2. Codierung/Codes
Codesicherung (EDC/ECC)
Daten-Codierung
Zeichencodierung

- Gray-Code, Tetradencodes, Zyklische Codes
- Hammingabstand, Parität, Lineare Codes, CRC
- Daten: ASN.1, BER, TLV, AVP, JSON
- Zeichencodes: ASCII, UCS, UNICODE, UTF-8, UTF-16, BASE-64

siehe auch: http://de.wikipedia.org/wiki/xxx

Tetraden Codes

Codes	Hexcode	Aiken	3-Exzeß	BCD	Gray	Glixon	White	Petherick
Stellenwert	(keine)	2421	(keine)	8421	(keine)	(keine)	(keine)	(keine)
0	0000	0=0000	PT	0=0000	0=0000	0000	0000	0101
1	0001	1=0001	PT	1=0001	1=0001	0001	0001	0001
2	0010	2=0010	PT	2=0010	3=0010	0011	0011	0011
3	0011	3=0011	0=0011	3=0011	2=0011	0010	0101	0010
4	0100	4=0100	1=0100	4=0100	7=0100	0110	0111	0110
5	0101	PT	2=0101	5=0101	6=0101	0111	1000	1110
6	0110	PT	3=0110	6=0110	4=0110	1010	1001	1010
7	0111	PT	4=0111	7=0111	5=0111	1101	1011	1011
8	1000	PT	5=1000	8=1000	PT	1100	1101	1001
9	1001	PT	6=1001	9=1001	PT	1000	1111	1101
A	1010	PT	7=1010	PT	PT	PT	PT	PT
B	1011	5=1011	8=1011	PT	PT	PT	PT	PT
C	1100	6=1100	9=1100	PT	8=1100	PT	PT	PT
D	1101	7=1101	PT	PT	9=1101	PT	PT	PT
E	1110	8=1110	PT	PT	PT	PT	PT	PT
F	1111	9=1111	PT	PT	PT	PT	PT	PT

Codierung – Gray Codes

Gray-Code = Code bei dem sich jeweils beim Übergang von einem Codewort zum folgenden nur genau eine Bitstelle ändert

Konstruktion:

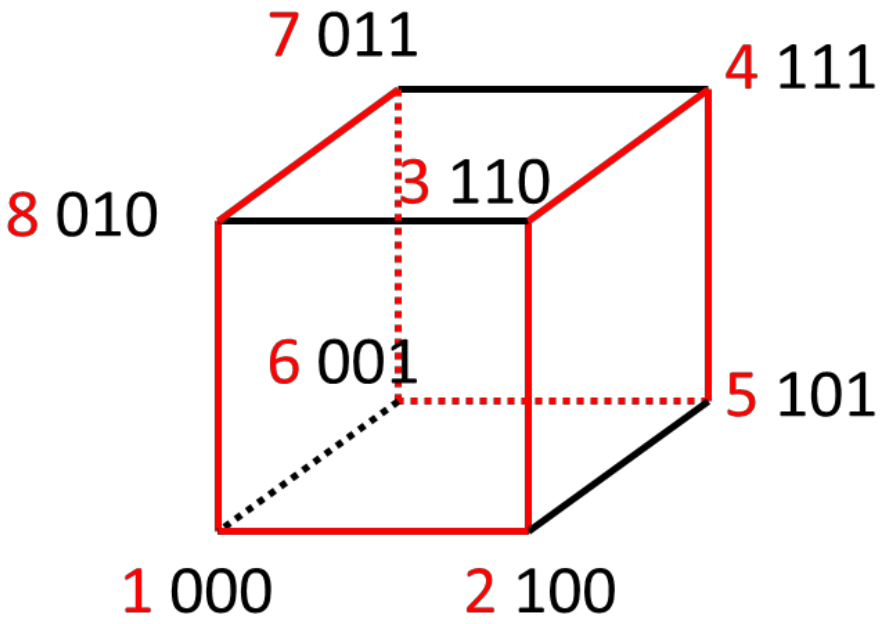
n=1:
0
1

n=2:
00
01
11
10

n=3:
000
001
011
010
110
111
101
100

Codierung – Gray Codes

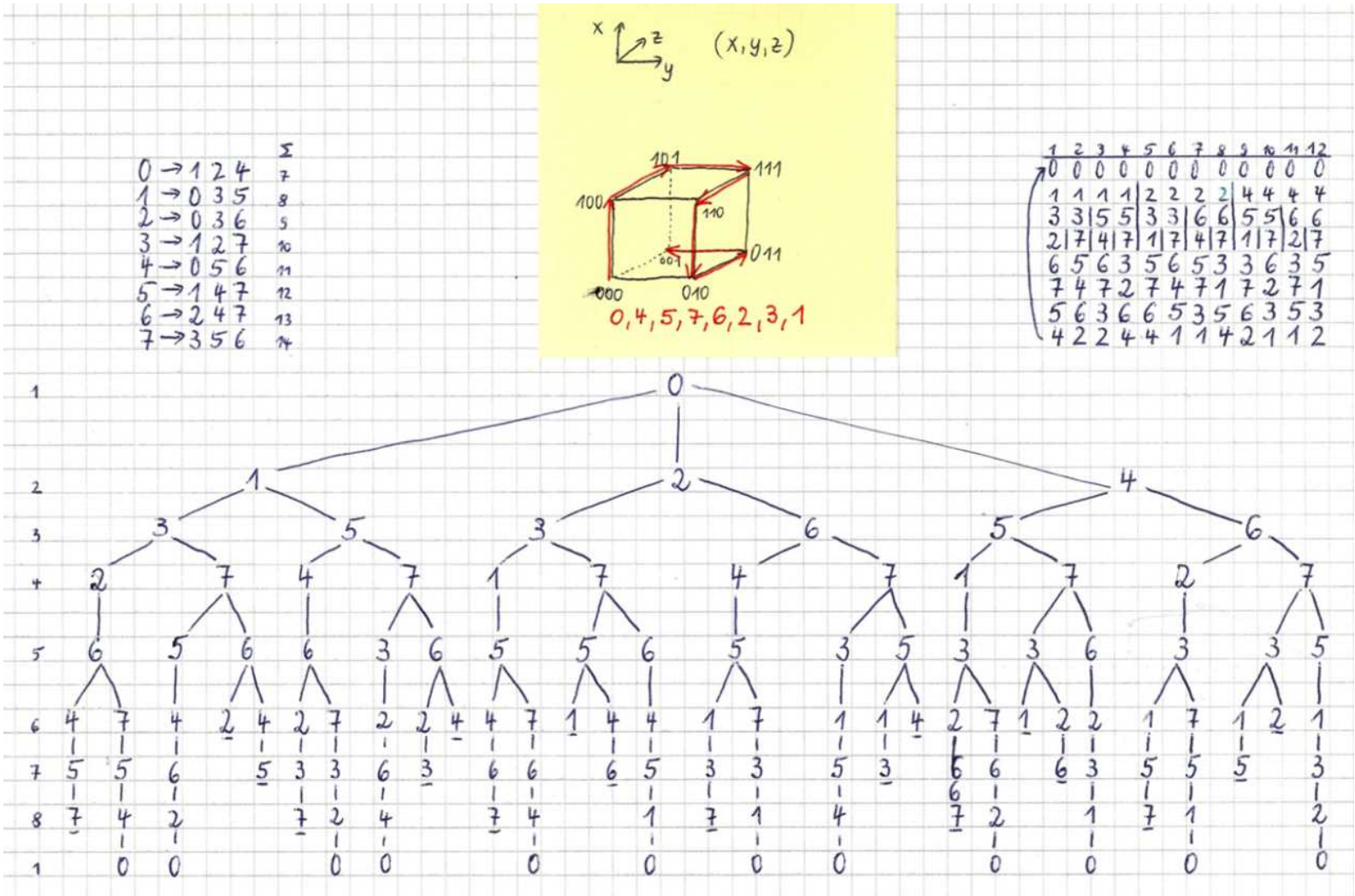
Gray-Codes für n=3 (Bits)



Eck-Nachbarn:

0 ➔ {1,2,4}
1 ➔ {0,3,5}
2 ➔ {0,3,6}
3 ➔ {1,2,7}
4 ➔ {0,5,6}
5 ➔ {1,4,7}
6 ➔ {2,4,7}
7 ➔ {3,5,6}

Codierung – Gray Codes



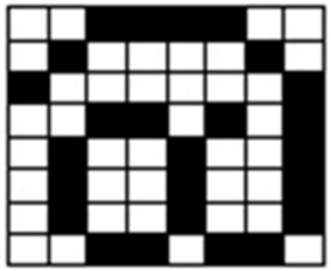
Codierung – Gray Codes

0 000	0 000	0 000	0 000	0 000	0 000	0 000	0 000	0 000	0 000	0 000	0 000
1 001	1 001	1 001	1 001	2 010	2 010	2 010	2 010	4 100	4 100	4 100	4 100
3 011	3 011	5 101	5 101	3 011	3 011	6 110	6 110	5 101	5 101	6 110	6 110
2 010	7 111	4 100	7 111	1 001	7 111	4 100	7 110	1 001	7 111	2 010	7 111
6 110	5 101	6 110	3 011	5 101	6 110	5 101	3 011	3 011	6 110	3 011	5 101
7 111	4 100	7 111	2 010	7 111	4 100	7 111	1 001	7 111	2 010	7 111	1 001
5 101	6 110	3 101	6 110	6 110	5 101	3 011	5 101	6 110	3 011	5 101	3 011
4 100	2 010	2 010	4 100	4 100	1 010	1 001	4 100	2 010	1 001	1 001	2 010
0 000	0 000	0 000	0 000	0 000	0 000	0 000	0 000	0 000	0 000	0 000	0 000

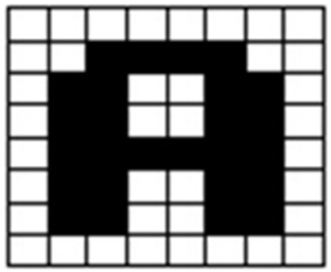
12 verschiedene Gray-Codes für n=3 (Bits)

Codierung

8x8 Pixelzeichensätze

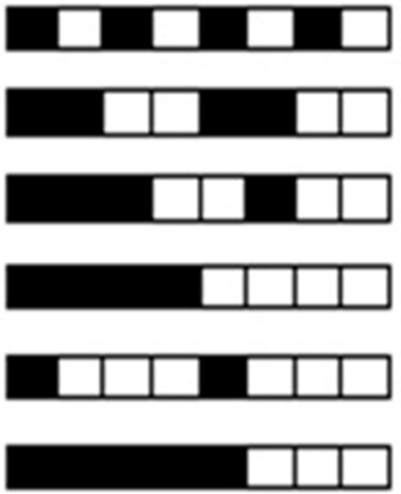


00111100	3C
01000010	42
10000001	81
00110101	35
01001001	49
01001001	49
01001001	49
00110110	36

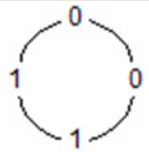
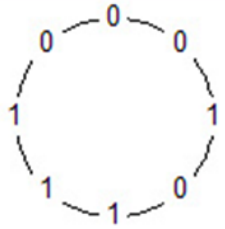
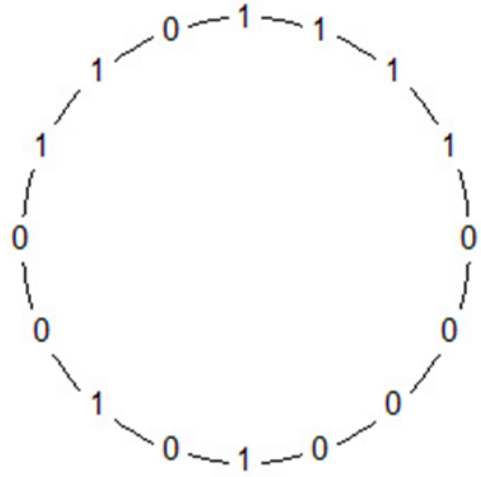


00
3C
66
66
7E
66
66
00

Linienarten



Codierung

(a) Gedächtnisrad für Pärchen		00 01 11 10	0 1 3 2
(b) Gedächtnisrad für Drillinge		001 010 101 011 111 110 100 000	1 2 5 3 7 6 4 0
(c) Gedächtnisrad für Vierlinge		1111 1110 1100 1000 0000 0001 0010 0101 1010 0100 1001 0011 0110 1101 1011 0111	15 14 12 8 0 1 2 5 10 4 9 3 6 13 11 7

Gedächtnisräder

Codierung

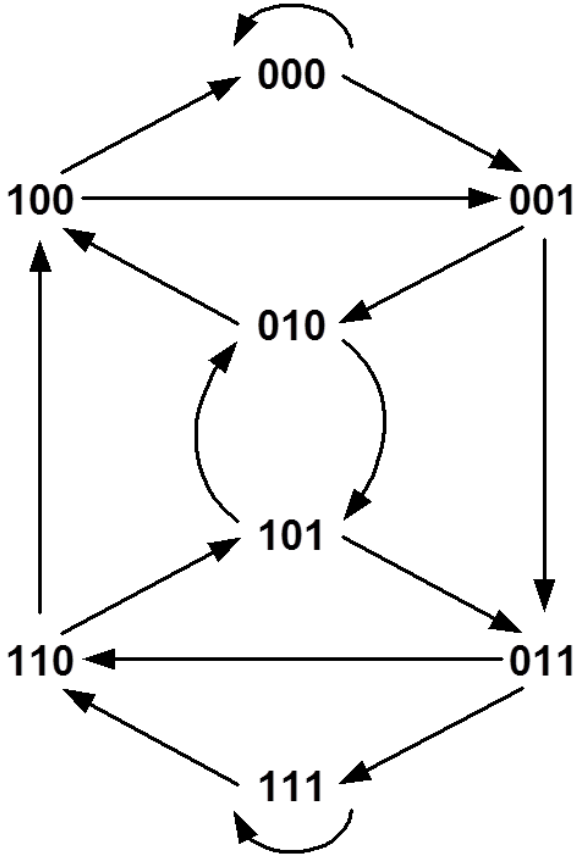
256 Gedächtnisräder für n=4

```
  1    2480   Bitfolge=1111011001010000 Wert=63056   OK!   15  14  13  11  6  12  9  2  5  10  4  8  0  1  3  7
  2    2540   Bitfolge=1111011000010100 Wert=62996   OK!   15  14  13  11  6  12  8  0  1  2  5  10  4  9  3  7
  3    2672   Bitfolge=1111010110010000 Wert=62864   OK!   15  14  13  10  5  11  6  12  9  2  4  8  0  1  3  7
  4    2684   Bitfolge=1111010110000100 Wert=62852   OK!   15  14  13  10  5  11  6  12  8  0  1  2  4  9  3  7
  5    2878   Bitfolge=1111010011000010 Wert=62658   OK!   15  14  13  10  4  9  3  6  12  8  0  1  2  5  11  7
  6    2896   Bitfolge=1111010010110000 Wert=62640   OK!   15  14  13  10  4  9  2  5  11  6  12  8  0  1  3  7
  7    3022   Bitfolge=1111010000110010 Wert=62514   OK!   15  14  13  10  4  8  0  1  3  6  12  9  2  5  11  7
  8    3028   Bitfolge=1111010000101100 Wert=62508   OK!   15  14  13  10  4  8  0  1  2  5  11  6  12  9  3  7
  9    3262   Bitfolge=1111001101000010 Wert=62274   OK!   15  14  12  9  3  6  13  10  4  8  0  1  2  5  11  7
 10    3376   Bitfolge=1111001011010000 Wert=62160   OK!   15  14  12  9  2  5  11  6  13  10  4  8  0  1  3  7
 11    3450   Bitfolge=1111001010000110 Wert=62086   OK!   15  14  12  9  2  5  10  4  8  0  1  3  6  13  11  7
 12    3558   Bitfolge=1111001000011010 Wert=61978   OK!   15  14  12  9  2  4  8  0  1  3  6  13  10  5  11  7
  |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
  |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
245   61979   Bitfolge=0000110111100101 Wert= 3557   OK!   0  1  3  6  13  11  7  15  14  12  9  2  5  10  4  8
246   62087   Bitfolge=0000110101111001 Wert= 3449   OK!   0  1  3  6  13  10  5  11  7  15  14  12  9  2  4  8
247   62161   Bitfolge=0000110100101111 Wert= 3375   OK!   0  1  3  6  13  10  4  9  2  5  11  7  15  14  12  8
248   62275   Bitfolge=0000110010111101 Wert= 3261   OK!   0  1  3  6  12  9  2  5  11  7  15  14  13  10  4  8
249   62509   Bitfolge=0000101111010011 Wert= 3027   OK!   0  1  2  5  11  7  15  14  13  10  4  9  3  6  12  8
250   62515   Bitfolge=0000101111001101 Wert= 3021   OK!   0  1  2  5  11  7  15  14  12  9  3  6  13  10  4  8
251   62641   Bitfolge=0000101101001111 Wert= 2895   OK!   0  1  2  5  11  6  13  10  4  9  3  7  15  14  12  8
252   62659   Bitfolge=0000101100111101 Wert= 2877   OK!   0  1  2  5  11  6  12  9  3  7  15  14  13  10  4  8
253   62853   Bitfolge=0000101001111011 Wert= 2683   OK!   0  1  2  5  10  4  9  3  7  15  14  13  11  6  12  8
254   62865   Bitfolge=0000101001101111 Wert= 2671   OK!   0  1  2  5  10  4  9  3  6  13  11  7  15  14  12  8
255   62997   Bitfolge=0000100111101011 Wert= 2539   OK!   0  1  2  4  9  3  7  15  14  13  10  5  11  6  12  8
256   63057   Bitfolge=0000100110101111 Wert= 2479   OK!   0  1  2  4  9  3  6  13  10  5  11  7  15  14  12  8
```

