

ZEICHENSÄTZE, ZEICHENSATZKODIERUNG, UNICODE UND KODIERUNGSKONVERSION

SEMINAR: KLASSIFIKATION

DOZENT: STEFAN LANGER

CIS, UNIVERSITÄT MÜNCHEN

WINTERSEMESTER 2022



Motivation

Dokumente können in verschiedensten Formaten auftreten

Der Text kann in verschiedensten Zeichensatzkodierungen vorliegen

Zeichensätze

Was ist ein Zeichensatz, eine Zeichensatzkodierung ...

- Terminologie
- Einige Beispiele

Unicode

- UTF-8
- UTF-16, UTF-32 u.a. Unicode-Kodierungen

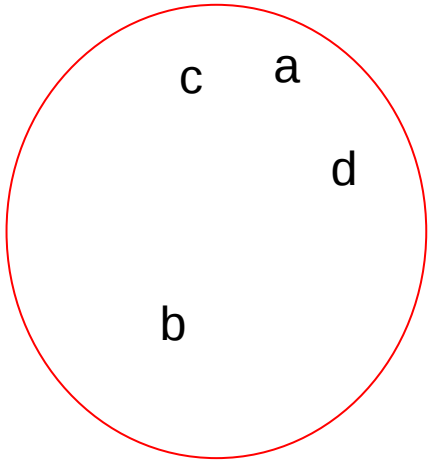
Andere Zeichensätze mit mehr als 256 Zeichen

Hebräisch, Arabisch (Schreibrichtung rechts nach links)

Kodierungskonversion

Zeichensatzerkennung

Terminologie

Zeichensatz character set character repertoire	nummerierter Zeichensatz coded character set	Zeichensatzkodierung character encoding (scheme)
	$a \rightarrow 1$ $b \rightarrow 2$ $c \rightarrow 3$ $d \rightarrow 4$	$a \rightarrow 1 \rightarrow 00000001$ $b \rightarrow 2 \rightarrow 00000010$ $c \rightarrow 3 \rightarrow 00000011$ $d \rightarrow 4 \rightarrow 00000100$

ASCII

ASCII

ASCII definiert nur den Bereich 0-127

0-31 ist mit Kontrollzeichen besetzt

20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
	!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
'	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	
p	q	r	s	t	u	v	w	x	y	z	{		}	~	

ISO-8859-1 versus Windows-1252

Iso-8859-1 (=Iso-Latin-1)

Iso-8859-X Zeichensätze benutzen nicht
den Bereich x80 bis x9F

A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
	í	φ	£	¥	¥	!	§	..	©	≡	«	¬	-	®	-
B0	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C0	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î
D0	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ
E0	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î
F0	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ

Windows-1252

Bereich x80-x9F benutzt

Sonst identisch zu ISO-8859-1

80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
€		,	f	,,	...	†	‡	^	%	Š	‹	Œ		Ž	
	¸	,	“	”	•	-	-	~	™	š	›	œ		ž	ÿ
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
	í	φ	£	¥	¥	!	§	..	©	≡	«	¬	-	®	-
B0	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C0	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î
D0	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ
E0	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î
F0	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ

Kyrillische Zeichensätze

80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF
C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF
E0	E1	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF
F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF

KOI8-R - Russisch

80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF
C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF
E0	E1	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF
F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF

KOI8-U - Ukrainisch

A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF
C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF
E0	E1	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF
F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF

ISO-8859-5

80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF
C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF
E0	E1	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF
F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF

Windows-1251

Für den die kyrillische Schrift existieren zahlreiche unterschiedliche Zeichensätze, die z.T. alle, z.T. nur den für eine Sprache relevanten Teil der kyrillischen Buchstaben enthalten.

Zeichensatzkodierungen in einem Byte

Alle Zeichensätze mit ≤ 256 Zeichen und einer entsprechenden Nummerierung 0-255 ($< 2^8$) können in einem Byte 1 zu 1 kodiert werden.

