Thomas Staub

Von ASCII zu Unicode

Anmerkung: Der Inhalt diese Dokuments stammt nicht ausschliesslich von mir, sondern ist ein zusammenschnitt verschiedenster Unterlagen. Hauptsächlich Wikipedia.

Inhalt

ASCII

Die beiden Grundeinheiten in jedem heutigen Computer sind die Einheiten **Bit** und **Byte**. Ein Byte ist bei den heute üblichen Systemen als Folge von 8 Bit definiert (man spricht auch von **Oktetts**). Da jedes Bit zwei Zustände haben kann, nämlich 0 oder 1 bzw. ja oder nein, lassen sich mit einer Folge von 8 Bit genau 256 (= 2^8) unterschiedliche Zustände realisieren. Ein Byte kann also 256 unterschiedliche Werte haben. Da im Computer immer auch die 0 dazugehört, können in einem Byte dezimal ausgedrückt Werte zwischen 0 und 255 stehen.

Wenn ein laufendes Programm im Computer eine Datei in den Arbeitsspeicher einliest, stehen im Arbeitsspeicher anschließend nur Byte-Werte. Von Zeichen unseres Alphabets ist auf dieser Ebene noch keine Rede. Damit aus den Byte-Werten lesbare Zeichen werden, die sich am Bildschirm darstellen lassen, braucht es eine Konvention, welches Zeichen mit welchem oder welchen Byte-Werten gespeichert wird. Diese Aufgabe haben die so genannten **Zeichenkodierungen**. Eine solche Zeichenkodierung greift auf eine Übersetzungstabelle (**Codetabelle**) zurück, die zunächst jedem Zeichen, das verwendet werden kann, eine fortlaufende Nummer (einen **Code**) zuweist. Die Menge der Zeichen in einer solchen Tabelle wird **Zeichenvorrat** genannt.

Die Kodierungen sowie deren Codetabellen sind EDV-historisch gewachsene Gebilde. Bis zum Aufkommen der Personal Computer benutzten viele Rechner noch 7 Bit lange Grundeinheiten, mit denen sich nur 128 unterschiedliche Zustände darstellen lassen. Noch früher waren es nur 6 und 5 Bit lange Grundeinheiten. Auf den 7 Bit langen Grundeinheit beruhten die ersten Kodierungen, die historisch den Durchbruch schafften: z.B. Die ASCII-Kodierung (American Standard Code for Information Interchange) Sie setzte sich vor allem durch, weil sie im erfolgreichen Unix-Betriebssystem und in den aufkommenden Personal Computern zum Einsatz kam.

	0	1	2	3	4	5	6	7
0	NUL	DLE	space	0	@	P	`	p
1	SOH	DC1 XON	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3 XOFF	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	del

In der ASCII-Codetabelle sind die ersten 32 Zeichen für Steuerzeichen reserviert, etwa für Tastatur-Impulse wie den Zeilenumbruch. Die Zeichen zwischen 32 und 127 sind darstellbare Zeichen, darunter alle Ziffern, Satzzeichen und Buchstaben, die ein Amerikaner so braucht (denn die ASCII-Kodierung kommt natürlich aus den USA). Das Umwandeln der Zeichen in Einsen und Nullen, also die eigentliche Kodierung, funktionierte einfach: Jedes Zeichen nahm bei der Speicherung genau 7 Bits in Anspruch und der binäre Zahlenwert dieser 7 Bits entsprach der Nummer des Zeichens in der ASCII-Codetabelle. Der lateinische Buchstabe "a"; beispielsweise hat in der ASCII-Codetabelle den dezimalen Wert 97, er wurde daher ASCII-kodiert als 1100001 gespeichert.