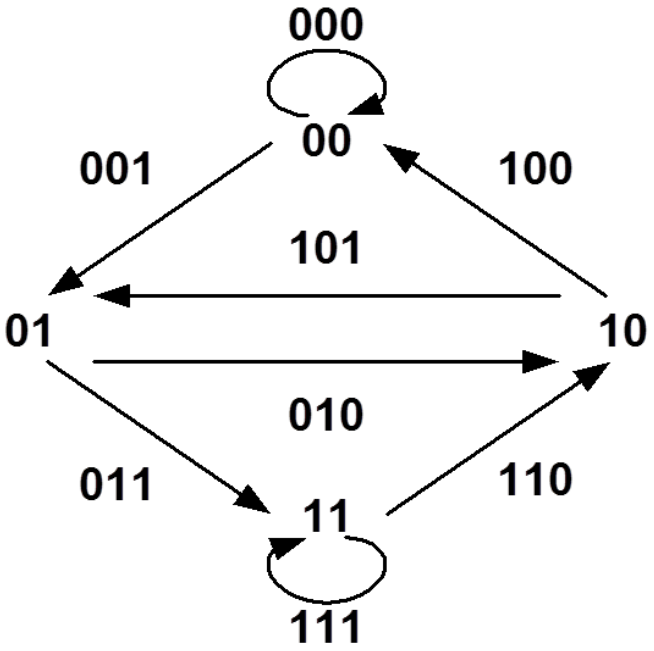
Zyklische Codes

Gedächtnisräder

als Hamilton Graph



als Euler Graph



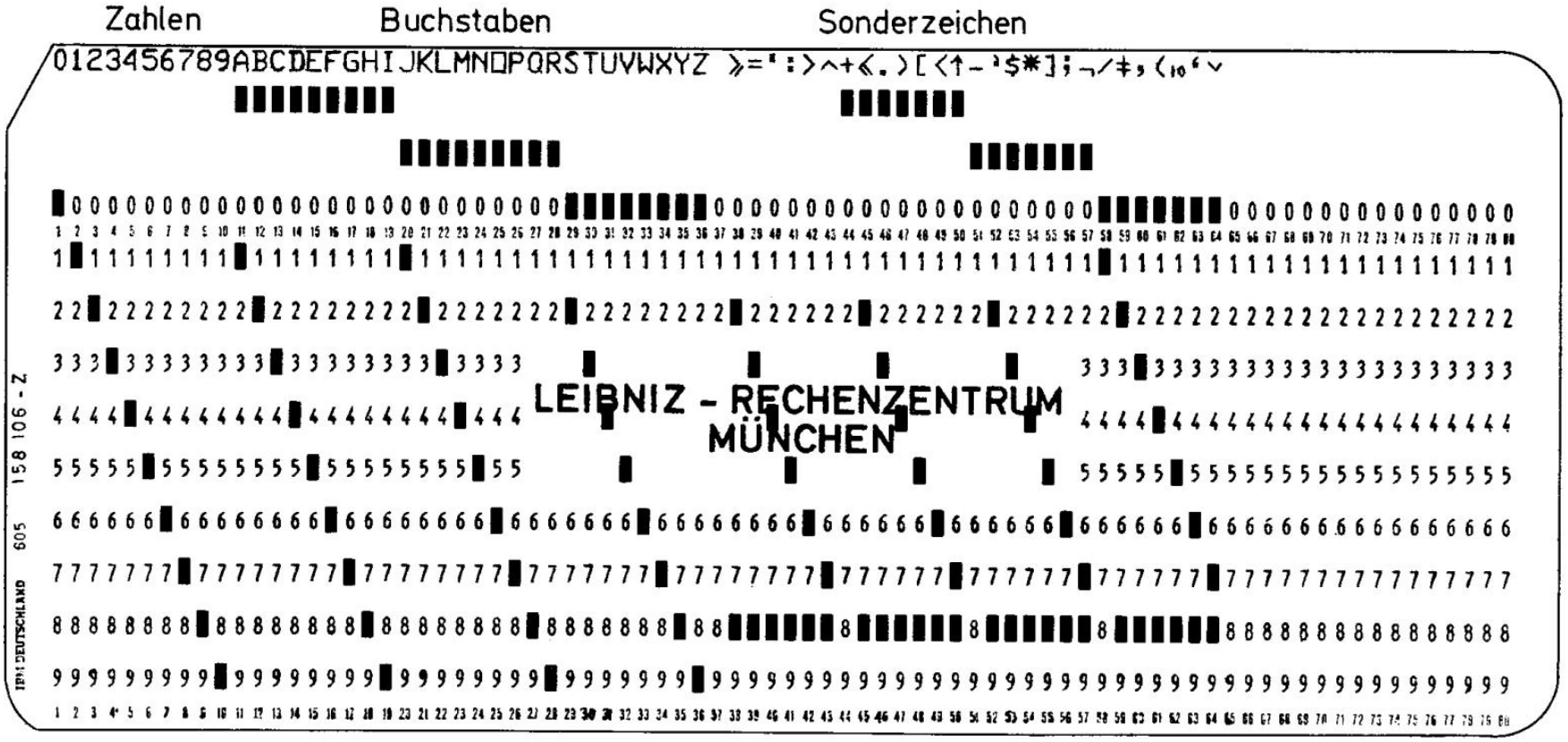
Codierung - Fernschreibcode

5-Kanal-Fernschreibercode: regellose Verschlüsselung (CCITT-2-Code)

26 der 31 Kombinationen (00000 ist nicht belegt) sind jeweils zweimal belegt und mittels des Umschaltzeichens
11111 von Ziffern auf Buchstaben,
und 11011 von Buchstaben auf Ziffern
zu erreichen.

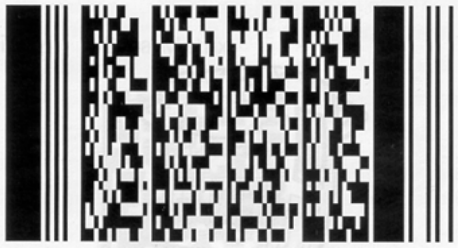
0=01101	1=11101	2=11001	3=10000	4=01010	5=00001	6=10101
7=11100	8=01100	9=00011	+=10001	-=11000	/=10111	.=00111
,=00110	(=11110	)=01001	==01111	'=10100		
A=11000	B=10011	C=01110	D=10010	E=10000	F=10110	G=01011
H=00101	I=01100	J=11010	K=11110	L=01001	M=00111	N=00110
O=00011	P=01101	Q=11101	R=01010	S=10100	T=00001	U=11100
V=01111	W=11001	X=10111	Y=10101	Z=10001		

Wagenrücklauf=00010 Zeilenvorschub=01000 Leerzeichen=00100



Zeichenvorrat des IBM Lochkartencodes (Schreiblocher IBM 029 Mod. A22, Zeichenbelegung DFG)

Codierung – Barcode/Matrixcode



Barcodes

Code 2/5 überlappt (1972/ITF)

CODABAR (1972)

Code 3 of 9 (39) (1974)

EAN-Code / DIN 66236 (1975)

PDF 417

Module 30x12 mils ECL = 2

Data Matrix

Module Size 24 mils ECC 200

Aztec Code

Module Size 28 mils EC = 23%

QR Code

Cell 0,6 mm ECL = 30%

Fehlererkennung
Fehlerkorrektur

Def.:

(binäres) Codewort = Kombination aus 0 und 1

Pseudoworte = unbenutzte Codeworte

Nutzworte = Codes, die einem zu codierenden Zeichen entsprechen

ungesicherte Codes: wenn Anzahl der Codeworte = Anzahl der Nutzworte

(d.h. keine Pseudoworte, keine Redundanz)

d:=Stellendistanz =Anzahl der verschiedenen Binärstellen zwischen 2 Nutzworten

im nachfolgenden Beispiel ist d=2 , i.a. ist d unterschiedlich groß

Def.:

Hammingdistanz h := geringste paarweise Stellendistanz zwischen allen Nutzworten eines Codes (h<=d)

Bsp.:

Die Anzahl der Bitstörungen e, die noch als Fehler erkannt werden können beträgt:

$$e = h - 1$$

Die Zahl k der mit Sicherheit richtig korrigierten Fehler beträgt:

$$k = \text{INT}\,[\,(\,h - 1\,)\,/\,2\,]$$

(falls bei Störung das am nächsten liegende Nutzwort gewählt wird)

d.h. mit ein und demselben Code sind doppelt soviele Fehler erkennbar als korrigierbar.

Falls keine Pseudoworte vorhanden sind, dann ist keine Fehlererkennung möglich, denn jede Bit-Störung führt zu einem anderen Nutzwort.

Paritätsbit

Prüfbit-Methode: Zusatz von einem Bit pro Nutzwort,
ermöglicht die sichere Erkennung eines Fehlers.

d.h. gesamte Quersumme = geradzahlig (even parity check) 1011010
= ungeradzahlig (odd parity check) 0110101

Sender und Empfänger einigen sich auf:

a) even-parity-check: Anzahl der Einsen im erw. Codewort ist gerade (=0Parität)
b) odd-parity-check: Anzahl der Einsen im erw. Codewort ist ungerade (=1Parität)

d.h.

a) EVEN:
Falls die Quersumme des Codewortes gerade => Anhängen des Prüfbits 0
Falls die Quersumme des Codewortes ungerade => Anhängen des Prüfbits 1
b) ODD:
Falls die Quersumme des Codewortes gerade => Anhängen des Prüfbits 1
Falls die Quersumme des Codewortes ungerade => Anhängen des Prüfbits 0

Falls eine bestimmte Zahl von Prüfbits zusammengefaßt werden, dann spricht man von Blockcodes mit einem Paritätsblock.

Man benutzt nur lineare Codes, d.h. Codes die sich mit einfachen Rechenverfahren erzeugen lassen.

Hierzu wird keine komplette Übersetzungsttabelle angegeben, sondern nur die Übersetzung der Datenwerte, die nur eine 1 enthalten, d.h. der Einheitsvektoren (z.B. 000100)

Dazu werden anstatt einer Codetabelle mit 2^n Einträgen nur noch n Zielcodes benötigt.

Die restlichen Quellwörter werden durch Linearkombination der "Einheitscodes" codiert

(d.h. die "logische Summe" zweier Codes stellt wiederum ein zulässiges Codewort dar,

und damit bilden die Codes mit der Operation XOR eine mathematische Gruppe):

Datenwert	Codewort	normalisierte Erzeugermatrix
1000	00000011	1000 0001
0100	00001100	0100 0010
0010	00110010	0010 0101
0001	11011000	0001 1110

Die 4 Codewörter nennt man die Erzeugermatrix.

Der Hammingabstand eines Codes ist invariant gegenüber

a) Bitvertauschung aller Zielcodes

b) Neuzuordnung von Codewörter

Ausserdem Beschränkung auf die "Normalform" der Erzeugermatrix

für 4 Bit lange Daten und 7 Bit lange Codes z.B.

```
1000xxx
0100xxx
0010xxx
0001xxx
```

Diese (permutierte, vertauschte) Normalform hat die gleiche Stabilität gegenüber Bitfehlern wie der Ausgangscode. Ausserdem ist die Dekodierung eines aus der Normalform resultierenden Codes trivial:

Bsp: 1000 1000001

0100 0100011

0010 0010010

0001 0001111

zu codierendes Quellwort: 1101

1 x 1000001

1 x 0100011

0 x

1 x 0001111

1101101

Die ersten 4 Bits entsprechen dem Quellwort, die restlichen 3 Bit könnten als Prüfbits interpretiert werden.

Fehlersuche durch Umformung der Normalform in eine Prüfmatrix :

Prüfmatrix für obiges Beispiel:

```
0001100
0111010
1101001
```

Mit Hilfe dieser Prüfmatrix Fehlererkennung für 1101101:

```
1101101 AND 0001100 = 0001100
1101101 AND 0111010 = 0101000
1101101 AND 1101001 = 1101001
```

In allen 3 Fällen liegt 0-Parität vor, also ist das Codewort korrekt

Codierung – „n TO m-Codes“

Geg.: n Quellbits → 2^n Quellzeichen

Ges.: m Codebits → 2^m Codes
so dass die paarweisen Stellendistanzen zwischen den Codeworten möglichst gleichmäßig groß sind.

