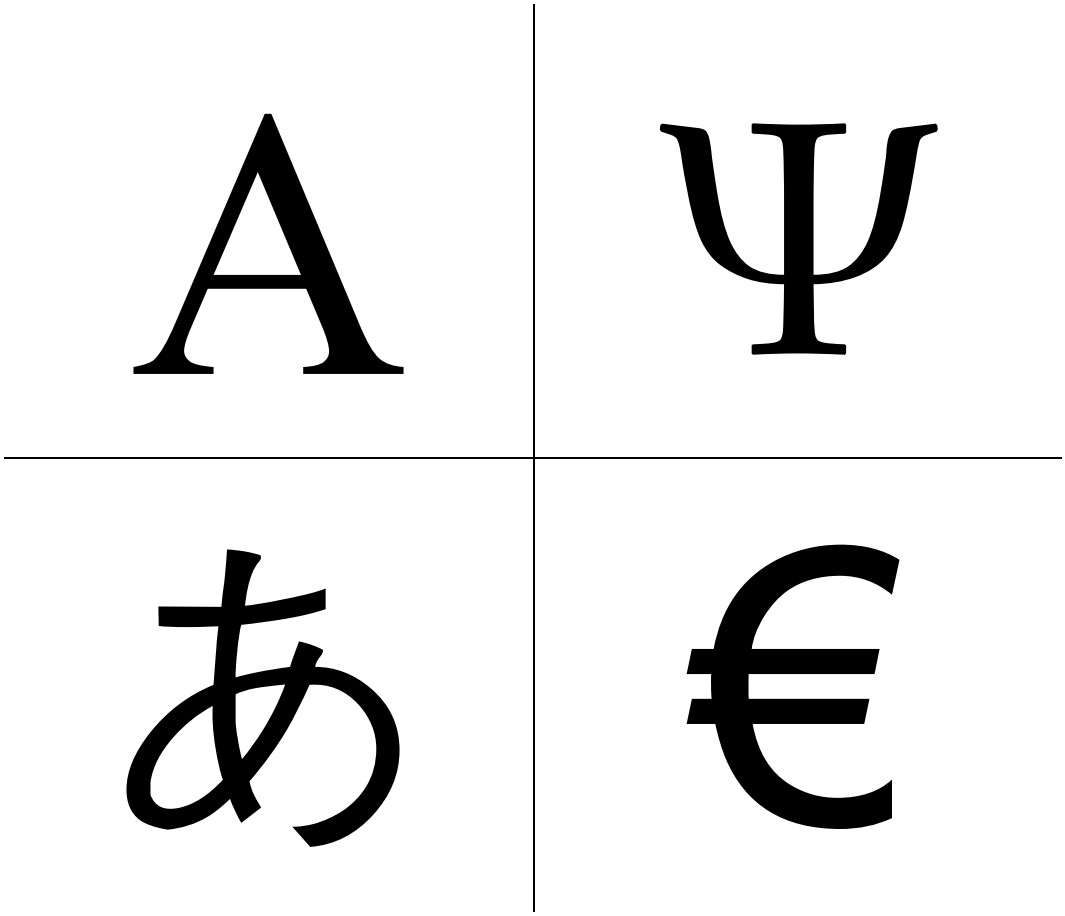


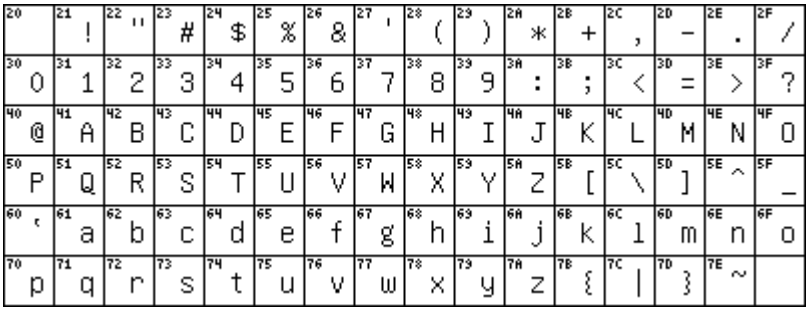
ZEICHENSAETZE & CO.

http://www.mehnle.net/papers/charsets+co

Julian Mehnle <*julian@mehnle.net*>

INHALT

1 ZEICHENSÄTZE, ZEICHENKODIERUNGEN, SCHRIFTEN, SCHRIFTARTEN



ISO-646-US (ASCII)

Am Anfang war der *ASCII*.

