

# Encodings im LMS der Berufsfachschule BBB

Michael Schneider [michael.schneider@bbb Baden.ch](mailto:michael.schneider@bbb Baden.ch)

31. Juli 2007

# Inhaltsverzeichnis

<b>1</b>	<b>Encodings</b>	<b>1</b>
1.1	ASCII	1
1.2	8-Bit Encodings	2
1.2.1	ISO 8859	2
1.2.2	Windows Codepages	3
1.3	Die Verschlimmbesserung: Unicode	3
1.3.1	UTF-16	4
1.3.2	UTF-8	4
<b>2</b>	<b>Encodierung am LMS</b>	<b>5</b>
2.1	Komponenten	5
2.1.1	SVN	5
2.1.2	PHP	5
2.1.3	Dateisystem	5
2.1.4	Webserver	5
2.1.5	Webseiten	6
2.1.6	Browser	6
2.2	Analyse	6
2.2.1	Ausgangsdaten	6
2.2.2	Erste Analyse	6
2.3	Reparatur	7

### **Zusammenfassung**

An der Berufsfachschule BBB arbeiten verschiedene Systeme zusammen um eine komplexe Lernlandschaft zu bilden. Moodle bindet als Frontsystem Verzeichnisse ein, die mit SVN aktuell gehalten werden. Dieser Datenaustausch funktioniert nur, wenn die Schnittstellen genau geregelt sind.

Ein grosses Problem sind die sogenannten *Encodings*. Dieser kurzbericht soll zeigen, welche Faktoren eine Rolle spielen, was Probleme verursachen kann und wie man diese Probleme umgeht.

# Kapitel 1

## Encodings

*Encodings* werden immer dann benötigt, wenn ein digitales System reine Texte speichern muss. Dies ist beispielsweise der Fall für:

- Emails
- Datei-/Verzeichnisnamen
- Webseiten
- Texte in Datenbanken
- ...

Worddateien sind ein Grenzfall, weil es sich nicht um reine Texte handelt sondern zusätzliche Metainformation wie Formatierungen, etc. mit in die Datei gepackt werden.

Grundsätzlich stehen einem digitalen System nur Nullen und Einen zur Verfügung um Informationen abzubilden. Dies bedeutet, dass Textdaten ebenfalls auf ein System aus Nullen und Einen abgebildet werden müssen. Die Abbildungen nennt man *Encodings* oder *Codepages*. Um andere Zahlen wie 0 und 1 zu erhalten, werden mehrere Bits miteinander verwendet. Das heisst, dass diese Encodings die Abbildung von Zeichen auf Zahlen regeln müssen.

Dies hat weitreichende Konsequenzen: Wie bei den einfachen Geheimschriften müssen Sender (Schreiber) eines Textes und Empfänger (Leser) mit dem gleichen Encoding arbeiten, sonst sehen sie nur bedeutungslose Zahlen. Weiter hat das Encoding einen Einfluss auf die Dateigrösse: Wenn eine A4 Seite ca. 2048 Zeichen enthält und mit einem ASCII Encoding gespeichert wird (Siehe Kapitel 1.1) werden pro Zeichen 7 bit benötigt und die Datei damit

$$2048 \cdot 7\text{Bit} = 14'336\text{Bit}$$

oder 1'792 Bytes gross (8 Bit = 1 Byte). Verwendet man Unicode (Siehe Kapitel 1.3 auf Seite 3) so werden im schlimmsten Falle 4 Bytes (32 Bit) pro Zeichen aufgewendet und somit wird die Datei

$$2048 \cdot 32\text{Bit} = 65'536\text{Bit}$$

oder 8'192 Bytes gross.

### 1.1 ASCII

ASCII war eine der ersten Encodings. Sie kommt mit 7 Bit aus und kann somit  $2^7 = 128$  verschiedene Zeichen darstellen. Das reicht für das Alphabet mit allen englischen Buchstaben in Gross- und

Kleinschreibung, Satzzeichen, Ziffern und hat dann sogar noch etwas Platz. Dieser Platz wurde für Kontrollzeichen verwendet um die Kommunikation zu vereinfachen. Mit ASCII war und ist die Welt noch in Ordnung, die meisten neueren Encodings sind in den ersten 128 Zeichen kompatibel zu ASCII. Abbildung 1.1 zeigt die Zuordnung.

