UTF-8 Kodierung

UTF-8 (Abk. für 8-bit Unicode Transformation Format) ist die am weitesten verbreitete Kodierung für Unicode Zeichen.

Unicode ist ein internationaler Standard, in dem langfristig für jedes sinntragende Zeichen bzw. Textelement aller bekannten Schriftkulturen und Zeichensysteme ein digitaler Code festgelegt wird. Ziel ist es, das Problem unterschiedlicher, inkompatibler Kodierungen in unterschiedlichen Ländern oder Kulturkreisen zu beseitigen.
In Unicode finden Zeichen der wichtigsten ISO-Zeichensätze wie die ISO-Normen der Serie 8859 eine 1:1-Entsprechung (das bedeutet, dass bei einer Konvertierung von ISO zu Unicode und zurück das gleiche Ergebnis herauskommt). Heute erledigen die meisten Webbrowser die Darstellung dieser Zeichensätze mit einer Unicode-kodierten Schrift in der Regel perfekt und vom Benutzer unbemerkt.

In UTF-8 wird jedem Unicode-Zeichen eine speziell kodierte Bytekette von variabler Länge zugeordnet. UTF-8 unterstützt bis zu vier Byte, auf die sich wie bei allen UTF-Formaten alle 1.114.112 Unicode-Zeichen abbilden lassen.

UTF-8 hat eine zentrale Bedeutung als globale Zeichenkodierung im Internet. Die Internet Engineering Task Force verlangt von allen neuen Internetkommunikationsprotokollen, dass die Zeichenkodierung deklariert wird und dass UTF-8 eine der unterstützten Kodierungen ist

Auch 2007 wird diese Empfehlung allerdings immer noch nicht universell befolgt.

Kodierung:

Unicode-Zeichen mit den Werten aus dem Bereich von 0 bis 127 (0 bis 7F hexadezimal) werden in der UTF-8-Kodierung als ein Byte mit dem gleichen Wert wiedergegeben. Insofern sind alle Daten, für die ausschließlich echte ASCII-Zeichen verwendet werden, in beiden Darstellungen identisch.

Unicode-Zeichen größer als 127 werden in der UTF-8-Kodierung zu Byteketten der Länge zwei bis vier kodiert.

Unicode-Bereich	**UTF-8-Kodierung**	**Bemerkungen**	**Möglichkeiten**	
0000 0000– 0000 007F	0xxxxxxx	In diesem Bereich (128 Zeichen) entspricht UTF-8 genau dem ASCII-Code: Das höchste Bit ist *0*, die restliche 7-Bit-Kombination ist das ASCII-Zeichen.	2^7	128
0000 0080– 0000 07FF	110xxxxx 10xxxxxx	Das erste Byte enthält binär 11xxxxxx, die folgenden Bytes 10xxxxxx; die *x* stehen für die fortlaufende Bitkombination des Unicode-Zeichens. Die Anzahl der Einsen ausgehend von der höchsten 0 im ersten Byte ist die Anzahl der Bytes für das Zeichen. (In Klammern jeweils die theoretisch maximal möglichen.)	$2^{11} - 2^7$ (2^{11})	1.920 (2.048)
0000 0800– 0000 FFFF	1110xxxx 10xxxxxx 10xxxxxx		$2^{16} - 2^{11}$ (2^{16})	63.488 (65.536)
0001 0000– 0010 FFFF [0001 0000– 001F FFFF]	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx		2^{20} (2^{21})	1.048.576 (2.097.152)

Der Algorithmus lässt theoretisch bis zu sieben Bytes lange Byteketten und dadurch mehrere Milliarden Zeichen zu ($2^7 + 2^{(1\cdot6+5)} + 2^{(2\cdot6+4)} + 2^{(3\cdot6+3)} + 2^{(4\cdot6+2)} + 2^{(5\cdot6+1)} + 2^{(6\cdot6+0)} = 70.936.234.112 > 2^{36}$), in seiner Verwendung als UTF-Kodierung ist er aber auf den gemeinsamen Coderaum aller Unicode-Kodierungen beschränkt, also von 0 bis 0010 FFFF (1.114.112 Möglichkeiten) und weist maximal vier Bytes lange Byteketten auf.