Der numerische Wert des Bytes ist gleich der der Kodeposition des Zeichens im nummerierten Zeichensatz

Wenn der Zeichensatz ASCII-kompatibel sein soll, sind nur 128 Positionen zusätzlich zu den 128 mit ASCII-Zeichen besetzten Positionen verfügbar (z.B. in Windows 1252).

ISO-8859-X-Zeichensätze haben nur 96 frei besetzbare Kodepositionen, da die Positionen 0-127 gleich wie ASCII ist, und die Positionen 128-160 nicht besetzt sind.

VISCII – die Grenzen des Bytes

Vietnamese – VISCII

Mehr als 128 zusätzliche Buchstaben; daher muss der Bereich < 128 (<x80) geändert werden →

6 zusätzliche Buchstaben im Bereich 0-31

Rest von ASCII ist unverändert



				02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
				Ä			Å	Å									
						14	15					19	1A			1E	1F
						ÿ						ÿ				ÿ	
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F		
	!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/		
30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F		
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?		
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F		
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O		
50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F		
P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_		
60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F		
,	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o		
70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F		
p	q	r	s	t	u	v	w	x	y	z	{		}	~			
80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F		
À	Á	Â	Ã	Ä	Å	Ä	Ä	Ä	Ä	Ä	Ä	Ä	Ä	Ä	Ä		
90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F		
Õ	Ö	Ö	Ö	Ö	Ö	Ö	Ö	Ö	Ö	Ö	Ö	Ö	Ö	Ö	Ö		
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF		
Ö	ä	ä	ä	ä	ä	ä	ä	ä	ä	ä	ä	ä	ä	ä	ä		
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF		
Ö	ö	ö	ö	ö	ö	ö	ö	ö	ö	ö	ö	ö	ö	ö	ö		
C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF		
Ä	Ä	Ä	Ä	Ä	Ä	Ä	Ä	Ä	Ä	Ä	Ä	Ä	Ä	Ä	Ä		
D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF		
Ö	ö	ö	ö	ö	ö	ö	ö	ö	ö	ö	ö	ö	ö	ö	ö		
E0	E1	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF		
ä	ä	ä	ä	ä	ä	ä	ä	ä	ä	ä	ä	ä	ä	ä	ä		
F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF		
d	ü	ö	ö	ö	ö	ö	ö	ö	ö	ö	ö	ö	ö	ö	ö		

Übungsaufgabe (5 min)

Die vorgestellten Methoden zur Zeichensatzkodierungen sind nicht geeignet für manche Sprachen – welche?

Welche Lösungen können Sie sich für solche Zeichensätze vorstellen?

Große Zeichensätze

Größere Zeichensätze

CJK – Chinesisch, Japanisch, Koreanisch – haben wesentlich mehr als 256 Zeichen.

Japanisch verwendet neben Hiragana und Katakana (Silbenschrift) auch Zeichen chinesischen Ursprungs (Kanji); in Koreanisch werden neben Hangul (s.u.) auch noch chinesische Schriftzeichen (Hanja) verwendet.

洗拘梳拮拮搥搥汜汎
沘沘汜汜汜汜汜汜
邗邗邗邗邗邗邗邗
境垠垠垠垠垠垠

Chinesisch

Japanisch

(Kanji+Hiragana/Katakana)

Koreanisch (Hangul)

全角ひらがな
全角カタカナ
全角英数
半角カタカナ
半角英数
直接入力

모든 인간은 태어날 때부터 자유로우며 그 존엄과 권리에 있어 동등하다. 인간은 천부적으로 이성과 양심을 부여받았으며 서로 형제애의 정신으로 행동하여야 한다.