Lange Zeit war ASCII der einzige verbreitete Standard. Da die neueren Computer aber 8 Bit lange Grundeinheiten hatten, war es folgerichtig, für die Byte-Werte zwischen 128 und 255 neue Verwendungszwecke zu finden. Dabei entwickelten sich jedoch proprietäre Lösungen. Microsoft DOS beispielsweise benutzt eine "erweiterte" ASCII-Codetabelle - dies ist aber nicht viel mehr als eine schöne Umschreibung für die Microsoft-eigene Belegung der Zeichen 128 bis 255 speziell für die Bedürfnisse von MS DOS.

iso-8859-1

+	0	1	2	3	4	5	6	7	8	9
160		¡	¢	£	¤	¥	¦	§	¨	©
170	ª	«	¬	-	®	¯	°	±	²	³
180	´	µ	¶	·	¸	¹	º	»	¼	½
190	¾	¿	À	Á	Â	Ã	Ä	Å	Æ	Ç
200	È	É	Ê	Ë	Ì	Í	Î	Ï	Ð	Ñ
210	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û
220	Ü	Ý	Þ	ß	à	á	â	ã	ä	å
230	æ	ç	è	é	ê	ë	ì	í	î	ï
240	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù
250	ú	û	ü	ý	þ	ÿ				

Um auch hierfür einen Standard zu schaffen, entwickelte die internationale Standardisierungs-Organisation ISO eine Reihe von Kodierungen, die sogenannte **ISO-8859-Familie**. Die Codetabellen dieser Kodierungen übernehmen für die Zeichen 0 bis 127 die ASCII-Codetabelle und definieren für die Werte zwischen 128 und 255 etliche Sonderzeichen und wichtige Alphabet Zeichen verschiedener europäischer Sprachen. Die in Mitteleuropa verbreitete Kodierung **ISO 8859-1**, auch **Latin-1** genannt, enthält etwa die deutschen Umlaute, die französischen Accent-Zeichen und spanische Zeichen mit Tilde. Dazu kommen diverse verbreitete kaufmännische und wissenschaftliche Zeichen.

In der Fachliteratur wird oft der Begriff "Zeichensatz" (englisch *character set*) benutzt, um sowohl Zeichenkodierung (engl. *character encoding*), die Zuordnungstabelle zwischen Zeichen und Zeichencode (engl. *character code*), als auch den Zeichenvorrat (engl. *character repertoire*) zu bezeichnen. Tatsächlich sind diese drei Konzepte notwendigerweise miteinander verbunden: Beispielsweise ISO 8859-1 benutzt einen Zeichenvorrat von 256 Zeichen (darunter a, b, c usw.). Diesen Zeichen werden jeweils in der zugehörigen Codetabelle Nummern zugeordnet (a = 97, b = 98, c = 99 usw.). Die Kodierung kümmert sich dann um die Speicherung dieser Nummern in Bytes (97 = 01100001, 98 = 01100010, 99 = 01100011 usw.).
Das Problem, welches aber nicht gelöst werden kann, ist: Was ist mit Sprachen mit mehr als 255 Zeichen, z.B. Chinesisch?

Unicode

Die Unicode-Kodierungen haben das Potenzial, mittelfristig die auf 256 Zeichen begrenzten Kodierungen abzulösen. Derzeit geschieht auch schon viel in dieser Richtung. Neuere Betriebssysteme bieten Schriftarten an, die den kompletten Unicode-Zeichenvorrat oder zumindest große Teile davon abdecken. Auch die meisten modernen Anwendungen können Texte mit der Unicode-Kodierung **UTF-8** speichern, wodurch ein Zeichen nicht mehr zwangsläufig genau einem Byte entspricht, sondern aus mehreren Bytes bestehen kann. Damit ist die Grundlage zur Unterstützung des Unicode-Systems gegeben.