Das erste Byte eines UTF-8-kodierten Zeichens nennt man dabei Start-Byte, weitere Bytes nennt man Folgebytes. Startbytes enthalten also die Bitfolge 11xxxxxx oder 0xxxxxxx, Folgebytes immer die Bitfolge 10xxxxxx.

- Ist das höchste Bit des ersten Byte 0, handelt es sich um ein gewöhnliches ASCII-Zeichen, da ASCII eine 7-Bit-Kodierung ist und die ersten 128 Zeichen des Unicode die ASCII-Zeichen sind. Damit sind alle ASCII-Dokumente automatisch aufwärtskompatibel zu UTF-8.
- Ist das höchste Bit des ersten Byte 1, handelt es sich um ein Mehrbytezeichen, also ein Unicode-Zeichen mit einer Zeichennummer größer als 127.
- Sind die höchsten beiden Bits des ersten Byte 11, handelt es sich um das Start-Byte eines Mehrbytezeichens, sind sie 10, um ein Folge-Byte.
- Die lexikalische Ordnung nach Byte-Werten entspricht der lexikalischen Ordnung nach Buchstaben-Nummern, da höhere Zeichennummern mit entsprechend mehr 1-Bits im Start-Byte kodiert werden.
- Bei den Start-Bytes von Mehrbyte-Zeichen gibt die Anzahl der höchsten 1-Bits die gesamte Bytezahl des als Mehrbyte-Zeichen kodierten Unicode-Zeichens an. Anders interpretiert, die Anzahl der 1-Bits nach dem höchsten 0-Bit entspricht der Anzahl an Folgebytes plus eins, z. B. 1110xxxx 10xxxxxx 10xxxxxx = drei Bits nach dem höchsten 0-Bit = drei Bytes insgesamt, zwei Bits nach dem höchsten 0-Bit vor dem höchsten 1-Bit = zwei Folgebytes.

- Start-Bytes (0xxx xxxx oder 11xx xxxx) und Folge-Bytes (10xx xxxx) lassen sich eindeutig voneinander unterscheiden. Somit kann ein Byte-Strom auch in der Mitte gelesen werden, ohne dass es Probleme mit der Dekodierung gibt, was insbesondere bei der Wiederherstellung defekter Daten wichtig ist. 10xxxxxx Bytes werden einfach übersprungen, bis ein 0xxxxxxx oder 11xxxxxx Byte gefunden wird. Könnten Start-Bytes und Folge-Bytes nicht eindeutig voneinander unterschieden werden, wäre das Lesen eines UTF-8-Datenstroms, dessen Beginn unbekannt ist, unter Umständen nicht möglich.

Zu beachten:

- Das gleiche Zeichen kann theoretisch auf verschiedene Weise kodiert werden (Zum Beispiel „a“ als *0*__1100001__ oder fälschlich als *110*0000__1__ *10*__100001__). Jedoch ist nur die jeweils kürzestmögliche Kodierung erlaubt.
- Bei mehreren Bytes für ein Zeichen werden die Bits *rechtsbündig* angeordnet – das rechte Bit des Unicode-Zeichens steht also immer im rechten Bit des letzten UTF-8-Bytes.
- Ursprünglich gab es auch Kodierungen mit mehr als vier Oktetts (bis zu sechs), diese sind jedoch ausgeschlossen worden, da es in Unicode keine korrespondierenden Zeichen gibt und ISO 10646 in seinem möglichen Zeichenumfang an Unicode angeglichen wurde.
- Für alle auf dem lateinischen Alphabet basierenden Schriften ist UTF-8 die platzsparendste Methode zur Abbildung von Unicode-Zeichen.
- Die Unicodebereiche U+D800–U+DBFF und U+DC00–U+DFFF sind ausdrücklich keine Zeichen, sondern dienen nur in UTF-16 zur Kodierung von Zeichen außerhalb der Basic Multilingual Plane, sie wurden früher als Low und High surrogates bezeichnet. Folglich sind Byte-Folgen, die diesen Bereichen entsprechen, kein gültiges UTF-8. Zum Beispiel wird U+10400 in UTF-16 als D801,DC00 dargestellt, sollte in UTF-8 aber als F0,90,90,80 und nicht als ED,A0,81,ED,B0,80 ausgedrückt werden. Java unterstützt dies seit der Version 1.5 [2]. Aufgrund der weiten Verbreitung der falschen Kodierung, insbesondere auch in Datenbanken, wurde diese Kodierung eigens als CESU-8 normiert.
- In UTF-8, UTF-16 und UTF-32 ist der gesamte Wertebereich von Unicode kodiert.