Bsp.:
„2 TO 3“: 4 Quellzeichen stehen 8 Codes gegenüber
(1 Bit mehr bedeutet doppelt so viele Codes)

„8 TO 10“: 256 Quellzeichen stehen 1024 Codes gegenüber
(2 Bit mehr bedeutet viermal so viele Codes)

„8 TO 14“: 256 Quellzeichen stehen 16384 Codes gegenüber

Codierung – Lineare Codes

Für lineare Codes benötigt man lediglich eine normalisierte Erzeugermatrix:

Bsp.: „2 TO 5“ - Code

$$\begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \end{pmatrix} \rightarrow \text{Codematrix} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \end{pmatrix}$$

Hieraus resultiert die Matrix der Stellendistanzen zwischen je 2 Codevektoren:

-	**3**	**3**	**4**
	-	4	3
		-	3
			-

Codierung – Lineare Codes

Lineare 3 TO 7-Codes mit Hammingabstand = 4

normalisierte Erzeugermatrix:
Hammingabstand=4

1 0 0 0 1 1 1 0 1 0 1 0 1 1 0 0 1 1 1 0 1	1 0 0 1 0 1 1 0 1 0 0 1 1 1 0 0 1 1 1 0 1	1 0 0 1 1 0 1 0 1 0 0 1 1 1 0 0 1 1 0 1 1	1 0 0 1 1 1 0 0 1 0 0 1 1 1 0 0 1 1 0 1 1
1 0 0 0 1 1 1 0 1 0 1 0 1 1 0 0 1 1 1 1 0	1 0 0 1 0 1 1 0 1 0 0 1 1 1 0 0 1 1 1 1 0	1 0 0 1 1 0 1 0 1 0 0 1 1 1 0 0 1 1 1 1 0	1 0 0 1 1 1 0 0 1 0 0 1 1 1 0 0 1 1 1 0 1
1 0 0 0 1 1 1 0 1 0 1 1 0 1 0 0 1 1 0 1 1	1 0 0 1 0 1 1 0 1 0 1 1 0 1 0 0 1 0 1 1 1	1 0 0 1 1 0 1 0 1 0 1 0 1 1 0 0 1 0 1 1 1	1 0 0 1 1 1 0 0 1 0 1 0 1 1 0 0 1 0 1 1 1
1 0 0 0 1 1 1 0 1 0 1 1 0 1 0 0 1 1 1 1 0	1 0 0 1 0 1 1 0 1 0 1 1 0 1 0 0 1 1 1 1 0	1 0 0 1 1 0 1 0 1 0 1 0 1 1 0 0 1 1 1 1 0	1 0 0 1 1 1 0 0 1 0 1 0 1 1 0 0 1 1 1 0 1
1 0 0 0 1 1 1 0 1 0 1 1 1 0 0 0 1 1 0 1 1	1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 0 1 0 1 1 1	1 0 0 1 1 0 1 0 1 0 1 1 1 0 0 0 1 0 1 1 1	1 0 0 1 1 1 0 0 1 0 1 1 0 1 0 0 1 0 1 1 1
1 0 0 0 1 1 1 0 1 0 1 1 1 0 0 0 1 1 1 0 1	1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 0 1 1 1 0 1	1 0 0 1 1 0 1 0 1 0 1 1 1 0 0 0 1 1 0 1 1	1 0 0 1 1 1 0 0 1 0 1 1 0 1 0 0 1 1 0 1 1

Codierung – Lineare Codes

Hammingabstände der 2^12=4096 linearen Codes:

0: 0
1: 721
2: 2385
3: 966
4: 24
5: 0
6: 0
7: 0

Summe der jeweiligen Stellendistanzen: $48 \leq s \leq 112$

Anzahl der erreichten Stellendistanzsummen:

48 (1x), 64 (28x), 80 (294x), 96 (1372x), 112 (2401x)

Codierung – Lineare Codes

Anzahl der erreichten Stellendistanzsummen:

48: 1	65: 0	82: 0	99: 0
49: 0	66: 0	83: 0	100: 0
50: 0	67: 0	84: 0	101: 0
51: 0	68: 0	85: 0	102: 0
52: 0	69: 0	86: 0	103: 0
53: 0	70: 0	87: 0	104: 0
54: 0	71: 0	88: 0	105: 0
55: 0	72: 0	89: 0	106: 0
56: 0	73: 0	90: 0	107: 0
57: 0	74: 0	91: 0	108: 0
58: 0	75: 0	92: 0	109: 0
59: 0	76: 0	93: 0	110: 0
60: 0	77: 0	94: 0	111: 0
61: 0	78: 0	95: 0	112: 2401
62: 0	79: 0	96: 1372	
63: 0	80: 294	97: 0	
64: 28	81: 0	98: 0	

Codierung – „8 TO 10-Code“

Code Group Name	Octet Value	Octet Bits HGF EDCBA	Current RD – abcdei fghj	Current RD + abcdei fghj
D8.6	C8	110 01000	111001 0110	000110 0110
D9.6	C9	110 01001	100101 0110	100101 0110
D10.6	CA	110 01010	010101 0110	010101 0110
D11.6	CB	110 01011	110100 0110	110100 0110
D12.6	CC	110 01100	001101 0110	001101 0110
D13.6	CD	110 01101	101100 0110	101100 0110
D14.6	CE	110 01110	011100 0110	011100 0110
D15.6	CF	110 01111	010111 0110	101000 0110
D16.6	D0	110 10000	011011 0110	100100 0110
D17.6	D1	110 10001	100011 0110	100011 0110
D18.6	D2	110 10010	010011 0110	010011 0110
D19.6	D3	110 10011	110010 0110	110010 0110
D20.6	D4	110 10100	001011 0110	001011 0110
D21.6	D5	110 10101	101010 0110	101010 0110
D22.6	D6	110 10110	011010 0110	011010 0110
D23.6	D7	110 10111	111010 0110	000101 0110
D24.6	D8	110 11000	110011 0110	001100 0110
D25.6	D9	110 11001	100110 0110	100110 0110
D26.6	DA	110 11010	010110 0110	010110 0110
D27.6	DB	110 11011	110110 0110	001001 0110
D28.6	DC	110 11100	001110 0110	001110 0110
D29.6	DD	110 11101	101110 0110	010001 0110
D30.6	DE	110 11110	011110 0110	100001 0110
D31.6	DF	110 11111	101011 0110	010100 0110
D0.7	E0	111 00000	100111 0001	011000 1110
D1.7	E1	111 00001	011101 0001	100010 1110

Code Group Name	Octet Value	Octet Bits HGF EDCBA	Current RD – abcdei fghj	Current RD + abcdei fghj
D26.7	FA	111 11010	010110 1110	010110 0001
D27.7	FB	111 11011	110110 0001	001001 1110
D28.7	FC	111 11100	001110 1110	001110 0001
D29.7	FD	111 11101	101110 0001	010001 1110
D30.7	FE	111 11110	011110 0001	100001 1110
D31.7	FF	111 11111	101011 0001	010100 1110
K28.0[1]	1C	000 11100	001111 0100	110000 1011
K28.1[1,2]	3C	001 11100	001111 1001	110000 0110
K28.2[1]	5C	010 11100	001111 0101	110000 1010
K28.3[1]	7C	011 11100	001111 0011	110000 1100
K28.4[1]	9C	100 11100	001111 0010	110000 1101
K28.5[2]	BC	101 11100	001111 1010	110000 0101
K28.6[1]	DC	110 11100	001111 0110	110000 1001
K28.7[1,2]	FC	111 11100	001111 1000	110000 0111
K23.7	F7	111 10111	111010 1000	000101 0111
K27.7	FB	111 11011	110110 1000	001001 0111
K29.7	FD	111 11101	101110 1000	010001 0111
K30.7	FE	111 11110	011110 1000	100001 0111

1— Reserved
2— Contains a comma

Codierung – „8 TO 14-Code“

0	00000000	01001000100000
1	00000001	10000100000000
2	00000010	10010000100000
3	00000011	10001000100000
4	00000100	01000100000000
5	00000101	00000100010000
6	00000110	00010000100000
7	00000111	00100100000000
8	00001000	01001001000000
9	00001001	10000001000000
10	00001010	10010001000000
11	00001011	10001001000000
12	00001100	01000001000000
13	00001101	00000001000000
14	00001110	00010001000000
15	00001111	00100001000000
16	00010000	10000000100000
17	00010001	10000010000000
18	00010010	10010010000000
19	00010011	00100000100000
20	00010100	01000010000000
21	00010101	00000010000000
22	00010110	00010010000000
23	00010111	00100010000000
24	00011000	01001000010000
25	00011001	10000000010000
26	00011010	10010000010000
27	00011011	10001000010000
28	00011100	01000000010000
29	00011101	00001000010000
30	00011110	00010000010000
31	00011111	00100000010000
32	00100000	00000000100000
33	00100001	10000100001000
34	00100010	00001000100000
35	00100011	00100100100000
36	00100100	01000100001000
37	00100101	00000100001000
38	00100110	01000000100000
39	00100111	00100100001000
40	00101000	01001001001000
41	00101001	10000001001000
42	00101010	10010001001000
43	00101011	10001001001000
44	00101100	01000001001000
45	00101101	00000001001000
46	00101110	00010001001000
47	00101111	00100001001000
48	00110000	00000100000000
49	00110001	10000010001000
50	00110010	10010010001000
51	00110011	10000100010000
52	00110100	01000010001000
53	00110101	00000010001000
54	00110110	00010010001000
55	00110111	00100010001000
56	00111000	01001000001000
57	00111001	10000000001000
58	00111010	10010000001000
59	00111011	10001000001000
60	00111100	01000000001000
61	00111101	00001000001000
62	00111110	00010000001000
63	00111111	00100000001000
64	01000000	01001000100100
190	10111110	00010000001001
191	10111111	00100000001001
192	11000000	01000100100000
193	11000001	10000100010001
194	11000010	10010010010000
195	11000011	00001000100100
196	11000100	01000100010001
197	11000101	00000100010001
198	11000110	00010010010000
199	11000111	00100100010001
200	11001000	00001001000001
201	11001001	10000100000001
202	11001010	00001001000100
203	11001011	00001001000000
204	11001100	01000100000001
205	11001101	00000100000001
206	11001110	00000010010000
207	11001111	00100100000001
208	11010000	00000100100100
209	11010001	10000010010001
210	11010010	10010010010001
211	11010011	10000100100000
212	11010100	01000010010001
213	11010101	00000010010001
214	11010110	00010010010001
215	11010111	00100010010001
216	11011000	01001000010001
217	11011001	10000000010001
218	11011010	10010000010001
219	11011011	10001000010001
220	11011100	01000000010001
221	11011101	00001000010001
222	11011110	00010000010001
223	11011111	00100000010001
224	11100000	01000100000010
225	11100001	00000100000010
226	11100010	10000100010010
227	11100011	00100100000010
228	11100100	01000100010010
229	11100101	00000100010010
230	11100110	01000000100010
231	11100111	00100100010010
232	11101000	10000100000010
233	11101001	10000100000100
234	11101010	00001001001001
235	11101011	00001001000010
236	11101100	01000100000100
237	11101101	00000100000100
238	11101110	00010000100010
239	11101111	00100100000100
240	11110000	00000100100010
241	11110001	10000010010010
242	11110010	10010010010010
243	11110011	00001000100010
244	11110100	01000010010010
245	11110101	00000010010010
246	11110110	00010010010010
247	11110111	00100010010010
248	11111000	01001000010010
249	11111001	10000000010010
250	11111010	10010000010010
251	11111011	10001000010010
252	11111100	01000000010010
253	11111101	00001000010010
254	11111110	00010000010010
255	11111111	00100000010010

Codierung – „8 TO 14-Code“

Eight-to-Fourteen-Modulation (EFM)
https://de.wikipedia.org/wiki/Eight-to-Fourteen-Modulation
http://www.physics.udel.edu/~watson/scen103/efm.html
http://www.laesieworks.com/digicom/Storage_CD_8to14.html
• ECMA-130 = CD-ROM im Yellow Book - 2nd Edition - June 1996
http://www.ecma-international.org/publications/standards/Ecma-130.htm
• EN 60908 = CD-Audio im Red Book
• EFM-Modulation deutsches Patent DE 3119529 (inzwischen erloschen)
• 1 = Übergang von Pit zu Land oder von Land zu Pit
• zwischen 2 Einsen mindestens 2 und maximal 10 Nullen (d/k-Bedingung)
• damit nur 1/3 lange Pits und Lands nötig (aber 75% mehr Bits)
• zwischen Codewörtern 3 Trennbits (000, 001, 010 oder 100)

Eight-to-Fourteen-Modulation-plus (EFMplus)
• auf DVDs benutzte Modulation des Datenstroms (aber es ist eine 8TO16-Modulation)
• ECMA-267 ist im Annex G „8-to-16 Modulation with RLL (2,10) requirements“ und im Kapitel 22 „Suppress control of the d.c. component“ beschrieben
• EFMplus Modulation: Europäischen Patente EP 0 745 254 und EP 0 789 910

CRC = Cyclic Redundancy Check

Falls diese linearen Codes zu aufwendig sind (für sehr lange Datenwerte), dann benutzt man zyklische Codes (CRC).

Idee: Vorgabe nur der Zielcodes für 100...0, 0100...0, 0000...10, 000...01 mit einem codeerzeugenden Bitmuster z.B. 1011 (Verschiebung an die entsprechende Bitposition)

Mit CRC-32 lassen sich

a) in einem Datenwort mit der Wortlänge bis 2^{32}Bit= 0,5GB zwei beliebige Fehler erkennen und ein Bitfehler korrigieren

b) Fehlerbündel (="Bursts"=bis zu 32 hintereinander auftretende Bit-Fehler) erkennen

(z.B. elektromagnetische Einstreuungen in Ethernetkabel)

Diese Burst-Sicherheit beruht darauf, dass jedes einzelne Datenbit sich im Codewort auf Stellen auswirkt,

die 32 Bits voneinander entfernt liegen.

CRC = Cyclic Redundancy Check

Nachrichtenpolynom: $N(x) = y_0x^{n-1} + y_1x^{n-2} + \ldots + y_{n-2}x + y_{n-1}$ (n-Bits)

Generatorpolynom: $G(x) = g_0x^k + g_1x^{k-1} + \ldots + g_{k-1}x + g_k$

Restpolynom: $R(x) = r_0x^{k-1} + r_1x^{k-2} + \ldots + r_{k-1}x + r_k$

mit $R(x) = N(x)*x^k \bmod G(x)$ und $(N(x)*x^k + R(x)) \bmod G(x) = 0$

Codewortpolynom: $C(x) = c_0x^{n+k-2} + c_1x^{n+k-3} + \ldots + c_{n+k-3}x + c_{n+k-2}$

mit $y_i , g_i , r_i , c_i \in \{0,1\}$ für i=0, 1, 2, …

CRC = Cyclic Redundancy Check

1 1 0 1 0 0 0 : 1 0 1 1 =

GenPolynom:='G(x)=x^3+x+1';
GenPolynom:='G(x)=$x^8+x^4+x^2+x+1$';
GenPolynom:='G(x)=x^8+1, (CRC-LRC-Code)';
GenPolynom:='G(x)=$x^{16}+x^{15}+x^2+1$ (CRC-16 for WAN)';
GenPolynom:='G(x)=$x^{16}+x^{12}+x^5+1$ (CRC-CCITT for WAN)';
GenPolynom:='G(x)=$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$ (CRC-32 for LAN)';

Codegruppe für 1011 ≙ $G(x) = x^3 + x + 1$ $(m=4,\ k=3,\ n=m+k=7)$

Nullpolynom $C_0(x) = 0$

Basis-Codeworte: $C_1(x) = x^6 + x^4 + x^3$ ≙ 1011000
$C_2(x) = x^5 + x^3 + x^2$ ≙ 0101100
$C_3(x) = x^4 + x^2 + x$ ≙ 0010110
$C_4(x) = x^3 + x + 1$ ≙ 0001011

$C_1(x)+C_2(x) = C_5(x)$	=	≙ 1110100
$C_1(x)+C_3(x) = C_6(x)$	=	≙ 1001110
$C_1(x)+C_4(x) = C_7(x)$	=	≙ 1010011
$C_2(x)+C_3(x) = C_8(x)$	=	≙ 0111010
$C_2(x)+C_4(x) = C_9(x)$	=	≙ 0100111
$C_3(x)+C_4(x) = C_{10}(x)$	=	≙ 0011101
$C_1(x)+C_2(x)+C_3(x) = C_{11}(x)$	=	≙ 1100010
$C_1(x)+C_2(x)+C_4(x) = C_{12}(x)$	=	≙ 1111111
$C_1(x)+C_3(x)+C_4(x) = C_{13}(x)$	=	≙ 1000101
$C_2(x)+C_3(x)+C_4(x) = C_{14}(x)$	=	≙ 0110001
$G(x)+C_2(x)+C_3(x)+C_4(x) = C_{15}(y)$	=	≙ 1101001

Stellendistanzen

x6	x5	x4	x3	x2	x1	x0	
3	3	3	3	4	4	4	4
4	4	4	3	3	3	3	7
3	4	4	4	3	3	3	3
4	3	4	4	4	3	3	3
3	4	3	4	4	4	3	3
4	3	4	3	4	4	4	3
3	4	3	4	3	4	4	4
4	3	4	3	4	3	4	4
3	4	3	4	3	4	3	4
4	3	4	3	4	3	4	3
3	4	3	4	3	4	3	4
4	3	4	3	4	3	4	3
3	4	3	4	3	4	3	4
4	3	4	3	4	3	4	3

$\cdot G(x)$

$(x^3+x^2+x^1+x^0)$	$\cdot (x^3+x+1)$
1000	1011
0100	1011
0010	1011
0001	1011
1100	1011
1010	1011
1001	1011
0110	1011
0101	1011
0011	1011
1110	1011
1101	1011
1011	1011
0111	1011
1111	1011

Fazit: Jedes gültige Codewort C(x) ist ohne Rest durch das Generatorpolynom G(x) teilbar.