GB 18030

Chinesisch: GB 18030

> 25000 Schriftzeichen

捌办贝壁并踩差扯趾描词答蹈
芭半背辟病睬岔车驰蠹瓷达捣
溴瓣丰必炳财察炒弛储慈搭刀
懊伴北弊饼才捺吵迟础辞错档
奥拌卑敝秉材碴巢池楚雌挫荡
傲扮悲闭丙裁查潮匙除磁措党
祇版碑痹柄猜茶嘲持潏茨搓挡
翱板杯庇冰擦荏朝痴雏疵撮当
熬颁爆币兵怖叉钞吃锄绰磋蛋
敖般鲍彪摈部插抄秤蹯戳寸弹
凹扳豹毙宾簿蹬超骋厨蠢存诞
盎搬暴毕滨步层倡逞榭纯村淡
昂斑报蔽濒布测唱承出淳翠悫
肮斑抱莧斌不册畅诚初唇淬但
案裨宝碧彬埠侧敞澄奥醇粹氮
胺拜饱彼瘥补策厂惩丑椿瘁旦
岸败堡笔别哺厕肠程瞅春脆胆
暗佰保鄙憋卜草俚乘绸垂催掸
按摆雹比鳖捕曹长呈仇捶崔郸
俺百薄鼻表驳槽常成筹捶推单
安柏剥逼膘泊糙尝橙愁炊窳丹
氨白衰迸彪渤操场城稠吹篡担
鞍爸包黼标脯藏猖称畴创蹕耽
隘罢胞泵遍脖沧昌撑畴闯促怠
爱霸苞甬辫舶仓颀衬酬床簇逮
碍坝谤绷辩帛舱阍趁抽幢醋待
艾耙傍崩辨伯苍产陈宠窗糙袋
矮把镑笨下箔灿铲沉崇疮凑贷
葛靶蚌本变铂惨缠忱虫串丛代
癌跋磅苯便搏搏恹恹晨冲喘从殆
皑拔棒奔扁勃残饯饯尘充船匆带
哀巴绑被贬博蚕蝉辰炽传鹵戴
唉庖膀焙编波参掺臣斥椽葱榛稻
哎八榜惫边钵餐搏榔翅穿聪歹到
挨笆榔备鞭拔蔡豺澈赤川次呆导
埃吧帮狈陛播菜柴彻尺揣赐大袴
阿叭邦倍避菠彩拆掣修处刺打岛
啊扒绊狈臂玻采瓷撒齿触此瘩倒

中国标准技术开发公司 北京中易电子有限公司 点阵字库样本

Der universelle Zeichensatz Unicode

Der größte Zeichensatz der Welt: Unicode

Übersicht – der Zeichensatz

Zeichensatzkodierungen für Unicode

- UTF-8
- UTF-16
- UCS-2, UTF-32 (= UCS-4)

Der größte aller Zeichensätze

Unicode – Universeller Zeichensatz

- Die Zeichen aller Sprachen, auch toter Sprachen
- größter existierender Zeichensatz
- Version 6.2. : 110 182 Schriftzeichen (>> 65535)

Unified Canadian Aboriginal Syllabic

1430	1440	1450	1460	1470	1480
1431	1441	1451	1461	1471	1481
1432	1442	1452	1462	1472	1482

Gurmukhi

	0A0	0A1	0A2	0A3	0A4	0A5	0A6	0A7
0		ਐ	ਠ	ਰ	ੀ			ੌ
1	ਏ		ਡ		ੌ			ੌ
2	ਏ		ਚ	ਲ	ੌ			ੌ
3	ਏ	ਓ	ਣ	ਲ				ੌ
4		ਐ	ਤ					ੌ
5	ਅ	ਕ	ਬ	ਵ				
6	ਆ	ਖ	ਦ	ਸ਼			ੌ	
7	ਇ	ਗ	ਧ		ੌ		ੌ	
8	ਈ	ਘ	ਨ	ਸ਼	ੌ		ੌ	
9	ਉ	ਛ		ਹ		ਖ	ੌ	
A	ਊ	ਚ	ਪ			ਗ	ੌ	
B		ਛ	ਫ		ੌ	ਜ਼	ੌ	
C		ਜ਼	ਬ	ੌ	ੌ	ੌ	ੌ	
D		ੌ	ੌ	ੌ	ੌ	ੌ	ੌ	
E		ੌ	ੌ	ੌ	ੌ	ੌ	ੌ	
F	ਏ	ੌ	ੌ	ੌ	ੌ	ੌ	ੌ	ੌ

