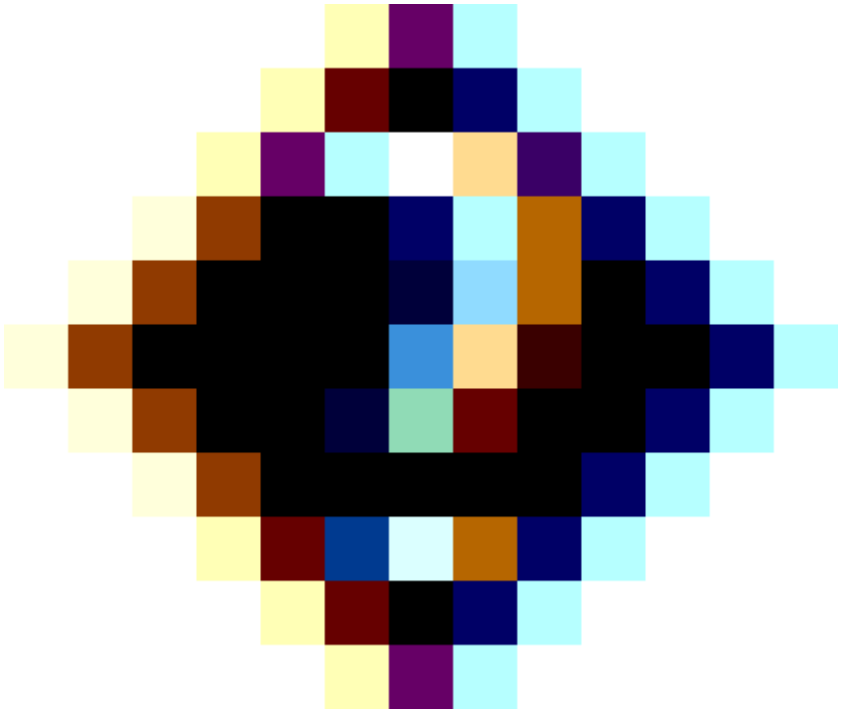
Unicode, PHP, and Character Set Collisions