Ein Generatorpolynom ist als Teiler von (x^n-1) und als irreduzibles Polynom zu wählen, d.h. es läßt sich nicht als Produkt zweier Polynome darstellen. ← Kreisteilung(sgleichung) $x^n = 1$ bzw. $x^n - 1 = 0$

Bsp: (x^7-1) besitzt 3 irreduzible Teiler: $(x+1)$, (x^3+x+1), (x^3+x^2+1)

$(x^7-1) : (x^3+x+1) = x^4+x^2+x+1$
$\oplus\ (x^7+x^5+x^4)$
(x^5+x^4+1)
$x^5+x^3+x^2$
$x^4+x^3+x^2+1$
$x^4 \quad +x^2+x$
x^3+x+1
Rest 0

d.h. (1011) ist Teiler von (10000001)

$(x^3+x+1) : (x+1) = x^2+x$
x^3+x^2
x^2+x+1
x^2+x
Rest 1
1011 : 11 = 110
11
11
00
1 = Rest ≠ 0

$(x^3+x+1) : (x^2+x) = x+1$
x^3+x^2
x^2+x+1
x^2+x
Rest 1
1011 : 110 = 11
110
111
110
01 = Rest ≠ 0

$(x^3+x+1) : (x^2+1) = x$
x^3+x
Rest 1
1011 : 101 = 10
101
010
000
11 = Rest ≠ 0

$(x^3+x+1) : (x^2+x+1) = x+1$
(x^3+x^2+x)
(x^2+1)
x^2+x+1
Rest x
1011 : 111 = 11
111
100
111
10 = Rest ≠ 0

also weder durch $x = 10$
$x+1 = 11$
$x^2 = 100$
$x^2+1 = 101$
$x^2+x = 110$
noch durch $x^2+x+1 = 111$ ohne Rest teilbar!

CRC-Algorithmus:

(z.B. für FCS=Frame Check Sequence - Berechnung nach IEEE802.3)

1. Die ersten 32 Bit der Quellbits werden invertiert.
2. Die n Bits des Frames werden als Koeffizienten des Polynoms M(x) des Grades n-1 betrachtet, d.h. das erste Bit der Destination-Address entspricht dem Term $x^{(n-1)}$ und das letzte Bit des Datenfeldes entspricht dem Term x^0 .
3. M(x) wird mit x^{32} multipliziert und durch G(x) dividiert, so daß ein Rest R(x) des Grades 31 verbleibt.
4. Die Koeffizienten von R(x) werden als 32-Bit-Sequenz angesehen, so daß eventuelle führende Leerstellen mit Nullen aufgefüllt werden. Somit entspricht das äußerste rechte Bit des Restes R(x) einem Bit der Wertigkeit 2^0.
5. Die 32-Bit-Sequenz wird invertiert. Das Ergebnis ist der CRC.

Daten

Datenelemente, Datendefinitionen,

Transfersyntax, einfache & strukturierte Daten

Datenhierarchie

einfache Datentypen (Boolean, ganze Zahlen/Integer, Gleitkommazahlen)

strukturierte Datentypen (RECORD/struct, SET, ARRAY, ENUM/enum)

ADT = abstrakte Datentypen (Liste, FCFS-Queue, LIFO-Stack, Baum, Hash, RingBuffer)

BER (Basic Encoding Rules)

ITU-T X.690 / ISO 8825-1 & X.892, X.694 (Mapping XML ⟶ ASN.1)

ASN.1 (Abstract Syntax Notation Number One): Standard zur Definition und Kodierung (Transfer Syntax) abstrakter Information

BER ⟶ CER (Canonical Encoding Rules)
BER ⟶ DER (Distinguised Encoding Rules)

BER = Kodierungsformat im ASN.1 Standard &

Beschreibung eines selbsterklärendes Formats zur Kodierung von Datenstrukturen

BER stellen eine Transfer Syntax in ASN.1 dar:

a) definieren die exakte Oktett Folge im Code (Data Stream)

b) Darstellung der Basis-Datentypen (primitives)

c) Struktur von Längeninformationen

d) Arten der Kompositionen von Basis-Datentypen zu komplexeren Datentypen

Datenelement = TypID | Längenbeschreibung | Daten | Ende Marke (falls notwendig)

= TLV = Typ|Length|Value

ASN.1 = Abstract Syntax Notation Number One (ITU-T X.690)

formale Definition einer abstrakten Syntax,
d.h. eine Metasprache zur Definition einer formalen Sprache:

a) mittels BNF-Grammatik (Backus-Naur Form) - Erweiterungen EBNF/ABNF

b) Syntaxdiagramme

c) OSI-Normierung ASN.1

Dienstelemente von ASN.1

Dienstelementtyp	Bedeutung
INTEGER	ganze Zahl beliebiger Länge
BOOLEAN	TRUE oder FALSE
BIT STRING	Liste mit 0 oder mehr Bits
OCTET STRING	Liste mit 0 oder mehr Bytes
ANY	Vereinigung aller Typen
NULL	keiner der Typen
OBJECT IDENTIFIER	Objektname

Hauptbestandteile von ASN.1

Baustein	Bedeutung
SEQUENCE	geordnete Liste mit verschiedenen Typen
SEQUENCE OF	geordnete Liste mit nur einem Typ
SET	ungeordnete Sammlung mit verschiedenen Typen
SET OF	ungeordnete Sammlung mit nur einem Typ
CHOICE	beliebiger Typ aus einer gegebenen Liste

Identifizierungskennzeichen (Tags) von ASN.1

Datentyp	Aufbau	Definition	Gültigkeit
UNIVERSAL	elementar, zusammen-gesetzt	vordefiniert	Internationaler Standard
APPLICATION	zusammen-gesetzt	selbstdefiniert	eine Anwend-ung, mehrere Organisationen
PRIVATE	zusammen-gesetzt	selbstdefiniert	eine Organisation

Bsp.:

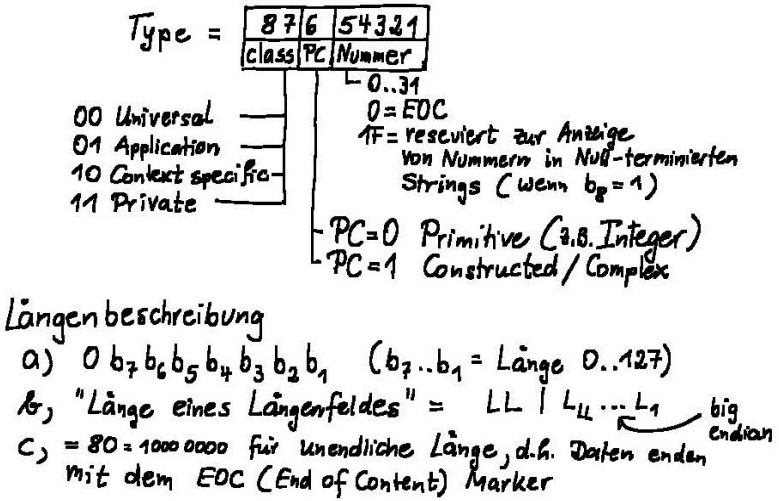
```
TestPDU ::= SEQUENCE {
                aNumber INTEGER,
                anotherNumber INTEGER,
                today UTCTime OPTIONAL,
                myText CHOICE {
                        multilingualText  ISO10646String,
                        standardText      VisibleString
                            }
                    }
```

Bsp.datensatz: {7,42,2018-04-26 07:30:11,"Hallo!"}

TLV (Typ, Length, Value)

Mittels eines TLV-Triples lassen sich (einfache oder kombinierte/komplexe) Datenelemente beschreiben/definieren. Hierbei steht T für eine Datentypangabe, L für eine Längenbeschreibung und V für den zu übertragenden/speichernden Datenwert.

Diese Art der Datencodierung eignet sich vor allem für binäre Daten (z.B. Optionen im TCP und IP-Protokollheader).



Nummer	Datentyp	P/C	Nummer	Datentyp	P/C
0	EOC = End of Content	0	10	SEQUENCE OF	1
1	BOOLEAN	0	11	SET OF	1
2	INTEGER	0	12	NumericString	0/1
3	BIT STRING	0/1	13	PrintableString	0/1
4	OCTET STRING	0/1	14	T61String	0/1
5	NULL	0	15	VideotexString	0/1
6	OBJECT IDENTIFIER	0	16	IA5String	0/1
7	OBJECT DESCRIPTOR	0	17	UTCTime	0/1
8	EXTERNAL	1	18	GeneralizedTime	0/1
9	REAL / FLOAT	0	19	GraphicString	0/1
A	ENUMERATED	0	1A	VisibleString	0/1
B	EMBEDDED PDV	1	1B	GeneralString	0/1
C	UTF8String	0/1	1C	UniversalString	0/1
D	Relative OID		1D	CHARACTER STRING	0/1
E			1E	BMPString	0/1
F			1F	reserviert	0/1

TLV = Typ | Length | Value

Datenelement:=

TypID | Längenbeschreibung | Daten | Ende Marke (falls notw.)

Bsp.datensatz: {7,42,2018-04-26 07:30:11,"Hallo!"}

32|30|31|38|30|34|32|36|30|37|33|30|31|31

Pasal → RECORD
C → struct

```
TestPDU ::=
    SEQUENCE {                                  30
        aNumber INTEGER,                        02
        anotherNumber INTEGER,                  02
        today UTCTime OPTIONAL,                 17
        myText CHOICE {                         1A
            multilingualText  ISO10646String,
            standardText      VisibleString
                              }
        }
```

C → union
Pascal → RECORD CASE

T	L	V
30	80	(undef. Länge)
02	01	07
02	01	2A
17	0E	32/30/31/38/30/34/32/36/30 37/33/30/31/31
1A	06	48 61 6C 6C 6F 21
00	00	(Nullterminiert)

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
61									6A		6C			6F

AVP (Attribut Value Pairs)

Mittels eines AVP Datenpaares lassen sich (einfache oder kombinierte/ komplexe) Datenelemente beschreiben/definieren. Hierbei werden die Objektbezeichner und deren Wert gegenübergestellt.

Diese Art der Datencodierung eignet sich vor allem für textbasierte Daten (z.B. OptionenParameter und deren Werte im EMail-Header oder SIP-Header)

EMail-Header

```
From - Thu Nov 24 13:40:25 2016
X-Account-Key: account1
X-UIDL: 42552
X-Mozilla-Status: 0003
X-Mozilla-Status2: 00000000
X-Mozilla-Keys:
Received: from SERU29.hft-leipzig.de ([fe80::1829:2039:7ab1:9eac]) by
 SERU43.hft-leipzig.de ([fe80::652a:feb:2649:b566%11]) with mapi id
 14.03.0319.002; Thu, 24 Nov 2016 13:32:46 +0100
From: "Saupe, Volker" <saupe@hft-leipzig.de>
To: Hochschullehrer <HSL@hft-leipzig.de>
CC: "xxxxxxx.yyyyyyy@telekom.de" <xxxxxxx.yyyyyyy@telekom.de>
Subject: xxxxxxxxxxxxxxxxx
Thread-Topic: xxxxxxxxxxxxxxxxx
Thread-Index: AdJGTo2t0ybXh6VVQeWcWeLZzrM7SA==
Date: Thu, 24 Nov 2016 13:32:45 +0100
Message-ID: <60FAFF9FFD57284EADFA03E6A87FD0B60D911042@seru29.hft-leipzig.de>
Accept-Language: de-DE, en-US
Content-Language: de-DE
X-MS-Exchange-Organization-AuthAs: Internal
X-MS-Exchange-Organization-AuthMechanism: 04
X-MS-Exchange-Organization-AuthSource: SERU43.hft-leipzig.de
X-MS-Has-Attach:
X-Auto-Response-Suppress: DR, OOF, AutoReply
X-MS-Exchange-Organization-SCL: -1
X-MS-TNEF-Correlator:
x-originating-ip: [195.145.74.98]
Content-Type: text/html; charset="iso-8859-1"
Content-Transfer-Encoding: quoted-printable
MIME-Version: 1.0
```

LDIF - Adressdaten

```
dn: cn='Michael Flegl',mail=flegl@hft-leipzig.de
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
objectclass: mozillaAbPersonAlpha
givenName: 'Michael
sn: Flegl'
cn: 'Michael Flegl'
mail: flegl@hft-leipzig.de
modifytimestamp: 1479839598
```

JSON - JavaScript Object Notation

2005 von Bob Ipplito vorgestellt

ECMA-262 (Juni 2016): ECMAScript(R) 2016 - Language Specification
https://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf

ECMA-404 (Okt.2013): The JSON Data Interchange Format
http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf

Website: JSON.ORG
MIME-Typ: application/json
RFC 4627 - The application_json Media Type for JavaScript Object Notation (JSON)
RFC 7159 - JSON_Interchange_Format

JavaScript on Demand (JOD)
JSON (JavaScript Object Notation) mittels eval() interpretierbares Javascript
JSON (2005 von Bob Ippolito)-RFC 4627
YAML (Markup-Sprache)
BISON (Binary Interchange Standard and Object Notation)=binäres JSON-Format
BERT steht für „Berkeley Reliability Tool“ und beschreibt ein binäres Dateiformat,
das auf der Programmiersprache Erlang basiert
DOM (Document Object Model)

Zeichencodes

Codierung – Zeichencodes

Codierung von Zeichen (characters) mittels einer Anzahl von Bits

- 7-Bit ASCII (z.B. für Email)
- ANSI
- IBM EBCDIC
- UNICODE
- UTF-8
- BASE-64, BASE-32, BASE-16
- BASE-85 (IPv6-Adressen)
- BASE-91, BASE-128
- Puny-Code (für Umlaute oder internationale Zeichen in Domains)
- Barcode, Matrixcode, Blindenschrift