Beispiele für UTF-8 Kodierungen:

Zeichen	**Unicode**	**Unicode binär**	**UTF-8 binär**	**UTF-8 hexadezimal**
Buchstabe *y*	U+0079	00000000 01111001	**0**1111001	0x79
Buchstabe *ä*	U+00E4	00000000 11100100	**110**00011 **10**100100	0xC3 0xA4
Zeichen für eingetragene Marke ®	U+00AE	00000000 10101110	**110**00010 **10**101110	0xC2 0xAE
Eurozeichen *€*	U+20AC	00100000 10101100	**1110**0010 **10**000010 **10**101100	0xE2 0x82 0xAC
Violinschlüssel *?*	U+1D11E	000000001 11010001 00011110	**11110**000 **10**011101 **10**000100 **10**011110	0xF0 0x9D 0x84 0x9E

Das letzte Beispiel liegt außerhalb des ursprünglich in Unicode (unter Version 2.0) enthaltenen Codebereiches (16 Bit), der in der aktuellen Unicode-Version als BMP-Bereich (Ebene 0) enthalten ist, im SMP-Bereich (Ebene 1). Da derzeit viele Schriftarten diese neuen Unicode-Bereiche noch nicht enthalten, können die dort enthaltenen Zeichen auf vielen Plattformen nicht korrekt dargestellt werden. Stattdessen wird ein Ersatzzeichen dargestellt, welches als Platzhalter dient.

UTF-8 und Java-Strings:

In Java werden Zeichenketten durch die Klasse String repräsentiert. Als Reihung von Elementen des Typs char ist sie die wichtigste Datenstruktur für alle Aufgaben, die etwas mit der Ein- und Ausgabe oder der Verarbeitung von Zeichen zu tun haben.

Da der char-Typ in Java durch ein Unicode-Zeichen repräsentiert wird, besteht auch ein String aus einer Kette von Unicode-Zeichen.

Java wurde mit dem Anspruch entworfen, bekannte Schwächen bestehender Programmiersprachen zu vermeiden, und der Wunsch nach Portabilität stand ganz oben auf der Liste der Designziele. Konsequenterweise wurde der Typ char in Java daher bereits von Anfang an 2 Byte groß gemacht und speichert seine Zeichen auf der Basis des Unicode-Zeichensatzes. Als einziger integraler Datentyp ist char nicht vorzeichenbehaftet.

In einem String können beliebige Unicode-Escape-Sequenzen der Form \uxxxx angegeben werden, wobei xxxx eine Folge von bis zu 4 hexadezimalen Ziffern ist. So steht beispielsweise \u000a für die Zeilenschaltung und \u0020 für das Leerzeichen.

Beispielcode:

```
001
002
003 import java.io.*;
004
005 public class Listing1902
006 {
007   public static void main(String[] args)
008   {
009     try {
010       DataOutputStream out = new DataOutputStream(
011                              new BufferedOutputStream(
012                              new FileOutputStream("test.txt")));
013       out.writeInt(1);
014       out.writeInt(-1);
015       out.writeDouble(Math.PI);
016       out.writeUTF("häßliches");
017       out.writeUTF("Entlein");
018       out.close();
019     } catch (IOException e) {
020       System.err.println(e.toString());
021     }
022   }
023 }
```

Das Programm erzeugt eine Ausgabedatei test.txt von 38 Byte Länge:

```
00 00 00 01 FF FF FF FF-40 09 21 FB 54 44 2D 18   ........@.!.TD-.
00 0B 68 C3 A4 C3 9F 6C-69 63 68 65 73 00 07 45   ..h....liches..E
6E 74 6C 65 69 6E                                 ntlein
```