Unicode ist ein System, in dem die Zeichen oder Elemente aller bekannten Schriftkulturen und Zeichensysteme festgehalten werden. Durch dieses System wird es möglich, einem Computer "weltweit" zu sagen, welches Zeichen man dargestellt bekommen will. Voraussetzung ist natürlich, dass der Computer bzw. das ausgeführte Programm das Unicode-System unterstützt.
Unicode strebt die möglichst vollständige Erfassung aller bekannten Zeichen aus gegenwärtigen und vergangenen Schriftkulturen an. Die Zeichen werden nach Klassen katalogisiert und erhalten eine Zeichennummer (Code). Alle nur erdenklichen Zeichen und Zeichensorten werden erfasst. Auch für Steuerzeichen wie Silbentrennzeichen, erzwungene Leerzeichen oder Tabulator-Zeichen gibt es Nummern. Die Zeichen mathematischer Formeln fehlen ebenso wenig wie die Silben- oder Wortzeichen fernöstlicher Schriftkulturen. Auch Einzelteile von Zeichen, so genannte diakritische Zeichen wie etwa die Doppelpunkte über den deutschen Umlauten, haben einen eigenen Code. Zeichen lassen sich auch dynamisch kombinieren - so gibt es zwar natürlich auch ein deutsches "ä", aber der gleiche Buchstabe lässt sich auch aus "a" und dem Element für Doppelpunkt über dem Zeichen erzeugen.
Neben der bloßen Adressierung eines Zeichens oder Elements ist im Unicode-System für jedes Zeichen auch ein Set von Eigenschaften definiert. Zur Eigenschaft eines Zeichens gehört z.B. die Schreibrichtung (bei arabischen Zeichen etwa ist die Schreibrichtung von rechts nach links). Insgesamt stecken hinter dem Unicode-System unzählige Forschungsergebnisse der weltweiten Sprachwissenschaft.
Das Unicode-Konsortium, das 1991 gegründet wurde und aus Linguisten und anderen Fachleuten besteht, ermittelt die aufzunehmenden Zeichen. Die vergebenen Zeichencodes haben verbindlichen Charakter. Seit Version 2.0 ist das Unicode-System auch mit der internationalen Norm ISO/IEC 10646 synchronisiert. Das ist insofern wichtig, als HTML seit Version 4.0 und auch XML ab Version 1.0 auf der Norm ISO/IEC 10646 aufsetzen. Wenn Sie also wissen wollen, wie man ein bestimmtes Zeichen in HTML oder XML notieren soll, müssen Sie in den Unicode-Zeichentabellen nachsehen, welche Zeichennummer das gewünschte Zeichen hat. Anschließend können Sie das gewünschte Zeichen durch eine numerische Notation wie z.B. `&#9871;` (dezimale Schreibweise) oder `&#x268F;` (hexadezimale Schreibweise mit `x`) im Quelltext der HTML- oder XML-Datei notieren.

Der Zeichenumfang ist dazu in 17 Ebenen (englisch planes) gegliedert, welche jeweils 2^{16} = 65.536 Zeichen umfassen. Sechs dieser Ebenen werden bereits verwendet, die restlichen sind für spätere Nutzung reserviert. Hier eine kleine Übersicht dieser Planes.

- Die Basic Multilingual Plane BMP auch als Plane 0 bezeichnet: Sie enthält hauptsächlich Schriftsysteme, die aktuell in Gebrauch sind.
- Die Supplementary Multilingual Plane SMP auch als Plane 1 bezeichnet: Sie enthält vor allem historische Schriftsysteme, aber auch größere Ansammlungen an Zeichen, die selten in Gebrauch sind, wie z. B. Domino- und Mahjonggsteine und Emoji.
- Die Supplementary Ideographic Plane SIP auch als Plane 2 bezeichnet: Sie enthält ausschließlich chinesische, japanische und koreanische Schriftzeichen, die selten benutzt werden.
- Plane 3 ist ebenfalls dafür reserviert.
- Die Supplementary Special-purpose Plane SSP auch als Plane 14 bezeichnet: Sie enthält einige wenige Kontrollzeichen zur Sprachmarkierung.
- Die letzten beiden Ebenen, jeweils Supplementary Private Use Area-A und -B (PUA; auch Plane 15 und Plane 16), stehen als privat nutzbare Bereiche zur Verfügung. Sie werden teilweise auch als Private Use Planes (PUP) bezeichnet.