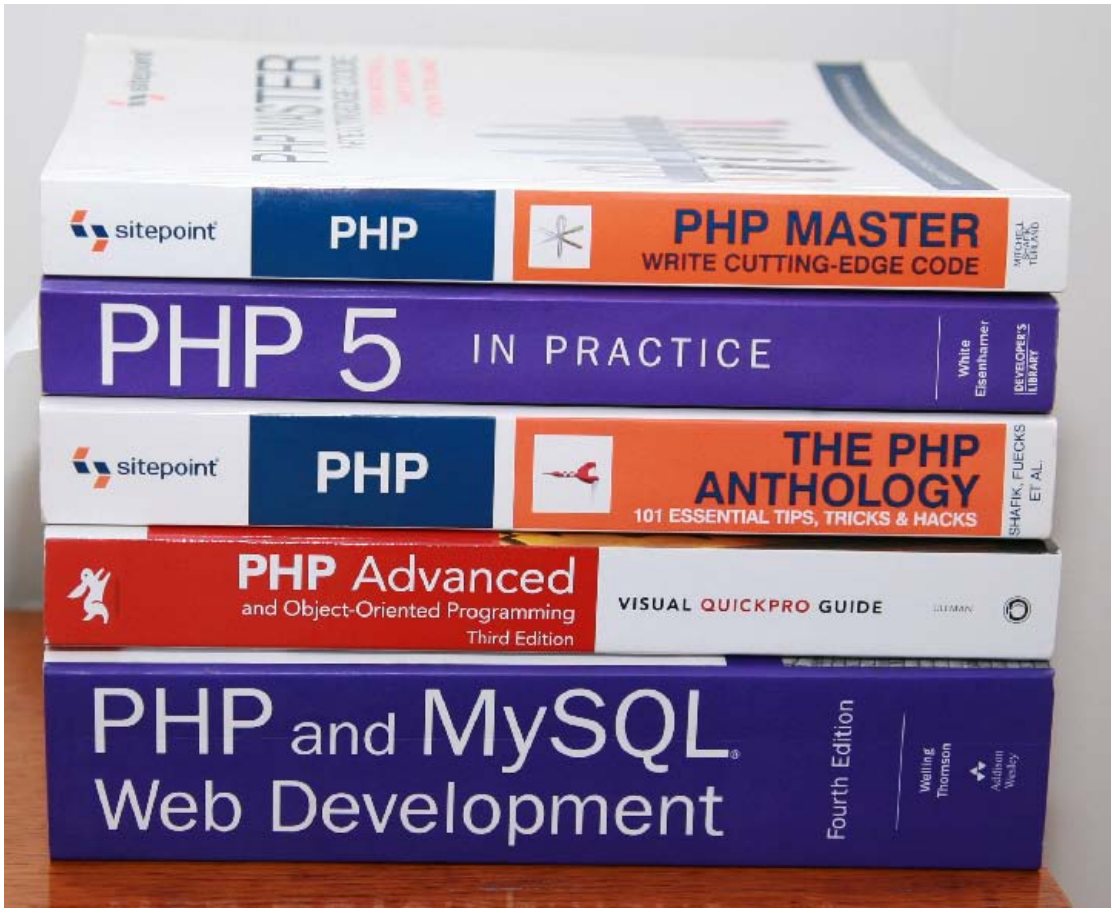
Ray Paseur

DC PHP Developers' Community

10th September 2014

Unicode, PHP, and Character Set Collisions

Some Books That Do Not Cover This Topic



Unicode, PHP, and Character Set Collisions

Signature of a Character Encoding Collision

The browser renders these oddities:

- Question marks inside black diamonds
- Inverted question marks
- Ã (the A-Tilde), or…
- Å (the A-Ring), plus…
- some drivel

Unicode, PHP, and Character Set Collisions

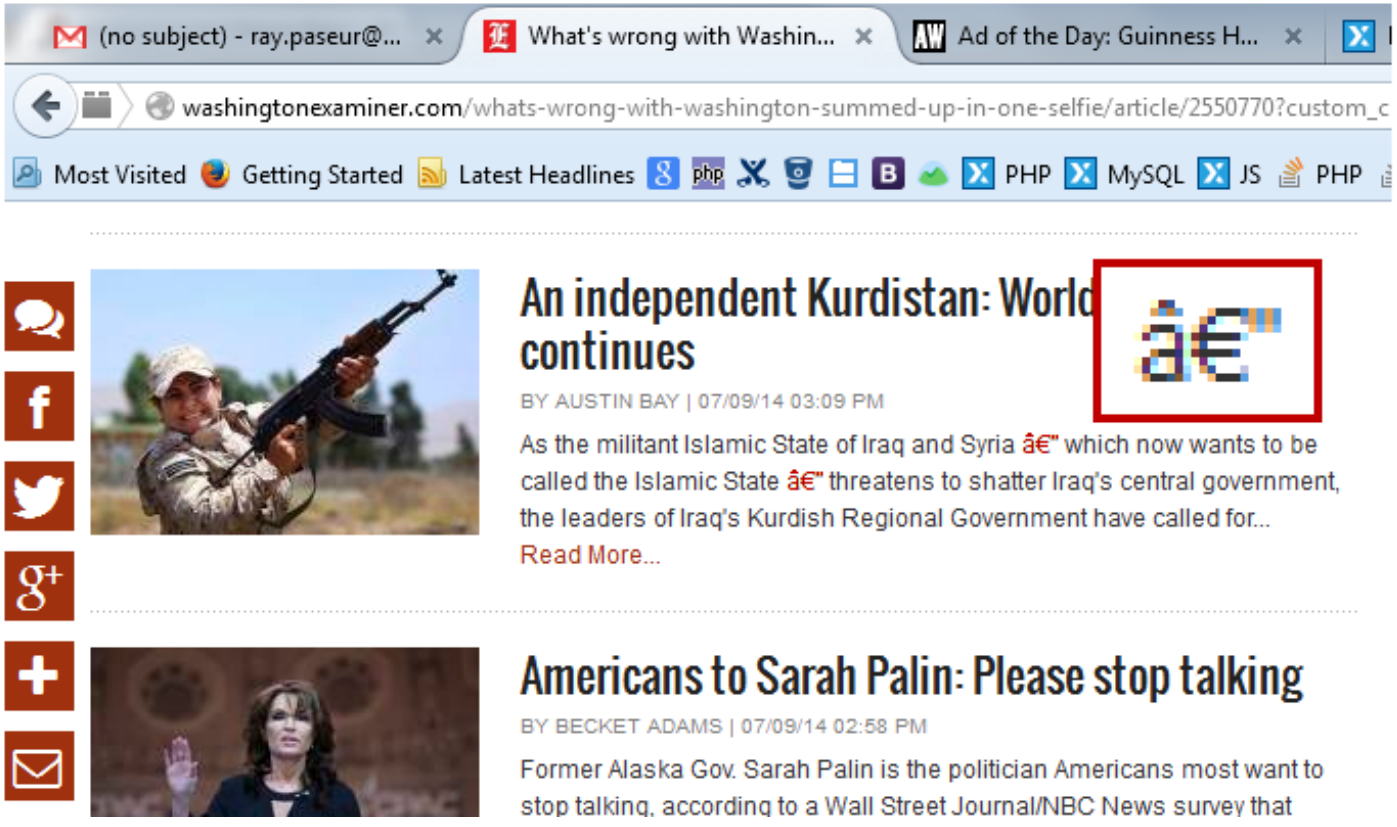
Signature of a Character Encoding Collision

