Fighting charsets and encodings

- Kore Nordmann
- International PHP Conference
  - Mainz, 29.10.08

http://kore-nordmann.de/portfolio.html
Contact: kore@php.net

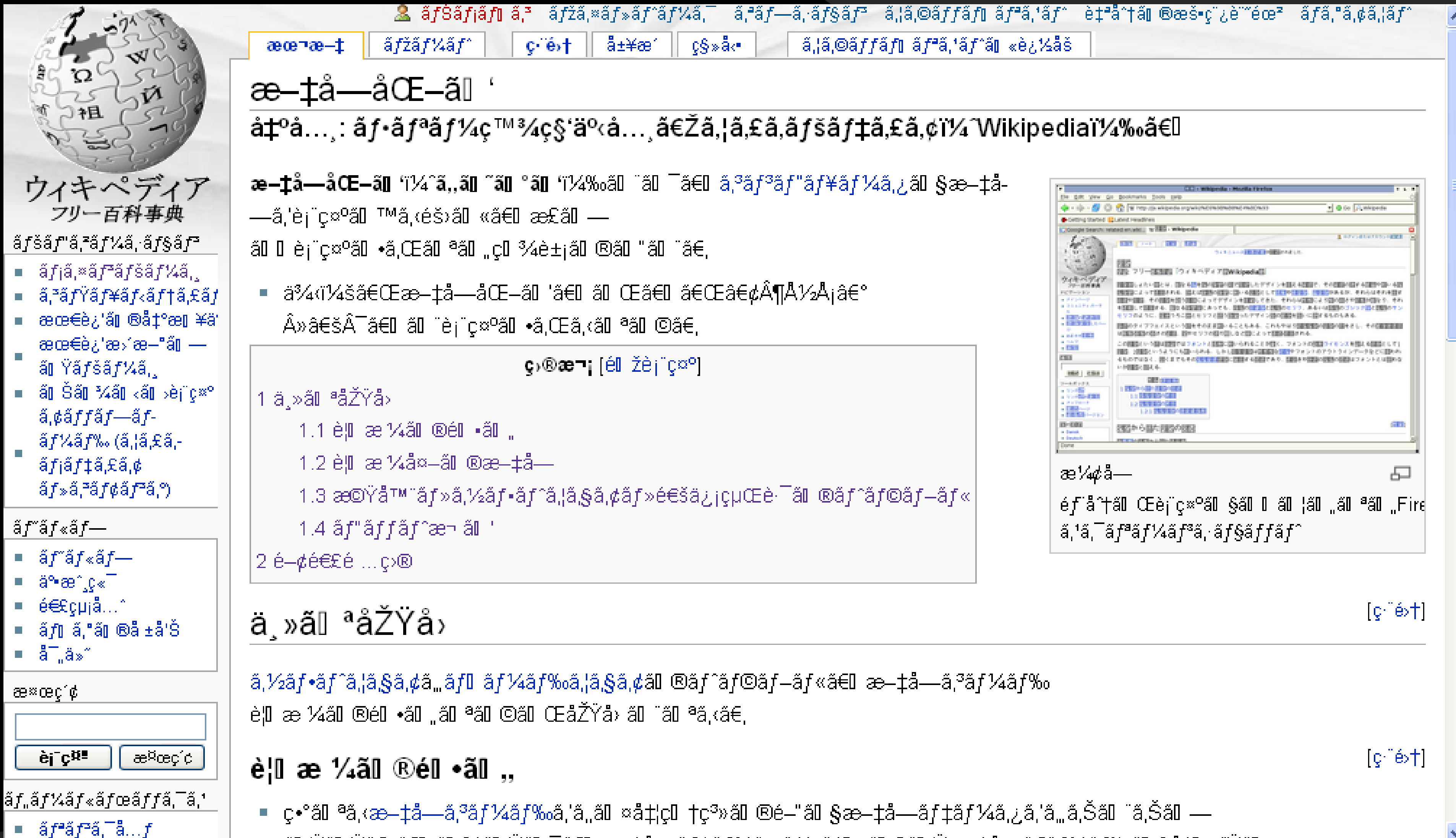
About me

- Kore Nordmann <kore@php.net>
- Long time PHP developer
- Regular speaker, author, etc.
- Studies computer science in Dortmund
- Consultancy and software development
- Active open source developer:
  - eZ Components, arbit, PHPUnit, Torii, PHPillow, KaForkL, Image_3D, WCV, ...