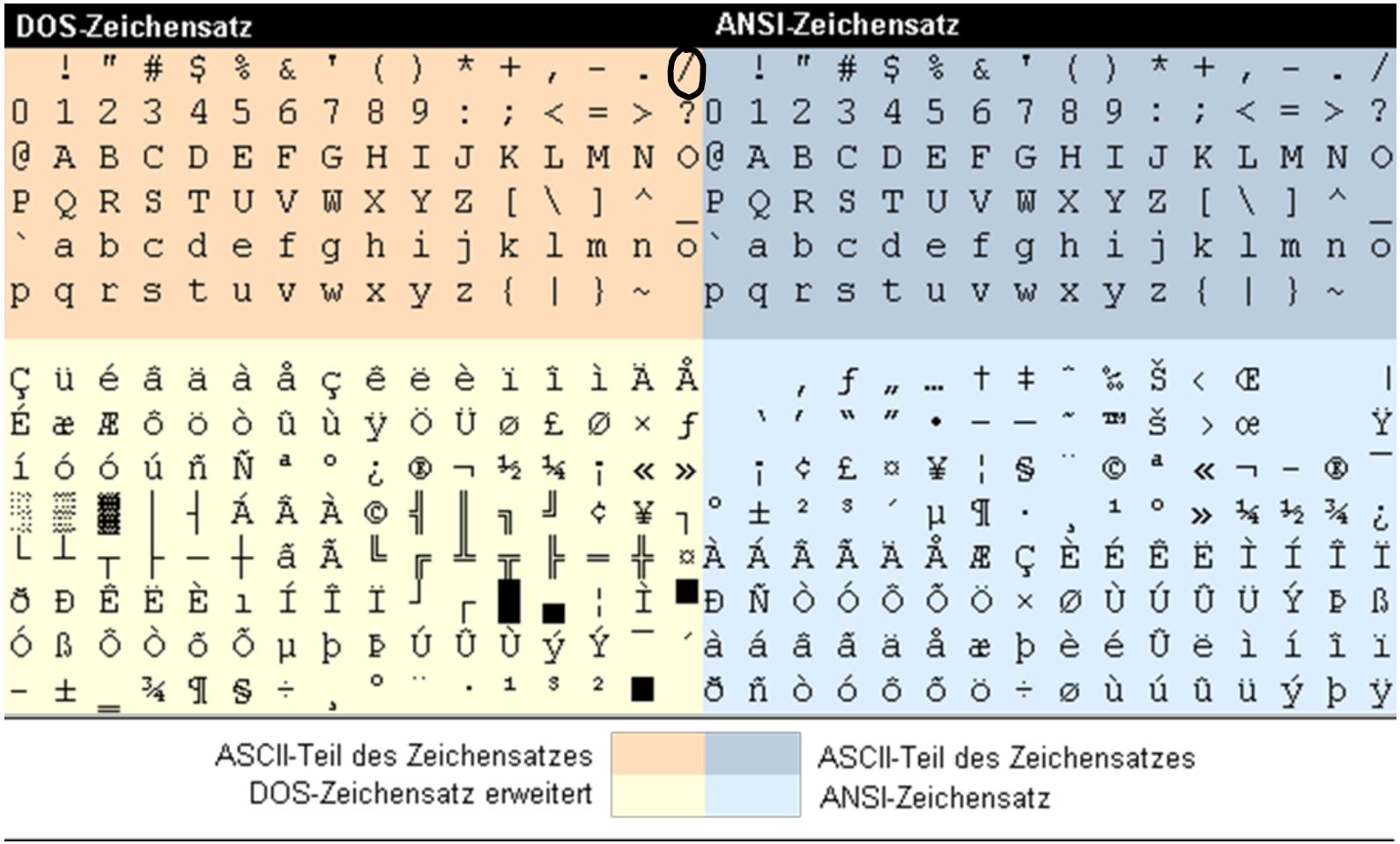
Character Encoding Model (RFC 2130)

- Abstract Character Repertoire (ACR)
- Coded Character Set (CCS)
- Character Encoding Form (CEF)
- Character Encoding Scheme (CES)
- Transfer Encoding Syntax (TES)
- Character Map (CM)
- ACR → serielle Byte Sequenz

Codierung – ASCII/ANSI

ASCII = American Standard Code of Information Interchange
ANSI = American National Standard Institute

60
70



IBM Codierung - EBCDIC

EBCDIC Code (Extended Binary Codes Decimal Interchange) Code

- Gruppe kodierter Zeichensätze für Mainframes bestehend aus 8-bit codes
- 00H bis 3FH , d.h. 64 Zeichen für Steuercodes
- 41H bis FEH für graphische Zeichen
- C1H bis C9H , D1H bis D9H, E2H bis E9H für englische Großbuchstaben
- 81H bis 89H, 91 bis 99H, A2H bis A9H für englische Kleinbuchstaben

				3	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
				2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
				1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
				0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
7	6	5	4		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	NUL	SOH	STX	ETX	PF	HAT	LC	DEL			SMM	VT	FF	CR	SO	SI
0	0	0	1	1	DLE	DC1	DC2	DC3	RES	NL	BS	IL	CAN	EM	CC		IFS	IGS	IRS	IUS
0	0	1	0	2	DS	SOS	FS		BYP	LF	EOB	PRE			SM			ENQ	ACK	BEL
0	0	1	1	3			SYN		PN	RS	UC	EOT					DC4	NAK		SUB
0	1	0	0	4	SP											.	<	(	+	|
0	1	0	1	5	&										!	$	*	)	;	
0	1	1	0	6	-	/										`	%	-	>	?
0	1	1	1	7											:	#	@	,	=	"
1	0	0	0	8		a	b	c	d	e	f	g	h	i						
1	0	0	1	9		j	k	l	m	n	o	p	q	r						
1	0	1	0	A			s	t	u	v	w	x	y	z						
1	0	1	1	B																
1	1	0	0	C		A	B	C	D	E	F	G	H	I						
1	1	0	1	D		J	K	L	M	N	O	P	Q	R						
1	1	1	0	E			S	T	U	V	W	X	Y	Z						
1	1	1	1	F	0	1	2	3	4	5	6	7	8	9						

ISO-8559 Codepages

ISO-8559-Familie wurde vom European Computer Manufacturer's Association (ECMA) entwickelt

- Set von standardisierten Zeichensätzen für alphabetische Schriften
  (lateinischen Schriften mit den Beinamen Latin-1 bis Latin-6., ...)
- 1 Byte zur Codierung eines Zeichens ? 256 Zeichen
- die ersten 128 stimmen mit ANSI/ASCII überein
- 128 bis 159 für besondere Zeichen

ISO-8859-x (International Standards Organization)

ISO 8859/1 ("Latin-1") Western (Albanisch, Dänisch, Deutsch, Englisch, Faraörisch, Finnisch, Französisch, Galizisch, Irisch, Isländisch, Italienisch, Katalanisch, Niederländisch, Norwegisch, Portugiesisch, Schwedisch und Spanisch)

ISO 8859/2 ("Latin-2") bzw. Windows 1250 - Zentral Europa + slawische Sprachen (Kroatisch, Polnisch, Rumänisch, Slowaisch, Slowenisch, Tschechisch und Ungarisch

ISO 8859/3 ("Latin-3") Esperanto, Galizisch, Maltesisch und Türkisch

ISO 8859/4 ("Latin-4")bzw. Windows 1257 - Baltic (Estonisch, Lättisch und Lithauisch) ähnlich 8559-10

ISO 8859/5 ("Latin-5")bzw. Windows 1251 - Kyrillisch (Bulgarisch, Mazedonisch, Russisch, Serbisch und Ukrainisch)

ISO 8859/6 ("Latin-6") arabische Schrift mit Schriftrichtung (von rechts nach links)

ISO 8859/7 ("Latin-7") bzw. Windows 1253 - Griechisch

ISO 8859/9 bzw. Windows 125? - Türkisch

Zeichencodierung – UCS – ISO/IEC 10646-1

Abstract
ISO/IEC 10646-1 defines a multi-octet character set called the Universal Character Set (UCS) which encompasses most of the world's writing systems. Multi-octet characters, however, are not compatible with many current applications and protocols, and this has led to the development of a few so-called UCS transformation formats (UTF), each with different characteristics. UTF-8, the object of this memo, has the characteristic of preserving the full US-ASCII range, providing compatibility with file systems, parsers and other software that rely on US-ASCII values but are transparent to other values. This memo updates and replaces RFC 2044, in particular addressing the question of versions of the relevant standards.

➔ ISO/IEC 10646-1 definiert eine multi-octet Zeichenmenge (Universal Character Set-UCS)

- Kodierung mittels 4-Oktett pro Zeichen: UCS-4
- Kodierung mittels 2-Oktett pro Zeichen: UCS-2
- (adressiert nur die ersten 65536=64K Zeichen der UCS, d.h. die sogenannte Basic Multilingual Plane= BMP)

UNICODE Version 5.2 (2009) 686 S.
Version 8 (2015)
Version 10 (2017)

Unicode Code Charts

http://www.unicode.org/charts/PDF/Unicode-5.2/

http://www.unicode.org/charts/

http://www.unicode.org/charts/unihanrsindex.html

Unicode terms & Unicode Character Name Index

http://www.unicode.org/glossary/

http://www.unicode.org/charts/charindex.html

Unicode Character Database

http://www.unicode.org/ucd/

Unicode Technical Standards and Unicode Technical Reports

http://www.unicode.org/reports/

Updates and Errata

http://www.unicode.org/reporting.html

http://www.unicode.org/errata/

UNICODE Version 5.2 (2009)
Version 8 (2015)
Version 10 (2017)

Unicode Technical Standards

UTS #6: A Standard Compression Scheme for Unicode

UTS #10: Unicode Collation Algorithm

UTS #18: Unicode Regular Expressions

UTS #22: Character Mapping Markup Language (CharMapML)

UTS #35: Unicode Locale Data Markup Language (LDML)

UTS #37: Unicode Ideographic Variation Database

UTS #39: Unicode Security Mechanisms

Unicode Technical Reports

UTR #16: UTF-EBCDIC

UTR #17: Unicode Character Encoding Model

UTR #20: Unicode in XML and Other Markup Languages

UTR #23: The Unicode Character Property Model

UTR #25: Unicode Support for Mathematics

UTR #26: Compatibility Encoding Scheme for UTF-16: 8-Bit (CESU-8)

UTR #33: Unicode Conformance Model

UTR #36: Unicode Security Considerations

UTR #45: U-Source Ideographs

Unicode Standard Annexes

UAX #9: Unicode Bidirectional Algorithm, Version 5.2.0

UAX #11: East Asian Width, Version 5.2.0

UAX #14: Unicode Line Breaking Algorithm, Version 5.2.0

UAX #15: Unicode Normalization Forms, Version 5.2.0

UAX #24: Unicode Script Property, Version 5.2.0

UAX #29: Unicode Text Segmentation, Version 5.2.0

UAX #31: Unicode Identifier and Pattern Syntax, Version 5.2.0

UAX #34: Unicode Named Character Sequences, Version 5.2.0

UAX #38: Unicode Han Database (Unihan), Version 5.2.0

UAX #41: Common References for Unicode Standard Annexes, Version 5.2.0

UAX #42: Unicode Character Database in XML, Version 5.2.0

UAX #44: Unicode Character Database, Version 5.2.0

Codierung – UNICODE

Conformance Requirements

- Code Points Unassigned to Abstract Characters
- Interpretation
- Modification
- Character Encoding Forms
- Character Encoding Schemes
- Bidirectional Text
- Normalization Forms
- Normative References
- Unicode Algorithms
- Default Casing Algorithms
- Unicode Standard Annexes

Code point type (seven fundamental classes)

- Graphic
- Format
- Control
- Private-Use
- Surrogate
- Noncharacter
- Reserved

Noncharacters

- Byte Order Mark (BOM) u+FEFF
- Layout and Format Control Characters
- The Replacement Character U+FFFD
- Object Replacement Character U+FFFC
- Control Codes (ISO/IEC 2022 framework kompatibel)
  U+0000..U+001F, U+007F..U+009F

Plane 0 (BMP) Basic Multilingual Plane

Plane 1 (SMP) Supplementary Multilingual Plane

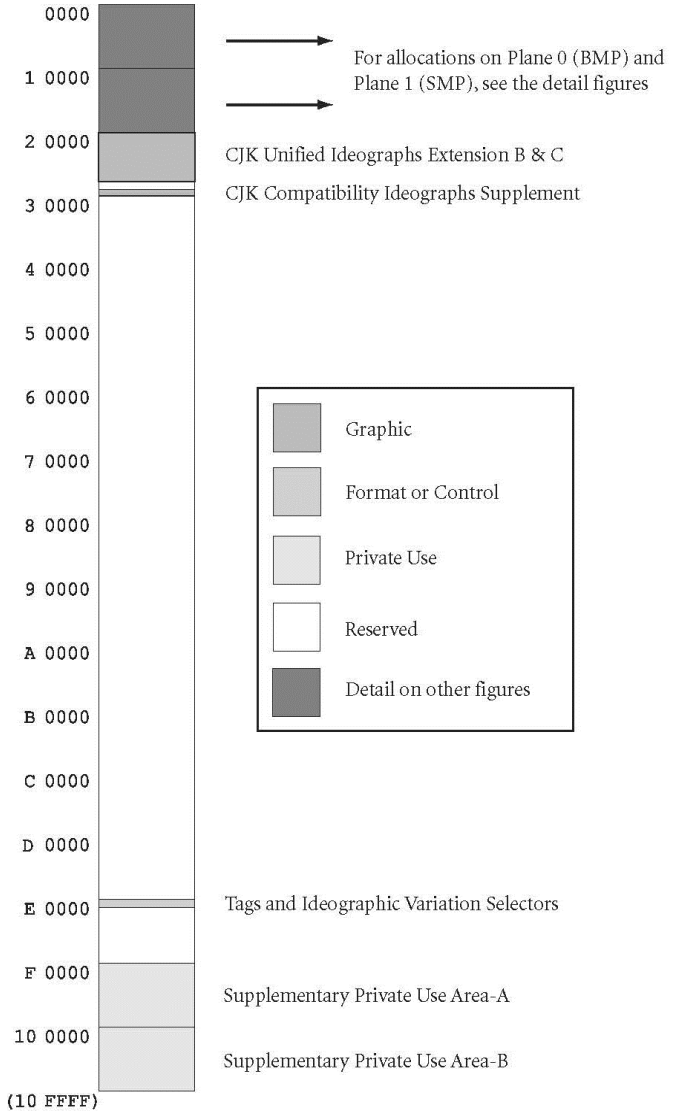
Plane 2 (SIP) - Supplementary Ideographic Plane

Codierung – UNICODE

Table 2-3. Types of Code Points

Basic Type	Brief Description	General Category	Character Status	Code Point Status
Graphic	Letter, mark, number, punctuation, symbol, and spaces	L, M, N, P, S, Zs	*Assigned to abstract character*	*Designated (assigned) code point*
Format	Invisible but affects neighboring characters; includes line/paragraph separators	Cf, Zl, Zp		
Control	Usage defined by protocols or standards outside the Unicode Standard	Cc		
Private-use	Usage defined by private agreement outside the Unicode Standard	Co		
Surrogate	Permanently reserved for UTF-16; restricted interchange	Cs	*Not assigned to abstract character*	
Noncharacter	Permanently reserved for internal usage; restricted interchange	Cn		
Reserved	Reserved for future assignment; restricted interchange			*Undesignated (unassigned) code point*