... Unicode: Historische Zeichensätze

					
16A0	16B0	16C0	16D0	16E0	16F0
					
16A1	16B1	16C1	16D1	16E1	
					
16A2	16B2	16C2	16D2	16E2	
					
16A3	16B3	16C3	16D3	16E3	
					
16A4	16B4	16C4	16D4	16E4	

Runen

	
10330	10340
	
10331	10341
	
10332	10342
	
10333	10343
	
10334	10344
	
10335	10345

Gotisch

													
1D100	1D110	1D120	1D130	1D140	1D150	1D160	1D170	1D180	1D190	1D1A0	1D1B0	1D1C0	1D1D0
													
1D101	1D111	1D121	1D131	1D141	1D151	1D161	1D171	1D181	1D191	1D1A1	1D1B1	1D1C1	1D1D1
													
1D102	1D112	1D122	1D132	1D142	1D152	1D162	1D172	1D182	1D192	1D1A2	1D1B2	1D1C2	1D1D2
													
1D103	1D113	1D123	1D133	1D143	1D153	1D163	1D173	1D183	1D193	1D1A3	1D1B3	1D1C3	1D1D3

Musik

								
2800	2810	2820	2830	2840	2850	2860	2870	2880
								
2801	2811	2821	2831	2841	2851	2861	2871	2881
								
2802	2812	2822	2832	2842	2852	2862	2872	2882
								
2803	2813	2823	2833	2843	2853	2863	2873	2883

Braille
(Blindenschrift)

Was Unicode nicht enthält...

- Klingon und einige andere künstliche Sprachen mit künstlichem Zeichensatz.

Unicode-Zeichen:Eigenschaften

Unicode-Zeichen haben Eigenschaften wie (Auswahl)

- Typ des Zeichens (z.B. Buchstabe oder Zahlwort oder Punctuation)
- Groß- oder Kleinschreibung

Außerdem haben viele Zeichen Beziehungen zu anderen Zeichen:

- Großbuchstabe $\leftrightarrow$ Kleinbuchstabe

Diese Eigenschaften sind in der UNICODE CHARACTER DATABASE (UCD) festgehalten. Die Eigenschaftstabellen sind als Textdateien kodiert.

