

Zeichenkodierung

Hausarbeit im Rahmen der Übung „Hilfsmittel II“ von Frau Vogel im Sommersemester 2005 am Seminar Japanologie der Fakultät für Kulturwissenschaften der Universität Tübingen

Stand: 29.06.2005
Adresse: Markus Stengel
Herrenberger Str. 22
72070 Tübingen

Inhaltsverzeichnis

1. Einleitung.....	3
2. Begriffserklärungen.....	3
2.1. Bit und Byte.....	3
2.2. Kodierung.....	3
2.3. Endian.....	3
2.4. Coded Character Set (CCS).....	4
2.5. Character Encoding Scheme (CES).....	4
2.6. Zeichenmenge (Charset).....	4
3. Probleme von Kodierungen.....	5
4. Kodierungen.....	5
4.1. Nichtjapanische Kodierungen.....	5
4.1.1. ASCII.....	5
4.1.2. Die ISO-8859-x-Normen.....	6
4.2. Japanische Kodierungen.....	7
4.3. Unicode.....	7
4.3.1. UTF-1.....	7
4.3.2. UCS-2 oder UTF-16.....	8
4.3.3. UCS-4.....	8
4.3.4. UTF-7.....	8
4.3.5. UTF-8.....	8
4.4. Kodierungen für das Internet.....	8
5. Schriftarten (Fonts).....	9
6. Literaturverzeichnis.....	10

1. Einleitung

In dieser Arbeit sollen die verschiedenen Zeichenkodierungen vorgestellt werden, wie sie unter anderem im Internet Verwendung finden.

Begonnen wird mit ein paar fundamentalen Begriffserläuterungen, die essentiell für das Verständnis der späteren Themen sind. Anschließend werden Kodierungen, ihre Probleme, nichtjapanische Kodierungen, japanische Kodierungen und Unicode vorgestellt. Abschließend wird noch kurz auf die Bedeutung der Schriftarten für die Darstellung von Text eingegangen.

2. Begriffserklärungen

2.1. Bit und Byte

Ein *Byte* besteht aus acht *Bit*¹, wobei ein Bit die kleinste Einheit ist, die durch einen Computer dargestellt werden kann. Vereinfacht gesagt: Ein Bit ist eine Variable oder Speicherstelle, die entweder den Wert 0 oder 1 annehmen kann.

Da ein Byte aus acht Bit besteht kann es den Wert einer Zahl von bis zu acht Stellen im *Binärsystem* (auch „Dualsystem“ oder „Zweiersystem“ genannt) annehmen. Umgerechnet auf das *Dezimalsystem* („Zehnersystem“) entspricht dies $2^8=256$ möglichen Zahlen, also den Zahlen von 0 bis 255.

Die Umrechnung von Zahlen aus dem Dezimalsystem in das Binärsystem ist denkbar einfach: Es wird fortwährend durch Zwei geteilt. Verbleibt ein Rest, ist die entsprechende Stelle auf Eins, ansonsten auf Null zu setzen. Beispiel mit der Zahl 137: $137=2^7+2^3+2^0$, also 10001001 in Bitschreibweise. Dabei steht links das höchstwertige Bit, rechts das Bit mit dem niedrigsten Wert.

2.2. Kodierung

Mit *Kodierung* wird der Vorgang bezeichnet, der Entitäten wie z.B. Schriftzeichen in eine für den Computer les- und verarbeitbare Form überführt. Im Normalfall ist damit eine Repräsentation durch Zahlen, die wiederum in Bytes abgespeichert werden, gemeint.

2.3. Endian

Solange nur positive Zahlen kleiner als 256 abgespeichert werden müssen gibt es keine Probleme. Sollen aber größere Zahlen – die z.B. Schriftzeichen kodieren – von einem Computer verarbeitet werden können, müssen sie irgendwie durch Bytes repräsentiert werden können. Dies wird dadurch realisiert, dass die gesamte Zahl auf das Binärsystem umgerechnet und anschließend in Blöcken von jeweils acht Bit, also einem Byte, abgespeichert wird. Beispiel mit der Zahl 2953: $2953=2^{11}+2^9+2^8+2^7+2^3+2^0$, also 0000101110001001 in Bitschreibweise. Dieser Block wird nun in Einheiten von acht Bit unterteilt und nacheinander abgelegt.