Nicht ganz. Denn eigentlich gab es auch davor schon Computer (und andere Maschinen), die mit Schriftzeichen umgehen konnten, aber zu jener Zeit war alles sehr chaotisch, weil jeder seine eigenen Zeichen benutzte. Darum waren die meisten Leute froh, als sich endlich ein paar kluge Köpfe vom *ANSI*, dem *American National Standards Institute*, hinsetzten und sich *ASCII* ['aski:] ausdachten.

ASCII steht für *American Standard Code for Information Interchange*, und bezeichnet zuerst einmal eine Menge der 128 in der Arbeit mit Computern am häufigsten benötigten Schrift- und Steuerzeichen. Allgemein nennt man eine solche Menge von Zeichen einen *Zeichensatz* (engl. *character set*). Zeichensätze sind meist auf eine bestimmte *Schrift* (engl. *script*) spezialisiert, wie z.B. ASCII auf die lateinische Schrift.

Zwar war mit ASCII somit schon einmal eine gemeinsame Menge von Zeichen gefunden, die jeder benutzen konnte. Nun arbeiteten Computer aber noch nie direkt mit grafischen Schriftzeichen, sondern mit Zahlen, weshalb seit jeher Schriftzeichen durch Zahlen kodiert werden, mit denen der Computer hantieren kann. Daraus ergibt sich implizit auch eine Ordnung innerhalb eines jeden Zeichensatzes. Allgemein nennt man eine solche Ordnung eine *Zeichenkodierung* (engl. *character encoding*). Der ASCII-Standard spezifiziert neben der blossen Menge von Zeichen auch deren Kodierung [2], welche im obigen Bild zu sehen ist. Da ASCII genau 128 Zeichen enthält, kann jedes Zeichen durch 7 Bits und somit in knapp einem Byte kodiert werden ($2^7 = 128$).

Das Wort "Hallo" wird also in ASCII durch Bytes mit den Hexadezimal-Werten

48 61 6C 6C 6F

repräsentiert.

Zeichensätze mit mehr als 256 Zeichen werden zwangsweise im Allgemeinen mit mehr als einem Byte pro Zeichen kodiert. Naive Kodierungen kodieren einfach jedes Zeichen eines solchen Zeichensatzes in so vielen Bits, wie der Zeichensatz "breit" ist, z.B. also 16 Bits = 2 Bytes für einen Zeichensatz mit $2^{16} = 65536$ Zeichenpositionen.

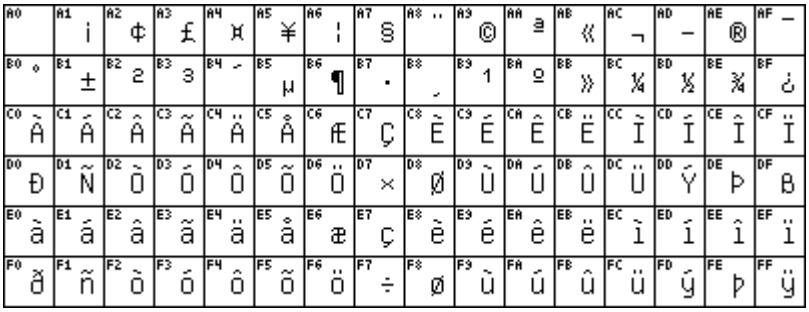
Um die Zeichen eines Zeichensatzes grafisch auf einem Ausgabemedium wie einem Bildschirm oder Drucker auszugeben, benötigt man ferner eine Beschreibung des Erscheinungsbildes der Zeichen: eine *Schriftart* (engl. *font*). Eine Schriftart definiert für jedes Zeichen eines Zeichensatzes genau, wie es grafisch darzustellen ist. Eine Schriftart muss nicht für jedes Zeichen eine grafische Beschreibung enthalten (wie man oft an generischen Platzhalter-Grafiken erkennen kann [8]) – ebenso kann eine Schriftart aber auch mehr als einen Zeichensatz beschreiben.