Aufbau des Unicode-Systems

Bei neuen Unicode-Versionen wird das Buch The Unicode Standard, herausgegeben vom Unicode-Konsortium, neu aufgelegt. Im internationalen Buchhandel ist dieses Buch erhältlich. Darin sind alle Zeichen, Zeichennummern, Zeichenklassen usw. genau aufgeschlüsselt und dargestellt. Dieses Buch ist das verbindliche Normwerk. Auf den Web-Seiten des Unicode-Konsortiums finden sich zum schnellen Nachschlagen PDF-Dateien mit den einzelnen Codetabellen. Das Unicode-System ist in Zahlenbereiche aufgeteilt. Die Zahlen selbst werden in der Form U+XXXX notiert. Das U steht für Unicode, und die X für je eine hexadezimale Ziffer. Zeichennummern sind in diesen Tabellen also hexadezimal dargestellt.
Die einzelnen Zeichen im Unicode-System sind nicht wahllos angeordnet. Das gesamte System ist in Zeichenbereiche (engl. *blocks*, Blöcke) aufgeteilt. Die Zeichenbereiche spiegeln jeweils eine bestimmte Schriftkultur oder ein Set von Sonderzeichen wider.

Unter Windows (ab Windows 2000) kann in einigen Programmen (genauer in RichEdit-Feldern) der Code dezimal als Alt+<dezimales Unicode> (bei eingeschaltetem Num-Lock) auf dem numerischen Tastaturfeld eingegeben werden. Dabei ist jedoch zu beachten, dass Zeichennummern kleiner als 1000 um eine führende Null zu ergänzen sind (z. B. Alt+0234 für Codepoint 234_{10} [ê]). Diese Maßnahme ist notwendig, da die (immer noch in Windows verfügbare) Eingabemethode Alt+<ein- bis dreistellige dezimale Zeichennummer ohne führende Null> bereits in MS-DOS-Zeiten genutzt wurde, um die Zeichen der Codepage 850 (v. a. bei früheren MS-DOS-Versionen auch Codepage 437) einzugeben.

Jedes im Unicode-Standard codierte elementare Zeichen ist einem Codepunkt (engl. code points) zugeordnet. Diese werden üblicherweise hexadezimal (mindestens vierstellig, d. h. ggf. mit führenden Nullen) und mit einem vorangestellten U+ dargestellt, z. B. U+0041 für das A.

	Markierung	Ebene	Zeichen	Unicode	In Windows (Dezimal)
	U+	00 - 10	0000 - FFFF		Alt+
Bsp: A	U+	00	0041	U+0041	65
Bsp: ∞	U+	00	221E	U+221E	8734
Bsp: 😼	U+	1	F640	U+1F640	128576

Der gesamte vom Unicode-Standard beschriebene Bereich umfasst **1.114.112** Codepunkte (U+0000 … U+10FFFF, 17 Ebenen zu je 2^16, d. h. 65536 Zeichen). Das dürfte noch für sehr lange ausreichen.

UTF (Unicode Transformation Format)

Es gibt folgende verwendete UTF Codierungen: (Kurzübersicht)

Codierung	Beschreibung
UTF-8	Die am häufigsten verwendete Codierung. (HTML, Mail) Die ersten 128 Zeichen entsprechen ASCII Variable Länge 1-4 Byte Platzsparend beim lateinischen Alphabet
UTF-16	Die älteste Abbildungsvariante von Unicode Es werden 2 Byte (16 Bit) zur Codierung verwendet (teilweise auch 4 Byte) Teilweise kürzer als UTF-8 (bei nicht lateinischen Alphabeten) Es gibt 2 Varianten (Little und Big Endian)
UTF-32	Fixe Länge von 32 Bit (4 Byte) Braucht immer am meisten Speicherplatz Einfach zu gebrauchen, da fixe Länge Es gibt 2 Varianten (Little und Big Endian)