Die Datei die die wichtigsten Eigenschaften aller Zeichen kodiert ist

- <http://www.unicode.org/Public/UNIDATA/UnicodeData.txt>

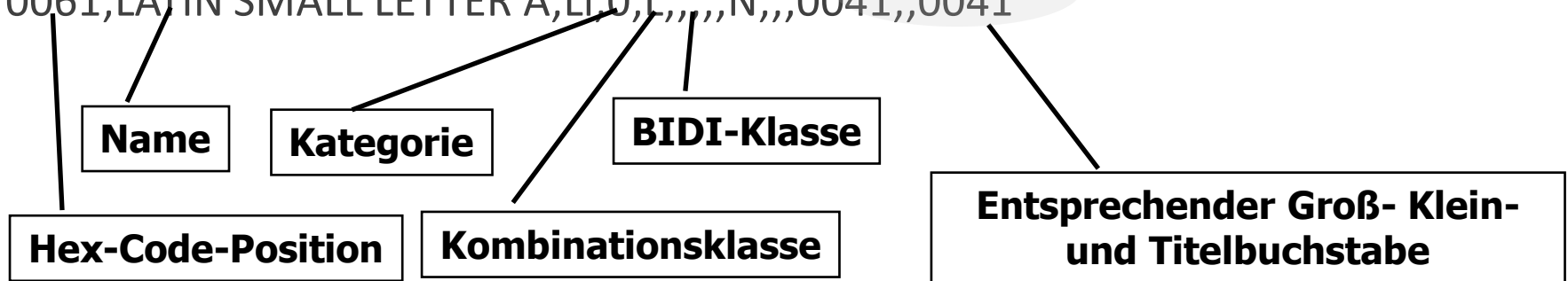
UnicodeData.txt

Name* (<reserved>)	M	N	(1) These names match exactly the names published in the code charts of the Unicode Standard.
General_Category (Cn)	E	N	(2) This is a useful breakdown into various character types which can be used as a default categorization in implementations.
Canonical_Combining_Class (0)	N	N	(3) The classes used for the Canonical Ordering Algorithm in the Unicode Standard.
Bidi_Class (L, AL, R)	E	N	(4) These are the categories required by the Bidirectional Behavior Algorithm in the Unicode Standard.
Decomposition_Type (None) Decomposition_Mapping (=)	E	N S	(5) This field contains both values, with the type in angle brackets.
Numeric_Type (None) Numeric_Value (Not a Number)	E	N N	(6) If the character has the <i>decimal digit</i> property then the value of that digit is represented with an integer value in fields 6, 7, and 8.
	E	N	(7) If the character has the <i>digit</i> property, but is not a decimal digit, then the value of that digit is represented with an integer value in fields 7 and 8.
	E	N	(8) If the character has the <i>numeric</i> property, as specified in Chapter 4 of the Unicode Standard, the value of that character is represented with an positive or negative integer or rational number in this field.
Bidi_Mirrored (N)	B	N	...
Unicode_1_Name (<none>)	M	I	(10) This is the old name as published in Unicode 1.0.
ISO_Comment (<none>)	M	I	(11) This is the ISO 10646 comment field.
Simple_Uppercase_Mapping	S	N	(12) Simple uppercase mapping (single character result).
Simple_Lowercase_Mapping	S	N	(13) Simple lowercase mapping (single character result).
Simple_Titlecase_Mapping	S	N	(14) Similar to Uppercase mapping (single character result).

Examples from UnicodeData.txt

0041;LATIN CAPITAL LETTER A;Lu;0;L;;;;;N;;;;;0061;

0061;LATIN SMALL LETTER A;Ll;0;l;;;;;N;;;0041;;0041



Numerischer Wert

0031;DIGIT ONE;Nd;0;EN;;1;1;1;N;;;;;

Zeichensatz**kodierung** für Zeichensätze mit mehr als 256 Zeichen am Beispiel Unicode

Methoden:

- Variable Länge, byte-basiert – z.B. UTF-8
 - Für jedes Zeichen werden ≥ 1 Byte verwendet
 - in den meisten Fällen ASCII-kompatibel
 - manchmal auch nicht (SHIFT-JIS)
- Variable Länge, aber immer > 1 Byte (z. B. UTF-16)
 - Selbe Länge (2 Byte) für fast alle Zeichen; einige selten verwendete Zeichen > 2 Byte
- Feste Länge (e.g. UCS-4 / UTF-32)
 - Selbe Bytezahl für alle Zeichen

Unicode-Zeichensatzkodierungen

UTF-7: Variable Länge, 1-4 Byte, nur 7 Bit; wenig verwendet

UTF-8: Variable Länge 1-4 Byte; sehr weit verbreitet

UTF-16: Variable Länge, aber meiste Zeichen mit 2 Byte kodiert

basiert auf UCS-2; Java verwendet UTF-16 als interne Zeichensatzkodierung

UCS-2 : Fixe Länge (2 Bytes max. 65535 Zeichen); VERALTET

UCS-4 ($\approx$ UTF-32) : Fixe Länge (4 Bytes), braucht viel Speicherplatz...

Mehrbyte-Zeichensatzkodierungen mit variabler Länge

Grundlagen zum Verständnis von UTF-8

Was heißt ASCII-kompatibel?

Eine Zeichensatzkodierung ist ASCII-kompatibel, wenn jedes Byte ≥ 0 & ≤ 127 den entsprechenden ASCII-Buchstaben repräsentiert. Alle Zeichen, die keine ASCII-Zeichen sind, werden durch Bytesequenzen repräsentiert, in denen jedes Byte einen Wert ≥ 128 (≤ 255)

→ das hat folgende Konsequenzen für Programme, die nur ASCII verarbeiten können:

- grundlegende Texttokenisierung kann vorgenommen werden
- z.B. funktioniert eine HTML-Parser
 - alle Funktionalitäten, die Bytes > 128 nicht speziell implementieren, sind verfügbar.