Obwohl die Begriffe *Zeichensatz*, *Zeichenkodierung*, *Schrift*, und *Schriftart* im Alltag leider regelmässig vertauscht und missverständlich angewendet werden, so bezeichnen sie doch vier unterschiedliche Konzepte.

2 BUCHSTABENSUPPE

Wie leicht zu bemerken ist, wird es einem schwer fallen, z.B. das Wort "Hallöchen" mittels des ASCII-Zeichensatzes zu kodieren. Entsprechend schnell schossen so auch Unmengen [1] abgewandelte und erweiterte Varianten von ASCII aus dem Boden: zuerst 7-bittige, bei denen einige Zeichen durch andere ersetzt wurden; später dann, als die vollständige Ausnutzung der 8 Bits eines Bytes in Mode kam, auch 8-bittige, bei denen die neuen Zeichen in den zusätzlichen 128 Positionen untergebracht wurden.

Die wichtigsten 7-bittigen ASCII-Varianten wurden von der *International Standardization Organization* (*ISO*) zum kombinierten internationalen Standard *ISO-646* erhoben [2]. *ISO-646-US* oder *US-ASCII* bezeichnet seitdem die ursprüngliche Version von ASCII. Ebenso wurden die wichtigsten 8-bittigen Varianten zum Standard *ISO-8859* [3], dessen Basis das von ASCII abgeleitete und extrem verbreitete *ISO-8859-1* oder *Latin-1* ist:



ISO-8859-1 (Latin-1)

Mit ISO-8859-1 kann man also problemlos das Wort "Hallöchen" kodieren:

48 61 6C 6C F6 63 68 65 6E

Die weiteren Varianten von ISO-8859 enthalten an einigen Positionen andere Sonderzeichen, wie z.B. *ISO-8859-2* (*Latin-2*) u.a. mit tschechischen, ungarischen, und polnischen Zeichen, *ISO-8859-5* (*Cyrillic*) u.a. mit russischen Zeichen, oder *ISO-8859-9* (*Latin-5*) mit türkischen Zeichen.

Spätestens seit 2002-01-01 gewinnt *ISO-8859-15*, auch *Latin-9* genannt, immer mehr Bedeutung und ersetzt oft sogar bereits ISO-8859-1, da es u.a. das kaum benötigte internationale Währungszeichen ¤ (Position *A4*) durch das Euro-Zeichen € ersetzt:

A0	A1 ¡	A2 ¢	A3 £	A4 €	A5 ¥	A6 Š	A7 §	A8 š	A9 ©	AA ª	AB «	AC ¬	AD -	AE ®	AF ¯
B0 °	B1 ±	B2 ²	B3 ³	B4 Ž	B5 µ	B6 ¶	B7 ·	B8 ž	B9 ¹	BA º	BB »	BC Œ	BD œ	BE Ÿ	BF ¿
C0 À	C1 Á	C2 Â	C3 Ã	C4 Ä	C5 Å	C6 Æ	C7 Ç	C8 È	C9 É	CA Ê	CB Ë	CC Ì	CD Í	CE Î	CF Ï
D0 Ð	D1 Ñ	D2 Ò	D3 Ó	D4 Ô	D5 Õ	D6 Ö	D7 ×	D8 Ø	D9 Ù	DA Ú	DB Û	DC Ü	DD Ý	DE Þ	DF ß
E0 à	E1 á	E2 â	E3 ã	E4 ä	E5 å	E6 æ	E7 ç	E8 è	E9 é	EA ê	EB ë	EC ì	ED í	EE î	EF ï
F0 ð	F1 ñ	F2 ò	F3 ó	F4 ô	F5 õ	F6 ö	F7 ÷	F8 ø	F9 ù	FA ú	FB û	FC ü	FD ý	FE þ	FF ÿ

ISO-8859-15 (Latin-9)

Neben den ISO-8859-Zeichensätzen gibt es auch noch eine Vielzahl anderer Zeichensätze, z.B. mit (traditionellen oder vereinfachten) chinesischen, oder japanischen Schriftzeichen.

