UTF-8 Kodierung sicherstellen

== Was wir wollen ==
Die Anforderung ist einfach. Wir wollen einen UTF-8 String erhalten - egal welche Zeichenkodierung die Quelle hatte.

Entweder der String ist korrekt und sieht folgendermaßen aus:
Quellcode

```
1. $string_correct = "umlaute läuten die wende ein.";
```

Oder er ist falsch und sieht zum Beispiel so aus:
Quellcode

```
1. $string_wrong = "umlaute l•uten die wende ein.";
```

== Anwenden der Funktion utf8_encode() auf fehlerhafte Kodierung ==
Wenn wir wissen, dass die Eingabe fehlerhaft kodiert ist, dann fällt die Wahl einfach. Wir wenden die Funktion utf8_encode() also nur auf den falschen String an:
Quellcode

```
1. $string_wrong = "umlaute l•uten die wende ein.";
2. echo utf8_encode($string_wrong);
3. //Ausgabe = umlaute läuten die wende ein.
```

== Anwenden der Funktion utf8_encode() auf korrekte Kodierung ==
Doch was geschieht eigentlich wenn wir die Funktion utf8_encode() auf eine Zeichenkette anwenden, die bereits in UTF-8 vorliegt? Man könnte erwarten, dass wir weiterhin einen gültigen UTF-8 String erhalten, doch leider ist dem nicht so.
Quellcode

```
1. $string_correct = 'umlaute läuten die wende ein.';
2. echo utf8_encode($string_correct);
3. //Ausgabe = umlaute lÃ¤uten die wende ein.
```

== Erkennen ob UTF-8 verwendet wird ==
Das Problem beschränkt sich also darauf zu erkennen ob der String in UTF-8 vorliegt. Dankenswerterweise liefert uns das PHP Manual die Lösung dazu und sagt uns welche Bit die Funktion utf8 selbst manipuliert.
easy-coding.de/Attachment/490/…1ca5d9e27a7998c8f1546ebc6

Wir erstellen also die folgende Funktion um zu erkennen ob es sich bei einem String um einen UTF-8 String handelt - die Lösung wurde bereits im Jahre 2004 von "bmorel at ssi dot fr" gepostet.
Quellcode

```
1. function seems_utf8($Str) {
2. for ($i=0; $i<strlen($Str); $i++) {
3. if (ord($Str[$i]) < 0x80) continue; # 0bbbbbbb
4. else if ((ord($Str[$i]) & 0xE0) == 0xC0) $n=1; # 110bbbbb
5. else if ((ord($Str[$i]) & 0xF0) == 0xE0) $n=2; # 1110bbbb
6. else if ((ord($Str[$i]) & 0xF8) == 0xF0) $n=3; # 11110bbb
7. else if ((ord($Str[$i]) & 0xFC) == 0xF8) $n=4; # 111110bb
8. else if ((ord($Str[$i]) & 0xFE) == 0xFC) $n=5; # 1111110b
```

```
else return false; // Does not match any model
for ($j=0; $j<$n; $j++) {
// n bytes matching 10bbbbbb follow ?
if ((++$i == strlen($Str)) || ((ord($Str[$i]) & 0xC0) != 0x80)) {
return false;
}
}
}
return true;
}
```

Alles anzeigen

== Implementierung und Test ==
Wir können die Funktion beliebig oft auf falsche und richtige String anwenden. Wir erhalten immer einen korrekten UTF-8-String

Quellcode

```
$string_correct = utf8_ensure($string_wrong);
echo $string_correct;
echo utf8_ensure(utf8_ensure(utf8_ensure($string_correct)));
function seems_utf8($Str) {
for ($i=0; $i<strlen($Str); $i++) {
if (ord($Str[$i]) < 0x80) continue; # 0bbbbbbb
else if ((ord($Str[$i]) & 0xE0) == 0xC0) $n=1; # 110bbbbb
else if ((ord($Str[$i]) & 0xF0) == 0xE0) $n=2; # 1110bbbb
else if ((ord($Str[$i]) & 0xF8) == 0xF0) $n=3; # 11110bbb
else if ((ord($Str[$i]) & 0xFC) == 0xF8) $n=4; # 111110bb
else if ((ord($Str[$i]) & 0xFE) == 0xFC) $n=5; # 1111110b
else return false; // Does not match any model
for ($j=0; $j<$n; $j++) {
// n bytes matching 10bbbbbb follow ?
if ((++$i == strlen($Str)) || ((ord($Str[$i]) & 0xC0) != 0x80)) {
return false;
}
}
}
return true;
}
function utf8_ensure($str) {
return seems_utf8($str)? $str: utf8_encode($str);
}
```

Alles anzeigen

== Aktuelle Vorgehensweise ==
Mit den PHP 4 Versionen >= 4.4.3 und den PHP 5 Versionen >= 5.1.3 wurde eine neue Methode eingeführt, die uns das lange Script erleichtert.
Wenn garantiert werden kann, dass diese Versionen installiert sind, kann daher folgende Version verwendet werden:

Quellcode

```
function utf8_ensure($str) {
return mb_check_encoding($str, 'UTF-8') ? $str : mb_convert_encoding($str, 'UTF-8', 'auto');
}
```