UTF-8 Codierung

UTF-8 ist die am weitesten verbreitete Kodierung für Unicode-Zeichen. UTF-8 ist in den ersten 128 Zeichen deckungsgleich mit ASCII und eignet sich mit in der Regel nur einem Byte Speicherbedarf für Zeichen vieler westlicher Sprachen besonders für die Kodierung englischsprachiger Texte, die sich im Regelfall ohne Modifikation daher sogar mit nicht-UTF-8-fähigen Texteditoren ohne Beeinträchtigung bearbeiten lassen.

In anderen Sprachen ist der Speicherbedarf in Byte pro Zeichen größer, wenn diese vom ASCII-Zeichensatz abweichen: Bereits die deutschen Umlaute erfordern zwei Byte;

kyrillische, fernöstliche und Sprachen aus dem afrikanischen Raum belegen bis zu 4 Byte je Zeichen.

Bei der UTF-8-Kodierung wird jedem Unicode-Zeichen eine speziell kodierte Zeichenkette variabler Länge zugeordnet. Dabei unterstützt UTF-8 Zeichenketten bis zu einer Länge von vier Byte, auf die sich – wie bei allen UTF-Formaten – alle Unicode-Zeichen abbilden lassen.

Schauen wir uns den Aufbau genauer an:

Beispiel mit UTF-8 mit 7 Bit Zeichen (ASCII). In diesem Fall stellt UTF eine führende 0 voran. Das bedeutet, dass die Zeichenkette 1 Byte lang ist.

Unicode Zeichen	UTF-8 Codierung	Verwendungszweck
0000 0000 - 0000 $007F_{16}$ ($7F_{16}=127_{10}$)	$0xxx\ xxxx_2$	UTF-8 gleich wie ASCII Zeichensatz 0-127
0000 0041_{16} $100\ 0001_2$	$0100\ 0001_2$	Bsp: A (7 Bit)

Beispiel mit UTF-8 mit 11 Bit Länge. In diesem Fall stellt UTF eine führende 110 voran. Das bedeutet, dass die Zeichenkette 2 Byte lang ist. Das zweite Byte wird mit 10 eingeleitet. Die Bits werden von rechts aufgefüllt und allenfalls mit führenden Nullen ergänzt.

Unicode Zeichen	UTF-8 Codierung	Verwendungszweck
0000 0080 - 0000 $07FF_{16}$ ($80_{16}=128_{10}$; $7FF_{16}=2047_{10}$ = $111\ 1111\ 1111_2$)	$110x\ xxxx\ 10xx\ xxxx_2$	Unicode Zeichen wird von rechts her aufgefüllt.
0000 $03A9_{16}$ 1110101001_2	$1100\ 1110\ \ 1010\ 1001_2$ $CEA9_{16}$	Bsp: Ω (Alt+937)

Beispiel mit UTF-8 mit 16 Bit Länge. In diesem Fall stellt UTF eine führende 1110 voran. Das bedeutet, dass die Zeichenkette inkl. Codierung 3 Byte lang ist. Das zweite und letzte Byte wird mit 10 eingeleitet. Die Bits werden von rechts aufgefüllt und allenfalls mit führenden Nullen ergänzt.