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	Space	!	64	40	100	@	0	96	60	140	96	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	97	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	98	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	99	c
4	4	004	EOT (end of transmission)	36	24	044	\$	\$	68	44	104	D	D	100	64	144	100	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	101	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	102	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	103	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	104	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	105	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	106	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	107	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	108	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	109	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	110	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	111	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	112	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	113	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	114	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	115	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	116	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	117	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	118	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	119	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	120	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	121	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	122	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	123	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174	124	
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	125	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	126	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	127	DEL

Source: [www.LookupTables.com](http://www.LookupTables.com)

Abbildung 1.1: Encoding nach ASCII

Wird nun der Text Hello! in ASCII als Email übertragen oder als Datei abgespeichert, so wird folgendes Bitmuster geschrieben (die Abstände dienen ausschliesslich der Lesbarkeit):

```
1001000 1100101 1101100 1101100 1101111 0100001
```

## 1.2 8-Bit Encodings

Kurz nach der Festlegung des ASCII-Codes wurde von der englischsprachigen Bevölkerung der Rest der Welt entdeckt. Ganz erschrocken haben sie festgestellt, dass die Deutschsprachigen Umlaute wie ä, ö und ü (Ä, Ö, Ü), komische Zeichen wie das sz (ß) verwenden, die Franzosen und Nordländer ihre Buchstaben mit Strichen und Kreisen verzieren (é, ô, ø, ...). Das war bei ASCII schlicht und einfach nicht drin, und genügend Platz hatte es auch nicht.

Die "Lösung" war, dass man ein Bit mehr spendierte (was 128 zusätzliche Zeichen bedeutete). Diese reichten aber nicht aus, um alle Landesspezifischen Symbole unterzubringen. Ab diesem Zeitpunkt hatte jede Region ihr eigenes Encoding und Zustände wie beim Turm zu Babel waren vorprogrammiert.

### 1.2.1 ISO 8859

Die ISO hat sich dem Problem angenommen und flugs den Standard ISO 8859 aus der Taufe gehoben. Dieser definiert die verschiedenen Encodings:

ISO 8859-1	Latin-1, Westeuropäisch
ISO 8859-2	Latin-2, Osteuropäisch
ISO 8859-3	Latin-3, Südeuropäisch
ISO 8859-4	Latin-4, Baltisch
ISO 8859-5	Kyrillisch
ISO 8859-6	Arabisch
ISO 8859-7	Griechisch
ISO 8859-8	Hebräisch
ISO 8859-9	Latin-5, Türkisch
ISO 8859-10	Latin-6, Nordisch
ISO 8859-11	Thai
ISO 8859-13	Latin-7, Baltisch
ISO 8859-14	Latin-8, Keltisch
ISO 8859-15	Latin-9, Westeuropäisch
ISO 8859-16	Latin-10, Südosteuropäisch

Daraus ergaben sich nun aber einige Probleme. Ist das Betriebssystem auf ISO 8859-1 konfiguriert (wie in unseren Breitengraden üblich) und man schreibt einen Text, so kann es sein dass er auf einem Computer in einem anderen Land, der auf ein anderes Encoding konfiguriert ist, die nicht-ASCII Zeichen falsch interpretiert.

Beispiel: Die Bitfolge 11100000 hat verschiedene bedeutungen in den Encodings:

**ISO 8859-1:** à

**ISO 8859-2:** í

**ISO 8859-3:** à

**ISO 8859-4:** ā

**ISO 8859-5:** p

Wenn also eine Webseite als ISO 8859-1 encodierte Bitfolge ausgeliefert wird und der Browser besteht darauf, alles als ISO 8859-5 zu interpretieren, so kann das zu Chaos im Text führen.

### 1.2.2 Windows Codepages

Eine Windowsspezialität sind die eigenen Encodings (beispielsweise Windows-1252) die *fast* den Isonormen entsprechen, aber die Steuerzeichen durch eigene, darstellbare Zeichen ersetzt haben.

## 1.3 Die Verschlimmbesserung: Unicode

Europa und Amerika haben begonnen, untereinander Dokumente auszutauschen und haben kurz darauf auch die restlichen Kontinente entdeckt. Um das Ganze nicht eskalieren zu lassen, hat man einen Strich gezogen und wollte wieder einen einheitlichen Standard, aber nicht ganz.