- Die ersten 4 Byte stellen den int-Wert 1 dar.
- Die nächsten 4 Byte repräsentieren die -1 (es wird die übliche Zweierkomplementdarstellung verwendet).
- Anschließend folgen 8 Byte mit der double-Darstellung der Zahl pi.
- Nun folgt die **Längeninformation 000B** des ersten UTF-8-Strings und danach dessen **11 Zeichen**. Man kann sehen, dass die beiden Umlaute **ä** und **ß** jeweils zwei Byte belegen, alle übrigen Zeichen jedoch nur eines.
- Schließlich folgt der zweite UTF-8-String. Er hat die **Länge 7**, und **alle Zeichen** werden mit einem Byte dargestellt.

Unicode Zeichen im Java Quellcode:

Ein Java -Programm besteht aus Zeichen der Unicode -Zeichenmenge. Java erlaubt es, diverse Sonderzeichen und Zeichen aus den Alphabeten vieler Sprachen direkt im Quelltext zu verwenden.

Da die Eingabe, Bearbeitung oder Anzeige aller Zeichen, die nicht bereits zum klassischen ASCII -Code gehören, Schwierigkeiten bereiten kann, sind diese Zeichen selten in Java -Programmen zu finden und werden auch in den Standard-Sprachelementen (wie den Schlüsselwörtern) nicht verwendet. Das große lateinische „A“ sieht beispielsweise ähnlich aus, wie das große griechische Alpha „A“, so dass beim Lesen von Quellcode die Zeichen auch nicht immer eindeutig erkannt werden können.

Der Java -Compiler kann Quellcode verschiedener Zeichencodierungen lesen. Als Vorgabe wird die Vorgabe- oder Standardeinstellung der aktuellen Umgebung verwendet. Andere Codierungen lassen sich mit der Option "`encoding`" angeben.

Nicht jeder Editor ermöglicht die Bearbeitung von Texten mit allen *Unicode* -Zeichen. Falls aber ein geeigneter Editor verwendet wurde, dann muss dem Java -Compiler die für die Datei verwendete Codierung mitgeteilt werden, wenn diese von der Vorgabe der Umgebung abweicht. Soll die Datei "`Hallo.java`" beispielsweise mit der oft für Unicode verwendeten Codierung "UTF-8" interpretiert werden, so kann das unter "`javacutf`" angegebene Kommando verwendet werden.

```
javacutf [Kommando]
      javac –encoding UTF-8 Hallo.java
```

Die möglichen Namen für Codierungen lassen sich dem Quellcode der Klasse "`sun.io.CharacterEncoding`" entnehmen.

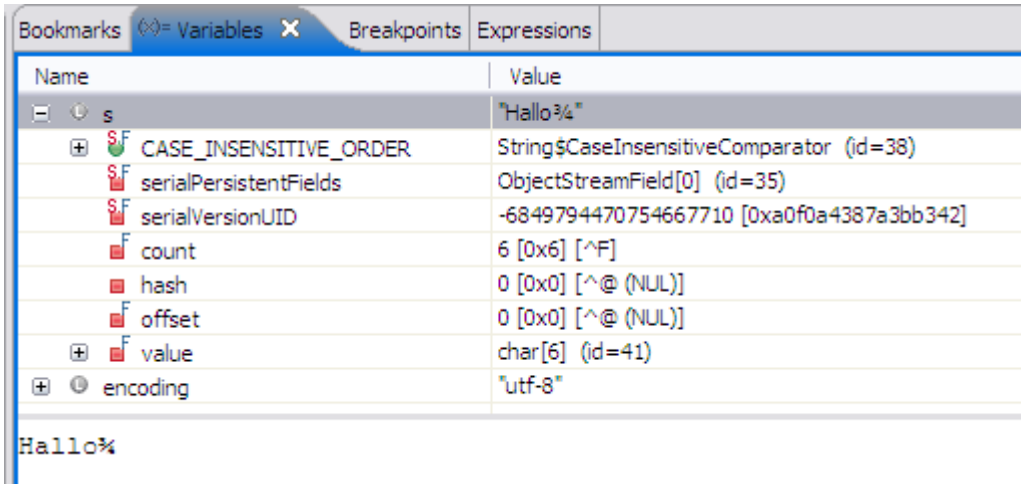
Die Tabelle "`Einige Codierungsnamen`" nennt einige Beispiele.

