícího znaku, není tedy BOM a žádná aplikace ji nerozpozná
– slovo "Dáša" může tedy v UTF-8 vypadat
  a) **44 C3 A1 C5 61**
  b) **EF BB BF 44 C3 A1 C5 61**
  ale nikdy jako
  c) **EF BF BE 44 C3 A1 C5 61**

Poznámka:
– existuje též kódovací schéma UTF-7, které využívá pouze 7 bitů bajtu
  – je popsáno v RFC-2152
  – prakticky by se použilo, pokud by (zastaralý) přenosový kanál dovoloval přenášet jen sedmibitové bajty
  – slovo "Dáša" v UTF-7 je: **44 2B 41 4F 45 42 59 51 2D 61** což znamená, že každý akcentovaný znak využívá čtyři bajty

Problém UTF-16

– dokud Unicode využívalo jen U+0000 až U+FFFF kódovací schéma UTF-16 neexistovalo a používalo se UCS-2 s pevnou šířkou 2 bajty
– po expanzi na 21 bitů (od Unicode verze 2.0) UCS-2 s pevnou šířkou přestává stačit, ale není možné jej opustit, protože je široce používáno
  – proto je od 1994 navrženo a od 1996 nastupuje UTF-16 (přijímané též ISO)
  – použil se stejný trik, jako ve vztahu ASCII a UTF-8, tj. vše staré musí zůstat nezměněno, nové se přidá zvětšením šířky
– UTF-16 je kódovací schéma s proměnnou šířkou, čímž se odlišuje od UCS-2 (pro znaky mimo BMP)
  – kódování znaků v BMP se nemění!!!

– možného zvětšení šířky se dosáhlo trikem, že byly vyhrazeny dva bloky kódových bodů 1024 velké, které se mohu vyskytovat pouze společně, tj. jeden znak je zakódován pomocí 2 x 16 bitů
  – to dává možnost zakódovat 1024 x 1024 znaků nad BMP
  – oba „znaky“ dohromady se nazývají "zástupné páry" (*surrogate pairs*)

– první blok je v rozsahu U+D800 až U+DBFF
  tj. 1101 1000 0000 0000 až 1101 1011 1111 1111
  (*high-half zone* nebo *high surrogate area* nebo *leading-surrogate code unit*)

- druhý blok je v rozsahu U+DC00 až U+DFFF
  tj. 1101 1100 0000 0000 až 1101 1111 1111 1111
  (*low-half zone* nebo *low surrogate area* nebo *trailing-surrogate code unit*)

– v každém bloku je volných 10 bitů, dohromady 20 bitů
  – Unicode ale potřebuje 21 bitů – použije se trik, kdy se využije toho, že BMP je kódováno jen 16 bity a proto lze 21 bitů "posunout dolů" odečtením 0x10000 (viz dále)
  – zástupné páry tedy umí zakódovat znaky U+010000 až U+10FFFF
– pro vyšší znaky nelze použít UTF-16 (ale není to třeba) a musí se použít UTF-32 nebo UCS-4

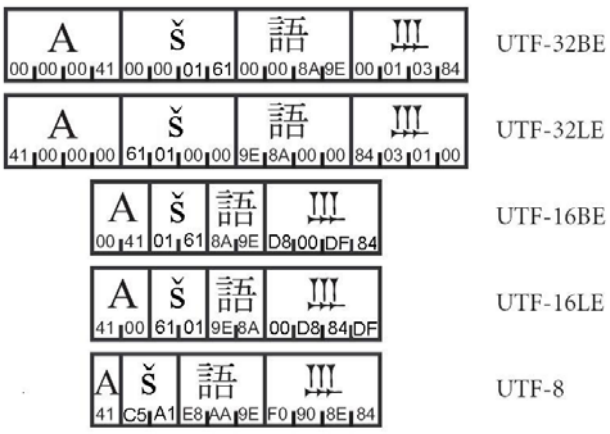
– způsob kódování znaků nad BMP, tj. U+010000 až U+10FFFF
  – od hodnoty znaku odečteme 0x10000, čímž se dostaneme do rozsahu 0x00000 až 0xFFFFF, binárně yyyy yyyy yyxx xxxx xxxx
  – přidáme tyto bity k základům zástupných párů, tj. k 0xD800 a 0xDC00, binárně 1101 10yy yyyy yyyy a 1101 11xx xxxx xxxx