UTF-8 : Wie funktioniert?

UTF-8 – Wie es funktioniert

Unicode – kurze Wiederholung

Jedes Unicode Zeichen hat einen numerischen Kode

Es gibt >> 65000 Unicode Zeichen (aber immer < 1 Mio)

- die numerischen Kodes < 128 sind ASCII-äquivalent
- die numerischen Kodes < 256 sind == Iso-8859-1

UTF-8: Anforderungen

Anforderungen an UTF-8

- Kodierte mehrere 100 000 Zahlen als Sequenzen von 1-4 Bytes
- Ascii-kompatibel
- Schnelles Auffinden des nächsten Zeichens an beliebiger Stelle einer Bytefolge.
- Einfach zu kodieren-dekodieren

UTF-8 Spezifikation 1

Für jeden Kode c produziere eine Bytesequenz folgender Form:

if $c < 128$ (2^7)

CS = c

if $c \geq 128$ and < 2048 (2^{11})

Das erste Byte ist **110YYYYY**, folgendes Byte ist **10XXXXXX**, c wird in den Y und X-Positionen kodiert, wobei die ersten ≥ 2 Bytes in die YYYYY Sequenz verschoben werden.

INTEGER **00000000 00000000 00000000 11111111** (= 255) wird

11000011 10111111

Wann ist Schluss für zwei Bytes? bei ≥ 2048 (2^{11})

UTF-8 Spezifikation 2

if $c \geq 2048$ (2^{11}) and $c \leq 65535$ (2^{16})

Erstes Byte ist **1110**YYYY, die folgenden zwei Bytes sind **10**XXXXXX, c wird wieder in YYYY und XXXXXX kodiert; wobei die führenden Bytes verschoben werden.

Die Integer **00000000 00000000 11111111 11111111** (= 65535)

wird

11101111 **10**111111 **10**111111

UTF-8 Spezifikation 3

if $c \geq 65536$ (2^{16}) (und, was immer der Fall ist < 2097152 (2^{21}))

erstes Byte ist **11110**YYY, die drei folgenden **10**XXXXXX

UTF-8 nach Unicode: in Perl

```
sub utf8_to_unicode
{
    my $sequence = shift;
    my @byte_sequence = split //, $sequence;
    my $i = 0;
    $val1 = unpack("C",$byte_sequence[$i]);
    if($val1 < 128)
    {
        return $val1;
    }
    elsif($val1 >= 128 && $val1 < 192) #Bits: 10??????
    {
        die "Invalid leadings byte in utf-8 value: $_";
    }
    elsif($val1 >= 192 && $val1 < 224) #Bits: 110?????
    {
        $i++;
        $val2 = unpack("C",$byte_sequence[$i]);
        if($val2 < 128 || $val2 >= 192)
        {
            die "Invalid 2-byte utf-8 value in $file , $line: $_";
        }
        my $unicode_value = (($val1^192)<<6) | ($val2^128);
        return $unicode_value;
    }
}
```

```
    elsif($val1 >= 224 && $val1 < 240) #Bits: 1110????
    {
        $i++;
        $val2 = unpack("C",$byte_sequence[$i]);
        $i++;
        $val3 = unpack("C",$byte_sequence[$i]);
        if($val2 < 128 || $val2 >= 192
           || $val3 < 128 || $val3 >= 192)
        {
            die "Invalid 3-byte utf-8 value in $file , $line: $_";
        }
        my $unicode_value = (($val1^224)<<12) |
            (($val2^128)<<6) | ($val3^128);
        return $unicode_value;
    }
    #4-Byte Sequenzen entsprechen; hier nicht behandelt
    else
    {
        die "Unicode values in this range not supported $_";
    }
}
```