Unicode Zeichen	UTF-8 Codierung	Verwendungszweck
0000 0800 - 0000 $FFFF_{16}$ ($800_{16}=2048_{10}$; $7FF_{16}=65535_{10}$ $=1111\ 1111\ 1111\ 1111_2$)	$1110\ xxxx\ 10xx\ xxxx\ 10xx\ xxxx_2$	Unicode Zeichen wird von rechts her aufgefüllt
0000 $221E_{16}$ $10\ 0010\ 0001\ 1110_2$	$1110\ 0010\ 1000\ 1000\ 1001\ 1110_2$ $E2889E_{16}$	Bsp: ∞(Alt+937)

Beispiel mit UTF-8 mit 16 Bit Länge. In diesem Fall stellt UTF eine führende 1110 voran. Das bedeutet, dass die Zeichenkette inkl. Codierung 3 Byte lang ist. Die drei folgenden Bytes werden mit 10 eingeleitet. Die Bits werden von rechts aufgefüllt und allenfalls mit führenden Nullen ergänzt.

Unicode Zeichen	UTF-8 Codierung	Verwendungszweck
0001 0000 - 0010 FFFF$_{16}$ (10000_{16}=65535_{10}; $10FFFF_{16}$= 1'114'111_{10})	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx $_2$	Unicode Zeichen wird von rechts her aufgefüllt
0001 F640$_{16}$ 11111011001000000$_2$	1111 0000 1001 1111 1001 1001 1000 0000$_2$ F09F9980$_{16}$	Bsp: 😈 (Alt+128576)

Probieren Sie es aus. Speichern Sie diese Zeichen mit Notepad++ und der Kodierung UTF-8 ohne BOM. Öffnen Sie die Datei in einem Hex Editor. Stimmt die Theorie mit der Praxis überein?

Als Byte Order Mark (BOM; deutsch Bytereihenfolge-Markierung) wird das Unicode-Zeichen U+FEFF (englisch zero width non-breaking space) am Anfang eines Datenstroms bezeichnet, wo es als Kennung zur Definition der Byte-Reihenfolge und Kodierungsform in Unicode-Zeichenketten, insbesondere Textdateien, verwendet wird.

UTF-16 Codierung

Bei der UTF-16-Kodierung wird jedem Unicode-Zeichen eine speziell kodierte Bytekette von zwei oder vier Byte Länge zugeordnet, so dass sich wie auch bei den anderen UTF-Formaten alle Unicode-Zeichen abbilden lassen.

Während UTF-8 eine zentrale Bedeutung in Internet-Protokollen hat, wird UTF-16 vielerorts zur internen Zeichenkettenrepräsentation verwendet, beispielsweise in aktuellen Versionen von Java.

Die Herstellung der UTF-16 Codierung funktioniert anders als bei UTF-8. Es gibt unterschiedliche Codierungen je nachdem in welcher *Ebene* sich das Zeichen befindet.

Bei der Basic Multilingual Plane BMP (Plane 0) wird der Unicode Wert direkt in 2 Bytes gespeichert.

Unicode Zeichen	UTF-16 Codierung (Bits)	
0000 0000 - 0000 $FFFF_{16}$ ($7FF_{16}=65535_{10}$ = $1111\ 1111\ 1111\ 1111_2$)	xxxx xxxx xxxx xxxx	Unicode Ebene 0 -> 2 Byte
$0000\ 0041_{16}$ $0100\ 0001_2$	0041_{16} $0000\ 0000\ 0100\ 0001_2$	Bsp: A (Ebene 0)
$0000\ 20AC_{16}$ $00100000\ 10101100_2$	$20AC_{16}$ $00100000\ 10101100_2$	Bsp: € (Ebene 0)

Nun mit einem Beispiel ausserhalb der BMP.

1. Wir nehmen den Unicode des Zeichens und subtrahieren 010000_{16}

Nun bleibt je nach Plane eine Maximal 20bit Nummer übrig.

2. Die höheren 10 Bits werden mit D80016 (11011000000000002) addiert. Diese 16 Bit werden als high surrogate bezeichnet.
3. Die tieferen 10 Bits werden mit DC0016 (11011100000000002) addiert. Diese 16 Bit werden als low surrogate bezeichnet.

Diese Addition entspricht wie bei UTF-8 einer Einleitung der Bitfolge.