– prakticky:
  – znak U+10384
    0x10384 – 0x10000 = 0x00384 = 0000 0000 00|11 1000 0100
    vyšší pár: 1101 10|00 0000 0000 = 0xD800
    nižší pár: 1101 11|11 1000 0100 = 0xDF84

– protože i UTF-16 potřebujeme serializovat, používá se BOM, respektive UTF-16 existuje ve verzích UTF-16, UTF-16LE, UTF-16BE (viz dříve)

UTF-32

– byl definován v 1999
– pro prvních 21 bitů se neliší od UCS-4
– nad 21 bitů není definován (narozdíl od UCS-4), což je ale pouze akademický problém
– opět existují tři verze UTF-32, UTF-32LE, UTF-32BE



Problémy pojmenování charsetů

– každý významnější uživatel (nikoli tvůrce!) charsetu považuje za svoji povinnost jej nově pojmenovat, nejlépe úplně odlišně od již užívaných pojmenování
  – vzniká situace, kdy jeden a týž charset má množství různých jmen
  – např. US-ASCII má dalších 14 oficiálně evidovaných jmen: ISO646-US, IBM367, ASCII, cp367, default, ascii7, ANSI_X3.4-1986, iso-ir-6, us, 646, iso_646.irv:1983, csASCII, ANSI_X3.4-1968, ISO_646.irv:1991
– takže i když je uvedeno jméno charsetu, příjemce dokumentu nemusí být schopen text přečíst, protože toto pojmenování (nikoli charset!) nezná

– pořádek v pojmenování zavádí *Internet Assigned Numbers Authority* (IANA) **`http://www.iana.org/assignments/character-sets`**

– rozlišuje v pojmenování základní (nejoficiálnější) jméno, které se nazývá "kanonické jméno" a ostatní evidovaná jména, kterým se říká "aliasy"
  – např. US-ASCII je kanonické jméno a pokud není speciální důvod, mělo by se používat
  – ANSI_X3.4-1986 je evidovaný alias od US-ASCII

– může se stát, že jméno charsetu není evidováno v IANA, ale charset je přesto podporován některými aplikacemi
  – pak se používá stejný princip kanonického jména a aliasů, ale kanonické jméno musí začínat znaky "x-" nebo "X-"
  – např. API Javy poporuje charset s kanonickým jménem x-MacCentralEurope a aliasem MacCentralEurope
  – API JDK 1.5.0 podporuje 148 charsetů a 655 aliasů

Praktický dopad na uživatele počítačů v České republice

– zcela běžně se můžeme setkat s dokumentem kódovaným těmito charsety
  – uvažujeme pouze dokumenty psané latinkou

1) **US-ASCII** – *American Standard Code for Information Interchange*
  aliasy: ISO646-US, IBM367, ASCII, cp367, default, ascii7, ANSI_X3.4-1986, iso-ir-6, us, 646, iso_646.irv:1983, csASCII, ANSI_X3.4-1968, ISO_646.irv:1991
  – základní sedmibitový charset
  – nemůže zobrazit akcenty
  – je běžný v dokumentech psaných angličtinou nebo ve zdrojových kódech programů

2) **ISO-8859-2** – *Latin Alphabet No. 2*
  aliasy: ibm912, l2, ibm-912, cp912, ISO_8859-2:1987, ISO_8859-2, latin2, csISOLatin2, iso8859_2, 912, 8859_2, ISO8859-2, iso-ir-101
  – základní osmibitový charset pro východeovropské země – mezinárodní standard dle ISO
  – na platformě Unix/Linux se používalo zcela výhradně, ale poslední verze mnoha distribucí implicitně používají UTF-8