Figure 2-13. Unicode Allocation



UNICODE

Plane 0 (BMP) Basic Multilingual Plane

- common-use worldwide characters
- many historical and rare characters

Plane 1 (SMP) Supplementary Multilingual Plane

- selten benutzte Zeichen
- Zeichen, die nicht mehr in die BMP passen
- pictographic symbol sets
- special-purpose invented scripts
- notational systems
- Gothic (historic)
- Shavian (special-purpose invented)
- Musical Symbols (notational system)
- Domino Tiles (pictographic)
- Ancient Greek Numbers (historic extension for Greek)

Plane 2 (SIP) - Supplementary Ideographic Plane

- more common CJK characters
- CJK-JRG (Chinese/Japanese/Korean Joint Research Group)
- extremely rare or of historical interest only

UNICODE Codeplanes

Figure 2-14. Allocation on the BMP *Plane 0*

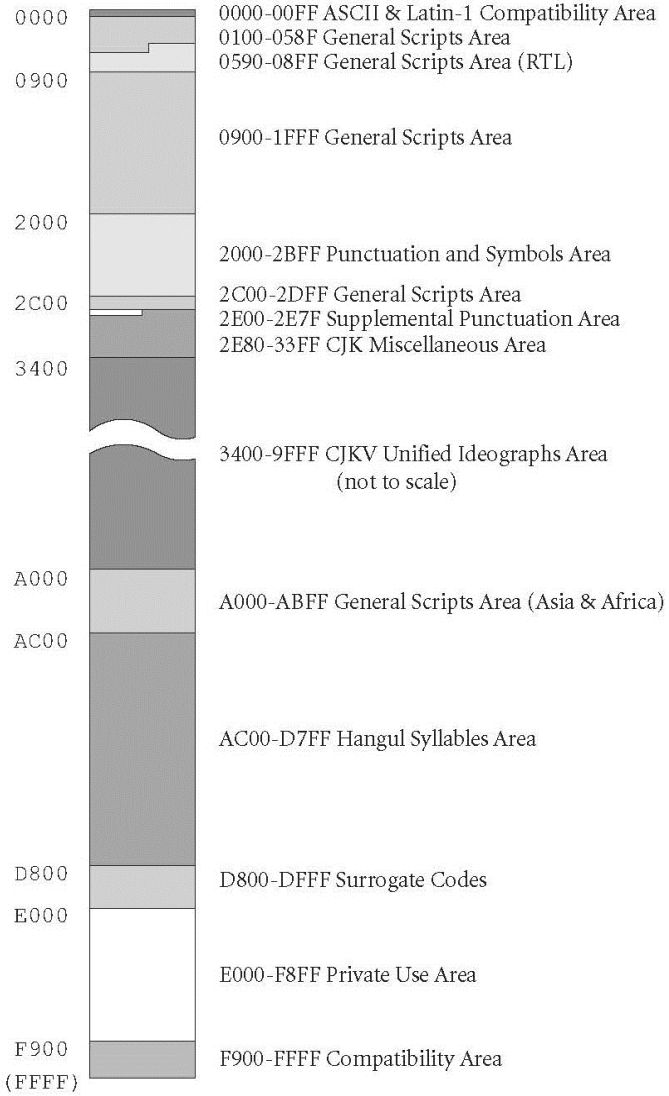
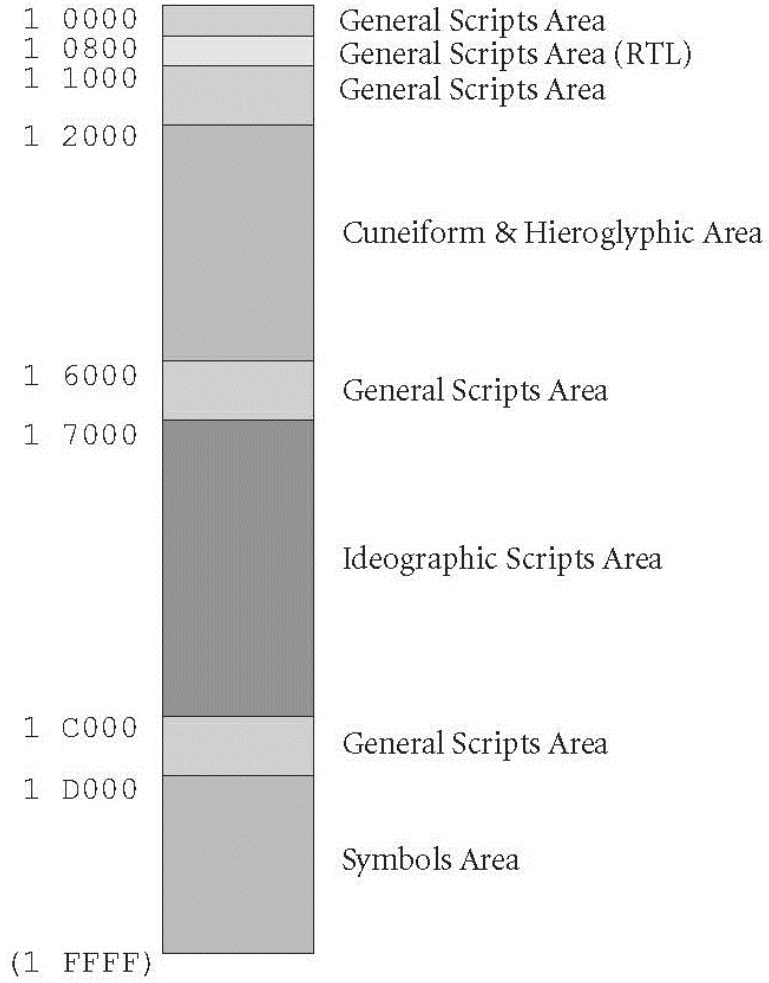


Figure 2-15. Allocation on Plane 1



UNICODE Codeplanes

Zeichen (Skripts) des Mittleren Ostens

Hebrew (U+0590–U+05FF v.r.n.l., ISO/IEC 8859-8—Part 8. Latin/Hebrew Alphabet)

Arabic (U+0600–U+06FF v.r.n.l., ASMO 449 => ECMA-114 => ISO/IEC 8859-6—Part 6. Latin/Arabic Alphabet)

Syriac (U+0700–U+074F v.r.n.l.,West & East Syriac Dialekt)

Samaritan (U+0800–U+083F v.r.n.l.)

Thaana (U+0780–U+07BF v.r.n.l., Malediven)

Symbols
- Currency symbols
- Geometrical symbols
- Letterlike symbols
- Miscellaneous symbols and dingbats
- Mathematical alphabets
- Enclosed and square symbols
- Number forms
- Braille patterns
- Mathematical symbols
- Western and Byzantine musical symbols
- Invisible mathematical operators
- Ancient Greek musical notation
- Technical symbols

Zeichen (Skripts) Süd Asien 1
- Sinhala
- Limbu
- Meetei
- Mayek
- Tibetan
- Syloti Nagri
- Ol Chiki
- Lepcha
- Kaithi
- Kharoshthi
- Phags-pa
- Saurashtra

East Asian Scripts
- Han (Chinese, Japanese, and Korean languages from Han Dynasty)
- Hiragana
- Hangul
- Bopomofo
- Katakana
- Yi

Zeichen (Skripts) Süd Asien 1
- Devanagari
- Gujarati
- Telugu
- Bengali
- Oriya
- Kannada
- Gurmukhi
- Tamil
- Malayalam

Ancient and Historic Scripts
- Ogham
- Cypriot Syllabary
- Inscriptional Pahlavi
- Old Italic
- Ancient Anatolian Alphabets
- Avestan
- Runic
- Old South Arabian
- Ugaritic
- Gothic
- Phoenician
- Old Persian
- Old Turkic
- Imperial Aramaic
- Sumero-Akkadian
- Linear B
- Inscriptional Parthian
- Egyptian Hieroglyphs

Southeast Asian Scripts
- Thai
- Tai Tham
- Buginese
- Lao
- Tai Viet
- Balinese
- Myanmar
- Kayah Li
- Javanese
- Khmer
- Cham
- Rejang
- Tai Le
- Philippine scripts
- Sundanese
- New Tai Lue

Additional Modern Scripts
- Ethiopic
- N'Ko
- Canadian Aboriginal Syllabics
- Mongolian
- Vai
- Deseret
- Osmanya
- Bamum
- Shavian
- Tifinagh
- Cherokee
- Lisu

European Alphabetic Scripts
- Latin
- Cyrillic
- Georgian
- Greek
- Glagolitic
- Modifier letters
- Coptic
- Armenian
- Combining marks

Special Areas and Format Characters
- Control codes
- Surrogates area (U+D800–U+DFFF)
- Private-use characters
- Layout controls
- Variation selectors
- Deprecated format characters
- Specials
- Noncharacters
- Deprecated tag characters

Codierung – UNICODE

Figure 2-1. Text Elements and Characters

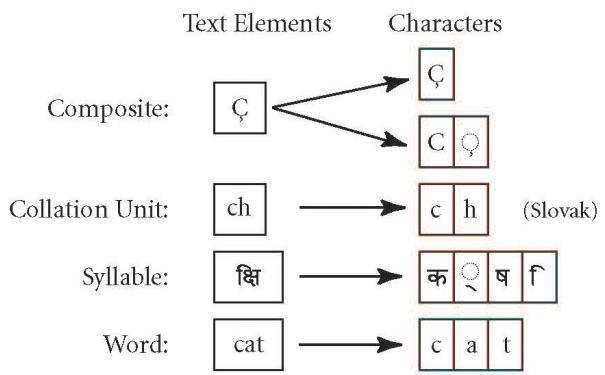


Figure 2-16. Writing Directions

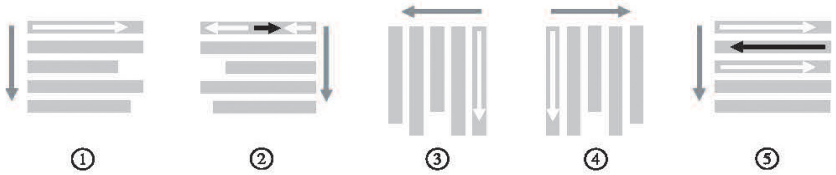


Figure 2-7. Dynamic Composition

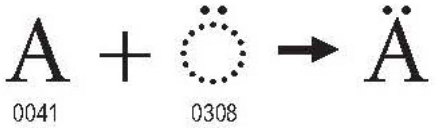
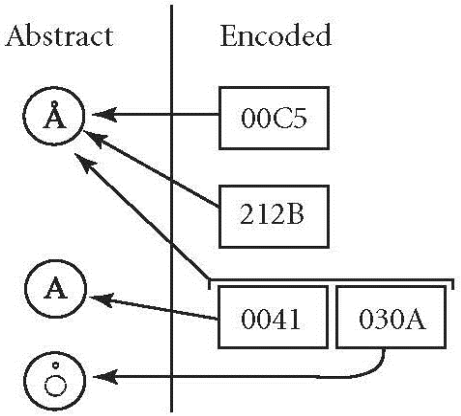


Figure 2-8. Abstract and Encoded Characters



Codierung – UNICODE

Figure 2-16. Writing Directions

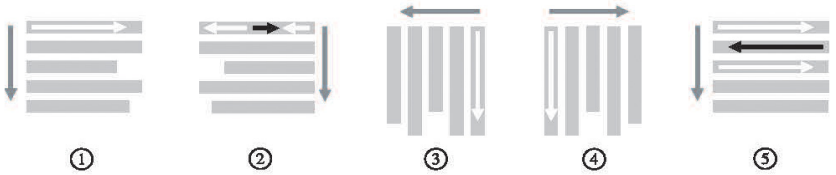


Figure 2-4. Bidirectional Ordering

Figure 2-5. Writing Direction and Numbers

① آ مغ سعفة ١١٢٣ من.

② .1123 נא ראה עמוד

③ راجع صفحة ١١٢٣ من فضلك.

④ Please see page 1123.

⑤ 1123ページをみてください。

⑥ 1123ページを
みてください。

Codierung – UNICODE

Figure 2-11. Unicode Encoding Forms

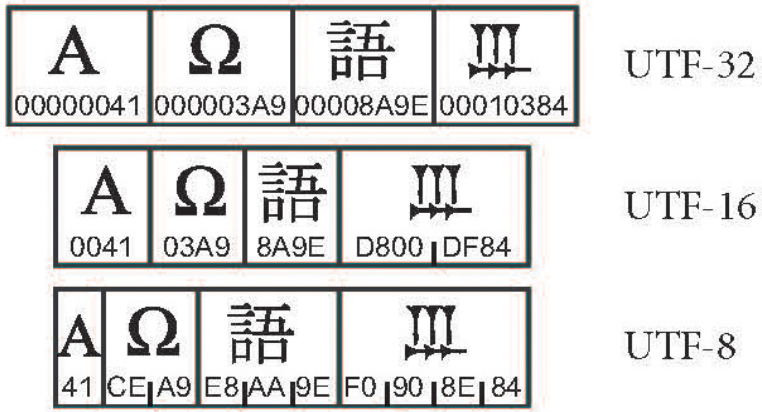
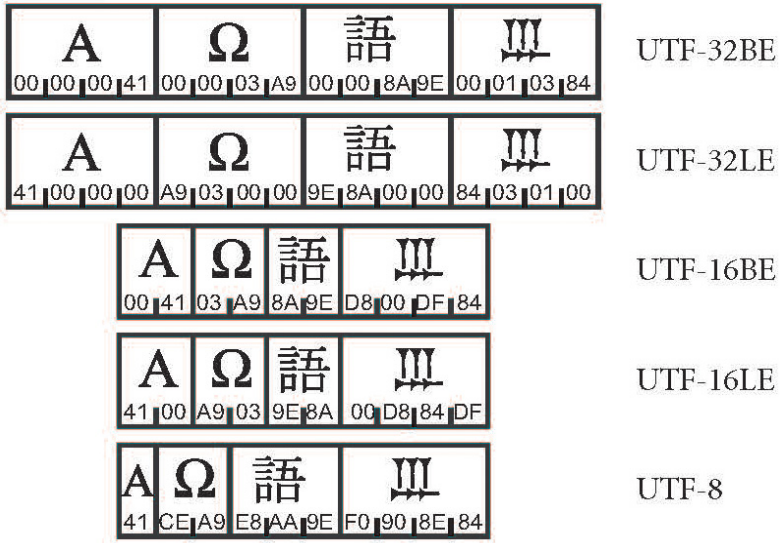


Figure 2-12. Unicode Encoding Schemes



	000	001	002	003	004	005	006	007
0	NUL 0000	DLE 0010	SP 0020	0 0030	@ 0040	P 0050	` 0060	p 0070
1	SOH 0001	DC1 0011	! 0021	1 0031	A 0041	Q 0051	a 0061	q 0071
2	STX 0002	DC2 0012	" 0022	2 0032	B 0042	R 0052	b 0062	r 0072
3	ETX 0003	DC3 0013	# 0023	3 0033	C 0043	S 0053	c 0063	s 0073
4	EOT 0004	DC4 0014	$ 0024	4 0034	D 0044	T 0054	d 0064	t 0074
5	ENQ 0005	NAK 0015	% 0025	5 0035	E 0045	U 0055	e 0065	u 0075
6	ACK 0006	SYN 0016	& 0026	6 0036	F 0046	V 0056	f 0066	v 0076
7	BEL 0007	ETB 0017	' 0027	7 0037	G 0047	W 0057	g 0067	w 0077
8	BS 0008	CAN 0018	(0028	8 0038	H 0048	X 0058	h 0068	x 0078
9	HT 0009	EM 0019	) 0029	9 0039	I 0049	Y 0059	i 0069	y 0079
A	LF 000A	SUB 001A	* 002A	: 003A	J 004A	Z 005A	j 006A	z 007A
B	VT 000B	ESC 001B	+ 002B	; 003B	K 004B	[005B	k 006B	{ 007B
C	FF 000C	FS 001C	, 002C	< 003C	L 004C	\ 005C	l 006C	| 007C
D	CR 000D	GS 001D	- 002D	= 003D	M 004D	] 005D	m 006D	} 007D
E	SO 000E	RS 001E	. 002E	> 003E	N 004E	^ 005E	n 006E	~ 007E
F	SI 000F	US 001F	/ 002F	? 003F	O 004F	_ 005F	o 006F	DEL 007F

C0 controls

Alias names are those for ISO/IEC 6429:1992. Commonly used alternative aliases are also shown.