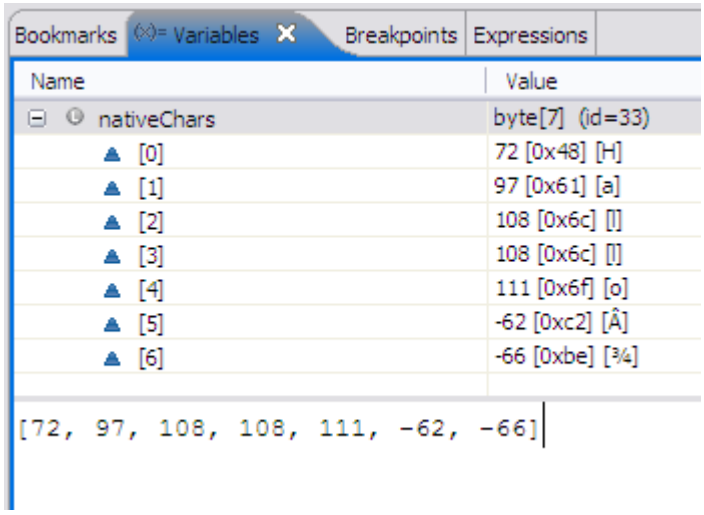
```
Einige Codierungsnamen [Tabelle]
      "Name",        "Bedeutung":
      "us-ascii",    "ANSI_X3.4-1968".
      "iso-8859-1", "ISO_8859-1:1987".
      "utf-8",       "UTF-8".
      "cp437",       "IBM437".
      "cp580",       "IBM580".
```

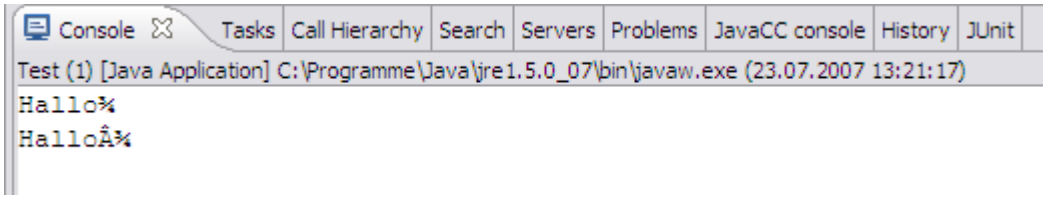
Unicode Zeichen in Eclipse:

Beispielcode

```
        String s = "Hallo\u00BE";
        String encoding = "utf-8";
        byte nativeChars[];
        System.out.println(s);
        try {
            nativeChars = s.getBytes( encoding );
            System.out.println( new String(nativeChars) );
        }
            catch (UnsupportedEncodingException e) {
```







Referenztabelle für die UTF-8 Kodierung:

http://www.utf8-zeichentabelle.de

http://www.utf8-zeichentabelle.de/hilfe-impressum.html

Diese Site ist als Referenztabelle für die UTF-8-Codierung von Unicode-Zeichen gedacht. Sie können die UTF-8_Codierung des Zeichens in verschiedenen Formaten (dezimal, hexadezimal, oktal, binär, als Perl-Stringliteral) darstellen und gleichzeitig testen, ob Ihr Browser die Glyphen des Zeichens auch tatsächlich anzeigt (falls nicht, wird je nach Browser oder Betriebssystem ein quadratisches Symbol oder ein Fragezeichen angezeigt).

Sie können auch die in HTML 4.0 definierten benannten HTML-Zeichenentities (soweit definiert) und/oder die numerische HTML-Darstellung des Unicode-zeichens (wahlweise dezimal oder hexadezimal) anzeigen.

Die Zeichen auf dieser Seite werden nur dann korrekt angezeigt, wenn Ihr Browser und Betriebssystem die UTF-8-Codierung unterstützt.