Unicode-Kodepunkt-nach UTF-8 (C++)

```
int convert_unicode_to_utf8(unsigned char *output_string,int max,int input_char)
{
    if(input_char < 128) {
        *output_string = input_char;
        output_string++;
        return 1;
    }
    else if(input_char < 2048 && max > 1) {
        //ending 6 bits will be written into second byte; set first two leading bits to 1
        *output_string = (unsigned char)192|input_char>>6;
        output_string++;
        //only take six last bits; set leading bit to 1; second bit to 0
        //~64 is 10111111; bitwise 'and' with this clears second bit
        *output_string = ((unsigned char)128|input_char)&(~64);
        output_string++;
        return 2; //number of bytes written
    }
    else if(input_char < 65536 && max > 2) {
        //set first 3 leading bits to 1
        *output_string = (unsigned char)224|input_char>>12;
        output_string++;
        //only take six last bits; set leading bit to 1; second bit to 0
        *output_string = ((unsigned char)128|input_char>>6)&(~64);
        output_string++;
        //only take six last bits; set leading bit to 1; second bit to 0
        *output_string = ((unsigned char)128|input_char)&(~64);
        output_string++;
        return 3; //number of bytes written
    }
    else if(input_char < 2097152 && max > 3) {
        //set first 4 leading bits to 1
        *output_string = (unsigned char)240|input_char>>18;
        output_string++;
        //only take six last bits; set leading bit to 1; second bit to 0
        *output_string = ((unsigned char)128|input_char>>12)&(~64);
        output_string++;
        //only take six last bits; set leading bit to 1; second bit to 0
        *output_string = ((unsigned char)128|input_char>>6)&(~64);
        output_string++;
        //only take six last bits; set leading bit to 1; second bit to 0
        *output_string = ((unsigned char)128|input_char)&(~64);
        output_string++;
        return 4; //number of bytes written
    }
    else
        return 0; //certainly not a valid unicode character
    }
```

Weitere Unicode-Kodierungen

Weitere Unicode-Kodierungen

UCS-2, UTF-32, UTF-16

UCS-2, UTF-32 (UCS-4)

UCS-2 (veraltet), UTF-32 ($\approx$ UCS-4)

Diese Zeichensatzkodierungen kodieren Unicode als Bytesequenzen fester Länge:

UCS-2 auf 2 Byte, max 2^{16} Zeichen, deshalb veraltet

- ersetzt durch UTF-16, s.u.

UTF-32 auf 4 Byte

Weitere Zeichensatzkodierungen für Unicode

UTF-16 – variable Bytelänge (2 oder 4 Byte)

UTF-16 (Erweiterung für UCS-2): kodiert nur Zeichen > 65535 as 4-Byte-Sequenzen; für alle anderen Zeichen, UCS-2-compatible, 2 Bytes pro Zeichen).

UTF-16 repräsentiert Buchstaben über UxFFFF (65535) als Surrogatpaare (surrogate pairs) aus dem Bereiche xD800-xDFFF (die in Unicode keinem Zeichen zugewiesen sind)

Endianness

Kodierungsproblem bei UTF-16, UTF-32 und UCS-4

Unterschiedliche Rechnerarchitekturen kodieren mehr-Byte-Zahlen unterschiedlich:

- **Little Endian:** Byte mit hohem Wert zuerst; Zahl eins auf zwei Byte kodiert als:

00000001 00000000

- **Big Endian:** Byte mit niedrigem Wert zuerst; Zahl eins:

00000000 00000001

- Daher wird bei USC-32, UCS-2 und UTF-16 (s.u.) am Anfang der Datei der Buchstabe FEFF (nicht sichtbarer Leerschritt) eingesetzt (Byte Order Mark oder BOM). Aus dem Umstand, ob er als FE-FF oder FF-FE erscheint, lässt sich die Endianness (Bytereihenfolge) erkennen.
- Alternative: Angabe der Zeichensatzkodierung als LE/BE, z.B. UTF-32LE, UTF-32BE
- dies ist nur beim Einlesen/Auslesen/Übermitteln von Binärdaten zwischen Rechnern relevant

Andere Mehrbyte-Kodierungen

Andere Mehrbytekodierungen

Neben Unicode gibts es noch andere Zeichensätze, die mehr als 256 Zeichen enthalten, und die daher nicht in einem Byte kodiert werden können. Die bekanntesten Zeichensätze dieser Art sind die Zeichensätze für Chinesisch, Japanisch und Koreanisch (engl. auch CJK).