0000 NUL <control>
= NULL

0001 SOH <control>
= START OF HEADING

0002 STX <control>
= START OF TEXT

0003 ETX <control>
= END OF TEXT

0004 EOT <control>
= END OF TRANSMISSION

0005 ENQ <control>
= ENQUIRY

0006 ACK <control>
= ACKNOWLEDGE

0007 BEL <control>
= BELL

0008 BS <control>
= BACKSPACE

0009 HT <control>
= CHARACTER TABULATION
= horizontal tabulation (HT), tab

000A LF <control>
= LINE FEED (LF)
= new line (NL), end of line (EOL)

000B VT <control>
= LINE TABULATION
= vertical tabulation (VT)

000C FF <control>
= FORM FEED (FF)

000D CR <control>
= CARRIAGE RETURN (CR)

000E SO <control>
= SHIFT OUT
• known as LOCKING-SHIFT ONE in 8-bit environments

000F SI <control>
= SHIFT IN
• known as LOCKING-SHIFT ZERO in 8-bit environments

0010 DLE <control>
= DATA LINK ESCAPE

0011 DC1 <control>
= DEVICE CONTROL ONE

0012 DC2 <control>
= DEVICE CONTROL TWO

0013 DC3 <control>
= DEVICE CONTROL THREE

0014 DC4 <control>
= DEVICE CONTROL FOUR

0015 NAK <control>
= NEGATIVE ACKNOWLEDGE

0016 SYN <control>
= SYNCHRONOUS IDLE

0017 ETB <control>
= END OF TRANSMISSION BLOCK

0018 CAN <control>
= CANCEL

0019 EM <control>
= END OF MEDIUM

001A SUB <control>
= SUBSTITUTE
→ FFFD replacement character

001B ESC <control>
= ESCAPE

001C FS <control>
= INFORMATION SEPARATOR FOUR
= file separator (FS)

001D GS <control>
= INFORMATION SEPARATOR THREE
= group separator (GS)

001E RS <control>
= INFORMATION SEPARATOR TWO
= record separator (RS)

001F US <control>
= INFORMATION SEPARATOR ONE
= unit separator (US)

ASCII punctuation and symbols

Based on ISO/IEC 646.

0020 SP SPACE
• sometimes considered a control code
• other space characters: 2000 – 200A
→ 00A0 no-break space
→ 2008 zero width space
→ 2060 word joiner
→ 3000 ideographic space
→ FEFF zero width no-break space

0021 ! EXCLAMATION MARK
= factorial
= bang
→ 00A1 ¡ inverted exclamation mark
→ 01C3 ǃ latin letter retroflex click
→ 203C ‼ double exclamation mark
→ 203D ⁽ interrobang
→ 2762 ❢ heavy exclamation mark ornament

0022 " QUOTATION MARK
• neutral (vertical), used as opening or closing quotation mark
• preferred characters in English for paired quotation marks are 201C “ & 201D ”
→ 02BA ˝ modifier letter double prime
→ 0308 ◌̈ combining double acute accent
→ 030E ◌̎ combining double vertical line above
→ 2033 ″ double prime
→ 3003 〃 ditto mark

0023 # NUMBER SIGN
= pound sign, hash, crosshatch, octothorpe
→ 2114 ℔ l b bar symbol
→ 266F ♯ music sharp sign

0024 $ DOLLAR SIGN
= milreis, escudo
• glyph may have one or two vertical bars
• other currency symbol characters: 20A0 ₠ – 20B5 ₵
→ 00A4 ¤ currency sign

0025 % PERCENT SIGN
→ 066A ٪ arabic percent sign
→ 2030 ‰ per mille sign
→ 2031 ‱ per ten thousand sign
→ 2052 ⁒ commercial minus sign

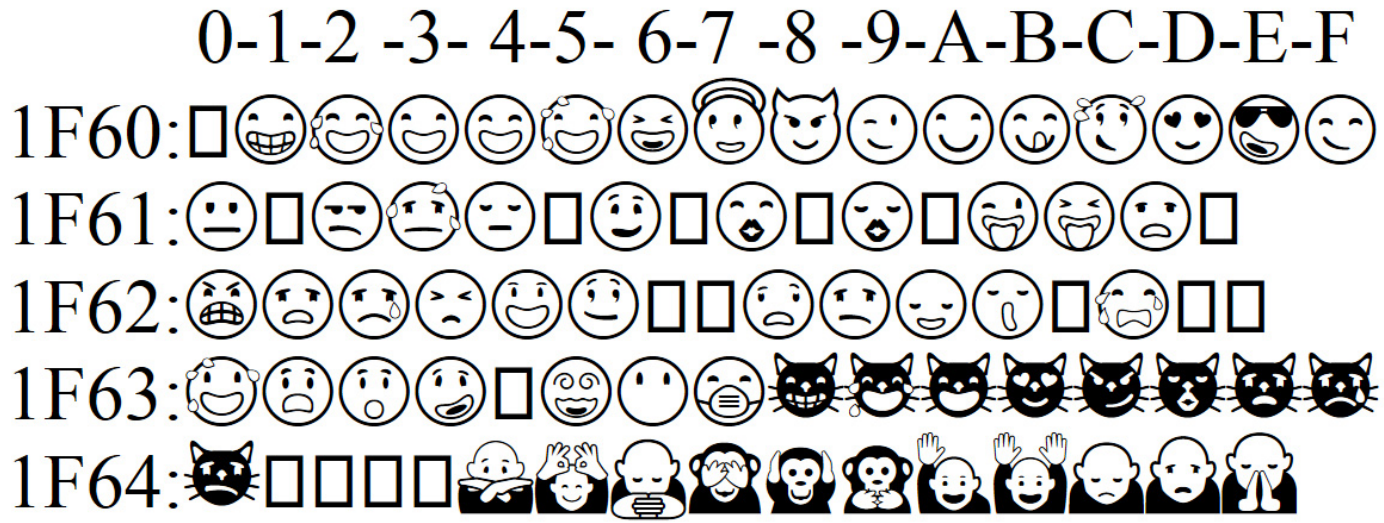
0026 & AMPERSAND
→ 204A ⅋ tironian sign et
→ 2148 ⅊ turned ampersand

UNICODE Emoticons

Code charts: https://www.unicode.org/charts/

Emoticons: https://www.unicode.org/charts/PDF/U1F600.pdf

(U+1F600-U+1F64F) d.h. 128512-128591




```
<HTML>
<HEAD></HEAD>
<BODY>
           0-1-2 -3- 4-5- 6-7 -8 -9-A-B-C-D-E-F<BR>
1F60:&#x1F600;&#x1F601;&#x1F602;&#x1F603;&#x1F604;&#x1F605;&#x1F606;&#x1F607;&#x1F608;&#x1F609;&#x1F60A;&#x1F60B;&#x1F60C;&#x1F60D;&#x1F60E;&#x1F60F;<BR>
1F61:&#x1F610;&#x1F611;&#x1F612;&#x1F613;&#x1F614;&#x1F615;&#x1F616;&#x1F617;&#x1F618;&#x1F619;&#x1F61A;&#x1F61B;&#x1F61C;&#x1F61D;&#x1F61E;&#x1F61F;<BR>
1F62:&#x1F620;&#x1F621;&#x1F622;&#x1F623;&#x1F624;&#x1F625;&#x1F626;&#x1F627;&#x1F628;&#x1F629;&#x1F62A;&#x1F62B;&#x1F62C;&#x1F62D;&#x1F62E;&#x1F62F;<BR>
1F63:&#x1F630;&#x1F631;&#x1F632;&#x1F633;&#x1F634;&#x1F635;&#x1F636;&#x1F637;&#x1F638;&#x1F639;&#x1F63A;&#x1F63B;&#x1F63C;&#x1F63D;&#x1F63E;&#x1F63F;<BR>
1F64:&#x1F640;&#x1F641;&#x1F642;&#x1F643;&#x1F644;&#x1F645;&#x1F646;&#x1F647;&#x1F648;&#x1F649;&#x1F64A;&#x1F64B;&#x1F64C;&#x1F64D;&#x1F64E;&#x1F64F;<BR>
</BODY>
</HTML>
```

The emoticons have been organized by mouth shape to make it easier to locate the different characters in the code chart.

Faces

1F600 GRINNING FACE
1F601 GRINNING FACE WITH SMILING EYES
1F602 FACE WITH TEARS OF JOY
1F603 SMILING FACE WITH OPEN MOUTH
→ 263A ☺ white smiling face
1F604 SMILING FACE WITH OPEN MOUTH AND SMILING EYES
1F605 SMILING FACE WITH OPEN MOUTH AND COLD SWEAT
1F606 SMILING FACE WITH OPEN MOUTH AND TIGHTLY-CLOSED EYES
1F607 SMILING FACE WITH HALO
1F608 SMILING FACE WITH HORNS
• commonly depicted as a (sinister) smiling version of 1F47F 👿 imp
1F609 WINKING FACE
1F60A SMILING FACE WITH SMILING EYES
1F60B FACE SAVOURING DELICIOUS FOOD
1F60C RELIEVED FACE
• indicates relief, not sleeping
1F60D SMILING FACE WITH HEART-SHAPED EYES
1F60E SMILING FACE WITH SUNGLASSES
1F60F SMIRKING FACE
1F610 NEUTRAL FACE
• used for the West Wind in some Mahjong annotation
1F611 EXPRESSIONLESS FACE
1F629 WEARY FACE
1F62A SLEEPY FACE
1F62B TIRED FACE
1F62C GRIMACING FACE
• should not be depicted with zipper mouth
→ 1F910 🤐 zipper-mouth face
1F62D LOUDLY CRYING FACE
1F62E FACE WITH OPEN MOUTH
1F62F HUSHED FACE
1F630 FACE WITH OPEN MOUTH AND COLD SWEAT
1F631 FACE SCREAMING IN FEAR
1F632 ASTONISHED FACE
1F633 FLUSHED FACE
• embarrassed
1F634 SLEEPING FACE
1F635 DIZZY FACE
1F636 FACE WITHOUT MOUTH
→ 2687 ⚇ white circle with two dots
1F637 FACE WITH MEDICAL MASK

Cat faces

1F638 GRINNING CAT FACE WITH SMILING EYES
1F639 CAT FACE WITH TEARS OF JOY
1F63A SMILING CAT FACE WITH OPEN MOUTH
1F63B SMILING CAT FACE WITH HEART-SHAPED EYES
1F63C CAT FACE WITH WRY SMILE
1F63D KISSING CAT FACE WITH CLOSED EYES
1F63E POUTING CAT FACE
• intended to depict pouting rather than simply anger

Codierung – UNICODE

Table 6-6. Names for the @

Language	Name and Comments
Chinese	= xiao laoshu (means "little mouse" in Mandarin Chinese), laoshu hao (means "mouse mark" in Mandarin Chinese)
Danish	= grishale, snabel-a (common, humorous slang)
Dutch	= apenstaartje (common, humorous slang)
Finnish	= ät, ät-merkki (Finnish standard) = kissanhäntä, miukumauku (common, humorous slang)
French	= arobase, arrobe, escargot, a crolle (common, humorous slang)
German	= Klammeraffe
Hebrew	= shtrudl ("Strudel", modern Hebrew) = krukhit (more formal Hebrew)
Hungarian	= kukac (common, humorous slang)
Italian	= chiocciola
Polish	= atka, małpa, małpka (common, humorous slang)
Portuguese	= arroba
Russian	= sobachka (common, humorous slang)
Slovenian	= afna (common, humorous slang)
Spanish	= arroba
Swedish	= snabel-a, kanelbulle (common, humorous slang)

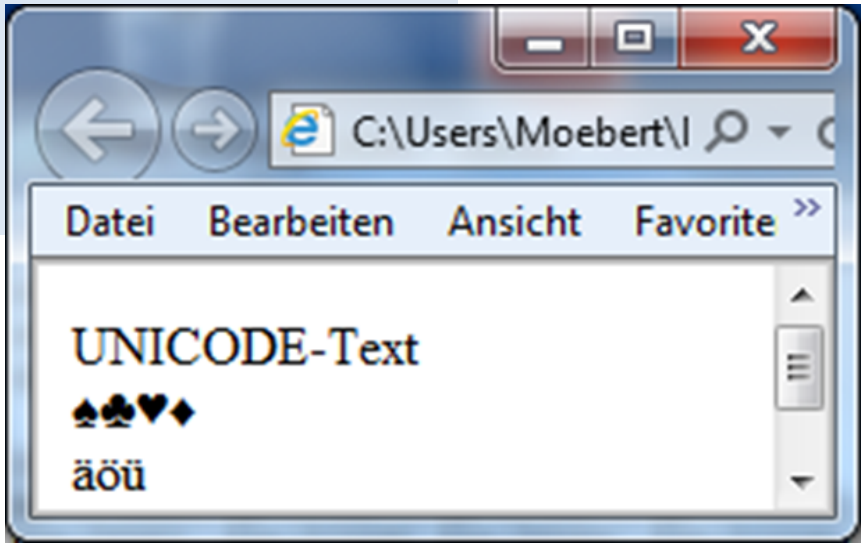
Zeichencodierung in HTML – UNICODE

```
<HTML>
<HEAD></HEAD>
<BODY>

UNICODE-Text<br>
&#x2660;&#x2663;&#x2665;&#x2666;<br>
&#x0061;&#x0308;&#x006F;&#x0308;&#x0075;&#x0308;<br>
&#0097;&#x0308;&#0111;&#x0308;&#0117;&#x0308;<br>



</BODY>
</HTML>
```



Codierung – UNICODE

- BYTE ORDER MARK (BOM) (U+FEFF / 0xFEFF)
  auch ZERO WIDTH NO-BREAK SPACE
- OBJECT REPLACEMENT CHARACTER (0xFFFC)
- LEFT-TO-RIGHT EMBEDDING (LRE) (U+202A+202C)
- RIGHT-TO-LEFT EMBEDDING (LRE) (U+202B+202C)
- POP DIRECTIONAL FORMATTING (PDF) (U+202C)
- LEFT-TO-RIGHT OVERWRITE (LRO) (U+202D bis U+202C)
- RIGHT-TO-LEFT OVERWRITE (RLO) (U+202E bis U+202C)

Codierung – UTF-8 (RFC 2279→3629)

UTF-8 = UNICODE Transformation Format

```
UCS-4 range (hex.) UTF-8 octet sequence (binary)
0000 0000-0000 007F   0xxxxxxx
0000 0080-0000 07FF   110xxxxx 10xxxxxx
0000 0800-0000 FFFF   1110xxxx 10xxxxxx 10xxxxxx
0001 0000-001F FFFF   11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
0020 0000-03FF FFFF   111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
0400 0000-7FFF FFFF   1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
                      11111110 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
                                                                    10xxxxxx
```

z.B. BOM = Byte Order Mark (0xFEFF)