3) **windows-1250** – *Windows Eastern European*
  aliasy: cp1250, cp5346
  – osmibitový proprietární charset fy Microsoft
  – podporován operačními systémy a aplikacemi této firmy
  – od ISO-8859-2 se liší pouze ve znacích š, Š, ž, Ž, ť, Ť

4) **IBM852** – *MS-DOS Latin-2*
  aliasy: 852, ibm-852, csPCp852, cp852, ibm852
  – osmibitový proprietární charset fy IBM
  – používaný charset v MS-DOSu a v konzolovém okénku Windows

5) **x-MacCentralEurope** – *Macintosh Latin-2*
  alias: MacCentralEurope
  – osmibitový proprietární charset fy Macintosh

6) **UTF-8** – *Eight-bit UCS Transformation Format*
  alias: UTF8, unicode-1-1-utf-8
  – základní charset pro uživatele znakové sady Unicode
  – znaky mají šířku 1 nebo 2 bajty
  – na začátku může mít BOM EF BB BF , který ale nemá pro charset význam (může mít význam pro aplikaci, která podle něj snadno pozná UTF-8)

7) **UTF-16** – *Sixteen-bit UCS Transformation Format, byte order identified by an optional byte-order mark*
  aliasy: utf16, UTF_16
  – charset pro uživatele znakové sady Unicode
  – v aplikacích se často označuje jako UCS-2, což není evidovaný alias
  – znaky mají šířku 2 bajty
  – na začátku může mít BOM FE FF nebo FF FE, který má význam pro určení pořadí bajtů
  – neobsahuje-li BOM, musí se pořadí bajtů (*big-endian* nebo *little-endian*) vyzkoušet

8) **UTF-16BE** – *Sixteen-bit Unicode Transformation Format, big-endian byte order*
  aliasy: X-UTF-16BE, UnicodeBigUnmarked, UTF_16BE, ISO-10646-UCS-2
  – charset pro uživatele znakové sady Unicode
  – znaky mají šířku 2 bajty a jsou vždy v *big-endian* pořadí
  – na začátku nesmí mít BOM, je-li tam, je ignorován ve verzi FEFF i FFFE

9) **UTF-16LE** – *Sixteen-bit Unicode Transformation Format, little-endian byte order*
  aliasy: UnicodeLittleUnmarked, X-UTF-16LE, UTF_16LE
  – charset pro uživatele znakové sady Unicode
  – znaky mají šířku 2 bajty a jsou vždy v *little-endian* pořadí
  – na začátku nesmí mít BOM, je-li tam, je ignorován ve verzi FEFF i FFFE

– jak vypadá bajtově dokument obsahující pouze slovo "Dáša"

1.	US-ASCII		**`44`** *nelze zobrazit akcenty* **`61`**
2.	ISO-8859-2		**`44 E1 B9 61`**
3.	windows-1250		**`44 E1 9A 61`**
4.	IBM852		**`44 A0 E7 61`**
5.	x-MacCentralEurope		**`44 87 E4 61`**
6.	UTF-8		**`44 C3 A1 C5 A1 61`**
7.	UTF-8	(s BOM)	**`EF BB BF 44 C3 A1 C5 A1 61`**
8.	UTF-16	(s BOM b-e)	**`FE FF 00 44 00 E1 01 61 00 61`**
9.	UTF-16	(s BOM l-e)	**`FF FE 44 00 E1 00 61 01 61 00`**
10.	UTF-16BE		**`00 44 00 E1 01 61 00 61`**
11.	UTF-16LE		**`44 00 E1 00 61 01 61 00`**

Zdroje

`Java\jdk1.5.0\docs\guide\intl\encoding.doc.html`
`Java\jdk1.5.0\docs\api\java\nio\charset\Charset.html`
`http://www.iana.org/assignments/character-sets`
`http://www.unicode.org/`
`http://www.unicode.org/faq/`
`http://www.unicode.org/reports/tr10/`
`http://www.unicode.org/reports/tr19/tr19-9.html`
`http://www.unicode.org/reports/tr17/tr17-5.html`