ASCII-kompatible Zeichensatzkodierungen

Dies meisten CJK-Zeichensatzkodierungen mit variabler Länge sind ASCII-kompatibel

- GB-1280, GBK, GB 18300 (Chinesisch)

- EUC-JP, EUC-KR (Japanisch/Koreanisch)

(funktionieren nach ähnlichen Prinzipien wie UTF-8 für Unicode)

Nicht Ascii-kompatible Mehrbyte-Zeichensatzkodierungen

Es gibt einige Multibyte-Zeichensatzkodierungen, die nicht ASCII-kompatibel sind.

a) Shift-Jis (Japanisch. Nur erstes Byte einer Folge muss ≥ 128 sein)

日本語、半角、Shift-JIS

インターネットで受け取ったメールが文字化けしていて、「半角カナを使っているから」といわれたことはありませんか？ あるいは送信コードを「Shift JISではなくJISにしてください」と注意された経験は？

b) Zeichensatzkodierungen, die ausschließlich im ASCII-Bereich arbeiten (7-Bit) und mit Escape-Sequenzen arbeiten, um zwischen ASCII und anderen Zeichensätzen zu wechseln)

ISO-2022-JP (Escape-Sequenzen in Rot)

```
<LI>$BI,MW$J%G!<%?$,>C5n$5$1$  
k (J)  
  
<LI>$BF|IU@_Dj$,$G$-  
$J$/$J$k (J)
```

- ISO-2022-(JP/KR/CN)

Andere Eigenschaften

Zusätzliche Eigenschaften von Zeichensatzkodierungen:

- Behandlung der Schriftrichtung (v.a. Hebräisch)

Hebräische Schriftrichtung

Visuelle vs. logisches Hebräisch (visual/logical)

- Hebräisch wird von rechts nach links geschrieben
- Die sogenannte visuelle Zeichensatzkodierung für das Hebräische (iso-8859-6) verwendet die falsche Byte-Reihenfolge – der erste Buchstabe der Bytesequenz, die eine Zeile repräsentiert ist das letzte Byte des Textes
- die kommt daher, das Browser früher den Text nicht von rechts nach links darstellen konnten; daher drehte man den Text einfach um
- funktioniert nur zeilenweise!!!
- zur Weiterverarbeitung und Konversion nach Unicode muss die Zeile umgedreht werden
- Für Arabisch (ebenfall rechts-links) gibt es keine visuelle Kodierung, da es auf frühen Browsern sowieso nicht dargestellt werden konnte.

א	05D0
ב	05D1
ג	05D2
ד	05D3
ה	05D4

הלכות תלמוד תורה

Schreibrichtung

Kodierungskonversion

Werkzeuge zur Kodierungskonversion:

- iconv (Programmbibliothek & ausführbares Programm)
Bsp: `iconv -f iso-8859-1 -t utf-8 < eingabedatei > ausgabedatei`
- recode (ausführbares Programm)
- Word: Abspeichern von Dateien als Text in beliebigem Zeichensatz (zur gelegentlichen Konversion; nicht zur Automatisierung geeignet)
- Eigenes Programm auf Basis einer Skriptsprache (s. nächste Folie)

Kodierungskonversion II

Kodierungskonversion in Programmiersprachen:

Unterstützung von Zeichensatzkodierung:

Beim Lesen und Abspeichern von Dateien

Konversion von Zeichenketten

Perl: Seit Perl 5.8. : Perl Encode

Python: codecs module

ICU-library <http://site.icu-project.org/> für C++ und Java