If your <meta charset> matches your data, things usually work out well. However, if there is a mismatch…

Consistently ISO *or* UTF-8	ISO-8859 Data Browser UTF-8	UTF-8 Data Browser ISO-8859
Françoise	Fran&#65533;oise	FranÃ§oise
Å-Ring	&#65533;-Ring	Ã…-Ring
ßeta or Beta?	&#65533;eta or Beta?	ÃŸeta or Beta?
Öh löök, umlauts!	&#65533;h l&#65533;&#65533;k, umlauts!	Ã–h lÃ¶Ã¶k, umlauts!
ENCYCLOPÆDIA	ENCYCLOP&#65533;DIA	ENCYCLOPÃ†DIA
A stealthy *ƒ*art	A stealthy &#65533;art	A stealthy Æ’art
Đe lónlí blú bojs	&#65533;e l&#65533;nl&#65533; bl&#65533; bojs	Ã&#65533;e lÃ³nlÃ blÃº bojs

Unicode, PHP, and Character Set Collisions

Signature of a Character Encoding Collision



Unicode, PHP, and Character Set Collisions

Brief History of Character Encoding

American Standard Code for Information Interchange

- 127 ASCII Characters 0000 0000 – 0111 1111
- 256 Extended ASCII Chars 1000 0000 – 1111 1111
  - Printable glyphs (mid-1980's)
  - Incompatible “standards”

¿ What of those funny ácceñted chars ?

- Emergence of Latin-1, ISO-8859-1, Windows-1252
- Gobbled up all the code points above *7F*
- Oops. What about the €uro? Maastricht 1992

Unicode, PHP, and Character Set Collisions

Brief History of Character Encoding

Realities of the 1990's

- Extended ASCII was adequate for most Western text
- Nascent WWW began to connect societies
- PHP was born with this in mind:

"A string is series of characters, where a character is the same as a byte. This means that PHP only supports a 256-character set, and hence does not offer native Unicode support."