Je nachdem, welches der beiden Bytes zuerst übertragen bzw. gespeichert wird, spricht man von *Big Endian* (UTF-16BE) oder *Little Endian* (UTF-16LE).
Bei Big Endian wird das höchstwertige Bit zuerst gespeichert. Bei Little Endian hingegen wird das tieferwertige Bit zuerst gespeichtert.

$0001\ 0000$ - $0010\ FFFF_{16}$ ($10000_{16}=65535_{10}$; $10FFFF_{16}$= $1'114'111_{10}$) Rechne: - $0001\ 0000_{16}$	1101 10xx xxxx xxxx 1101 11xx xxxx xxxx	Unicode Ebene 1 Berechnung: Unicode - 10000_{16} Aufteilung auf 2 Byte
Unicode: $0001\ F36B_{16}$ - $0001\ 0000_{16}$ $F36B_{16}$ $0000\ 1111\ 00\ 11\ 0110\ 1011_2$	$1101\ 1000\ 0011\ 1100_2$ $1101\ 1111\ 0110\ 1011_2$ entspricht: $D83C\ DF6B_{16BE}$ entspricht: $3CD8\ 6BDF_{16LE}$	Bsp: Schokolade (Alt+127851)
Unicode: $0002\ 4B62_{16}$ - $0001\ 0000_{16}$ $1\ 4B62_{16}$ $0001\ 0100\ 10\ 11\ 0110\ 0010_2$	$1101\ 1000\ 0101\ 0010_2$ $1101\ 1111\ 0110\ 0010_2$ entspricht:$D852DF62_{16BE}$ entspricht: $52D8\ 62DF_{16LE}$	Bsp:𠮢 (Alt+150370)

UTF32-Codierung

UTF-32 ist eine Methode zur Kodierung von Unicode-Zeichen, bei der jedes Zeichen mit vier Byte (32 Bit) kodiert wird. Sie kann deshalb als die einfachste Kodierung bezeichnet werden, da alle anderen UTF-Kodierungen variable Bytelängen benutzen. Die Herstellung der UTF-32 Codierung ist denkbar einfach, es wird einfach der Code aus Unicode übernommen. Die Länge ist fix auf 32 Bits definiert.

Der entscheidende Nachteil von UTF-32 ist der hohe Speicherbedarf. Bei Texten, die überwiegend aus lateinischen Buchstaben bestehen, wird – verglichen mit dem verbreiteten UTF-8- oder den ISO-8859-Zeichensätzen – etwa der vierfache Speicherplatz belegt. Deshalb wird es auch kaum zum externen Speichern verwendet. Ein weiterer Nachteil ist die fehlende Abwärtskompatibilität zu ASCII, wie sie z. B. mit UTF-8 gegeben ist.

Unicode Zeichen	UTF-32 Codierung (Bits)	
$0000\ 0041_{16}$ $0100\ 0001_2$	$0000\ 0041_{16}$ $0000\ 0000\ \ 0000\ 0000\ 0000\ 0000\ 0100\ 0001_2$	Bsp: A (Ebene 0)
Unicode: $0001\ F36B_{16}$ $1\ 1111\ 0011\ 0110\ 1011_2$	$0000\ 0000\ 0000\ 0001\ 1111\ 0011\ 0110\ 1011_2$	Bsp:  Schokolade