Der Coderaum von Unicode umfasste ursprünglich 65'536 Zeichen (UCS-2, 16 Bit oder 2 Bytes). Bald aber stellte sich dieses als unzureichend heraus. In Version 2.0 wurde der Codebereich um weitere 16 gleich grosse Bereiche, sogenannte Planes (Ebenen) erweitert. Bislang, in Unicode 5.0, sind 99'089 Zahlen individuellen Zeichen zugeordnet.

Soweit die schöne Theorie, die Praxis ist aber etwas hässlicher. Das eigentliche Encoding wurde leider nicht geregelt. Das heisst: Wie schreibe ich "das 946. Zeichen in der Plane 0"? (Übrigens ein  $\beta$ ).

Hier beginnt das Übel, denn da haben sich leider die Geister geschieden. Es gibt folgende Encodings für Unicode:

- Unicode Transformation Format (UTF), wobei UTF-8 das gebräuchlichste ist, z.B. im Internet und in fast allen Betriebssystemen. Neben UTF-8 hat UTF-16 eine grosse Bedeutung, so z.B. als Zeichencodierung in Java.
- SCSU (Standard Compression Scheme for Unicode, früher auch als RCSU ? Reuters' Compression Scheme for Unicode ? bezeichnet) ist eine Methode zur platzsparenden Speicherung, welche die Anordnung der verschiedenen Alphabete in Blöcken ausnutzt.
- UTF-EBCDIC ist eine Unicode-Erweiterung, die auf dem proprietären EBCDIC-Format von IBM-Grossrechnern aufbaut.
- Punycode dient dazu, Domainnamen mit nicht-ASCII-Zeichen zu kodieren.
- BOCU (Binary-Ordered Compression for Unicode) ist ein MIME-kompatibles Unicode-Kompressionsformat.
- Ausserdem gibt es die Formate CESU-8 und GB18030.

Hier wollen wir uns nur auf zwei Encodings beschränken.

### 1.3.1 UTF-16

UTF-16 ist das älteste aller Unicode-Kodierungsverfahren. Es codiert die meistbenutzten Zeichen mit 2 Bytes (16 Bits) und weniger oft benutzte mit 4 Bytes (32 Bits). Durch einen etwas komplexen Algorithmus werden die Zeichen markiert die 4 Bytes brauchen. Das decoden ist also nicht unbedingt trivial.

### 1.3.2 UTF-8

UTF-8 kann ein Unicodezeichen in 1 bis 4 Bytes ablegen. Die ersten 128 Zeichen entsprechen der ASCII-Codierung, darum hat man damit überhaupt keine Probleme. Man beachte: Hier sind sich ISO 8859, Windows und UTF-8 einig und darum sind diese Zeichen sehr gutmütig. Die restlichen Zeichen werden in speziell markierte Sequenzen von 2 bis 4 Bytes verpackt.

## Kapitel 2

# Encodierung am LMS

## 2.1 Komponenten

### 2.1.1 SVN

SVN speichert verschiedene Metadaten (*nicht* Dateinhalte) als UTF-8. Unter Anderem auch Pfadnamen. Viele Clients sind nicht auf Unicode UTF-8, sondern auf Windows Codepages oder ISO-8854 konfiguriert. SVN bemüht sich, die Zeichen umzuwandeln. Zitat aus der Doku:

Subversion internally handles certain bits of data – for example, property names, path names, and log messages – as UTF-8 encoded Unicode. This is not to say that all your interactions with Subversion must involve UTF-8, though. As a general rule, Subversion clients will gracefully and transparently handle conversions between UTF-8 and the encoding system in use on your computer, if such a conversion can meaningfully be done (which is the case for most common encodings in use today).

### 2.1.2 PHP

PHP hat keine, in die Basis eingebaute Unterstützung für Unicode. Jedes Zeichen hat 8 Bit und PHP ist es egal, was diese Bits bedeuten. Auf Geheiss gibt es sie aus und überlässt es der darstellenden Instanz irgendein Zeichen für den Code zu setzen.

### 2.1.3 Dateisystem

Auch dem Dateisystem ist das Encoding letztendlich egal. Es speichert einfach die Bitfolgen als Namen und überlässt es auch der darstellenden Instanz ein Zeichen auszuwählen.