Agenda

- Terms
- Usage in PHP
- HTTP & HTML
- Databases

The problem



http://kore-nordmann.de/portfolio.html
Contact: kore@php.net

Terms

- Character vs. Byte
  - Character: a
  - Byte : 01100001, \x61
- Character set
  - Defines a set of characters
- Encoding
  - Defines a mapping from characters to byte(s)
- Collations

Multibyte encoding

- Multibyte encodings map a character to ***multiple*** bytes
  - Character: ☯
  - UTF-8 encoded: 0xE2 0x98 0xAF
  - UTF-16 encoded: 0x26 0x2F

http://kore-nordmann.de/portfolio.html
Contact: kore@php.net

Unicode vs. UTF-8 / UTF-16 / …

- Unicode collects „all“ characters worldwide
  - a-Z     
- Unicode encodings: UTF-8, UTF-16, UTF-32, UCS2, UCS4, …
  - UTF-* differ in the amount of used bytes

Why is UTF-8 so popular?

- Same encoding for ASCII character set
  - ISO-8859-* compatible for \x00-\x7f
- Variable length results in low overhead for common western texts
- Encodes full unicode

How to determine the encoding?

- What is the encoding of some string?
  - Theoretically ***impossible*** to know.
  - There are several single byte encodings using the full byte range.
  - Multibyte encodings like UTF-16 or UCS2 use the full byte range.

How to guess the encoding.

- If you know the content...
  - Perform statistical analysis on byte sequences
- Check for UTF-8 characteristics:
  - ```
    (^(?:
     [\x00-\x7f] |
     [\xc0-\xdf][\x80-\xff] |
     [\xe0-\xef][\x80-\xff]{2} |
     [\xf0-\xf7][\x80-\xff]{3}
    )*$)x
    ```
    ```

String transliteration

- Required for conversions between charsets
  - Destination charset knows less characters the source charset
    - f.e.: UTF-8 => ASCII
- Example:
  - ```
    <?php
    $string = "Some string: äöü?";
    echo iconv( "UTF-8", "ASCII//TRANSLIT",
                $string );
    // Output: Some string: aeoeue?
    ```
    ```

Usage in PHP

- Encoding of strings in PHP?
  - **None.**

Encoding in PHP

- No encoding / charset information is maintained.
- Strings are arrays of bytes, *not characters*.
  - PHP 6 is totally different.

Change encoding of a string

- Problems with encoding conversions
  - Character set incompabilities
  - Invalid characters in source string
- Possibilities in PHP
  - iconv(), mb_convert_encoding()
    - Available encodings may be system dependent
  - utf8_encode() / utf8_decode()
    - Only: ISO 8859-1 ↔ UTF-8
    - No transliteration, unknown characters result in „?“

Iterating characterwise

```php
<?php
$string = 'Some string?';
$fixed  = iconv( 'UTF-8', 'UTF-32', $string );
$chars  = strlen( $fixed );

for ( $char = 0; $char < $chars; $char += 4 )
{
    echo iconv( 'UTF-32', 'UTF-8',
                substr( $fixed, $char, 4 )
            ), ' ';
}

// Output: S o m e   s t r i n g ?
```

String length

- Bytes or characters?
- Length in bytes: strlen()
- Length in chharacters:
  - int iconv_strlen (string str [, string charset])
  - int mb_strlen (string str [, string encoding])

Encodings in PHP applications

- Ensure same encoding in whole application
  - Input
  - Services
  - Storage backend
  - Output
- No „strange“ characters anymore.