Unicode:	FE	FF	(16 Bits)
	1111 1110	1111 1111	
UTF-8:	1110 1111	10 11 1011	10 11 1111
	EF	BB	BF

weitere: UTF-1, UTF-7, UTF-16, UTF-32, UTF-16BE, UTF-16LE, …

Codierung – UTF-16 (RFC 2781)

- jedes Zeichen wird mittels 2 oder 4 Bytes codiert

  0x0000 – 0xFFFF: 2 Bytes (für BMP)

  0x10000 – 0x10FFFF: 4 Bytes

  (2 Bytes high surrogate area/high half zone von 0xD800 - 0xDBFF

  2 Bytes low surrogate area/low half zone von 0xDC00 - 0xDFFF)

- sei U´ = U – 0x10000 <= 0xFFFFF (also 20 Bits nötig)

  mit U´ = yyyyyyyyyyxxxxxxxxxx ➔

  1101 10yy yyyy yyyy + 1101 11xx xxxx xxxx (20 Bits frei)

- UTF-16BE beginnt mit U+FEFF (Big Endian)
- UTF-16LE beginnt mit U+FFFE (Little Endian)

Codierung – UTF-32

- jedes UNICODE Zeichen wird mittels 4 Bytes codiert

- UTF-32BE beginnt mit 00 00 FE FF (Big Endian)
- UTF-32LE beginnt mit FF FE 00 00 (Little Endian)

"Hardcopy versions of the Unicode Standard have been among the most crucial and most-heavily used reference books in my personal library for years. Unicode allows me to celebrate the fact that computer science is a vast worldwide collaboration. And Unicode is perhaps the best tool I know to help bring understanding between people of different cultures."
—Donald E. Knuth, Professor Emeritus of The Art of Computer Programming
Stanford University

wichtig
↳ riesig, gewaltig, weit, enorm

Codierung – BASE64 (RFC 3548/4648)

Wert	Zeichen	Wert	Zeichen	Wert	Zeichen	Wert	Zeichen
0	A	17	R	34	i	51	z
1	B	18	S	35	j	52	0
2	C	19	T	36	k	53	1
3	D	20	U	37	l	54	2
4	E	21	V	38	m	55	3
5	F	22	W	39	n	56	4
6	G	23	X	40	o	57	5
7	H	24	Y	41	p	58	6
8	I	25	Z	42	q	59	7
9	J	26	a	43	r	60	8
10	K	27	b	44	s	61	9
11	L	28	c	45	t	62	+
12	M	29	d	46	u	63	/
13	N	30	e	47	v		
14	O	31	f	48	w	(pad)	=
15	P	32	g	49	x		
16	Q	33	h	50	y		

Codierung – BASE32 (RFC 3548/4648)

BASE-32-Alphabet

Value	Encoding	Value	Encoding	Value	Encoding	Value	Encoding
0	A	9	J	18	S	27	3
1	B	10	K	19	T	28	4
2	C	11	L	20	U	29	5
3	D	12	M	21	V	30	6
4	E	13	N	22	W	31	7
5	F	14	O	23	X		
6	G	15	P	24	Y	(pad)	=
7	H	16	Q	25	Z		
8	I	17	R	26	2		

BASE-16-Hex-Alphabet

Value	Encoding	Value	Encoding
0	0	8	8
1	1	9	9
2	2	10	A
3	3	11	B
4	4	12	C
5	5	13	D
6	6	14	E
7	7	15	F

Codierung – BASE85 (RFC 1924)

(RFC 1924 - A Compact Representation of IPv6 Addresses-1st April 1996
Funny RFC – FYE = For your enjoyment)

```
BASE-85-Alphabet (0 .. 84)

'0'..'9', 'A'..'Z', 'a'..'z',
'!', '#', '$', '%', '&', '(',
')', '*', '+', '-', ';', '<',
'=', '>', '?', '@', '^', '_',
'`', '{', '|', '}', and '~'
```

Codierung – basE91

(entw. von Joachim Henke unter BSD-Lizenz)

0	A	0x41	13	N	0x4E	26	a	0x61	39	n	0x6E	52	0	0x30	65	%	0x25	78	>	0x3E
1	B	0x42	14	O	0x4F	27	b	0x62	40	o	0x6F	53	1	0x31	66	&	0x26	79	?	0x3F
2	C	0x43	15	P	0x50	28	c	0x63	41	p	0x70	54	2	0x32	67	(	0x28	80	@	0x40
3	D	0x44	16	Q	0x51	29	d	0x64	42	q	0x71	55	3	0x33	68	)	0x29	81	[	0x5B
4	E	0x45	17	R	0x52	30	e	0x65	43	r	0x72	56	4	0x34	69	*	0x2A	82	]	0x5D
5	F	0x46	18	S	0x53	31	f	0x66	44	s	0x73	57	5	0x35	70	+	0x2B	83	^	0x5E
6	G	0x47	19	T	0x54	32	g	0x67	45	t	0x74	58	6	0x36	71	,	0x2C	84	_	0x5F
7	H	0x48	20	U	0x55	33	h	0x68	46	u	0x75	59	7	0x37	72	.	0x2E	85	`	0x60
8	I	0x49	21	V	0x56	34	i	0x69	47	v	0x76	60	8	0x38	73	/	0x2F	86	{	0x7B
9	J	0x4A	22	W	0x57	35	j	0x6A	48	w	0x77	61	9	0x39	74	:	0x3A	87	|	0x7C
10	K	0x4B	23	X	0x58	36	k	0x6B	49	x	0x78	62	!	0x21	75	;	0x3B	88	}	0x7D
11	L	0x4C	24	Y	0x59	37	l	0x6C	50	y	0x79	63	#	0x23	76	<	0x3C	89	~	0x7E
12	M	0x4D	25	Z	0x5A	38	m	0x6D	51	z	0x7A	64	$	0x24	77	=	0x3D	90	"	0x22

Codierung – BASE-128

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
"!"	"Ð"	"#"	"$"	"%"	"&"	"'"	"("	")"	"+"	","	"-"	"½"	"/"	"0"	"1"
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
"2"	"3"	"4"	"5"	"6"	"7"	"8"	"9"	":"	";"	"<"	"="	">"	"?"	"@"	"A"
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
"B"	"C"	"D"	"E"	"F"	"G"	"H"	"I"	"J"	"K"	"L"	"M"	"N"	"O"	"P"	"Q"
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
"R"	"S"	"T"	"U"	"V"	"W"	"X"	"Y"	"Z"	"["	"÷"	"]"	"^"	"_"	"`"	"a"
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
"b"	"c"	"d"	"e"	"f"	"g"	"h"	"i"	"j"	"k"	"l"	"m"	"n"	"o"	"p"	"q"
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
"r"	"s"	"t"	"u"	"v"	"w"	"x"	"y"	"z"	"{"	"|"	"}"	"~"	"€"	"¾"	"ſ"
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
"‚"	"…"	"†"	"‡"	"ˆ"	"‰"	"Š"	"‹"	"Œ"	"¥"	"£"	"¤"	"¦"	"§"	"¨"	"©"
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
"°"	"±"	"²"	"³"	"´"	"µ"	"¶"	"·"	"¸"	"¹"	"º"	"»"	"¼"	"¬"	""	"®"

Codierung - Punycode

RFC3492: "Punycode: A Bootstring encoding of Unicode for Internationalized Domain Names in Applications (IDNA)“
März 2003

Ziel: Codierung von INTERNET-Domains mit Umlauten oder internationalen Zeichen (37 Punycodezeichen= a..z,0..9,-)

Bsp: xn--mbert-jua.de (Punycode-IDN)
xn--gaszhler-3za
xn--rhrlffel-q4a0d

Punycodekonverter: http://idnaconv.phlymail.de/

```
# XML-Umlaute in ANSI-Umlaute
$_ =~s/Ã¤/ä/g;
$_ =~s/Ã¶/ö/g;
$_ =~s/Ã¼/ü/g;
$_ =~s/ÃŸ/ß/g;
$_ =~s/Ã„/Ä/g;
$_ =~s/Ã–/Ö/g;
$_ =~s/Ãœ/Ü/g;
$_ =~s/\&quot\;/\"/g;
```

```
# XML-Umlaute in HTML-Umlaute
$_ =~s/Ã¤/\&auml\;/g;
$_ =~s/Ã¶/\&ouml\;/g;
$_ =~s/Ã¼/\&uuml\;/g;
$_ =~s/ÃŸ/\&szlig\;/g;
$_ =~s/Ã„/\&Auml\;/g;
$_ =~s/Ã–/\&Ouml\;/g;
$_ =~s/Ãœ/\&Uuml\;/g;
$_ =~s/\&quot\;/\"/g;
```

```
# DOS-Umlaute zu Windows-Umlaute
$_ =~tr/„” áŽ™š/äöüßÄÖÜ/;
# Windows-Umlaute zu DOS-Umlaute
$_ =~tr/äöüßÄÖÜ/„” áŽ™š/;
```

```
# HTML-Umlaute zu Windows-Umlaute
$_ =~s/\&auml\;/ä/g;
$_=~s/\&ouml\;/ö/g;
$_ =~s/\&uuml\;/ü/g;
$_ =~s/\&szlig\;/ß/g;
$_ =~s/\&Auml\;/Ä/g;
$_ =~s/\&Ouml\;/Ö/g;
$_ =~s/\&Uuml\;/Ü/g;
$_ =~s/\&quot\;/\"/g;
```

```
# Windows-Umlaute zu HTML-Umlaute
$_ =~s/ä/\&auml\;/g;
$_ =~s/ö/\&ouml\;/g;
$_ =~s/ü/\&uuml\;/g;
$_ =~s/ß/\&szlig\;/g;
$_ =~s/Ä/\&Auml\;/g;
$_ =~s/Ö/\&Ouml\;/g;
$_ =~s/Ü/\&Uuml\;/g;
$_ =~s/\"/\&quot\;/g;
```

```
# Mail-Zeichenkonvertierung
$_ =~s/=E4/ä/g;
$_ =~s/=F6/ö/g;
$_ =~s/=FC/ü/g;
$_ =~s/=DF/ß/g;
$_ =~s/=C4/Ä/g;
$_ =~s/=D6/Ö/g;
$_ =~s/=DC/Ü/g;
$_ =~s/=20/ /g;
$_ =~s/=09/\t/g;  #Tabulator
$_ =~s/=2E/./g;   #Punkt
$_ =~s/=3D/=/g;
$_ =~s/=B0/°/g;   #Grad
$_ =~s/=B4/´/g;
$_ =~s/=A7/§/g;
```

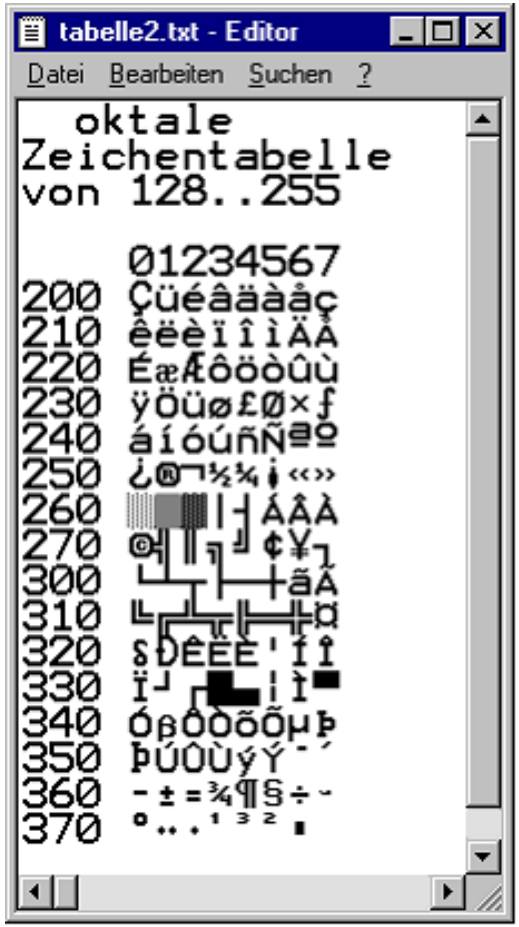
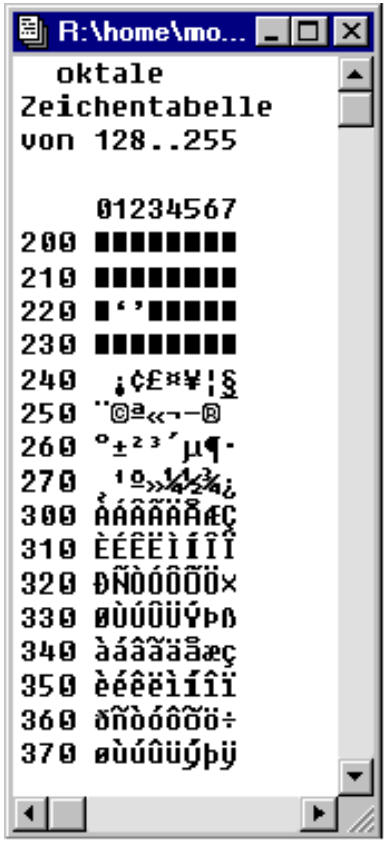


tabelle2.txt - Editor
Datei Bearbeiten Suchen ?
oktale
Zeichentabelle
von 128..255
01234567
200 Çüéâäàåç
210 êëèïîìÄÅ
220 ÉæÆôöòûù
230 ÿÖÜø£Ø×ƒ
240 áíóúñÑªº
250 ¿®¬½¼¡«»
260 ▒▓│┤ÁÂÀ
270 ©╣║╗╝¢¥┐
300 └┴┬├─┼ãÃ
310 ╚╔╩╦╠═╬¤
320 ðÐÊËÈ'ÍÎÏ
330 Ï┘┌▄¦Ì▀
340 ÓßÔÒõÕµÞ
350 ÞÚÛÙýÝ¯´
360 -±=¾¶§÷¸
370 °¨·¹³²■


R:\home\mo...
oktale
Zeichentabelle
von 128..255
01234567
200 ■■■■■■■■
210 ■■■■■■■■
220 ■‘’■■■■■
230 ■■■■■■■■
240 ¡¢£¤¥¦§
250 ¨©ª«¬¬-®
260 °±²³´µ¶·
270 ¸¹º»¼½¾¿
300 ÀÁÂÃÄÅÆÇ
310 ÈÉÊËÌÍÎÏ
320 ÐÑÒÓÔÕÖ×
330 ØÙÚÛÜÝÞß
340 àáâãäåæç
350 èéêëìíîï
360 ðñòóôõö÷
370 øùúûüýþÿ


Eingabeaufforderung
C:\>type tabelle2.txt
oktale
Zeichentabelle
von 128..255
01234567
200 Çüéâäàåç
210 êëèïîìÄÅ
220 ÉæÆôöòûù
230 ÿÖÜø£Ø×ƒ
240 áíóúñÑªº
250 ¿®¬½¼¡«»
260 ▒▓│┤ÁÂÀ
270 ©╣║╗╝¢¥┐
300 └┴┬├─┼ãÃ
310 ╚╔╩╦╠═╬¤
320 ðÐÊËÈ'ÍÎÏ
330 Ï┘┌▄¦Ì▀
340 ÓßÔÒõÕµÞ
350 ÞÚÛÙýÝ¯´
360 -±=¾¶§÷¸
370 °¨·¹³²■