Unicode, PHP, and Character Set Collisions

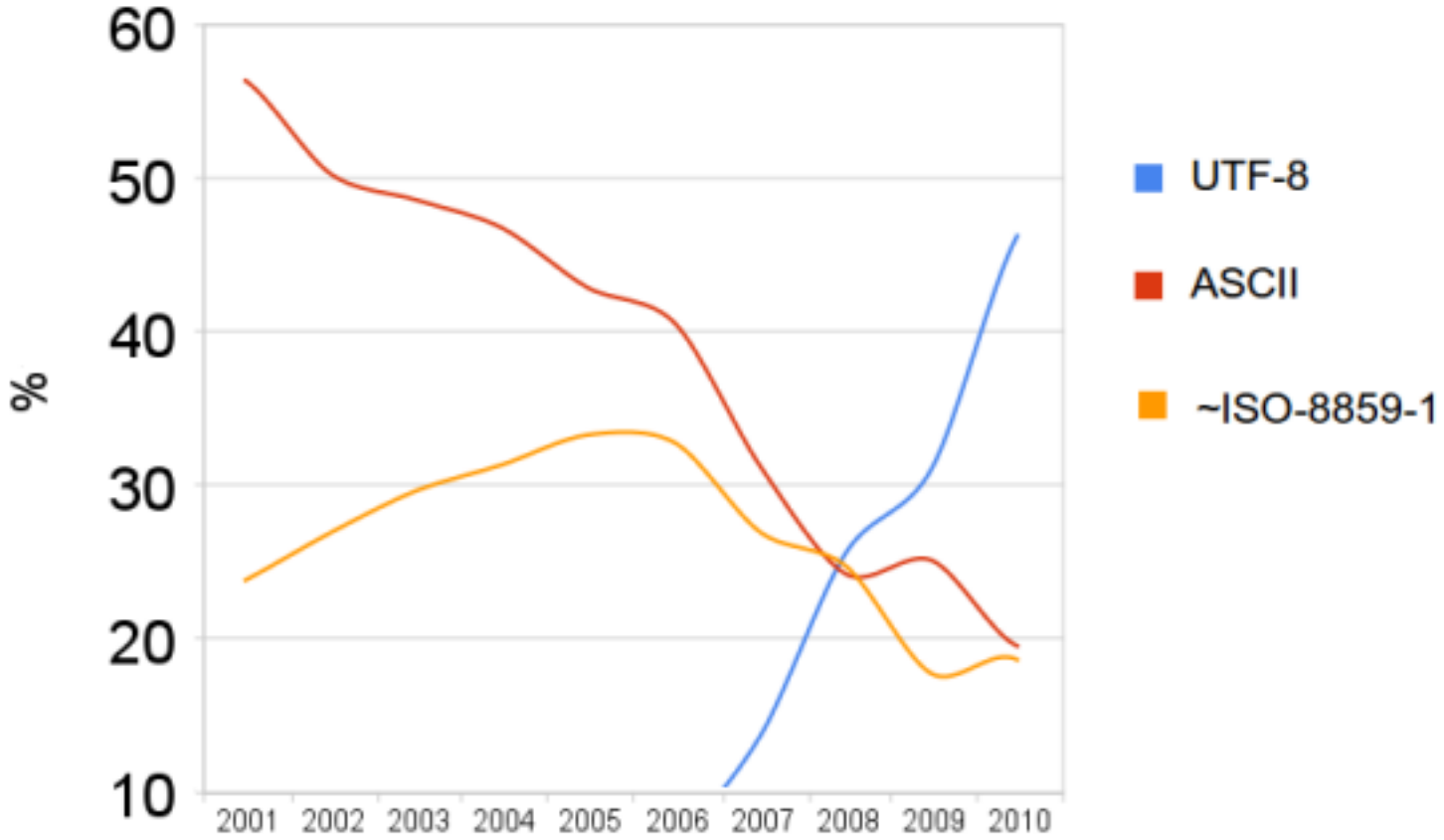
Brief History of Character Encoding

Realities of the 1990's and beyond

- Overwhelming dependence on (Extended) ASCII
- Many conflicting and unwieldy encoding schemes
- Byte-Order Marks and Endianness
- 1992: Thompson and Pike described UTF-8
- 2003: RFC 3629 – UTF-8 allowed 1,000,000+ chars
- 2006: UTF-8 "took off"
- 2008: UTF-8 Most. Popular. Encoding. Ever.