¹ Warum es ausgerechnet acht und nicht mehr oder weniger Bit hat, hat historische Gründe

Doch es stellt sich die Frage, in welcher Reihenfolge dies geschehen soll: Zuerst das Byte, in dem sich die höheren Werte befinden, die so genannte *Big Endian* Anordnung, oder das Byte mit den niederwertigeren, die *Little Endian* Anordnung (vgl. Endian)? Eigentlich ist dies mehr eine philosophische Frage als eine technisch relevante. In der Praxis ist dies aber ein häufig ein großes Problem, wenn zwischen verschiedenen Computerarchitekturen Daten ausgetauscht werden sollen, da bei Unklarheit über die Anordnung Daten falsch interpretiert werden können.

2.4. Coded Character Set (CCS)

Ein *Coded Character Set* ist eine Abbildung von ganzen Zahlen auf Zeichen bzw. umgekehrt (vgl. Czyborra [2]). Vereinfacht darstellbar ist dies durch eine Tabelle:

<i>Ganze Zahl</i>	<i>Zeichen</i>	<i>Ganze Zahl</i>	<i>Zeichen</i>
...
65	A	97	a
66	B	98	b
67	C	99	c
...

Eine wichtige Eigenschaft von CCS ist in obiger Tabelle erkennbar: Groß- und Kleinbuchstaben sind – wie alle Einträge der Tabelle – unterschiedliche Entitäten für den Computer. Dementsprechend müssen z.B. bei einer Suche in einem Text ohne Unterscheidung von Groß- und Kleinbuchstaben dem Computer Informationen gegeben werden, wie er mit ihnen umzugehen hat. Beispiele für CCS sind ASCII, ISO-8859-1 und Unicode.

2.5. Character Encoding Scheme (CES)

Das *Character Encoding Scheme* legt fest, in welcher Form die Zahlen, die im CCS die Zeichen repräsentieren, gespeichert werden, also ob Big Endian oder Little Endian verwendet wird, ob es spezielle, platzsparende Varianten gibt usw. Beispiele für CES sind UCS2 und UTF-8 (vgl. Czyborra [2]).

2.6. Zeichenmenge (Charset)

Ein *Charset* besteht aus einem CCS und einem CES (vgl. Czyborra [2]). Viele verschiedene Charsets haben zwar das gleiche CCS aber ein anderes CES, z.B. haben alle Unicodevarianten das CCS Unicode und als CES UTF-8, UTF-16 usw. Oft wird deswegen das Charset synonym mit dem CES verwendet, was allerdings nicht ganz korrekt ist.

Die meisten verbreiteten Schriftsysteme sind durch bis zu 256 Zeichen abdeckbar, zum Beispiel werden für Deutsch nur 59 Zeichen benötigt: 26 Großbuchstaben, 26 Kleinbuchstaben, ä, Ä, ö, Ö, ü, Ü und ß. Da es so wenige sind, reicht für ihre Darstellung ein sehr einfaches CES, welches mit höchstens einem Byte pro Buchstaben auskommt.

Bei umfangreicheren Schriftsystemen wie z.B. dem Japanischen oder dem Chinesischen werden jedoch Tausende von Zeichen benötigt, weswegen größere Zahlen und somit auch mehr Byte zur Kodierung benötigt werden. Da nach Möglichkeit versucht wird, so wenig Speicherplatz wie nötig für die

Repräsentation zu verwenden, können die CES mitunter sehr komplex werden.

3. Probleme von Kodierungen

Die unterschiedlichen im Einsatz befindlichen Kodierungen sind die Ursache für vielfältige Probleme (vgl. Czyborra [2]). Dabei sind weniger Probleme wie die bereits zuvor aufgeführte Unterscheidung zwischen Groß- und Kleinschreibung gemeint. Schwerwiegender sind Fehlinterpretationen verwendeter Daten, wie im folgenden Beispiel, in dem die deutsche Kodierung ISO-646-DE fälschlicherweise als die amerikanische Kodierung *ISO-646-US (ASCII)* interpretiert wird:

<i>gelesener Wert</i>	<i>korrektes Charset: Zeichen</i>	<i>falsches Charset: Zeichen</i>
70	f	f
125	ü	}
114	r	r

In diesem Beispiel wird also das Wort „für“ als Zeichenfolge „f}r“ fehlinterpretiert. Auch wenn sich in diesem Fall, zumindest durch einen Menschen, die eigentliche Bedeutung noch erraten lässt, ist es bei komplizierteren Kodierungen völlig unmöglich, noch den richtigen Inhalt zu erraten, wenn sich vorher ein Fehler eingeschlichen hat.