Bis auf die Identität mit ASCII in den ersten 128 Zeichen haben alle diese Zeichensätze aber nichts miteinander gemein und sind somit immer grösstenteils inkompatibel zueinander. Da ein Benutzer oder gar ein Computer allein aus einer Reihe von Bytes im Allgemeinen nicht schliessen kann, welcher Zeichensatz und welche Zeichenkodierung zur Kodierung verwendet wurde, ist es essentiell, bei allen technischen Speicher- und Übertragungsverfahren auf die eine oder andere Weise Zeichensatz und -kodierung mitanzugeben. Leider wird das oft vergessen – viele Leute (Benutzer wie Programmierer) sind sich der Problematik auch gar nicht bewusst – und so kommt es leicht immer wieder zu geradezu babylonischen Sprachverwirrungen, bei denen nicht in ASCII enthaltene Sonderzeichen durcheinander gewürfelt werden.

3 UNICODE

3.1 ÜBERBLICK

Am Anfang war der *Unicode*.

Nicht ganz. Denn eigentlich gab es auch davor schon Computer (und andere Maschinen), die mit Schriftzeichen umgehen konnten, aber zu jener Zeit war alles sehr chaotisch, weil jeder seine eigenen Zeichensätze benutzte. Darum waren die meisten Leute froh, als sich endlich ein paar kluge Köpfe vom *Unicode Consortium* und von der *International Standardization Organization*, hinsetzten und sich *Unicode* ['junicoud] ausdachten.

Na ja, nicht die meisten. Leider haben es viel zu viele Leute noch gar nicht mitbekommen, welch schöne Erfindung Unicode ist – nicht zuletzt darum auch dieses Schriftstück. ;-)

Unicode, von der ISO unter dem Namen *ISO-10646* ("ISO-646 plus 10000") standardisiert [4,5], ist ein allumfassender Zeichensatz, der es sich zum Ziel gesetzt hat, die Farce der ganzen herkömmlichen, miteinander inkompatiblen Zeichensätze ein für alle mal zu überwinden. Unicode ist 32 Bits "breit", d.h. es bietet theoretisch Platz für über 4 Milliarden Zeichen. Aktuell (2002-08) liegt Unicode in der Version 3.1 vor und enthält tatsächlich 94140 verschiedene Zeichen: u.a. lateinische, griechische, kyrillische, arabische, hebräische, japanische, und chinesische Zeichen, und sogar Runen und technische, mathematische, und astrologische Sonderzeichen.

Der 32-bittige Zeichenraum von Unicode ist mehrfach in unterschiedlich wichtige Bereiche abgestuft:

Die ersten 2^8 = 256 Zeichen entsprechen genau denen von ISO-8859-1 – somit ist ganz am Anfang des Zeichenraums auch ASCII enthalten.

Die ersten 2^{16} = 65536 Zeichenpositionen, welche zusammen *Basic Multilingual Plane* (*BMP*) genannt werden, enthalten alle Sonderzeichen der übrigen ISO-8859-Zeichensätze sowie eine Menge weiterer, wichtiger Zeichen.

Die ungebräuchlicheren Zeichen sind in weiteren Ebenen (engl. *planes*) untergebracht. Zukünftige Zeichen haben aber sehr wohl noch die Chance, in der BMP platziert zu werden.

3.2 UCS-4, UCS-2, UTF-16

Da Unicode 2^{32} verschiedene Zeichenpositionen hat, benötigt eine naive Kodierung 32 Bits = 4 Bytes pro Zeichen. Diese simple Zeichenkodierung heisst dementsprechend *UCS-4* (*Universal Character Set*), und das Wort "Hallöchen" würde durch folgende Bytes repräsentiert (in sog. *Big Endian*-Reihenfolge):

00 00 00 48 00 00 00 61 00 00 00 6C 00 00 00 6C 00 00 00 F6
00 00 00 63 00 00 00 68 00 00 00 65 00 00 00 6E

Für einfachere Zwecke und ein wenig Platzersparnis gibt es noch die Zeichenkodierung *UCS-2*, welche nur genau die 2^{16} Zeichen der BMP mit je 16 Bits = 2 Bytes kodiert:

00 48 00 61 00 6C 00 6C 00 F6 00 63 00 68 00 65 00 6E

Um eine 2-Byte-Konvertierung verwenden und dennoch alle Unicode-Zeichen kodieren zu können, gibt es noch *UTF-16*, welches im Grunde *UCS-2* entspricht, nur dass die Zeichenpositionen *D800* bis *DFFF* nicht mit regulären Zeichen besetzt sind, sondern gewissermassen als Escape-Zeichen verwendet werden, um Byte-Sequenzen einzuleiten, welche die restlichen, ausserhalb der BMP befindlichen Zeichen kodieren – ähnlich wie für UTF-8 im folgenden Abschnitt beschrieben.

3.3 UTF-8

Leider haben die vorgenannten Unicode-Kodierungsmethoden UCS-4, UCS-2, und UTF-16 allesamt den Nachteil, dass sie durch die vielen eingestreuten Null-Bytes erstens inkompatibel zu ASCII und den ISO-8859-Zeichensätzen sind, zweitens unnötig viel Platz verbrauchen, und drittens ungeeignet sind für Software, die ein Null-Byte in einer Zeichenkette (engl. *string*) als Ende derselbigen interpretiert – dies ist in der Tat auch der gravierendste Hinderungsgrund, diese Kodierungen zu benutzen.

Eine wesentlich geschicktere Methode, den Unicode-Zeichensatz zu kodieren, ist *UTF-8* (*UTF* steht für *Unicode Transformation Format*):

Die ersten 128 Unicode-Zeichen, welche ja dem ASCII-Zeichensatz entsprechen, werden direkt mit den Byte-Werten *00* bis *7F* kodiert, weshalb ASCII-Strings implizit auch gültige und gleichbedeutende UTF-8-Strings sind.

Alle übrigen Unicode-Zeichen werden als Sequenzen von mehreren Bytes kodiert, bestehend aus den Byte-Werten *80* bis *FD*, und zwar nach folgendem Schema:

Unicode-Zeichen	Byte-Sequenz
00000000 - 0000007F	*0xxxxxxx*
00000080 - 000007FF	*110xxxxx 10xxxxxx*
00000800 - 0000FFFF	*1110xxxx 10xxxxxx 10xxxxxx*
00010000 - 001FFFFF	*11110xxx 10xxxxxx 10xxxxxx 10xxxxxx*
00200000 - 03FFFFFF	*111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx*
04000000 - 7FFFFFFF	*1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx*

Die mit *x* markierten Bits werden dabei durch den binären Zeichencode ersetzt.

Das Euro-Zeichen z.B., das den Unicode-Code *20AC* (hexadezimal), oder *0010000010101100* (binär), hat, wird also durch 3 Bytes kodiert:

1110xxxx	*10xxxxxx*	*10xxxxxx*
0010	*000010*	*101100*
11100010	*10000010*	*10101100*

...also durch die Byte-Sequenz *E2 82 AC*.

Durch diese ausgeklügelte Kodierungsweise können Unicode-Texte auf eine Weise gespeichert und übertragen werden, die mit ASCII kompatibel ist, die für die wichtigsten Zeichen am wenigsten Platz benötigt, und die keine missverständlichen Null-Bytes verwendet. Ausserdem kann ein UTF-8-Dekoder sofort feststellen, ob ein Byte ein ASCII-Zeichen kodiert (das höchstwertige Bit, Bit *7*, ist gleich *0*) oder ob es zu einer UTF-8-Byte-Sequenz gehört, die ein höher positioniertes Zeichen kodiert (Bit *7* ist gleich *1*) – und er kann sogar auch erkennen, ob ein an einer UTF-8-Byte-Sequenz beteiligtes Byte den Anfang der Sequenz darstellt (Bits *7* und *6* sind gleich *11*) und wie viele Bytes die gesamte Sequenz in diesem Fall umfasst (Anzahl der gesetzten Bits von Bit *7* bis zum höchstwertigen gelöschten Bit).