### 2.1.4 Webserver

Apache möchte bzw. sollte dem Webbrowser einen Hinweis geben, wie er die Bitfolgen in darstellbare Zeichen übersetzen kann. Er macht dies mittels HTTP und sendet folgenden header auf moodle.bbbaden.ch:

```
Content-Type: text/html; charset=utf-8
```

### 2.1.5 Webseiten

Die Webseiten können sich selber nochmals ein Encoding zuweisen<sup>1</sup>. Die Moodleseiten machen dies mit der HTML-Zeile:

```
<meta http-equiv="content-type" content="text/html; charset=utf-8" />
```

### 2.1.6 Browser

Der Browser entscheidet nun nach Lust und Laune wie er die über das Netz empfangenen Bitfolgen interpretieren und darstellen soll. Glücklicherweise lassen sich die meisten Browser von der HTTP und HTML-Vorgabe überzeugen und interpretieren nach UTF-8.

## 2.2 Analyse

### 2.2.1 Ausgangsdaten

Folgender Kurs kann Umlaute korrekt darstellen:

**Name:** ET-AU-AC

**URL:** – Link gesnippt für Blog-Version –

**SVN:** Automatik > Elektrotechnik-Wechselstrom

Folgender Kurs kann Umlaute *nicht* korrekt darstellen:

**Name:** ET-AU-DC

**URL:** – Link gesnippt für Blog-Version –

**SVN:** Automatik > Elektrotechnik-Grundlagen

### 2.2.2 Erste Analyse

Wird im Kurs ET-AU-DC im Browser auf das ISO-8859-1 Encoding umgeschaltet, so werden alle Umlaute korrekt dargestellt.

Dies bestätigt ebenfalls ein Browsen unter – Link gesnippt für Blog-Version – welches direkt die ausgecheckten Verzeichnisse zeigt. Ebenfalls unterscheidet sich die Codierung beim Betrachten von der Kommandozeile aus. Diese hat allerdings mühe mit dem Unicode-Repository, weil sie auf ISO-8859-1 eingestellt ist (locales, aber das ist ein anderes Thema...).

Das Arbeitsverzeichnis Automatik > Elektrotechnik-Grundlagen muss also mit ISO-8859-1 Encoding ausgecheckt worden sein.

Um Verzeichnisse auf der Kommandozeile mit UTF-8 auszuchecken muss folgendes eingegeben werden:

```
export LC_ALL=de_CH.utf8
export LANG=de_CH.utf8
```

---

<sup>1</sup> Was die Frage aufwirft, wie der Inhalt interpretiert werden kann solange das Encoding noch nicht bekannt ist. Wenn also in einem Brief auf hebräisch ganz oben in hebräisch steht, dass der Brief auf hebräisch geschrieben wurde ist es etwas sinnlos. Naja, anscheinend vertrauen auch hier alle auf ASCII...

Von der Moodle-SVN Konsole aus lassen sich die Verzeichnisse als UTF-8 auschecken.

Was zum falschen Encoding auf Dateisystemebene geführt hat, lässt sich im Nachhinein nicht mehr rekonstruieren. Auschecken mit falschen Locales auf dem Server, ein verwirrtes SVN, das nicht richtig konvertiert hat oder missgelaunte Encoding-Götter, wer weiss..?

## 2.3 Reparatur

Zuerst immer alle Arbeitsverzeichnisse aus dem SVN mit UTF-8 auschecken.

Der Download hat noch ein paar Spezialitäten. Der MSIE akzeptiert Dateinamen nur als ISO oder URL-Encoded, Mozilla hingegen als UTF-8 direkt. Will man also Dateien mit Umlauten richtig downloaden so muss man noch folgende Codeänderung einbauen:

Datei `lib/filelib.php`, Funktion `send_file`:

```
1  if (check_browser_version('MSIE')) {
2      $dfilename=urlencode($filename);
3  } else {
4      $dfilename=$filename;
5  }
6
7  if ($forcedownload) {
8      @header('Content-Disposition:_attachment;_filename="'. $dfilename. '");
9  } else {
10     @header('Content-Disposition:_inline;_filename="'. $dfilename. '");
11 }
```