Oftmals wurden Kodierungen so angelegt, dass sie eigentlich mehr als genug Platz bieten würden. Später stellte sich jedoch heraus, dass es doch nicht ausreicht, beispielsweise im Fall von Unicode: Als die ursprüngliche Variante UCS-2 entworfen wurde glaubte man, dass 65536 Zeichen – so viele, wie sich in zwei Bytes (daher die „2“ von UCS-2) kodieren ließ – ausreichen würde, um für die Zeichen aller Schriftsysteme Platz zu bieten. Doch dies war ein Trugschluss, weshalb UCS-4 als Nachfolger an seine Stelle tritt.

Schließlich mussten Kodierungen noch ein Kriterium erfüllen, um für die Praxis relevant zu werden: Sie mussten möglichst sparsam in Hinblick auf den Speicherbedarf sein. Zunächst erscheint es vielleicht unbedeutend, für jedes Schriftzeichen zwei oder mehr Bytes zu benötigen. Wird eine solche Kodierung jedoch tatsächlich verwendet, bedeutet es effektiv wenigstens eine Halbierung des tatsächlich nutzbaren Speicherplatzes: Alle Dateien wären von mindestens zweifacher Größe, nur die Hälfte oder weniger könnte an Daten gleichzeitig verarbeitet werden usw.

4. Kodierungen

4.1. Nichtjapanische Kodierungen

4.1.1. ASCII

Mit Ausnahmen von IBM's *EBCDIC* ist die 1968 unter der Bezeichnung *American National Standards Institute X3.4 (ANSI X3.4)* veröffentlichte *American Standard Code for Information Interchange (ASCII)* Kodierung (vgl. Czyborra [1]). Er nutzt nur sieben Bit und ist so gut wie überall im Einsatz; insbesondere ist er die Standardkodierung für sämtliche offiziellen Internetprotokolle. Er sieht wie folgt aus:

4.2. Japanische Kodierungen

Das 1976 eingeführte *JIS X 0208* CCS war das erste, das mehr als 8 Bit umfasste (vgl. Czyborra [2]). Da es somit also mehr als ein Byte benötigte, wird es auch als *Double Byte Character Set (DBCS)* bezeichnet. Da es nicht nur Japanische Schriftzeichen, sondern auch lateinische, griechische und kyrillische enthielt, kann es auch als eine Art früher *Unicode* bezeichnet werden.

Eine weitere Besonderheit dieses CCS ist, dass es in 94 Zeilen („ku“) und 94 Spalten („ten“) aufgeteilt und auf Schriftzeichen dementsprechend mittels Zeilen- und Spaltenposition zugegriffen wird, also nicht mehr über eine fortlaufende Nummer. Die Einteilung in 94 Zeilen und 94 Spalten hat ferner zur Folge, dass das CCS maximal $94 \cdot 94 = 8836$ Zeichen umfassen kann.

Es gibt viele verschiedene CES für dieses CCS: Das emailsichere *ISO-2022(-JP)*, 8-Bit-Abbildungen wie das *Extended Unix Code (EUC) EUC-JP* oder das *Shift-JIS (SJIS, CP932)* von Microsoft. Chinesische und koreanische Standardisierer folgten dem Beispiel Japans und definierten ihre eigenen Tabellen: *ISO-2022-CN, EUC-CN, ISO-2022-TW, EUC-TW, ISO-2022-KR* und *EUC-KR*.

Insbesondere das EUC verdient besondere Beachtung: Das von AT&T entwickelte CES ist der Ahne der meisten Unixumwandlungsformate und verliert heute wegen der zunehmenden Verbreitung von *Unicode* zwar immer mehr an Bedeutung, ist aber vor allem im Internet noch sehr stark verbreitet. Vorteilhaft an diesem CES ist, dass es für ASCII-Zeichen dieselben Werte verwendet wie ASCII selbst, also auch mit nur einem Byte auskommt. Werden jedoch die nicht im ASCII enthaltenen Zeichen wie z.B. Kanji verwendet, werden zwei Bytes zur Kodierung benötigt. Diese variable Länge spart zwar Speicherplatz, bringt jedoch ein Problem mit: Sollte an irgendeiner Stelle die Synchronisation nicht klappen, also ein oder mehr Zeichen verloren gehen, so werden von dieser Stelle an – im schlimmsten Fall alle – Zeichen falsch interpretiert.