Unicode, PHP, and Character Set Collisions

Growth of UTF-8 on the Web



Unicode, PHP, and Character Set Collisions

Genius of UTF-8 Encoding

All one-byte ASCII Characters Preserved 1:1

Self-Evident with no BOM or Endian

UTF-8 Encoding

bytes	bits*	representation			
1	7	0bbb bbbb			
2	11	110b bbbb	10bb bbbb		
3	16	1110 bbbb	10bb bbbb	10bb bbbb	
4	21	1111 0bbb	10bb bbbb	10bb bbbb	10bb bbbb

***bits used in character, aside from the UTF-8 signal bits**

2^7 = 128 chars. 2^8 = 256 chars. *2^21 = 2.1MM chars.*

But…

Unicode, PHP, and Character Set Collisions

Downside of UTF-8 Encoding

If a byte has the high-order bit **on**, the byte is part of a UTF-8 multi-byte character.

Ergo: All one-byte Extended ASCII characters are lost.

"A string is series of characters, where a character is the same as a byte. This means that PHP only supports a 256-character set, and hence does not offer native Unicode support."

Unicode, PHP, and Character Set Collisions

Most Common PHP UTF-8 Encoding Issues

Western-European accented chars stored in ISO-8859-1

Example: ***Æ*** (AE Ligature) character

- decimal code point 198, hex ***C6***, binary 1100 0110
- Two high-order bits imply a two-byte UTF-8 character
- UTF-8 AE Ligature is hexadecimal ***C386***

Similar collisions occur with accents, umlauts, tildes, rings and some currency symbols

Unicode, PHP, and Character Set Collisions

The PHP Recondite Conundrum

In UTF-8 *a character is **not the same** as a byte*!

PHP does not dictate a specific encoding for strings

- Is the string **á** one-byte hex ***E1*** (ISO-8859-1)?
- Is the string **á** two-byte hex ***C3A1*** (UTF-8)?

It depends! What character encoding *was in use* at the time the string literal was created? Check your IDE or Editor settings. PHP mb_detect_encoding() knows **á** is not an ASCII character, but will be unable to distinguish between ISO-8859-1 and UTF-8.

Unicode, PHP, and Character Set Collisions

Changing Posture at Release 5.4+

PHP htmlentities(), htmlspecialchars() default charset

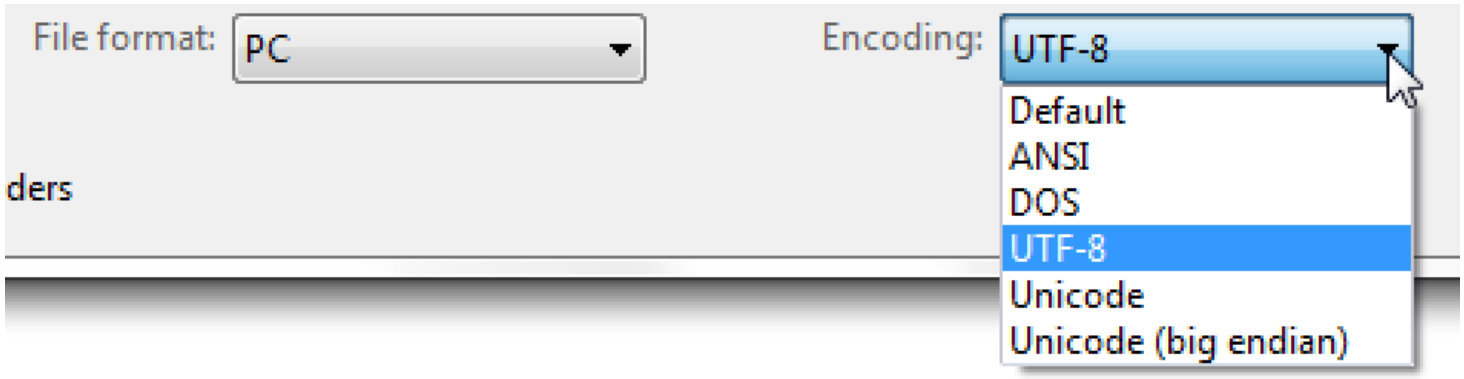
- "Optional" 3rd argument for default charset
- PHP < 5.4.0 = ISO-8859-1
- PHP >= 5.4.0 = UTF-8
- PHP >= 5.6.0 = configuration option (Sheesh!)