3.4 UTF-8 IN HTML- UND XML-DATEIEN

In einer HTML-Datei kann wie folgt über ein <*meta*>-Tag spezifiziert werden, dass das Dokument in einer UTF-8-Kodierung vorliegt:

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

Idealerweise würde man allerdings eher den HTTP-Server so konfigurieren, dass er grundsätzlich den richtigen HTTP-"*Content-Type*"-Header ausliefert.

In einer XML-Datei (und somit auch in einer XHTML-Datei) kann eine UTF-8-Kodierung wie folgt in der *Text Declaration* spezifiziert werden:

<?xml version="1.0" encoding="UTF-8" ?>

3.5 UNICODE & UTF-8 UNTER WINDOWS

3.5.1 Überblick

Alle Windows-Versionen, die auf Windows NT basieren (Windows NT, Windows 2000, Windows XP), arbeiten ohnehin intern grundsätzlich mit UCS-2-kodiertem Unicode. Auch die meisten Microsoft-Programme, wie z.B. die Office-Suites seit Office 97, und viele Programme von Drittherstellern kodieren ihre Dokumente mit UCS-2. Im Übrigen können die meisten bei Windows und

Office mitgelieferten Schriftarten auch grosse Teile des Unicode-Zeichensatzes darstellen.

Obwohl es nur selten Bedarf für einen direkten Umgang mit UTF-8 gibt, kann für den Fall der Fälle der Windows-zugehörige Editor *Notepad* Text-Dateien in MS-ANSI (einem erweiterten ISO-8859-1), UCS-2, und UTF-8 lesen und schreiben.

Für UTF-8 relevant sind hier hauptsächlich Email- und News-Clients, die leider nahezu allesamt standardmässig ausgehende Mails in ISO-8859-1 kodieren. Hier soll nur kurz auf die beiden Microsoft-Programme *Outlook* und *Outlook Express* eingegangen werden.

3.5.2 *Outlook*

Outlook stellt man folgendermassen auf die Kodierung mit UTF-8 für ausgehende Mails ein:

Extras →
Optionen →
E-Mail-Format →
Nachrichtenformat →
Internationale Optionen →
Codierung für ausgehende Nachrichten =
"Unicode (UTF-8)"
Briefpapier und Schriftarten →
Schriftarten →
Internationale Schriftarten →
"Unicode" →
Codierung = "Unicode (UTF-8)"
Als Standard!

3.5.3 *Outlook Express*

Bei Outlook Express sehen die nötigen Einstellungen so aus:

Extras →
Optionen →
Senden →
Senden →
Internationale Einstellungen →
Standardzeichensatz = "Unicode (UTF-8)"
Lesen →
Schriftarten →
"Unicode" →
Codierung = "Unicode (UTF-8)"
Als Standard!

3.6 UNICODE & UTF-8 UNTER UNIX/LINUX

3.6.1 Überblick

Viele Programme unter Unix und Linux, z.B. *ls*, *cat*, oder *grep*, können aufgrund ihrer Zeichenkodierungs-unabhängigen Arbeitsweise ohne weiteres mit UTF-8 umgehen. Viele andere, wichtige Programme, wie z.B. *less*, oder die *bash*, unterstützen auch bereits explizit UTF-8 [6].

Generell bringt man die meisten Programme dazu, korrekt mit UTF-8 umzugehen, indem man die nötigen Ländereinstellungen (engl. *locales*) installiert (bitte fragen Sie Ihren Systemadministrator), und z.B. in der eigenen *.bashrc*-Datei eine der Umgebungsvariablen *LANG* oder *LC_ALL* setzt – am besten die erstere:

export LANG=de_DE.UTF-8

Statt *de_DE* kann man auch *en_US* angeben, dann werden US-amerikanische Ländereinstellungen verwendet. Systemglobal, d.h. als Voreinstellung für alle Benutzer und Dienste, lässt sich diese Einstellung auch in der Datei */etc/environment* bzw. */etc/profile* vornehmen:

LANG=de_DE.UTF-8

Wenn man unter *X* arbeitet, sollte man natürlich auch Unicode-fähige Schriftarten installiert haben. Die *'-Misc-Fixed'*-Schriftarten von *XFree86 4* enthalten bereits viele Unicode-Zeichen. Eine weitere, sehr umfangreiche Unicode-Schriftart ist *Unifont* [7] (unter Debian in Form des Pakets *"unifont"* verfügbar).