4.3. Unicode

Unicode, der *Universal Code* oder offiziell *ISO-10646*, ist noch recht neu (vgl. Unicode Standard; vgl. Czyborra [2]; vgl. Unicode). Er besteht aus dem CCS „Unicode und verschiedenen CES wie *UTF-1*, *UTF-8*, *UTF-16* usw. Aufgrund seiner späten Einführung – die erste Veröffentlichung des Standards war 1991 – ist die Unterstützung allgemein noch recht dürftig. Allerdings wird sie ständig ausgebaut und dient vielen modernen Systemen wie JAVA oder Windows XP intern als Grundlage.

4.3.1. UTF-1

Das *Unicode Transformation Format 1 (UTF-1)* ist der ursprüngliche ISO-10646 Entwurf (vgl. Czyborra [2]). Im Optimalfall benötigt es nur ein Byte zur Kodierung, besitzt aber deutlich mehr Nachteile. So ist eine verlorene Synchronisation kaum wiederherstellbar, es werden also bei auch nur einer Fehlinterpretation oder Verlust auch nur eines Bytes anschließend nur noch unbrauchbare Zeichenketten dekodiert. Zudem benötigt es zur Dekodierung intern Divisionen, was es sehr langsam in der Verarbeitung macht. Die variable Länge verhindert einen direkten Zugriff auf ein Zeichen, es ist also nicht möglich, auf z.B. das 50. Zeichen eines Textes zuzugreifen, ohne alle anderen vorher zu durchlaufen und zu

dekodieren. Schließlich verwendet UTF-1 zur Kodierung von manchen Zeichen den Schrägstrich („/“), was die Verwendung von UTF-1 als Kodierung für Dateinamen in vielen Systemen unmöglich macht².

4.3.2. UCS-2 oder UTF-16

Das *Universal Character Set Version 2 (UCS-2)*, auch als *Universal Transformation Format 16 Bits (UTF-16)* bezeichnet, wurde 1993 veröffentlicht (vgl. Unicode Standard; vgl. Czyborra [2]; vgl. Unicode). Es ist ein sehr einfaches CES für das CCS Unicode, bei dem jedes Zeichen durch zwei Bytes kodiert wird; es können also 65536 Zeichen repräsentiert werden. Da stets zwei Bytes verwendet werden ist Direktzugriff auf gewünschte Zeichen möglich, dafür verschwendet es allerdings auch viel Speicherplatz im Falle von ASCII-Texten. UTF-16 ist das am weitesten verbreitete Unicode CES.

4.3.3. UCS-4

UCS-4 ist beinahe identisch zu UCS-2, nur werden anstatt zwei Byte vier Byte zur Kodierung verwendet, und das Problem der Platzverschwendung ist noch viel größer (vgl. Unicode Standard; vgl. Czyborra [2]; vgl. Unicode).

4.3.4. UTF-7

UTF-7 wurde 1994 veröffentlicht (vgl. Unicode Standard; vgl. Czyborra [2]; vgl. Unicode). Vorteilhaft an diesem CES ist, dass es mailsicher ist und gegenüber den anderen Verfahren sich für den Emailversand effizienter kodieren lässt. Allerdings ist es ein sehr kompliziertes und somit auch fehleranfälliges Verfahren, dass zudem Suchen ohne eine vorige Umwandlung in eine andere Kodierung nicht ermöglicht.

4.3.5. UTF-8

UTF-8, auch unter den Bezeichnungen *UTF-2* und *Filesystem-Safe UTF (FSS-UTF)* bekannt, wurde 1996 veröffentlicht (vgl. Unicode Standard; vgl. Czyborra [2]; vgl. Unicode). Es ist ein CES, dass die Vorteile des geringen Speicherbedarfs von UTF-1 beibehalten und seine Nachteile und Einschränkungen beheben sollte. Insbesondere hervorzuheben ist seine Transparenz für ASCII-Zeichen, Selbstsynchronisation und seiner Bytereihenfolgenunabhängigkeit.