"Although this argument is *technically optional*, you are **highly encouraged to specify** the correct value for your code." How many of your code points does this touch?

Unicode, PHP, and Character Set Collisions

Confusing the Browsers with *Charset*

Your text editor / IDE settings are in play



Unicode, PHP, and Character Set Collisions

Converting Existing Data

PHP utf8_encode() converts single-byte characters above code point 127 to UTF-8 multibyte characters*

PHP strlen() increases because it is a byte count

PHP mb_strlen() returns character count

http://php.net/manual/en/ref.mbstring.php

* This is a mung and cannot be re-run on the same string

Unicode, PHP, and Character Set Collisions

Converting Existing Data

PHP utf8_decode() *tries* to convert UTF-8 characters to their ISO-8859-1 equivalents. Does not always work – many more UTF-8 characters!

Failure results include missing or garbled text.

€ may be best represented by **€** and **ƒ** by **ƒ**

See list of named HTML entities in the article.

Unicode, PHP, and Character Set Collisions

Converting Existing Data

Assumptions:

- UTF8_decode() *assumes* input is UTF-8
- UTF8_encode() *assumes* input is ISO-8859-1
- Where does that leave Windows-1252?
- Maybe worth considering iconv()?

PHP substr() may split multibyte characters.

Use mb_substr() instead. How many code points…?

Unicode, PHP, and Character Set Collisions

The BOM is not *Da Bomb*

Byte Order Marks are out-of-place in UTF-8 documents

Unfortunately, Notepad® may create and insert BOM

UTF-8 BOM hex value: **EF BB BF**

UTF-8 BOM browser visual: **ï»¿** at the start of string

```
if(substr($data, 0, 3) == pack('CCC', 239, 187, 191))
```

Unicode, PHP, and Character Set Collisions

Character Sets in MySQL

```
mysql_set_charset(‘utf8mb4’);

$mysqli = new mysqli(‘localhost’, ‘user’, ‘pass’, ‘dbn’);
$mysqli->set_charset(‘utf8mb4’);

$pdo = new PDO(
‘mysql:host=localhost;dbname=dbn;charset=utf8mb4’,
‘user', ‘pass');
```

Unicode, PHP, and Character Set Collisions

Converting Existing Tables in MySQL

May want to use ALTER TABLE to widen the columns?

Charset name is *not* **utf8** – only 3 characters
Charset name is **utf8mb4** – gives 4 characters

MySQL can return the right characters, even if you have Extended ASCII in the database

Run the query after *set_charset()*. Query, *set_charset(), data_seek()* will return old Latin1 data set

Unicode, PHP, and Character Set Collisions

Loading tables with PDO

Unlike MySQLi, you *must* use UTF-8 input data

With Extended ASCII input data, data loss occurs

Column is truncated at first invalid UTF-8 character

Silent: No errors or exceptions

Unicode, PHP, and Character Set Collisions

Microsoft “Productivity” Software

General Assumptions:

- Your data is ASCII
- You want it rendered in CP-1252
- If you say “Unicode” you mean UTF-16

Expected (but unwanted) results:

- UTF-8 characters will be garbled

Watch out for **.csv** files destined for Excel spreadsheets

Unicode, PHP, and Character Set Collisions

Microsoft "Productivity" Software

Excel does not recognize UTF-8 data (?)

…Unless you tell Excel that it's UTF-8 data (?)

…But UTF-8 is self-evident (?)

Princeton University (home to Einstein) says:
http://www.itg.ias.edu/content/how-import-csv-file-uses-utf-8-character-encoding-0

Unicode, PHP, and Character Set Collisions

Microsoft “Productivity” Software

Actual quotes from a support dialog:

“I would like to save a csv file from an Excel 2013 sheet with utf-8 encoding. Does someone know how to do this?” - *Peter*

“I would need more details to assist you: What difference are you trying to achieve…” - *Aravinda*