Wer mit einem *X*-Server arbeitet, der bei Tastatur-Eingabe das €-Zeichen nicht erkennt (z.B. *X-Win32*), kann die Tastenkombination *Alt-Gr+E* mit dem €-Zeichen belegen:

$ xmodmap -e 'keycode 26 = e E EuroSign'

3.6.2 *xterm*

xterm muss entweder in einer solcherart eingestellten Umgebung gestartet werden, oder mit der Option *-u8*. Ausserdem sollte man eine Unicode-fähige Schriftart spezifizieren (z.B. *'-Misc-Fixed-Medium-R-SemiCondensed--13-120-75-75-C-60-ISO10646-1'*):

$ LANG=de_DE.UTF-8 xterm -fn '<schriftart>'
$ xterm -u8 -fn '<schriftart>'

Leider ist das derzeit nur die hehre Theorie, denn die aktuelle Version (*"4.1.0(165)"*) von *xterm* scheint in der automatischen Aktivierung der UTF-8-

Unterstützung einen Bug zu haben. Dies erfordert es explizit, *xterm* mit der Option *-u8* in einer Umgebung zu starten, die kein UTF-8 impliziert, und dann in der Shell innerhalb des *xterm*s eine UTF-8-Umgebung zu setzen:

```
$ LANG=de_DE  xterm -u8 -fn '<schriftart>'
$ LANG=de_DE.UTF-8
```

3.6.3 *konsole*

Der UTF-8-Modus des KDE-Terminals *konsole* kann mit dem Bash-Skript *konsole-utf8* aktiviert werden, das nichts anderes macht, als die Steuerzeichensequenz *'\033%G'* auszugeben:

```
$ konsole-utf8 on
```

3.6.4 *gnome-terminal*

Das Gnome-Terminal unterstützt UTF-8 derzeit leider nicht vernünftig. Gnome 2 unterstützt es aller Voraussicht nach aber.

3.6.5 *vi*

Der beliebte Editor *vi* unterstützt in der aktuellen Version automatisch UTF-8.

3.6.6 *recode*

Um Text-Dateien zwischen verschiedenen Zeichenkodierungen umzukodieren, gibt es das Programm *recode*, das Hunderte verschiedene Kodierungen kennt:

```
$ recode <von>..<nach>  <datei>
```

Z.B. lässt sich eine ISO-8859-1-kodierte Datei nach UTF-8 wie folgt umkodieren:

```
$ recode ISO-8859-1..UTF-8 <datei>
```

recode unterstützt insbesondere auch UCS-4 und UCS-2. Eine Auflistung aller unterstützten Kodierungen erhält man durch:

```
$ recode -l
```

Für die meisten Kodierungsbezeichnungen gibt es auch nützliche Abkürzungen, z.B. *u8* für *UTF-8*, oder *l1* für *ISO-8859-1* (*Latin-1*).

VERWEISE

1 Offizielle Zeichensatz-Namen
(Internet Assigned Numbers Authority, IANA)
http://www.iana.org/assignments/character-sets

2 ISO-646: ASCII und Varianten
http://czyborra.com/charsets/iso646.html

3 ISO-8859: Latin-1 und Varianten
http://czyborra.com/charsets/iso8859.html

4 ISO-10646: Unicode
http://czyborra.com/unicode/characters.html

5 UTF-8 and Unicode FAQ
http://www.cl.cam.ac.uk/~mgk25/unicode.html

6 UTF-8 and Unicode FAQ: UTF-8 enabled applications
http://www.cl.cam.ac.uk/~mgk25/unicode.html#apps

7 Unifont
http://czyborra.com/unifont/HEADER.html

8 Edison & Co. – Münchens erstes Erfinderlokal: Mittagsmenü-Karte
Man beachte das missgebildete €-Zeichen! ;-)
http://www.edisonundco.de/mittag.htm