UTF-8-Unterstützung wird zwar stetig ausgebaut, doch ist es noch nicht so weit verbreitet wie z.B. UTF-16. Auch hat es leider nicht nur Vorzüge, sondern auch Nachteile: Da es variable Länge zur Kodierung verwendet, sind Direktzugriffe nicht möglich. Auch können ungültige Bytesequenzen vorkommen, was insbesondere bei Dateinamen zu großen Problemen führen kann. Schließlich hat es gegenüber den ISO-8859-Kodierungen noch den zusätzlichen Nachteil, dass nichtlateinische Buchstaben anstatt einem Byte zwei benötigen.

4.4. Kodierungen für das Internet

Der Großteil der Internetinfrastruktur verwendet die 7-Bit-ASCII-Kodierung um Daten zu übertragen und auszutauschen (vgl. Tschabitscher; vgl. UUencode; vgl. Uuencode). Darum können oben aufgeführte

² In Unixsystemen wird der Schrägstrich („/“) als Trennzeichen für die unterschiedlichen Verzeichnisse verwendet, vergleichbar mit dem umgekehrten Schrägstrich („\“) bei Windowssystemen

Kodierungen nicht direkt verwendet werden, sondern müssen für die Übertragung in die sogenannte *Base64*-Kodierung umgewandelt werden.

Die Base64-Kodierung verdankt ihren Namen der Tatsache, dass sie lediglich 6 Bit eines Bytes verwendet und somit $2^6=64$ Zeichen kodieren kann³. Ihre Aufgabe ist es sicher zu stellen, dass die übertragenen Daten druckbar und von einem Menschen lesbar sind, also keine speziellen Steuerzeichen enthalten. Dabei werden jeweils drei Datenbytes in vier Base64-Bytes umgewandelt: $3*8 \text{ Bit} = 24 \text{ Bit} = 4*6 \text{ Bit}$. Insbesondere für den Emailversand ist diese Kodierung üblich, was zum Beispiel im Fall von Dateianhängen (*Attachments*) zu einer Vergrößerung der übertragenen Datenmenge führt.

5. Schriftarten (Fonts)

Für die korrekte Darstellung von Zeichen auf dem Computer ist es nicht nur wichtig, dass das verwendete System und die verwendeten Programme das entsprechende Charset unterstützen, sondern dass auch eine entsprechende Schriftart installiert ist. Soll beispielsweise ein Zeichen dargestellt werden, das in der Schriftart nicht vorhanden ist, setzt der Computer im besten Fall ein leeres Zeichen – oft in Form eines leeren Kästchens – ein, im schlimmsten Fall unterschlägt er es einfach. Darum empfiehlt sich auch die Installation und die Nutzung der sogenannten „Unicode Fonts“, die eine breite Unterstützung der meisten verwendeten Zeichen bieten (vgl. Unicode Fonts).

³ Base64 kennt die 26 Großbuchstaben und 26 Kleinbuchstaben des lateinischen Alphabets, das Pluszeichen („+“) und den Schrägstrich („/“)

6. Literaturverzeichnis

Folgende Primärliteratur wurde verwendet:

[Unicode Standard]. 2002-03-29. <<http://www.unicode.org/standard/standard.html>> (2005-06-29)

Folgende Sekundärliteratur wurde verwendet:

Czyborra, Roman [1]. 1998-11-30. „ASCII and its variants“.

<<http://czyborra.com/charsets/iso646.html>> (2005-06-29)

Czyborra, Roman [2]. 1998-11-30. „Unicode Transformation Formats: UTF-8 & Co.“.

<<http://www.czyborra.com/utf/>> (2005-06-29)

Czyborra, Roman [3]. 1998-12-01. „The ISO 8859 Alphabet Soup“.

<<http://czyborra.com/charsets/iso8859.html>> (2005-06-29)

[Endian]. 2005-06-11. „Byte-Reihenfolge“. <<http://de.wikipedia.org/wiki/Endian>> (2005-06-29)

Tschabitscher, Heinz. „How Base64 Encoding Works“.

<http://email.about.com/cs/standards/a/base64_encoding.htm> (2005-06-29)

[Unicode]. „Unicode“. <<http://en.wikipedia.org/wiki/Unicode>> (2005-06-29)

[Unicode Fonts]. 2005-06-26. „Free Software Unicode Fonts“.

<http://en.wikipedia.org/wiki/Free_software_Unicode_fonts> (2005-06-29)

[Uuencode]. „Uuencode“. <<http://www.webopedia.com/TERM/U/Uuencode.html>> (2005-06-29)

[UUencode]. „Uuencode“. 2005-04-12. <<http://de.wikipedia.org/wiki/UUencode>> (2005-06-29)