Unicode, PHP, and Character Set Collisions

Microsoft “Productivity” Software

Copy **Word®** / Paste into HTML **<textarea>**

Usually OK if you’re rendering ISO-8859-1

Usually garbled if you’re rendering UTF-8

“Fixed” by PHP get_html_translation_table() with named character entities. Eg: “ becomes ***«***

But strlen() increases. Check SQL column widths

Unicode, PHP, and Character Set Collisions

Text Editors

Symptom:

The requested URL
/genealogy/Letters/Ã¢â‚¬Å“1890-Dec-7.jpg"
was not found on this server

Cause:

Original

Diagnosis:

TextWrangler uses Microsoft-style quotes

Unicode, PHP, and Character Set Collisions

Malformed Characters in JSON Strings

json_decode() returns NULL

PHP5.3 json_last_error()
PHP5.5 json_last_error_message()
JSON_ERROR_UTF8 (Malformed UTF-8 characters possibly incorrectly encoded)

Locate the bad character(s)?

Unicode, PHP, and Character Set Collisions

Malformed Characters in JSON Strings

Locate the bad character(s)?

```
1...5...10...15...20...25...30
{"A1":"  ","A2":"$¢€₹!"}
7243232c222432322cae8afaaa227
b2112a222c2122a242222c04d212d
```

But what if the JSON string is thousands of bytes long?

Unicode, PHP, and Character Set Collisions

Malformed Characters in JSON strings

```
POSSIBLE UTF-8 ERRORS
BYTE: 7, CHR: Â, ORD: 194

ENTIRE STRING IN SINGLE BYTES
BYTE: 0, CHR: {, ORD: 123, HEX: 7B
BYTE: 1, CHR: ", ORD: 34, HEX: 22
BYTE: 2, CHR: A, ORD: 65, HEX: 41
BYTE: 3, CHR: 1, ORD: 49, HEX: 31
BYTE: 4, CHR: ", ORD: 34, HEX: 22
BYTE: 5, CHR: :, ORD: 58, HEX: 3A
BYTE: 6, CHR: ", ORD: 34, HEX: 22
BYTE: 7, CHR: Â, ORD: 194, HEX: C2, BIN: 11000010, ERROR IN BYTE 8: 00100010
BYTE: 8, CHR: ", ORD: 34, HEX: 22
BYTE: 9, CHR: ,, ORD: 44, HEX: 2C
BYTE: 10, CHR: ", ORD: 34, HEX: 22
BYTE: 11, CHR: A, ORD: 65, HEX: 41
BYTE: 12, CHR: 2, ORD: 50, HEX: 32
BYTE: 13, CHR: ", ORD: 34, HEX: 22
BYTE: 14, CHR: :, ORD: 58, HEX: 3A
BYTE: 15, CHR: ", ORD: 34, HEX: 22
BYTE: 16, CHR: $, ORD: 36, HEX: 24
BYTE: 17, CHR: Â, ORD: 194, HEX: C2, BIN: 11000010
BYTE: 18, CHR: ¢, ORD: 162, HEX: A2, BIN: 10100010
BYTE: 19, CHR: â, ORD: 226, HEX: E2, BIN: 11100010
BYTE: 20, CHR: ‚, ORD: 130, HEX: 82, BIN: 10000010
BYTE: 21, CHR: ¬, ORD: 172, HEX: AC, BIN: 10101100
BYTE: 22, CHR: ð, ORD: 240, HEX: F0, BIN: 11110000
BYTE: 23, CHR: ¤, ORD: 164, HEX: A4, BIN: 10100100
BYTE: 24, CHR: , ORD: 173, HEX: AD, BIN: 10101101
BYTE: 25, CHR: ¢, ORD: 162, HEX: A2, BIN: 10100010
BYTE: 26, CHR: !, ORD: 33, HEX: 21
BYTE: 27, CHR: ", ORD: 34, HEX: 22
BYTE: 28, CHR: }, ORD: 125, HEX: 7D
```

Unicode, PHP, and Character Set Collisions

Summary

http://www.reddit.com/r/MapPorn/comments/1dqh7d/after_seeing_a_recent_post_about_the_population/

Unicode, PHP, and Character Set Collisions

http://iconoun.com/articles/collisions/