HTTP & HTML

- Most common input & output layer

User input encodings

- The browser *generally* sends user input in application encoding
- Application encoding determined by:
  - ```
    header(
      'Content-Type: text/html; charset=utf-8'
    );
    ```
  - ```
    <meta
        http-equiv="Content-Type"
        content="text/html; charset=utf-8" />
    ```

Specify form encoding

- You can explicitely specify the form encoding, like:
  - `<form ... accept-charset="UTF-8">`
- But it won't help that much...
  - IE does not care at all
  - Site: UTF-8, Input encoding: ISO-8859-15
    - Opera, Firefox, Konqueror(, Lynx)
  - Manually changed site encoding still affects user input

From encoding: The example

```
<form ... accept-charset="$encoding">
   <legend>Test form using $encoding</legend>
   <input type="hidden" name="form"
          value="$encoding" />
   <input type="hidden" name="hidden_input"
          value="öäü" />
   <label>Input string
     <input type="text" name="user_input"
            value="öäüß☠ (by user)" />
   </label>
   <label>
     Submit
     <input type="submit" value="Submit" />
   </label>
</form>
```

Form encoding: The results

- Firefox/3.0.3 | ISO-8859-15 | ??? | ????☠
  Firefox/3.0.3 | UTF-16 | öäü | öäüß☠
  Firefox/3.0.3 | UTF-8 | öäü | öäüß☠
- Opera/9.61 | ISO-8859-15 | ??? | ????☠
  Opera/9.61 | UTF-16 | öäü | öäüß☠
  Opera/9.61 | UTF-8 | öäü | öäüß☠
- Konqueror/4.1 | ISO-8859-15 | ??? | ????☠
  Konqueror/4.1 | UTF-8 | öäü | öäüß☠
- MSIE 7.0 | ISO-8859-15 | öäü | öäüß☠
  MSIE 7.0 | UTF-16 | öäü | öäüß☠
  MSIE 7.0 | UTF-8 | öäü | öäüß☠
- Firefox/3.0.3 | UTF-8/ISO | öäü | ????☠
  Firefox/3.0.3 | ISO-8859-15/1 | ö?¤?¼ | ?

Handle user input

- Determine supported encodings of client
  - Accept-Charset header
    - ***Not*** Accept-Encoding
  - `iso-8859-1, utf-8, utf-16, *;q=0.1`
- Send all proper encoding headers
- Remove invalid characters in input handler
  - *Always validate user input*

Sanitize user input

- Convert to your application encoding
  - `iconv( 'utf-8', 'utf-8//IGNORE', $input );`
  - `iconv( $clientEncoding, 'utf-8//TRANSLIT', $input );`
- Might throw warnings on invalid input
- Check for same legth after conversion

HTML Output

- With unicode encodings there is no need for „HTML-Entities“
  - XML-Entities of course still required.
- htmlspecialchars() has an encoding parameter
  - ```
    <?php
    var_dump( htmlentities( 'Ü' ) );
    // string(9) "Ã"

    var_dump( htmlspecialchars( 'Ü', ENT_QUOTES,
                                'UTF-8' ) );
    // string(2) "Ü"
    ```
    ```

PHP 6 – Outlook

- Differs between unicode and binary strings
  - Internal encoding (libicu): UCS2
- Input conversions are done by PHP
- (Fast) String character iterators available
- „Most“ functions unicode aware

Databases

- Do not loose information while storing

Example: MySQL

- Most relevant, the connection encoding:
  - my.cnf
    ```
    [mysql]
    default-character-set=utf8
    ```
  - Per connection
    ```
    SET NAMES utf8
    ```
- Connection encoding specifies the query string encoding
  - MySQL internally converts to the table / column encoding

And the table encoding?

- Should match the expected character set
  - Only use multibyte encodings, if multibyte strings are really expected
    - Often not required for f.e. user names
  - Otherwise use ASCII
    - Requires much less space allocation
    - Set it per column!

Works similar in other RDBMS

- PostgreSQL
  - ```
    SET CLIENT_ENCODING TO 'UTF-8';
    SET NAMES 'UTF-8';
    ```
    ```

Resources

- Shameless plug:
  - http://kore-nordmann.de/blog/php_charset_encoding_FAQ.html

Thank you.

- Any question?
- Further remarks?

http://kore-nordmann.de/portfolio.html
Contact: kore@php.net