Zusammenfassung

UTF-8

Unicode Zeichen	UTF-8 Codierung	Verwendungszweck
0000 0000 - 0000 $007F_{16}$ ($7F_{16}=127_{10}$)	$0xxx\ xxxx_2$	UTF-8 gleich wie ASCII Zeichensatz 0-127
$0000\ 0041_{16}$ $100\ 0001_2$	$0100\ 0001_2$	Bsp: A (7 Bit)
0000 0080 - 0000 $07FF_{16}$ ($80_{16}=128_{10}$; $7FF_{16}=2047_{10}$ = $111\ 1111\ 1111_2$)	$110x\ xxxx\ 10xx\ xxxx_2$	Unicode Zeichen wird von rechts her aufgefüllt.
$0000\ 03A9_{16}$ 1110101001_2	$1100\ 1110\ \ 1010\ 1001_2$ $CEA9_{16}$	Bsp: Ω (Alt+937)
0000 0800 - 0000 $FFFF_{16}$ ($800_{16}=2048_{10}$; $7FF_{16}=65535_{10}$ $=1111\ 1111\ 1111\ 1111_2$)	$1110\ xxxx\ 10xx\ xxxx\ 10xx\ xxxx_2$	Unicode Zeichen wird von rechts her aufgefüllt
$0000\ 221E_{16}$ $10\ 0010\ 0001\ 1110_2$	$1110\ 0010\ 1000\ 1000\ 1001\ 1110_2$ $E2889E_{16}$	Bsp: ∞(Alt+937)
0001 0000 - 0010 $FFFF_{16}$ ($10000_{16}=65535_{10}$; $10FFFF_{16}$= 1'114'111_{10})	$11110xxx\ 10xxxxxx\ 10xxxxxx\ 10xxxxxx_2$	Unicode Zeichen wird von rechts her aufgefüllt
$0001\ F640_{16}$ 11110110010000002	$1111\ 0000\ \ 1001\ 1111\ \ 1001\ 1001\ 1000\ 0000_2$ $F09F9980_{16}$	Bsp: 😈 (Alt+128576)

UTF-16

Unicode Zeichen	UTF-16 Codierung (Bits)	
0000 0000 - 0000 FFFF$_{16}$ ($7FF_{16}$=65535_{10} = 1111 1111 1111 1111_2)	xxxx xxxx xxxx xxxx	Unicode Ebene 0 -> 2 Byte
0000 0041_{16} $0100\ 0001_2$	0041_{16} $0000\ 0000\ 0100\ 0001_2$	Bsp: A (Ebene 0)
0000 $20AC_{16}$ $00100000\ 10101100_2$	$20AC_{16}$ $0010\ 0000\ 10101100_2$	Bsp: € (Ebene 0)
0001 0000 - 0010 FFFF$_{16}$ (10000_{16}=65535_{10}; $10FFFF_{16}$= 1'114'111_{10}) Rechne: - 0001 0000_{16}	1101 10xx xxxx xxxx 1101 11xx xxxx xxxx	Unicode Ebene 1 Berechnung: Unicode - 10000_{16} Aufteilung auf 2 Byte
Unicode: 0001 $F36B_{16}$ - 0001 0000_{16} $F36B_{16}$ $0000\ 1111\ 00\ 11\ 0110\ 1011_2$	$1101\ 1000\ 0011\ 1100_2$ $1101\ 1111\ 0110\ 1011_2$ entspricht: D83C $DF6B_{16BE}$ entspricht: 3CD8 6BDF $_{16LE}$	Bsp: Schokolade (Alt+127851)
Unicode: 0002 $4B62_{16}$ - 0001 0000_{16} 1 $4B62_{16}$ $0001\ 0100\ 10\ 11\ 0110\ 0010_2$	$1101\ 1000\ 0101\ 0010_2$ $1101\ 1111\ 0110\ 0010_2$ entspricht:$D852DF62_{16BE}$ entspricht: 52D8 $62DF_{16LE}$	Bsp:𠮷 (Alt+150370)

UTF-32

Unicode Zeichen	UTF-32 Codierung (Bits)	
0000 0041_{16} $0100\ 0001_2$	$0000\ 0041_{16}$ $0000\ 0000\ \ 0000\ 0000\ 0000\ 0000\ 0100\ 0001_2$	Bsp: A (Ebene 0)
Unicode: 0001 $F36B_{16}$ $1\ 1111\ 0011\ 0110\ 1011_2$	$0000\ 0000\ 0000\ 0001\ 1111\ 0011\ 0110\ 1011_2$	Bsp: Schokolade